Clos

4D Programmiersprache

- Einleitung
- Grundbegriffe
- Debugging
- 4D Umgebung
- Arrays
- Auswahl
- Backup
- Benutzer und Gruppen
- Benutzerformulare
- Benutzeroberfläche
- Berechnungen
- Bilder
- BLOB
- Boolean
- Cache Verwaltung
- Collections
- Compiler
- Datenbankmethoden
- Datensatz sperren
- Datensätze
- Datum und Zeit
- Diagramme
- Drag and Drop
- Drucken
- Eingabe
- Eingabekontrolle
- Fenster
- Formel
- Formulare
- Formularereignisse
- Hierarchische Listen
- HTTP Client
- Import und Export
- JSON
- Kommunikation
- LDAP
- Listbox
- Mathematische Funktionen
- Mehrfachstil Text
- Meldungen
- Mengen
- Menüs
- Objekte (Formulare)
- Objekte (Sprache)
- Operatoren
- ORDA - DataClass
- ORDA - DataClassAttribute
- ORDA - DataStore
- ORDA - Entity
- ORDA - EntitySelection
- Pasteboard
- PHP
- Programmiersprache
- Prozess (Kommunikation)
- Prozesse
- Prozessoberfläche
- Rechtschreibprüfung
- Ressourcen
- Schnellbericht
- SQL
- String
- Strukturzugriff
- Suchen
- SVG
- Systemdokumente
- Systemumgebung
- Tabelle
- Temporäre Auswahl
- Transaktionen

- Trigger
- Unterbrechungen
- Untertabellen
- Variablen
- Verknüpfungen
- Verschlüsselung
- Web Area
- Web Server
- Web Services (Client)
- Web Services (Server)
- Werkzeuge
- XML
- XML DOM
- XML SAX
- Zugriff Designobjekte
- Konstantenthemen
- Fehlermeldungen
- Zeichensätze
- Neuerungen der Version
- Überholte Befehle
- Alphabetische Liste der Befehle

Einleitung

Copyrights und Rechte

Vorwort

4D hat seine eigene Programmiersprache. Diese integrierte Sprache mit über 1000 Befehlen macht 4D zu einem leistungsstarken Entwicklungswerkzeug für Datenbank-Anwendungen auf Desktop Rechnern. Sie können die 4D Programmiersprache für ganz unterschiedliche Operationen einsetzen - von der einfachen Berechnung bis hin zu individuell gestalteten komplexen Benutzeroberflächen. Sie können zum Beispiel:

- Über Programmierung den Zugriff auf die verschiedenen Editoren steuern, z.B. Sortiereditor, Menüeditor.
- Mit den Informationen aus der Datenbank komplexe Berichte und Etiketten erstellen und drucken.
- Mit anderen Systemen kommunizieren
- Dokumente verwalten,
- Daten zwischen 4D Datenbanken und anderen Anwendungen importieren und exportieren,
- Methoden aus anderen Sprachen in die 4D Programmiersprache integrieren.

4D bietet mit der umfassenden und flexiblen Programmiersprache optimale Voraussetzungen zur Entwicklung und Implementierung von Informationssystemen aller Größen. Einsteiger können mühelos Berechnungen ausführen. Versierte Benutzer können ihre Datenbank ohne Programmierkenntnisse an die eigenen Bedürfnisse anpassen. Profi-Entwickler können mit dieser leistungsstarken Programmiersprache ihren Datenbanken ausgeklügelte Funktionalitäten hinzufügen, inkl. File-Transfer und Kommunikation mit anderen Systemen. Entwickler mit Programmiererfahrung in anderen Sprachen können der 4D Programmiersprache eigene Befehle hinzufügen.

Die 4D Programmiersprache wird erweitert, sobald Sie ein 4D Plug-In in die Anwendung aufnehmen. Jedes Plug-In hat seine eigenen Programmierbefehle.

Über die Handbücher

Die hier aufgeführten Handbücher gelten für 4D und 4D Server gleichermaßen. Für 4D Server gibt es zusätzlich ein 4D Serverhandbuch, das die serverspezifischen Funktionalitäten, beschreibt.

- Das Handbuch *4D Self Training* macht Sie mit den Grundkonzepten der Datenbankentwicklung vertraut. Anhand von Beispielen erlernen Sie das Arbeiten mit 4D.
- Das Handbuch *4D Designmodus* beschreibt die Editoren und Werkzeuge auf Strukturebene sowie die Eingabe und Bearbeitung von Daten in der Datenbank.
- Das Handbuch *4D Programmiersprache* enthält alle Befehle und Funktionen von 4D mit Code-Beispielen.
- Das Handbuch *4D Server* erläutert die Verwaltung von Datenbanken im Mehrplatzbetrieb.

Über dieses Handbuch

Dieses Handbuch beschreibt die 4D Programmiersprache. Es setzt voraus, dass Sie mit Begriffen wie Tabelle, Datenfeld oder Formular vertraut sind. Bevor Sie mit diesem Handbuch arbeiten, sollten Sie die Beispiele im Handbuch *4D Self Training* durcharbeiten und Ihre eigene Datenbank erstellen. Ziehen Sie bei Bedarf das Handbuch *4D Designmodus* zu Rate. Die 4D Web Site bietet auch zahlreiche Beispiele zum Einstieg in 4D (siehe Ressourcen).

Schreibregeln

Dieses Handbuch verwendet folgende Schreibweisen:

- Analog zum 4D Methodeneditor werden Befehle in Großbuchstaben und in Sonderschrift geschrieben, z.B. **CLOSE DOCUMENT**. Funktionen, d.h. Befehle die einen Wert zurückgeben, beginnen mit einem Großbuchstaben, z.B. **Change string**.
- Parameter, die in der Befehlssyntax optional sind, werden in geschweifte Klammern gesetzt. **Beispiel: SET DEFAULT CENTURY (Jahrhundert{; Schlüsseljahr})** bedeutet, dass der Parameter **Schlüsseljahr** beim Aufrufen des Befehls wegbleiben kann.
- Das Zeichen |gibt in der Befehlssyntax eine Alternative an. **Beispiel: Table name (TabelleNum | TabellePtr)** gibt an, dass die Funktion als Parameter entweder eine Tabellennummer oder einen Zeiger akzeptiert.
- In manchen Beispielen läuft eine Code-Zeile aus Platzgründen über mehrere Zeilen. Beim Programmieren müssen solche Code-Zeilen jedoch in einer Zeile, also ohne Zeilenschaltung geschrieben werden.

Wie geht es weiter?

Benutzen Sie dieses Handbuch zum ersten Mal, lesen Sie den Abschnitt **Einführung in die Programmiersprache**.

Einführung in die Programmiersprache

Dieses Kapitel führt Sie in die 4D Programmiersprache ein. Es ist in folgende Abschnitte gegliedert:

- Definition und Verwendung der Programmiersprache,
- Methoden verwenden,
- Eine Anwendung mit 4D entwickeln.

Was ist eine Sprache?

Die Sprache von 4D unterscheidet sich kaum von der gesprochenen Sprache im Alltag. Es ist eine Kommunikationsart, um Ideen auszudrücken, Anweisungen zu erteilen und Informationen weiterzugeben. Wie die gesprochene Sprache hat 4D ein eigenes Vokabular, sowie eine eigene Grammatik und Syntax, die Sie einsetzen, um Ihre Datenbank und Ihre Daten zu verwalten.

Sie müssen nicht die komplette Sprache kennen, um effektiv mit 4D arbeiten zu können. Schon ein kleiner Teil reicht aus, um produktiv zu arbeiten. Weitere Teile können Sie je nach Bedarf dazulernen.

Warum eine Sprache verwenden?

Auf den ersten Blick erscheint der Einsatz einer Programmiersprache in 4D nicht notwendig. Die Designumgebung bietet flexible Tools, mit denen sich ohne Programmierung unzählige Vorgänge in der Datenverwaltung ausführen lassen. Grundlegende Operationen wie Daten eingeben, Berichte erstellen, Suchen und Sortieren lassen sich mühelos bewerkstelligen. Außerdem stehen viele weitere Funktionalitäten zur Verfügung. Sie können z.B. die Dateneingabe und -sicherung durch Zusätze steuern, ja sogar Diagramme und Etiketten generieren.
Programmieren müssen Sie erst dann, wenn Ihre Anwendung mehr als die Standardfunktionen von 4D enthalten soll. Dazu gehören

- Automatische Berechnung eines Feldinhaltes: Beispielsweise der Gesamtpreis (Stückpreis mal Anzahl).
- Automatischer Ablauf sich wiederholender Vorgänge, wie z.B. Generieren komplexer Berichte
- Anlegen einer eigenen Oberfläche: Beispielsweise soll die Bildschirmoberfläche genau Ihren Bedürfnissen entsprechen, einige Funktionen von 4D sollen für bestimmte Anwender gesperrt sein oder neue Funktionen hinzukommen.
- Erweitern der in 4D integrierten Webdienste: Sie wollen eigene dynamische HTML Seiten erstellen, zusätzlich zu den Seiten, die 4D automatisch anhand der Formulare einrichtet.

In diesen Fällen können Sie Ihre Datenbank genau auf Ihre Bedürfnisse hin programmieren.
Mit der Programmiersprache steuern Sie die Struktur und Funktionsweise Ihrer Datenbank selbst.

Die Daten steuern

Die Sprache von 4D eignet sich für Einsteiger und für erfahrene Anwendungsentwickler gleichermaßen. Sie bietet einen reibungslosen Übergang von den vorgegebenen Funktionen der Datenbank zu einer vollständig angepassten Datenbank.

Mit den Befehlen der 4D Programmiersprache haben Sie Zugriff auf die Ihnen bereits bekannten Editoren zur Datenverwaltung. Verwenden Sie zum Beispiel den Befehl **QUERY**, erhalten Sie den Sucheditor. Dieser Programmierbefehl unterscheidet sich kaum vom Befehl Suchen im Menü Datensätze. Der Unterschied liegt darin, dass Sie mit dem Programmierbefehl **QUERY** nach spezifischen Daten suchen können. So findet z.B. die Anweisung **QUERY** *([Kunden];[Kunden]Nachname="Schmidt")* in Ihrer Datenbank alle Kunden mit dem Nachnamen Schmidt.

Die 4D Programmiersprache ist sehr leistungsstark—ein Befehl ersetzt oft viele Zeilen Code in herkömlichen Computersprachen. Außerdem sind die Befehlsnamen klar verständlich, da sie aus ganzen Wörtern bestehen. Für einen Suchlauf verwenden Sie den Befehl **QUERY**; um einen neuen Datensatz hinzuzufügen, verwenden Sie den Befehl **ADD RECORD**.

Die Sprache ist so aufgebaut, dass Sie fast jeden Vorgang mühelos bewältigen können. Vorgänge wie Datensätze hinzufügen, Datensätze sortieren, nach Datensätzen suchen, lassen sich mit einfachen und direkten Befehlen festlegen. Die Sprache kann aber genauso die serielle Schnittstelle steuern, Dokumente auf der Festplatte lesen, ausgeklügelte Übertragungsprozesse kontrollieren u.v.m.

Die 4D Programmiersprache führt selbst überaus komplexe Aufgaben relativ einfach aus. Für viele wäre die Durchführung solcher Tasks ohne die Programmiersprache schier undenkbar.
Trotz der leistungsstarken Programmierbefehle können manche Vorgänge umfassend und schwierig sein, so dass die Ausführung mit einem Tool allein nicht möglich ist; der Vorgang selbst ist zu umfassend, so dass das Tool den Prozessablauf lediglich erleichtern kann. So lässt sich ein Buch mit einen Textprogramm zwar schneller und einfacher erstellen, das Programm schreibt aber nicht das Buch für Sie. Mit der 4D Programmiersprache lassen sich Ihre Daten leichter verwalten und komplizierte Vorgänge vereinfachen.

Ist 4D eine "traditionelle" Computersprache?

Wenn Sie traditionelle Computersprachen kennen, ist dieser Abschnitt sicher interessant für Sie. Ansonsten können Sie ihn überblättern.

Die 4D Programmiersprache ist keine traditionelle Computersprache. Sie gehört zu den innovativsten und flexibelsten Sprachen, die für Rechner zur Verfügung stehen. Sie ist so konzipiert, dass sie nach Ihrer Art und Weise arbeitet und nicht umgekehrt.

Bei traditionellen Sprachen müssen Sie ausführlich planen. Natürlich zählt Planen zu den wichtigsten Schritten beim Entwickeln. Mit 4D können Sie die Sprache zu jeder Zeit und an beliebiger Stelle in Ihrer Datenbank einsetzen. Sie können z.B. einem Formular eine Methode hinzufügen, und später weitere hinzunehmen. Wird Ihre Datenbank komplexer, können Sie eine über

Menü gesteuerte Projektmethode anfügen. Sie können so viel oder so wenig von der Sprache einsetzen, wie Sie gerade benötigen. Sie müssen nicht – wie in vielen anderen Datenbanken – nach der Devise "Alles oder nichts" arbeiten.

In traditionellen Sprachen sind Sie gezwungen, Formularobjekte in formalen syntaktischen Begriffen zu definieren und vorab zu deklarieren. In 4D legen Sie einfach ein Formularobjekt an, beispielsweise eine Schaltfläche und arbeiten damit. Sie zeichnen Ihre Schaltfläche in einem Formular und geben ihr einen Namen. 4D verwaltet das Objekt automatisch für Sie. Klickt der Benutzer auf die Schaltfläche, ruft die Sprache automatisch die entsprechenden Methoden auf.

Traditionelle Sprachen sind oft starr und unflexibel, Befehle müssen in genau vorgeschriebener Form und Stil eingegeben werden. Die 4D Programmiersprache macht Schluss damit und Sie profitieren davon.

Methoden sind die Verbindung zur Sprache

Eine Methode ist eine Reihe Anweisungen, die bewirken, dass 4D ein Task ausführt. Eine Zeile einer Anweisung wird Statement genannt. Jedes Statement besteht aus Teilen der Programmiersprache.

Wir gehen davon aus, dass Sie das Handbuch *4D Self Training* kennen (Sie haben es doch durchgearbeitet, oder ?). Dort haben Sie bereits Methoden erstellt und angewendet.

Mit 4D können Sie fünf Methodenarten erstellen:

- **Objektmethoden**: Kurze Methoden zum Steuern von Formularobjekten.
- **Formularmethoden**: Verwalten das Anzeigen oder Drucken von einem Formular.
- **Tabellenmethoden/Trigger**: Unterstützen die Regeln für Ihre Datenbank.
- **Projektmethoden**: Methoden, die für die gesamte Datenbank zur Verfügung stehen, z.B. Methoden, die Menüs zugeordnet sind.
- **Datenbankmethoden**: Führen Initialisierungen oder bestimmte Aktionen durch, wenn eine Datenbank geöffnet bzw. geschlossen wird oder sich ein Web Browser an die Datenbank anmelde, die als Web Server im Internet oder Intranet veröffentlicht ist.
  Diese Methoden werden nachfolgend ausführlich beschrieben. Sie erfahren auch, wie Sie damit Ihre Datenbank automatisieren können.

Objektmethoden einsetzen

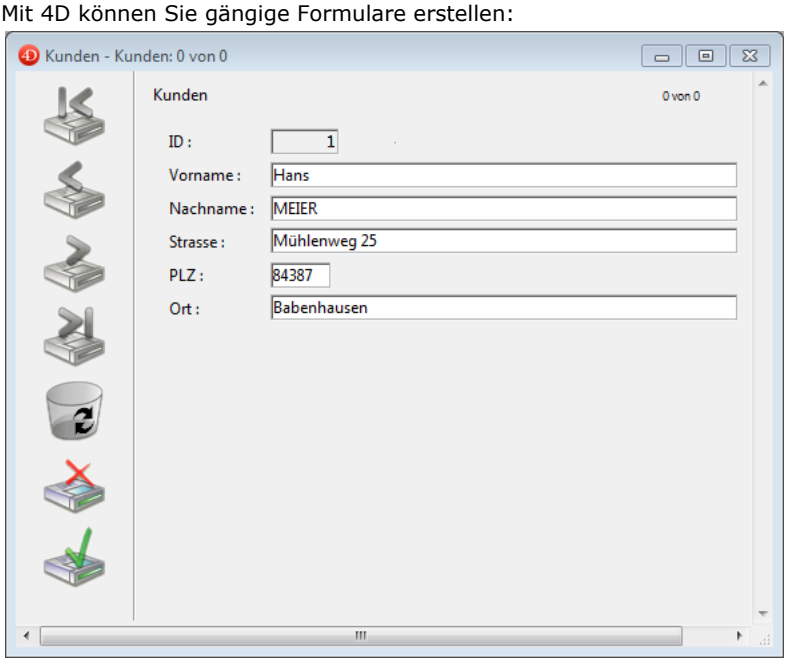
Sie können jedem Formularobjekt, das eine Aktion ausführen kann, d.h. jedem aktiven Objekt eine Methode zuweisen. Eine Objektmethode steuert das aktive Objekt während der Dateneingabe und beim Drucken. Kopieren oder verschieben Sie das aktive Objekt, bleibt die zugeordnete Objektmethode damit verbunden. So können Sie wiederverwendbare Bibliotheken für Objekte mit Objektmethoden erstellen. Die Objektmethode steuert genau dann, wenn sie gebraucht wird.

Objektmethoden sind die wichtigsten Tools zum Steuern der Benutzerschnittstelle, dem Tor zu Ihrer Datenbank. Die Benutzerschnittstelle besteht aus Prozeduren und Konventionen, durch die ein Rechner mit dem Benutzer kommuniziert. Ziel dabei ist, das Arbeiten mit der Benutzeroberfläche Ihrer Datenbank so einfach wie möglich zu machen. Für den Benutzer sollte die Arbeit mit dem Rechner ein angenehmer Prozess sein, der ihm Freude macht oder den er gar nicht bemerkt.

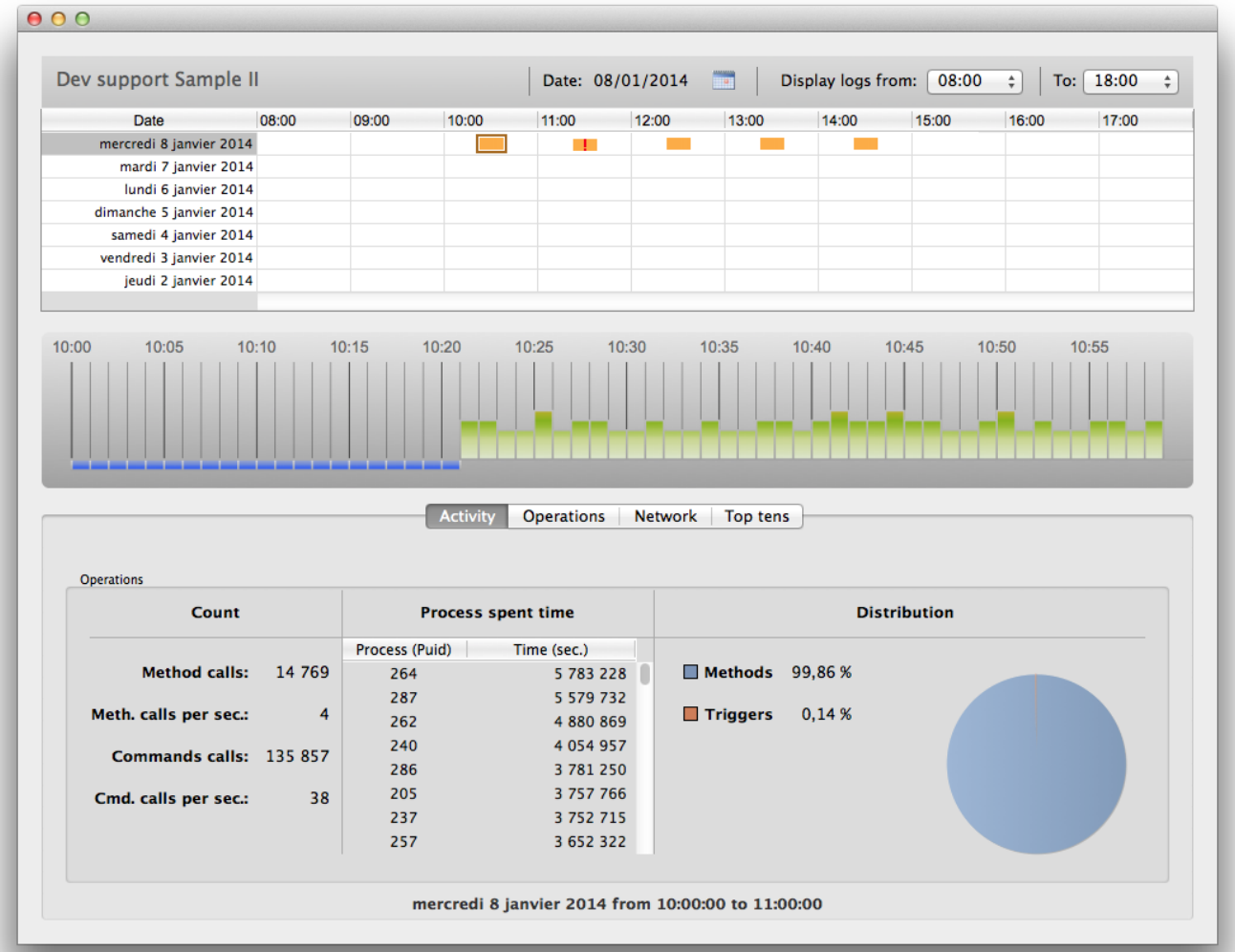
In einem Formular gibt es zwei Grundtypen für aktive Objekte:

- Typ 1 zum Eingeben, Anzeigen und Speichern von Daten; wie Datenfelder und Unterdatenfelder,
- Typ 2 zum Steuern; wie eingebbare Bereiche, Schaltflächen, rollbare Bereiche, hierarchische Listen und Laufbalken.

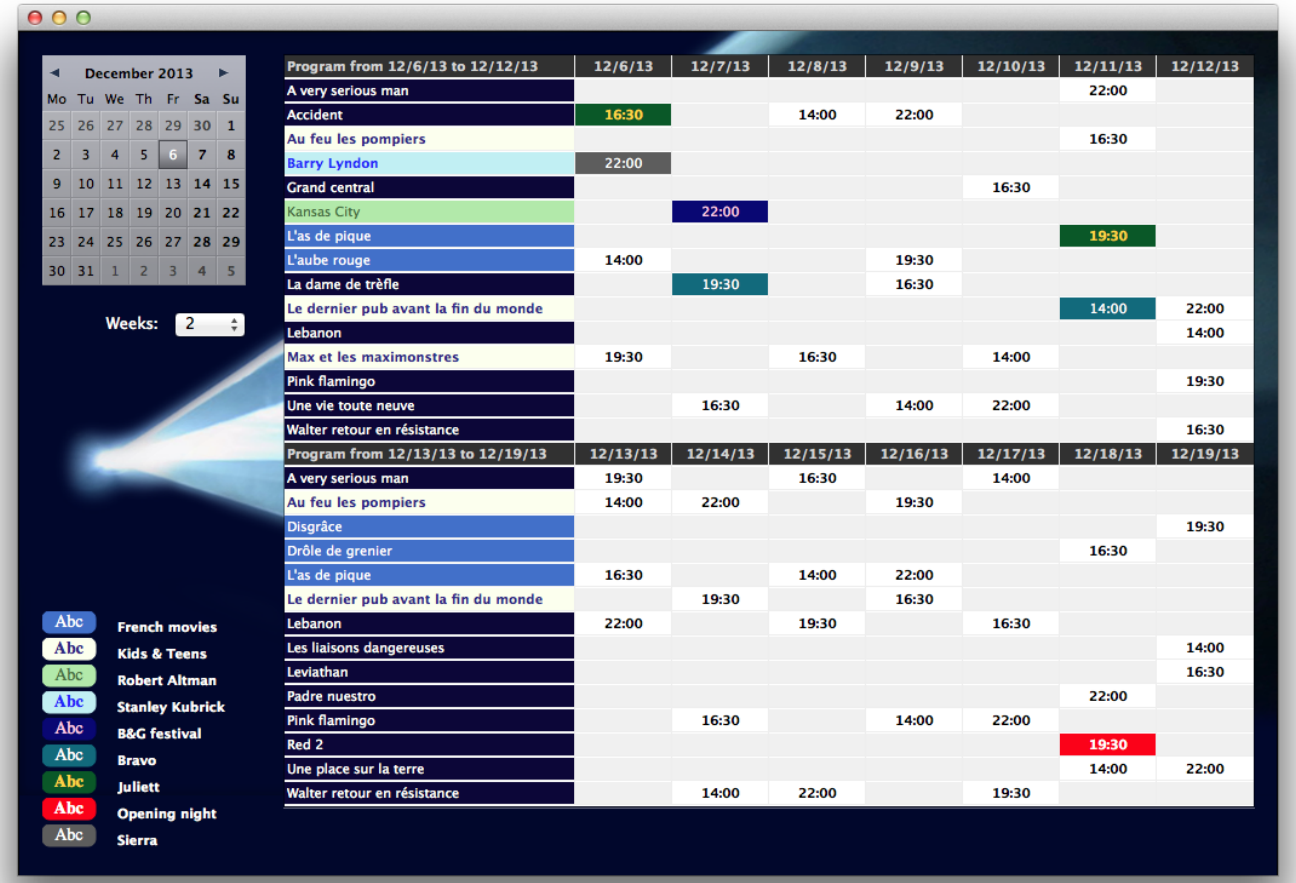
Mit 4D können Sie gängige Formulare erstellen:



Sie können auch ein Formular mit grafischer Steuerung erstellen:



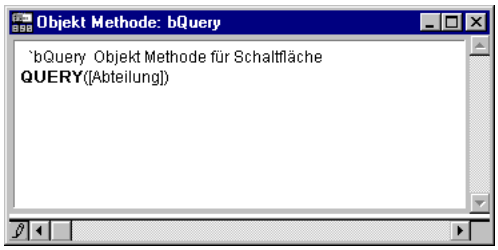
Sie können genausogut Formulare einrichten, die Sie nach Ihren eigenen Vorstellungen grafisch gestalten.:



Unabhängig vom Aussehen des Formulars haben alle aktiven Objekte vorgegebene Hilfen, wie Bereichsprüfung und Eingabefilter für die Eingabebereiche, automatische Aktionen zum Steuern, für Menüs und Schaltflächen. Verwenden Sie diese Hilfen, bevor Sie Objektmethoden hinzufügen. Vorgegebene Hilfen sind den Methoden ähnlich. Sie bleiben mit dem aktiven Objekt verbunden und werden nur aktiv, wenn dieses benutzt wird. Zum Steuern der Benutzeroberfläche verwenden Sie eine Mischung aus vorgegebenen Hilfen und Objektmethoden.

Objektmethoden können aktiven Objekten für die Dateneingabe (Datenfelder bzw. Variablen) zugeordnet sein. Die Methode kann Daten bestätigen, Daten formatieren oder Berechnungen durchführen. Sie kann sogar verknüpfte Information aus anderen Dateien abrufen. Einige dieser Aufgaben lassen sich natürlich auch mit den vorgegebenen Eingabehilfen für Objekte ausführen. Verwenden Sie Objektmethoden, wenn die Aufgabe mit den vorgegebenen Hilfen nicht zu bewältigen ist. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Objektmethoden können aktiven Objekten für die Steuerung (Schaltflächen) zugeordnet sein. Damit navigieren Sie in der Datenbank. Mit Schaltflächen können Sie in Datensätzen und Formularen blättern, sowie Daten hinzufügen bzw. löschen. Sie vereinfachen den Gebrauch der Datenbank und verkürzen die Lernzeit für die Anwendung. Auch für Schaltflächen gibt es vorgegebene Hilfen, die Sie zuerst einsetzen sollten. Über Objektmethoden können Sie eigene Aktionen hinzufügen. Nachfolgende Abbildung zeigt die Objektmethode für eine Schaltfläche, die bei Anklicken den Sucheditor aufruft.



Mit der Zeit werden Sie feststellen, dass bestimmte Aktionen immer wieder vorkommen. Dafür können Sie Bibliotheken mit Objekten und zugeordneteten Methoden anlegen. Diese Objekte mit entsprechenden Methoden können Sie dann zwischen Formularen, Tabellen sowie Datenbanken kopieren und einsetzen.

Formulare mit Formularmethoden steuern

Sie können für jedes Formular eine Formularmethode einrichten. In einem Formular legen Sie fest, wie Daten eingegeben, gelesen und gedruckt werden. Sie können die Daten in unterschiedlicher Form darstellen. Der Benutzer erhält so übersichtliche und leicht zu handhabende Eingabemasken und Berichte zum Drucken. Eine Formularmethode überwacht und verwaltet die Anwendung eines eigenen Formulars zum Eingeben von Daten und zum Ausdrucken.

Formularmethoden arbeiten auf einer höheren Ebene wie Objektmethoden. Objektmethoden werden nur aktiv, wenn das entsprechende Objekt benutzt wird, Formularmethoden werden dagegen aktiv, wenn irgendetwas im Formular benutzt wird. Formularmethoden steuern die Interaktion zwischen den einzelnen Objekten und dem gesamten Formular.

Da Formulare auf ganz unterschiedliche Weise verwendet werden, müssen Sie prüfen können, was während dem Benutzen eines Formulars abläuft. Dafür verwenden Sie **Formularereignisse**. Sie zeigen an, was gerade im Formular passiert. Jede Art von Ereignis (Klick, Doppelklick, Tastaturkürzel...) aktiviert oder deaktiviert die Ausführung der Formularmethode bzw. der Objektmethode für ein bestimmtes Objekt im Formular.

Weitere Informationen dazu finden Sie im Abschnitt **QR SET REPORT KIND**.

Tabellenmethoden/Trigger unterstützen die geltenden Regeln für Ihre Datenbank

Ein Trigger ist einer Tabelle zugeordnet, von daher heißt er auch Tabellenmethode. Trigger werden automatisch von der Engine der 4D Datenbank abgerufen, immer wenn die Datensätze in der Tabelle bearbeitet werden (Hinzufügen, Löschen, Ändern und Laden). Trigger sind Methoden, die "illegale" Operationen mit Datensätzen in Ihrer Datenbank verhindern. Ein Trigger sorgt zum Beispiel in einem Rechnungssystem dafür, dass der Benutzer eine Rechnung nur hinzufügen kann, wenn er auch den Kunden einträgt, an den die Rechnung gestellt wird. Trigger sind ein leistungsstarkes Tool: Sie beschränken Operationen in einer Tabelle und verhindern, dass Daten versehentlich verloren gehen oder beschädigt werden. Sie können ganz einfache Trigger schreiben, und diese dann nach und nach erweitern.

Weitere Informationen dazu finden Sie im Kapitel **Trigger**.

Projektmethoden für die gesamte Datenbank einsetzen

Objektmethoden, Formularmethoden und Trigger sind einem bestimmten Objekt, Formular bzw. einer Tabelle zugeordnet. Im Gegensatz dazu gelten Projektmethoden für die gesamte Datenbank. Sie sind wiederverwendbar und stehen für den Einsatz in jeder anderen Methode zur Verfügung. Soll ein Task wiederholt werden, müssen Sie nicht für jeden Fall die gleiche Methode schreiben. Sie können Projektmethoden immer da aufrufen, wo sie gebraucht werden – aus anderen Projektmethoden oder aus Objekt- bzw. Formularmethoden. Rufen Sie eine Projektmethode auf, läuft sie ab, als ob Sie die Methode an der aufgerufenen Stelle geschrieben hätten. Projektmethoden, die von anderen Methoden aufgerufen werden, werden oft auch "Unterroutinen" genannt.

Sie können Projektmethoden auch Menübefehlen zuordnen. Die Methode wird dann ausgeführt, wenn das Menü aufgerufen wird. Der Menübefehl ruft sozusagen die Projektmethode auf.

Arbeitssitzungen mit Datenbankmethoden verwalten

Methoden, die der Datenbank zugeordnet sind, und abgerufen werden, wenn ein Sitzungsereignis eintritt, heißen **Datenbankmethoden**. Sie wollen zum Beispiel bei jedem Öffnen der Datenbank einige Variablen initialisieren, die während der ganzen Sitzung verwendet werden. Dafür setzen Sie die **Datenbankmethode On Startup** ein. 4D ruft diese Methode beim Öffnen der Datenbank automatisch auf.
Weitere Informationen dazu finden Sie im Abschnitt **CLEAR SEMAPHORE**.

Ihre Datenbank entwickeln

Eine Datenbank entwickeln heißt, die Datenbank mit der Programmiersprache und den integrierten Tools individuell gestalten.

Schon beim Erstellen einer Datenbank verwenden Sie die Programmiersprache. Alle Teile Ihrer Datenbank – Tabellen und Datenfelder, Formulare und die dazugehörigen Objekte, sowie Menüs sind mit der Programmiersprache verbunden. Die 4D Programmiersprache "kennt" all diese Teile der Datenbank.

Vielleicht verwenden Sie die Sprache zum ersten Mal, wenn Sie einem Formularobjekt eine Methode zur Eingabekontrolle zuweisen. Später fügen Sie eine Formularmethode hinzu, die die Anzeige Ihres Formulars steuert. Wird die Datenbank dann komplexer, können Sie eine eigene Menüleiste mit entsprechenden Projektmethoden anlegen, um Ihre Datenbank an Ihre Bedürfnisse anzupassen.

Wie bei anderen Aspekten von 4D ist auch Entwickeln ein ganz flexibler Prozess. Bis auf wenige Grundregeln gibt es keine festgelegte Vorgehensweise – Sie können frei nach Ihren Vorstellungen entwickeln. Es gelten folgende Grundregeln:

- Ausführung: Sie legen die Struktur in der Designumgebung fest.
- Testen: Sie testen die Struktur und eigene Elemente im Modus Anwendung testen, der die Anwendungsumgebung aufruft.
- Anwendung: Haben Sie Ihre Datenbank komplett nach eigenen Vorstellungen erstellt, starten Sie diese direkt in der Anwendungsumgebung.
- Korrekturen: Fehler in der Datenbank beheben Sie in der Designumgebung.

4D enthält spezielle Hilfswerkzeuge zum Entwickeln, die nur bei Bedarf in Erscheinung treten. Je mehr Sie mit der Programmiersprache arbeiten, desto mehr werden Sie feststellen, wie sehr diese Tools die Entwicklung vereinfachen. Der Methodeneditor erkennt z.B. Tippfehler und formatiert Ihre Anweisungen; der Interpreter, d.h. die Engine, die die Sprache aktiviert, erkennt Syntaxfehler und zeigt an, wo sie liegen und wie sie aussehen; der Debugger überwacht die Ausführung Ihrer Methoden, er erkennt Fehler in der Struktur.

Anwendungen strukturieren

Sie sind mit dem allgemeinen Gebrauch einer Datenbank vertraut, dazu gehören Vorgänge wie Daten eingeben, Sortieren, Suchen und Berichte erstellen. Sie haben diese Aufgaben mit Hilfe der vorgegebenen Menüs und Editoren ausgeführt.

Beim Arbeiten mit der Datenbank wiederholen sich einige Abläufe immer wieder. Sie suchen z.B. in einer Datenbank "Personen" nach Ihren Geschäftspartnern, sortieren sie nach dem Nachnamen und drucken, immer wenn sich Informationen geändert haben, einen entsprechenden Bericht aus. Solche Abläufe sind zwar nicht schwierig, sie beanspruchen jedoch immer eine gewisse Zeit, und das summiert sich nach 20 Mal. Führen Sie diese Operation nur alle paar Wochen aus, erinnern Sie sich auch nicht mehr an den genauen Ablauf. Sie können die einzelnen Schritte in einer Methode aneinanderhängen, so dass ein einziger Befehl den kompletten Vorgang automatisch ausführt. Sie müssen die einzelnen Schritte nicht mehr im Kopf haben.

Anwendungen haben eigene Menüs und führen Abläufe aus, die genau auf die Bedürfnisse des Benutzers zugeschnitten sind. Eine Anwendung enthält alle Teile Ihrer Datenbank: Die Struktur, Formulare, Objekt-, Formular- und Projektmethoden sowie Menüs und Kennwörter.

Mit 4D Compiler kompilieren Sie Ihre Datenbank und erstellen eigenständige Anwendungen für Windows und Macintosh. Durch Kompilieren erhöhen Sie die Ausführungsgeschwindigkeit der Programmiersprache, Ihre Datenbanken sind geschützt und Sie können vollkommen unabhängige Anwendungen erstellen. Der integrierte Compiler überprüft auch die Syntax und die Einheitlichkeit der Variablen.

Eine Anwendung kann nur aus einem einzigen Menü bestehen, mit dem Sie Namen eingeben und einen Bericht ausdrucken können. Sie kann aber ebenso ein ganzer Komplex sein aus Rechnungswesen, Lagerhaltung und Steuersystem. Ihrer Phantasie sind keine Grenzen gesetzt. Oft wird eine Anwendung von einer Datenbank, die in der Benutzerumgebung angewendet wird zu einer individuell angepassten Datenbank, die nur noch über eigene Menüs gesteuert wird.

Wie geht es weiter?

- Sie entscheiden selbst, ob Sie eine einfache oder eine komplexe Anwendung entwickeln. Für einen kurzen Überblick über eine einfache 4D Anwendung gehen Sie zum Abschnitt **Eine 4D Anwendung aufbauen**.
- Arbeiten Sie zum ersten Mal mit 4D, lesen Sie das Kapitel **Grundbegriffe**: Beginnen Sie mit **Einführung in die 4D Sprache**.

Eine 4D Anwendung aufbauen

Eine Anwendung ist eine Datenbank für einen ganz bestimmten Zweck. Ihre Benutzeroberfläche ist genau auf die Wünsche des Benutzers zugeschnitten. Die Vorgänge, die Sie mit dieser Anwendung durchführen können, beschränken sich auf die gestellten Anforderungen. Mit 4D erstellen Sie eine Anwendung leichter als mit der herkömlichen Programmierung. Sie können Anwendungen für ganz verschiedene Bereiche erstellen:

- Rechnungswesen
- Warenwirtschaft
- Buchhaltung/Finanzwesen
- Lohn- und Gehaltsverwaltung
- Personalverwaltung
- Kundenverwaltung
- Datenbank im Internet bzw. Intranet

Natürlich kann eine einzelne Anwendung auch all diese Teile beinhalten. Die oben genannten Bereiche sind typisch für Datenbankanwendungen. Mit den in 4D integrierten Tools können Sie aber auch innovative Anwendungen erstellen:

- Dokumentenverwaltung
- Bildarchivierung und -verwaltung
- Katalogproduktion
- Serielle Schnittstellenverwaltung und -überwachung
- Elektronisches Mail-System (eMail)
- Flug-/Fahrplanverwaltung im Mehrplatzbetrieb
- Verzeichnisse, z.B. für Menüs, Video- oder Musiksammlungen

Eine Anwendung startet normalerweise als eine Datenbank, die in der Designumgebung eingesetzt wird. Sie entwickelt sich dann weiter zu einer individuell angepassten Anwendung. In einer Anwendung ist das System zum Verwalten der Datenbank für den Benutzer nicht zugänglich. Die Datenbankverwaltung ist automatisiert, die Benutzer führen spezifische Vorgänge über die Menüs aus.

Verwenden Sie eine 4D Datenbank in der Designumgebung, müssen Sie die Schritte kennen, die zu einem bestimmten Ergebnis führen. In einer Anwendung legen Sie eigene Menüs in der Anwendungsumgebung an, die alle Aspekte verwalten, die in der Designumgebung automatisch ablaufen. Dazu gehören:

- Navigieren in Tabellen: Der Benutzer hat keinen Zugriff auf die **Tabellenliste**. Die Funktion *Zuletzt verwendete Tabellen* oder Steuerungsschaltflächen sind für den Benutzer nicht verfügbar. Sie steuern das Navigieren zwischen Tabellen über Menübefehle und Methoden.

- Menüs: In der Anwendungsumgebung gibt es nur das Standardmenü **Datei/Ablage** mit den Befehlen **Beenden**, **Bearbeiten** sowie das Menü Modus und **Hilfe** (auf Mac OS zusätzlich das Menü Application). Weitere Menüs für die Anwendung müssen Sie mit Hilfe der 4D Methoden oder Standardaktionen erstellen und verwalten.

- Editoren: In der Anwendungsumgebung stehen die Editoren für Such- und Sortierläufe nicht automatisch zur Verfügung. Sie müssen diese über 4D Methoden aufrufen.
  Im Folgenden wird anhand von Beispielen erläutert, wie die Programmiersprache die Verwendung der Datenbank automatisieren kann.

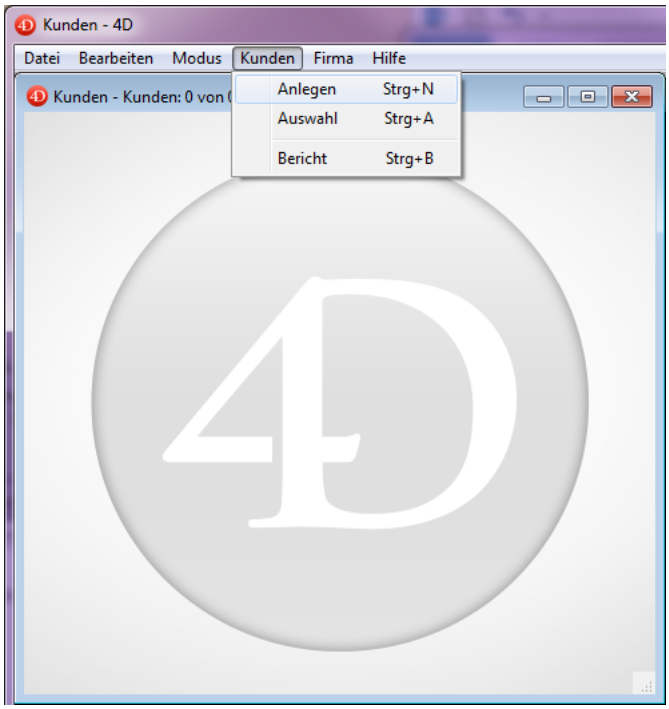
Anwendungsumgebung: Ein Beispiel

Eigene Menüs sind die Basis einer Anwendung. Sie erleichtern dem Benutzer das Erlernen und Arbeiten mit der Datenbank. Eigene Menüs anzulegen ist ganz einfach—Sie müssen nur jedem Menübefehl im Methodeneditor eine Methode oder automatische Aktionen zuweisen.

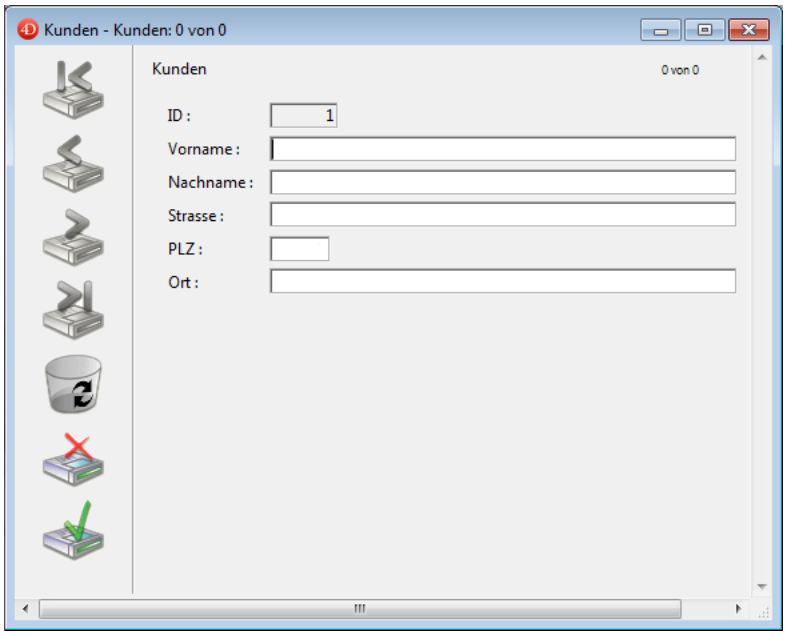
Der nächste Abschnitt "Aus der Sicht des Benutzers" beschreibt, was passiert, wenn der Benutzer einen Menübefehl auswählt. Der darauffolgende Abschnitt "Hinter den Kulissen" beschreibt die zugrundegelegte Struktur, damit das alles so funktioniert. Auch wenn das Beispiel einfach ist, sollte daraus ersichtlich werden, wie eigene Menüs das Erlernen und Handhaben der Datenbank vereinfachen. Im Gegensatz zu den "generischen" Tools und Menübefehlen in der Designumgebung sieht der Benutzer nur die Teile, die er für seine spezifischen Bedürfnisse benötigt.

Aus der Sicht des Benutzers

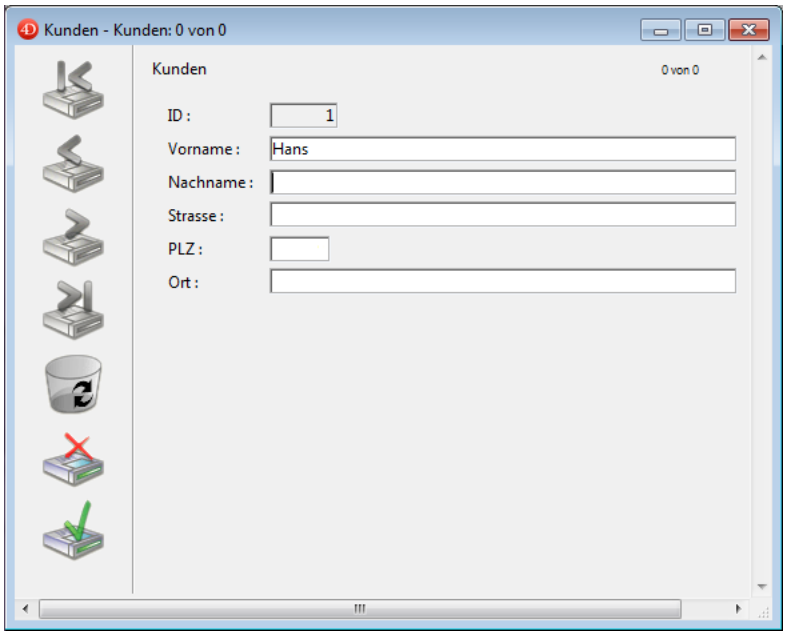
Der Benutzer wählt im Menü **Kunden** den Befehl **Anlegen**, um einen neuen Kunden in die Datenbank aufzunehmen.



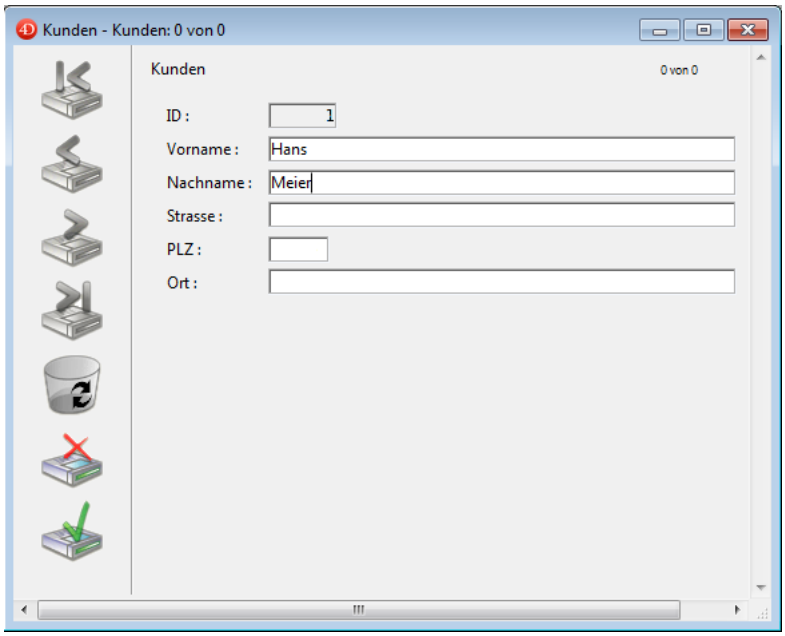
Auf dem Bildschirm erscheint das Eingabeformular für die Tabelle Kunden.



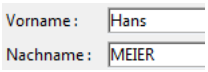
Der Benutzer gibt den Vornamen ein und klickt in das nächste Datenfeld.



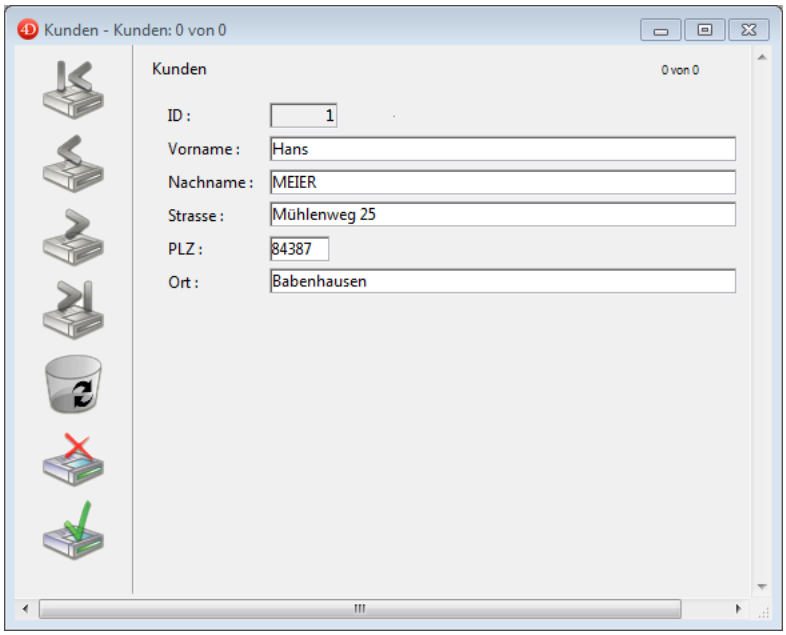
Der Benutzer gibt den Nachnamen ein.



Der Benutzer klickt in das nächste Datenfeld. Der Nachname wird in Großbuchstaben umgewandelt.



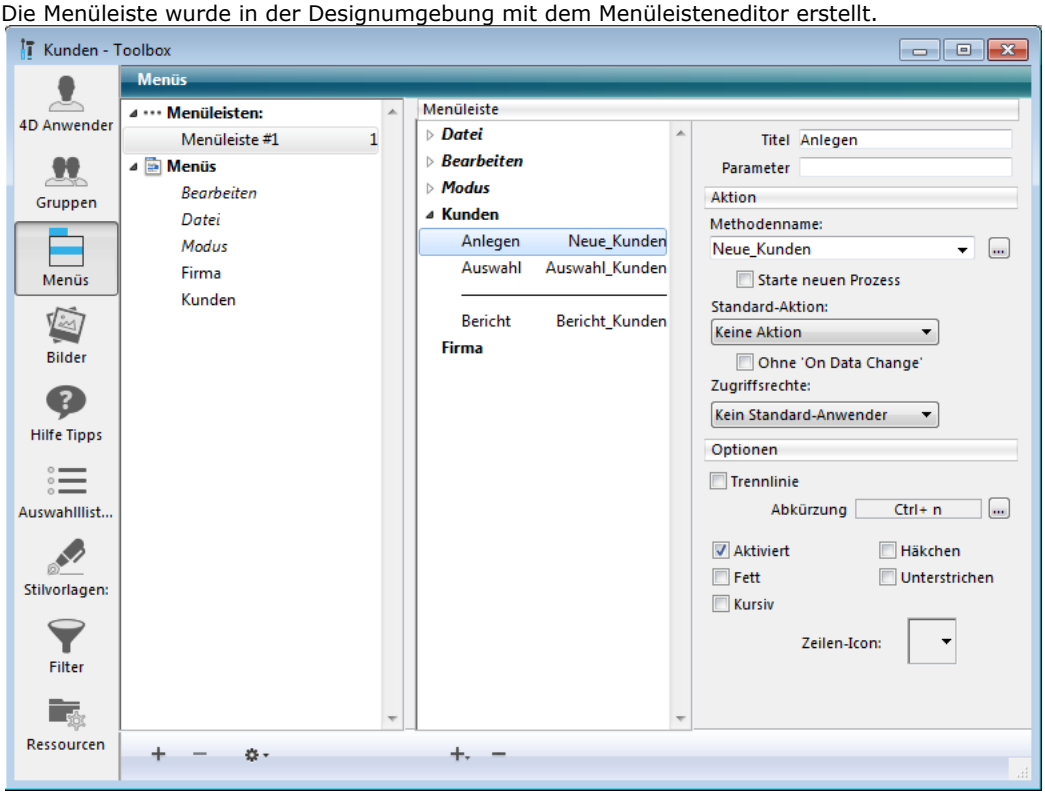
Der Benutzer gibt die restlichen Daten ein und klickt auf die Schaltfläche **Bestätigen**. Das ist normalerweise die letzte Schaltfläche in der Schaltflächenleiste.



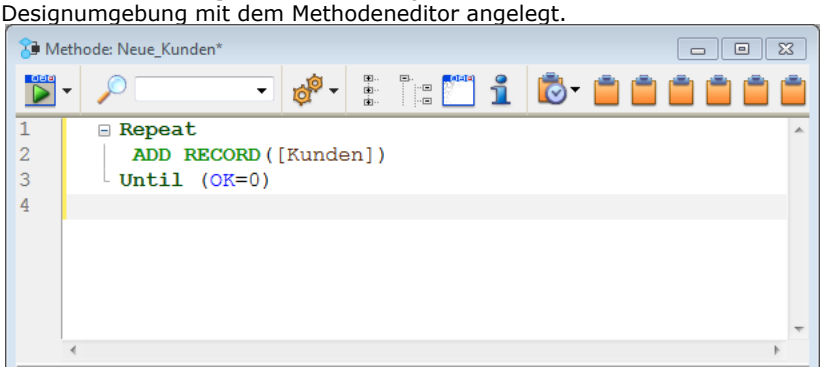
Auf dem Bildschirm erscheint ein leerer Datensatz. Der Benutzer klickt auf die Schaltfläche **Abbrechen**. Das ist die zweite Schaltfläche, gekennzeichnet mit "X". Damit beendet er die Dateneingabe und kehrt zur Menüleiste zurück.

Hinter den Kulissen

Die Menüleiste wurde in der Designumgebung mit dem Menüleisteneditor erstellt.



Der Menüzeile **Anlegen** wurde die Projektmethode mit Namen ***Neue_Kunden*** zugeordnet. Die Methode wurde in der Designumgebung mit dem Methodeneditor angelegt.



Wählt der Benutzer diesen Menübefehl, führt die Methode ***Neue_Kunden*** folgendes aus:

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

Die Schleife *Repeat...Until* zusammen mit dem Befehl **ADD RECORD** innerhalb dieser Schleife arbeitet genauso wie der Menübefehl **Neuer Datensatz** in der Designumgebung. Sie zeigt das Eingabeformular an, so dass der Benutzer einen neuen Datensatz eingeben kann. Sichert der Benutzer den Datensatz, erscheint ein neuer leerer Datensatz. Die Schleife **ADD RECORD** wird solange ausgeführt, bis der Benutzer auf die Schaltfläche **Abbrechen** klickt.

Ist der Datensatz eingegeben, passiert folgendes:

- Dem Datenfeld *Vorname* ist keine Methode zugeordnet, also passiert nichts.
- Dem Datenfeld *Nachname* ist eine Methode zugeordnet. Diese Objektmethode wurde in der Designumgebung dem Feld im Formular zugewiesen. Sie führt folgendes aus:

```
[Kunden]Nachname:=Uppercase([Kunden]Nachname)
```

Diese Zeile konvertiert den Inhalt des Datenfeldes *Nachname* in Großbuchstaben.

Ist der Datensatz ausgefüllt und klickt der Benutzer auf die Schaltfläche **Abbrechen**, wird die Variable OK auf Null gesetzt. Dadurch wird die Ausführung der Schleife **ADD RECORD** beendet.

Da keine weiteren Statements auszuführen sind, wird die Ausführung der Methode ***Neue_Kunden*** beendet, die Steuerung kehrt zur Menüleiste zurück.

Programmierten Task mit Operationen in der Designumgebung vergleichen

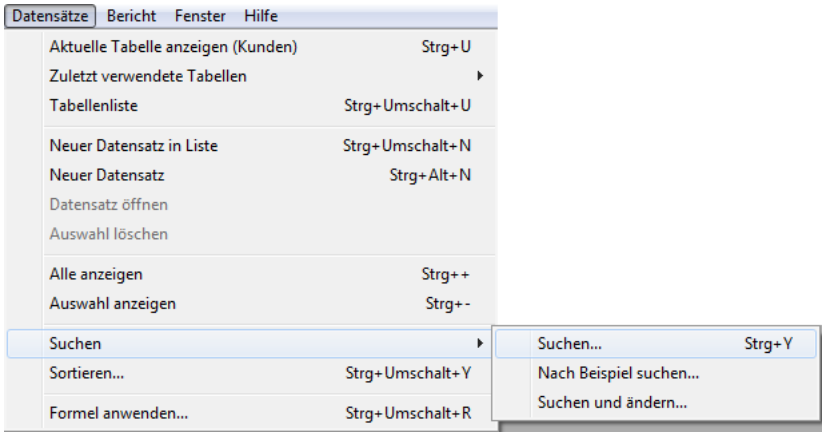
Im Folgenden vergleichen wir, wie derselbe Vorgang in der Designumgebung und über die Programmiersprache ausgeführt wird. Es geht um folgenden Vorgang:

- Finde eine Gruppe von Datensätzen
- Sortiere sie
- Drucke einen Bericht

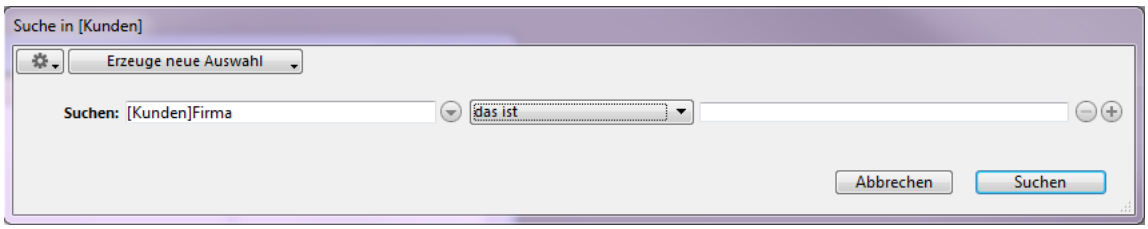
Der Abschnitt "Datenbank in der Designumgebung verwenden" erläutert, wie dieser Vorgang in der Designumgebung abläuft.
Der Abschnitt "Die integrierten Editoren in der Anwendungsumgebung verwenden" erläutert, wie dieser Vorgang in einer Anwendung abläuft.
Beide Methoden führen denselben Vorgang aus. Der Unterschied liegt darin, dass im zweiten Fall die einzelnen Schritte über die Programmiersprache automatisch ablaufen.

Datenbank in der Designumgebung verwenden

Der Benutzer wählt im Menü **Datensätze** den Befehl **Suchen ->Suchen**.

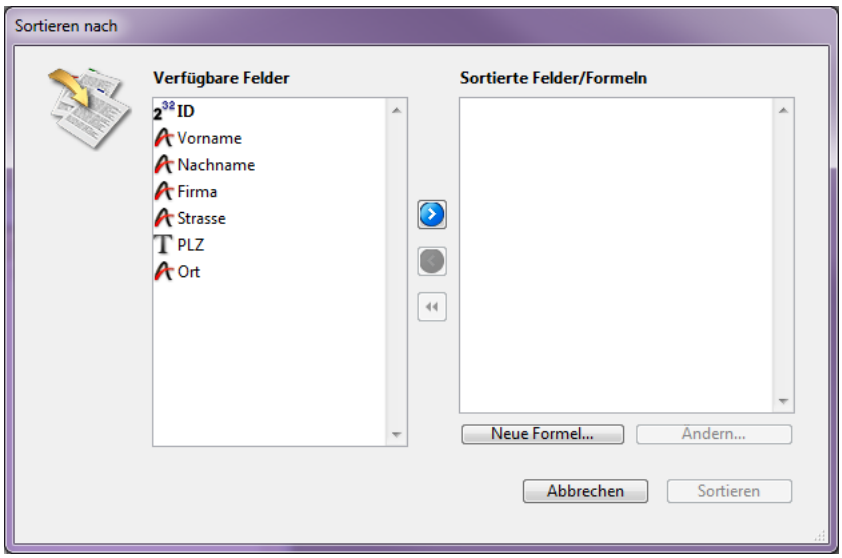


Auf dem Bildschirm erscheint der **Sucheditor:**



Der Benutzer gibt die Suchkriterien ein und klickt auf die Schaltfläche **Suchen**. Die Suche wird ausgeführt.
Der Benutzer wählt im Menü **Datensätze** den Befehl **Sortieren**.

Auf dem Bildschirm erscheint der **Sortiereditor:**



Der Benutzer gibt die Sortierkriterien ein und klickt auf die Schaltfläche **Sortieren**. Der Sortierlauf wird ausgeführt.
Zum Drucken der Datensätze sind noch folgende Schritte erforderlich:

- Der Benutzer wählt im Menü **Datei/Ablage** den Befehl **Drucken**.
- Auf dem Bildschirm erscheint das Dialogfenster **Drucke Formular**, da der Benutzer festlegen muss, welches Formular er drucken möchte.
- Die Druckdialoge erscheinen. Der Benutzer wählt die gewünschten Einstellungen und klickt auf OK. Der Bericht wird gedruckt.

Die integrierten Editoren in der Anwendungsumgebung verwenden

In der Anwendungsumgebung läuft derselbe Vorgang folgendermaßen ab:

Der Benutzer wählt im Menü **Kunden** den Menübefehl **Bericht**.

Schon bei diesem Schritt ist eine Anwendung für den Benutzer einfacher—er musste gar nicht wissen, dass der Vorgang mit Suchen beginnt!

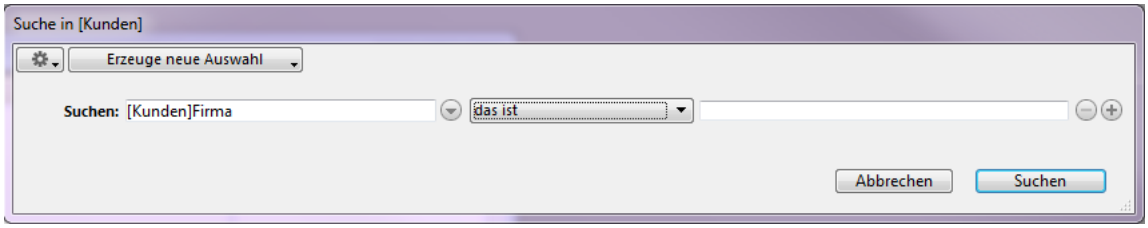
Dem Menübefehl wurde die Methode mit Namen ***Bericht_Kunden*** zugeordnet; sie sieht folgendermaßen aus:

```
QUERY([Kunden])
ORDER BY([Kunden])
FORM SET OUTPUT([Kunden];"Bericht")
PRINT SELECTION([Kunden])
```

Die erste Zeile wird ausgeführt:

```
QUERY([Kunden])
```

Auf dem Bildschirm erscheint der **Sucheditor**.



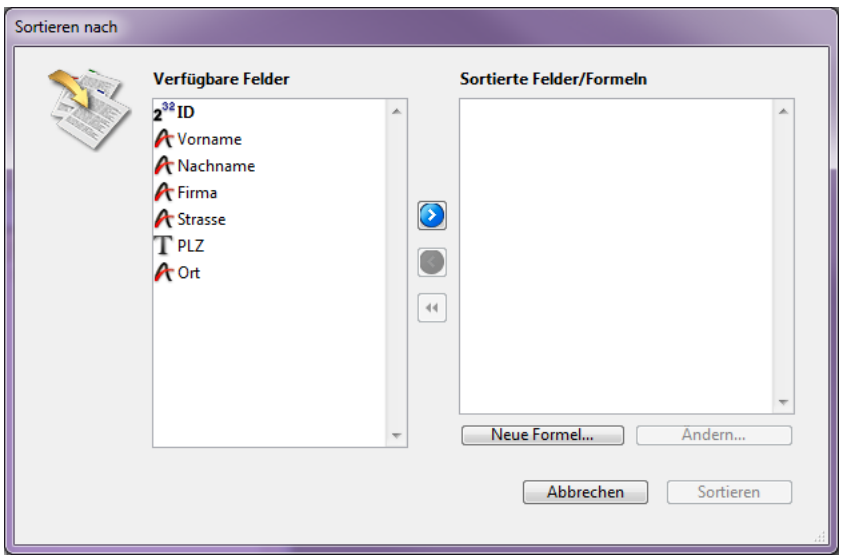
Der Benutzer gibt die Suchkriterien ein und klickt auf die Schaltfläche **Suche**. Die Suche wird ausgeführt.

Die zweite Zeile der Methode ***Bericht_Kunden*** wird ausgeführt:

```
ORDER BY([Kunden])
```

Der Benutzer musste nicht wissen, dass im nächste Schritt die Datensätze sortiert werden.

Auf dem Bildschirm erscheint der **Sortiereditor**.



Der Benutzer gibt die Sortierkriterien ein und klickt auf die Schaltfläche **Sortieren**. Der Sortierlauf wird ausgeführt.

Die dritte Zeile der Methode ***Bericht_Kunden*** wird ausgeführt:

```
FORM SET OUTPUT([Kunden];"Bericht")
```

Wieder musste der Benutzer nicht wissen, was als nächstes passiert. Das erledigt alles die Methode.

Die letzte Zeile der Methode ***Bericht_Kunden*** wird ausgeführt:

```
PRINT SELECTION([Kunden])
```

Die Druckdialoge erscheinen. Der Benutzer wählt die gewünschten Einstellungen und klickt auf OK. Der Bericht wird gedruckt.

Die Anwendung weiter automatisieren

Sie können die Anwendung mit den Befehlen aus obigem Beispiel weiter automatisieren.

Die Methode ***Bericht_Kunden*** wurde erweitert.

Der Benutzer wählt im Menü **Kunden** den Befehl **Bericht**. Diesem Befehl wurde die Methode ***Bericht_Kunden2*** hinzugefügt. Sie lautet:

```
QUERY([Kunden];[Kunden]Firma="ComputerAG")
ORDER BY([Kunden];[Kunden]Nachname;>;[Kunden]Vorname;>)
FORM SET OUTPUT[Kunden];"Bericht")
PRINT SELECTION([Kunden];*)
```

Die erste Zeile wird ausgeführt:

```
QUERY([Kunden];[Kunden]Firma="ComputerAG")
```

Der **Sucheditor** wird nicht angezeigt. Stattdessen wird die Suche definiert und vom Befehl **QUERY** ausgeführt. Der Benutzer muss gar nichts tun.

Die zweite Zeile der Methode ***Bericht_Kunden2*** wird ausgeführt:

```
ORDER BY([Kunden];[Kunden]Nachname;>;[Kunden]Vorname;>)
```

Der **Sortiereditor** wird nicht angezeigt, die Sortierung läuft automatisch ab. Wieder muss der Benutzer gar nichts tun.

Die letzten Zeilen der Methode ***Bericht_Kunden2*** werden ausgeführt:

```
FORM SET OUTPUT[Kunden];"Bericht")
PRINT SELECTION([Kunden];*)
```

Die Druckdialoge erscheinen nicht. Der Befehl **PRINT SELECTION** erlaubt einen optionalen Parameter. Der Stern (*) weist den Befehl an, die Druckereinstellungen zu verwenden, die beim Erstellen des Berichts gewählt wurden. Der Bericht wird gedruckt.

Diese zusätzliche Automatisierung erspart dem Benutzer die Eingabe in drei Dialogfenstern. Die Vorteile liegen auf der Hand:

- Die Suche läuft automatisch ab: Benutzer könnten ein falsches Suchkriterium auswählen.
- Die Sortierung läuft automatisch: Benutzer könnten ein falsches Sortierkriterium auswählen.
- Das Drucken läuft automatisch: Benutzer könnten eine falsche Einstellung auswählen.

Unterstützung beim Entwickeln von 4D Anwendungen

Beim Entwickeln von 4D Anwendungen werden Sie viele weitere Funktionalitäten entdecken. Durch Hinzufügen von Tools und Plug-Ins können Sie Ihre 4D Entwicklungsumgebung kontinuierlich erweitern.

Plug-ins 4D

4D bietet verschiedene Werkzeuge und Plug-Ins zum Ausbau der Funktionalitäten Ihrer 4D Anwendungen an:

- **4D Write**: Textverarbeitung
- **4D View:** Tabellenkalkulation und Listeneditor
- **4D Internet Commands** (integriert)**:** Kommunikations-Tools für das Internet

- **4D ODBC Pro**: Connectivity via ODBC
- **4D for OCI**: Connectivity mit ORACLE Call Interface

Weitere Informationen erhalten Sie bei 4D und ihren Partnern. Sie finden uns auch im Web unter: http://www.4d.com/de

Die 4D Gemeinschaft und Tools von Drittanbietern

Es gibt weltweit eine rege 4D Gemeinschaft aus Benutzergruppen, elektronischen Foren und 4D Partnern. 4D Partner produzieren auch **Third Party Tools**. Das Diskussionsforum finden Sie unter:

http://forums.4d.fr/MyHome/DE/

4D bietet Demos und umfassende Dokumentation zur 4D Produktpalette. Mehr dazu finden Sie im Internet. Sie können die Mailing-Listen von 4D Deutschland abonnieren, mehr dazu unter:

http://www.4d.com/de/support.html

Die 4D Gemeinschaft liefert Tipps und Tricks, Infos, Lösungen und zusätzliche Tools, mit denen Sie Zeit und Energie sparen und Ihre Produktivität steigern können

Grundbegriffe

Einführung in die 4D Sprache

Eine Programmiersprache zu lernen, ist ähnlich wie eine Fremdsprache zu erlernen. Sie müssen zuerst den Wortschatz und die Grammatik einüben, d. h. die korrekte Methode, die Wörter sinnvoll aneinanderzureihen. In den folgenden Abschnitten erklären wir Ihnen die Grundwerkzeuge, die mit einigen Abweichungen in allen Programmiersprachen vorkommen:

- **Datentyp**: Typ definiert die verschiedenen Werte, die ein Datenfeld oder eine Variable (alphanumerisch, numerisch, boolean…) annehmen können. Weitere Informationen dazu finden Sie im Abschnitt **Datentypen**.
- **Variable**: Speicherplatz, der durch einen Namen bestimmt ist und einen zeitlich begrenzten Wert enthält. Im Gegensatz zum Datenfeld, wird der Inhalt einer Variablen nicht auf der Festplatte gespeichert und ist nicht Teil eines Datensatzes oder einer Datei. Weitere Informationen dazu finden Sie im Abschnitt **Variablen**.
- **Operator**: Symbol, das den Wert eines Ausdrucks berechnet (Addition, Multiplikation). Weitere Informationen dazu finden Sie nachfolgend und im Abschnitt **Operatoren**.
- **Ausdruck**: Zusammensetzung aus verschiedenen Komponenten, die einen Wert zurückgeben. Weitere Informationen dazu finden Sie nachfolgend.
- **Befehl**: Führt eine Anweisung aus. Alle 4D Befehle, wie z.B. **ADD RECORD**, werden in diesem Handbuch beschrieben, sie sind nach Themen zusammengefasst. Über 4D Plug-Ins fügen Sie in Ihre 4D Entwicklungsumgebung weitere Befehle ein. Haben Sie zum Beispiel das Plug-In 4D Write in Ihr 4D System integriert, stehen die 4D Write Befehle zum Erstellen und Verwalten von Textdokumenten zur Verfügung.
- **Vordefinierte Konstanten**: Konstantenwerte, die nach Name verfügbar sind. Beispiel: XML DATA ist eine Konstante (Wert 6). Vordefinierte Konstanten machen den Code lesbarer. Sie werden bei den Befehlen, die sie verwenden, beschrieben. Die komplette Übersicht finden Sie im Kapitel **Konstantenthemen**.
- **Methode**: Anweisung, die Sie aus den Elementen der Programmiersprache zusammensetzen. Weitere Informationen dazu finden Sie im Abschnitt **Methoden**.

Der folgende Abschnitt beschreibt Datentypen, Operatoren und Ausdrücke. Die anderen Begriffe werden unter dem jeweiligen Thema beschrieben.

Zusatzinformationen:

- Für Komponenten der Programmiersprache, z.B. Variablen gelten bestimmte Namenskonventionen. Weitere Informationen dazu finden Sie im Abschnitt **Namenskonventionen**.
- Weitere Informationen zu den Variablen vom Typ Array finden Sie im Kapitel **Arrays**.
- Wollen Sie Ihre Datenbank kompilieren, gehen Sie zum Abschnitt **Compilerbefehle** oder schlagen Sie im Handbuch *4D Designmodus* nach.

Sprache für Befehle und Konstanten

Der 4D Methodeneditor verwendet ab Version 15, unabhängig von der 4D Version oder lokalen Systemeinstellung, standardmäßig internationale Formate. Das neutralisiert regionale Unterschiede, die beim Austausch von Code zwischen 4D Entwicklern zu Problemen führen können, wie Datumsformate. Befehls- und Konstantennamen in französischen Versionen erscheinen jetzt in Englisch.

Diese Standardeinstellung bietet 4D Entwicklern zwei bedeutsame Vorteile:

- Es vereinfacht die gemeinsame Nutzung von Code zwischen Entwicklern, unabhängig von dem Land, den regionalen Einstellungen oder der eingesetzten 4D Version. Eine 4D Methode lässt sich einfach per Copy/Paste, oder in einer Textdatei gesichert, austauschen. Es gibt keine Kompatibilitätsprobleme mehr.
- 4D Methoden lassen sich in Source-Control Tools einsetzen, die oft Exporte, unabhängig von regionalen Einstellungen und Programmiersprachen, benötigen.

Der Entwickler kann diese Standardeinstellung im Dialogfenster **4D Einstellungen** auf der **Seite Methoden** deaktivieren.

Eingabeprinzipien in Englisch-US

Die Standardeinstellung Englisch-US hat verschiedene Auswirkungen auf die Art, Methoden zu schreiben. Das gilt sowohl für Code, der im Entwicklungsmodus geschrieben wird, als auch für Formeln, die in Anwendungen im Einsatz geschrieben werden.

Es gelten folgende Regeln:

- In Zahlen müssen Dezimaltrenner in allen Versionen ein Punkt sein ("."), nicht Kommas (",") unabhängig von der Einstellung des Betriebssystems.
- Datumskonstanten müssen in allen Versionen das ISO Format (!YYYY-MM-DD!) verwenden.
- Befehls- und Konstantennamen müssen in Englisch sein (betrifft nur die französischen Versionen von 4D).

Hinweise: Der Methodeneditor enthält spezifische Mechanismen, die inkorrekte Eingaben bei Bedarf automatisch korrigieren:
- Wird auf einem deutschen System bei der Code-Eingabe ein Komma als Dezimaltrenner verwendet, wird es automatisch mit Punkt ersetzt.
- Eine Datumseingabe im Format !DD.MM.YYYY! wird automatisch umgewandelt in !YYYY-MM-DD!
- Eingaben der Anwender in Felder vom Typ Zahl oder Datum bleiben unverändert.

Nachfolgende Tabelle erläutert die Unterschiede zwischen Code ab 4D v15 und in früheren Versionen:

	Beispiel für Code in Methoden/Formeln
4D v15 (Standardmodus, alle Versionen)	a:=12.50 b:=!2013-12-31! Current date
4D v14 oder 4D v15 (Einstellung markiert, alle Versionen außer Französisch)	a:=12.50 b:=!12/31/2013! Current date
4D v14 oder 4D v15 (Einstellung markiert, französische Version)	a:=12,50 b:=!31/12/2013! Date du jour

Hinweis: Ist die Einstellung "Verwende regionale Systemeinstellungen" markiert, richten sich die Formate vom Typ Zahl und Datum nach den Systemeinstellungen.

Datentypen

In einer 4D Anwendung lassen sich verschiedenen Arten von Daten speichern, das sind die Datentypen. Es gibt grundlegende Typen (String, Zahl, Datum, Zeit, Boolean, Bild, Zeiger, Arrays) und komplexe Datentypen (BLOB, Objekt, Collection).

Die Datentypen alphanumerisch und numerisch enthalten mehrere Arten. Werden Daten in ein Feld eingetragen, konvertiert die Programmiersprache die Daten automatisch in den korrekten Typ des Datenfelds. So werden Daten in einem Datenfeld vom Typ Ganzzahl automatisch als numerisch eingestuft. Sie müssen also nicht darauf achten, dass ähnliche Datenfeldtypen vermischt werden, denn das erledigt die Programmiersprache für Sie.

Sie dürfen jedoch nicht verschiedene Datentypen miteinander mischen. Es macht keinen Sinn, die Zeichenkette “ABC” in einem Datenfeld vom Typ Datum einzutragen oder umgekehrt 01.01.16 in ein alphanumerisches Feld zu setzen. 4D toleriert das in den meisten Fällen und versucht, Ihr Vorgehen zu interpretieren. Fügen Sie z.B. eine Zahl in ein Datumsfeld ein, geht 4D davon aus, dass Sie eine Anzahl Tage zum Datum addieren wollen. Versuchen Sie dagegen, eine Zeichenkette in ein Datumsfeld einzufügen, erhalten Sie die Meldung, dass die Operation nicht durchgeführt werden kann.

Es kann auch vorkommen, dass Sie Daten in einem bestimmten Typ abspeichern und in einem anderen Typ verwenden müssen. Das ist mit der 4D Programmiersprache kein Problem. Sie können z.B. eine Artikelnummer mit Zahlen und Buchstaben erstellen:

```
[Products]Part number:=String(Zahl)+"abc"
```

Lautet die Zahl 17, werden die Buchstaben hinzugefügt, die [Products]Part number lautet “17abc”.

Weitere Informationen zu den Datentypen finden Sie im Abschnitt **Datentypen**.

Operatoren

Ein Operator ist ein Symbol für Berechnungen bei Ausdrücken, wie * als Multiplikationszeichen zur Berechnung zweier numerischer Werte. 4D hat Operatoren zur Berechnung von Zahlenwerten, aber auch von Zeichenketten, Daten, booleschen Ausdrücken, ja selbst Bildern.

Die gängigen Operatoren sind:

Operator	Operation	Beispiel
+	Addition	1 + 2 ergibt 3
–	Subtraktion	3 – 2 ergibt 1
*	Multiplikation	2 * 3 ergibt 6
/	Division	6 / 2 ergibt 3

Je nach Datentyp führt dasselbe Symbol unterschiedliche Operationen aus. Das Pluszeichen (+) kann folgende Operationen ausführen:

Datentyp	Operation	Beispiel
Numerisch	Addition	1 + 2 addiert die Zahlen, das Ergebnis ist 3
Alphanumerisch	Zusammenfügen	“Hello ” + “there” fügt die Zeichenketten zusammen, das Ergebnis ist “Hello there”
Datum und Zahl	Datumsaddition	!1998-1-1! + 20 addiert 20 Tage zum Datum 1.Januar 1998, das Ergebnis ist 21.Januar 1998

Weitere Informationen zu den Operatoren finden Sie im Kapitel **Operatoren**.

Ausdrücke

Ausdrücke werden gesetzt und geben einen Wert zurück. Beim Arbeiten mit der 4D Programmiersprache verwenden Sie stets Ausdrücke, auch wenn man meinen könnte, dass es der dargestellte Wert ist. Ausdrücke werden manchmal auch als Formeln dargestellt.

Ausdrücke können fast alle Komponenten der Programmiersprache enthalten, also Befehle, Operatoren, Variablen und Datenfelder. Mit Ausdrücken erstellen Sie Statements, das sind Programmierzeilen, die wiederum zu einer Methode zusammengesetzt werden. Immer wenn die Programmiersprache Daten benötigt, arbeitet sie mit Ausdrücken.

Parameter sind nur in folgenden Fällen eigenständig:

- Im Dialogfenster Nach Formel suchen
- Im Debuggerfenster, der den Wert der Ausdrücke prüft
- Im Dialogfenster Formel anwenden
- Im Schnellberichteditor als Formel für eine Spalte

Ein Ausdruck kann auch nur eine Konstante sein, also die Zahl 4 oder die Zeichenkette “Hello”. Wie der Name bereits sagt, bleibt der Wert der Konstanten immer gleich. Ausdrücke werden interessant in Verbindung mit Operatoren. Im vorangegangenen Abschnitt haben Sie bereits Beispiele kennengelernt. 4 + 2 ist ein Ausdruck, der über den Operator Plus (+) zwei Zahlen addiert und das Ergebnis 6 zurückgibt.

Sie beziehen sich auf einen Ausdruck gemäß dem Datentyp, den er zurückgibt. Es gibt folgende Arten:

- Alphanumerischer Ausdruck
- Numerischer Ausdruck (auch vom Typ Zahl)
- Ausdruck vom Typ Datum
- Ausdruck vom Typ Zeit
- Ausdruck vom Typ Boolean
- Ausdruck vom Typ Bild
- Ausdruck vom Typ Zeiger
- Ausdruck vom Typ Objekt
- Ausdruck vom Typ Collection

Nachfolgend sehen Sie ein Beispiel für jede Art:

Ausdruck	**Typ**	**Erklärung**
“Hello”	Alpha	Konstante vom Typ alphanumerisch, sie steht zwischen Anführungszeichen.
“Hello ” + “there”	Alpha	Zwei Zeichenketten, “Hello ” und “there”, werden mit dem Operator Plus (+) zusammengefügt. Der Ausdruck gibt "Hello there" zurück.
“Mr. ” + [People]Name	Alpha	Zwei Zeichenketten werden zusammengefügt, “Mr. ” und der aktuelle Wert des Datenfelds Name in der Tabelle People. Enthält das Feld “Schmidt”, gibt der Ausdruck “Mr. Schmidt” zurück.
Uppercase (“Schmidt”)	Alpha	Dieser Ausdruck verwendet die Funktion **Uppercase**, um “Schmidt” in Gross-buchstaben zu setzen. Er gibt “SCHMIDT” zurück.
4	Zahl	Die konstante Zahl 4.
4 * 2	Zahl	Die beiden Zahlen 4 und 2 werden mit dem Operator für Multiplikation (*) multipliziert. Das Ergebnis ist die Zahl 8.
MeineSchaltfläche	Zahl	Name einer Schaltfläche. Er gibt den aktuellen Wert der Schaltfläche zurück: 1 Angeklickt, 0 nicht Angeklickt.
!1998-01-25!	Datum	Konstante vom Typ Datum für das Datum 25.1.98 (25.Januar 1998).
Current date + 30	Datum	Ausdruck vom Typ Datum, der über die Funktion **Current date** das aktuelle Datum erhält. Er addiert 30 Tage zum aktuellen Datum und gibt das neue Datum zurück.
?8:05:30?	Zeit	Konstante vom Typ Zeit mit den Werten 8 Stunden, 5 Minuten und 30 Sekunden.
?2:03:04? + ?1:02:03?	Zeit	Dieser Ausdruck addiert zwei Zeiten und gibt die Zeit 3:05:07 zurück.
True	Boolean	Diese Funktion gibt den Boolean Wert TRUE zurück.
10 # 20	Boolean	Logischer Vergleich zwischen zwei Zahlen. Das Nummernzeichen (#) bedeutet “ungleich”. Da 10 “ungleich” 20 ist, gibt der Ausdruck TRUE zurück.
“ABC” = “XYZ”	Boolean	Logischer Vergleich zwischen zwei Zeichenketten. Da sie nicht gleich sind, gibt der Ausdruck FALSE zurück.
MyPicture + 50 Pixel		Dieser Ausdruck wählt das Bild in MyPicture, verschiebt es 50 Pixel nach rechts und gibt das verschobene Bild zurück.
->[People]Name	Zeiger	Dieser Ausdruck gibt einen Zeiger auf das Datenfeld [Personen]Name zurück.
Table (1)	Zeiger	Dieser Ausdruck gibt einen Zeiger auf die erste Tabelle zurück.
JSON Parse(MyString)	Objekt	Dieser Befehl gibt MyString als Objekt zurück (bei passendem Format)
JSON Parse(MyJSONArray)	Collection	Dieser Befehl gibt MyJSONArray als Collection zurück (bei passendem Format)

Datentypen

Übersicht über Datentypen für Datenfelder, Variablen und Ausdrücke von 4D:

Datentyp	Datenfeld	Variable	Ausdruck
Alphanumerisch (siehe 1.)	Ja	Ja	Ja
Zahl (siehe 2.)	Ja	Ja	Ja
Datum	Ja	Ja	Ja
Zeit	Ja	Ja	Ja
Boolean	Ja	Ja	Ja
Bild	Ja	Ja	Ja
Zeiger	Nein	Ja	Ja
BLOB (siehe 3.)	Ja	Ja	Nein
Array (siehe 4.)	Nein	Ja	Nein
Ganzzahl 64 bits (siehe 5.)	Ja	Nein	Nein
Fließkomma (siehe 5.)	Ja	Nein	Nein
Objekt	Ja	Ja	Ja
Collection	Nein	Ja	Nein
Undefiniert	Nein	Ja	Ja
Null	Nein	Nein	Ja

Hinweise

1. Alphanumerisch beinhaltet Datenfelder vom Typ alphanumerisch, Variablen mit fester Länge und Datenfelder bzw. Variablen vom Typ Text.
2. Numerisch beinhaltet Datenfelder und Variablen vom Typ Zahl, Ganzzahl und Lange Ganzzahl.
3. BLOB ist die Abkürzung für Binary Large OBject. Weitere Informationen dazu finden Sie im Kapitel **BLOB**.
4. Array beinhaltet alle Arten von Arrays. Weitere Informationen dazu finden Sie im Kapitel **Arrays**.
5. Ganzzahl 64 bits und Fließkomma werden nur von SQL verwaltet. Wir empfehlen, diese Datentypen nicht in der 4D Programmiersprache zu verwenden, da sie intern in den Typ Zahl umgewandelt werden und das die Genauigkeit beeinträchtigen kann.

String

String ist der Oberbegriff für:

- Datenfelder oder Variablen vom Typ alphanumerisch: Ein Datenfeld vom Typ alphanumerisch kann 0 bis 255 Zeichen enthalten (Limit wird bei der Felddefinition festgelegt)
- Felder bzw. Variablen vom Typ Text: Ein Feld, eine Variable oder ein Ausdruck vom Typ Text kann 0 bis 2 GB an Text enthalten
- Jeder Ausdruck vom Typ alphanumerisch oder Text

Es gibt keinen Unterschied zwischen einer Variablen vom Typ alphanumerisch oder Text.

Sie können einen alphanumerischen Wert einem Textfeld zuordnen und umgekehrt. 4D erledigt die Konvertierung und kürzt bei Bedarf. In Ausdrücken können Sie alphanumerisch und Text gleichzeitig verwenden.

Hinweis: Bei der Beschreibung von Befehlen in diesem Handbuch gilt der Begriff String für beide Arten von Ausdrücken, wenn nichts anderes angegeben ist.

Numerisch

Numerisch ist der Oberbegriff für:

- Datenfeld, Variable oder Ausdruck vom Typ Zahl
- Datenfeld, Variable oder Ausdruck vom Typ Ganzzahl
- Datenfeld, Variable oder Ausdruck vom Typ Lange Ganzzahl

Der Typ Zahl umfasst den Bereich ±1,7e±308 bei 13 Stellen Genauigkeit.
Der Typ Ganzzahl umfasst den Bereich -32 768 bis 32 767 (2 Bytes).
Der Typ Lange Ganzzahl umfasst den Bereich -2 147 483 648 bis 2 147 483 647 (4 Bytes).

Sie können numerische Datentypen untereinander zuweisen. 4D erledigt die Konvertierung und kürzt oder rundet bei Bedarf. Bei Werten außerhalb des entsprechenden Bereichs liefert die Konvertierung keinen gültigen Wert. Sie können numerische Datentypen in Ausdrücken auch miteinander mischen.

Hinweis: Bei der Beschreibung von Befehlen in diesem Handbuch gilt der Begriff numerisch für alle Datentypen der Art Zahl, Ganzzahl und Lange Ganzzahl, wenn nichts anderes angegeben ist.

Datum (6 Bytes)

- Datenfelder, Variablen oder Ausdrücke vom Typ Datum umfassen den Bereich 1.1.100 bis 31.12.32.767.
- Das Datum wird in der deutschen Version von 4D im Format Tag/Monat/Jahr dargestellt, in der internationalen Version im Format Monat/Tag/Jahr.

- Eine Jahreszahl mit zwei Stellen betrifft standardmäßig das Jahr 1900, außer die Einstellung wurde mit dem Befehl **SET DEFAULT CENTURY** geändert.
- Auch wenn die Datumsdarstellung über C_DATE bis zum Jahr 32 767 funktioniert, können über das Betriebssystem laufende Operationen ein niedrigeres Jahr vorschreiben.

Konvertieren von Datum in JavaScript

Da Datum in JavaScript ein Objekt ist, wird es an 4D, wie jedes andere Objekt, als Text mit JSON Formular gesendet. Dieses Prinzip wird insbesondere bei 4D Mobile oder **Web Area** implementiert.
Das JSON Formular des Objekts JavaScript Datum berücksichtigt den Standard ISO 8601, z.B. "2013-08-23T00:00:00Z"
Sie müssen selbst für die Konvertierung in ein 4D Datum (C_DATE) sorgen. Es gibt zwei Lösungen:

- Über den Befehl **JSON Parse**:

```
C_TEXT($1) // Ein Datum in ISO Format empfangen
C_DATE($d)
$d:=JSON Parse("\""+$1+"\"";ls date))
```

- Über die Funktion **Date**:

```
C_TEXT($1) // Ein Datum in ISO Format empfangen
C_DATE($d)
$d:=Date($1)
```

Es gibt folgende Unterschiede: **JSON Parse** berücksichtigt den Konvertierungsmodus, der über den Befehl **SET DATABASE PARAMETER** (sofern vorhanden) gesetzt wurde. Die Funktion **Date** berücksichtigt dagegen immer die lokale Zeitzone.

Hinweis: Ist in 4D v16 R6 "Datumstyp" die aktuelle Einstellung zum Speichern des Datums, verwalten die Funktionen **JSON Parse** und **Date** ein JSON Datum im Format "YYYY-MM-DD" automatisch als Datumswert. Weitere Informationen dazu finden Sie auf der **Seite Kompatibilität** unter der Option *Verwende Datumstyp statt ISO Datumsformat in Objekten*.

Zeit (4 Bytes)

- Datenfelder, Variablen oder Ausdrücke vom Typ Zeit umfassen den Bereich 00:00:00 bis 596.000:00:00.
- Die Zeit wird in der deutschen und der internationalen Version von 4D im Format Stunde:Minute:Sekunde dargestellt.
- Zeiten werden im 24 Stundenformat angezeigt.
- Ein Wert vom Typ Zeit kann wie eine Zahl behandelt werden. Ein Zeitausdruck als Zahl gibt die Zeit in Sekunden an. Weitere Informationen dazu finden Sie im Abschnitt **Zeitoperatoren**.

Boolean

Datenfelder, Variablen oder Ausdrücke vom Typ Boolean können WAHR oder FALSCH sein.

Bild

Datenfelder, Variablen oder Ausdrücke vom Typ Bild können sowohl im Windows- als auch im Macintosh-Format sein. Dazu gehört jedes Bild, das Sie in die Zwischenablage legen oder mit Befehlen von 4D bzw. Plug-Ins von der Festplatte lesen können.

Zeiger

Variablen oder Ausdrücke vom Typ Zeiger verweisen auf eine andere Variable (inkl. Arrays oder Tabellenelemente), Tabelle bzw. Datenfeld. Ein Datenfeld kann nicht vom Typ Zeiger sein. Weitere Informationen dazu finden Sie im Abschnitt **Zeiger**.

BLOB

Datenfelder oder Variablen vom Typ BLOB sind eine Reihe Bytes (von 0 bis 2 GB Länge), die Sie individuell oder über BLOB Befehle (siehe Kapitel **BLOB**) zuweisen können. Ein Ausdruck kann nicht vom Typ BLOB sein.

Objekt

Variablen, Felder oder Ausdrücke vom Typ Objekt können verschiedene Datentypen enthalten. Die Struktur von "native" 4D Objekten basiert auf dem klassischen Prinzip von "Eigenschaft/Wert" bzw. "Attribut/Wert" Paaren. Die Syntax dieser Objekte basiert auf JSON, folgt ihr aber nicht komplett.

- Ein Eigenschaftenname ist immer ein Text, z.B. "Name" (bis 255 Zeichen, unterscheidet zwischen Groß- und Kleinschreibung)

- Der Wert einer Eigenschaft kann einer der folgenden Typen sein:
  - Zahl (Zahl, Ganzzahl, etc.)
  - Text
  - Array (Text, Zahl, Boolean, Objekt, Zeiger)
  - Null
  - Boolean
  - Zeiger (als solcher gespeichert, wird beim Verwenden der Funktion **JSON Stringify** oder beim Kopieren bewertet)
  - Datum (Datumstyp oder ISO Datumsformat - siehe **Seite Kompatibilität**, Option "Verwende Datumstyp statt ISO Datumsformat in Objekten")

- Objekt (Objekte können in mehrere Stufen verschachtelt sein)
- Bild (*)
- Collection

Warnung: Beachten Sie, dass Attributnamen zwischen Groß- und Kleinschreibung unterscheiden.

Objekte vom Typ Variable, Felder oder Ausdruck verwalten Sie über Befehle im Kapitel **Objekte (Sprache)**, wie z.B. **OB Get** und **OB SET**, oder über die Objektnotation (siehe **Objektnotation verwenden**). Über spezifische Befehle im Kapitel **Suchen** wie **QUERY BY ATTRIBUTE**, **ORDER BY ATTRIBUTE** und **QUERY SELECTION BY ATTRIBUTE** können Sie mit Objektfeldern arbeiten.

Da Objektfelder in der Regel auf Text basieren, erscheint der Inhalt eines Objektfeldes in einem 4D Formular standardmäßig als Text und ist in JSON formatiert.

(*) Wenn als Text im Debugger dargestellt oder in JSON exportiert, zeigen die Eigenschaften des Objekts Bild "[Objekt Bild]" an. Beachten Sie, dass danach Sichern des Datensatzes den String "[Objekt Bild]" in der Eigenschaft sichert.

Hinweis: Zum Arbeiten mit JSON Objekten können Sie die Befehle im Kapitel **JSON** verwenden.

Collection

Eine Collection Variable kann eine Sammlung von Werten unterschiedlicher Typen enthalten. Hierzu ein Beispiel:

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

Unterstützt werden die Typen Text, Zahl, Objekt, Boolean, Collection oder Null. Es gibt keinen Ausdruck oder Feld vom Typ Collection.

Zum Verwalten von Collection Variablen benötigen Sie Objektnotation (siehe **Objektnotation verwenden**) und die Befehl im Kapitel **Collections**.

- Über die Elementnummer (Index) greifen Sie auf Collection Elemente zu
- Zur Angabe eines Elements verwenden Sie folgende Syntax: myCollection[*N*], wobei *N* der Index des Collection Elements ist
- **Warnung:** Der Index von Collection Elementen startet bei 0

Beispiel:

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

Collection Variablen speichern JSON Arrays. Ein JSON Array ist eine Sammlung von Werten beliebiger Typen, getrennt durch Komma. Sie können JSON Arrays in Collections speichern: Beispiele:

```
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"]
```

Undefiniert

Undefiniert ist kein Datentyp. Damit wird eine Variable bezeichnet, die noch nicht definiert ist. Eine Funktion (eine Projektmethode, die ein Ergebnis zurückgibt) gibt einen undefinierten Wert zurück, wenn in der Methode dem Ergebnis der Funktion (*$0*) ein undefinierter Ausdruck zugewiesen wurde (Ausdruck mit mindestens einer undefinierten Variablen). Ein Datenfeld kann nicht undefiniert sein (die Funktion **Undefined** gibt für ein Datenfeld immer **False** zurück).

Null

Null ist ein spezieller Datentyp mit nur einem möglichen Wert: **Null**. Dieser Wert wird von einem Ausdruck zurückgegeben, der gar keinen Wert enthält.

Aus Sicht der 4D Datenbank bedeutet ein **Nullwert**, dass der Datenwert unbekannt ist. Es bedeutet NICHT, dass kein Wert existiert, oder dieser leer ist ("" für String oder 0 für Lange Ganzzahl sind leere Werte). In der 4D Datenbank werden Nullwerte in Feldern (außer Objekt Feldattribute) nur von der SQL Engine verwaltet. Über eine spezifische Option für Felder können Sie einstellen, wie die Datenbank diesen Wert behandeln soll (**NULL Werten leere Werte zuordnen**) und Sie können Nullwerte über die Befehle **Is field value Null** und **SET FIELD VALUE NULL** setzen oder lesen.

In der 4D Programmiersprache und für Objekt Feldattribute werden **Nullwerte** über die Funktion **Null** verwaltet. Sie lässt sich mit folgenden Ausdrücken zum Setzen oder Vergleichen des Nullwerts verwenden:

- Objekt Attribute
- Collection Elemente
- Variablen vom Typ Objekt, Collection, Zeiger oder Bild.

Array

Ein Array ist kein Datentyp. Unter diesem Oberbegriff werden Arrays der Art Zahl, Text etc. zusammengefasst. Arrays sind Variablen — Datenfelder und Ausdrücke können nicht vom Typ Array sein. Weitere Informationen dazu finden Sie im Kapitel **Arrays**.

Hinweis: Im Allgemeinen werden Array Parameter in Befehlsbeschreibungen als Arrays bezeichnet, in spezifischen Fällen wird Array Text, Array Zahl, etc. verwendet

Datentypen konvertieren

Die 4D Programmiersprache enthält Operatoren und Befehle, um bei Bedarf zwischen Datentypen zu konvertieren. Die 4D Programmiersprache unterstützt die Überprüfung von Datentypen. Sie können nicht schreiben: *"abc"+0.5+!12/25/96!-?00:30:45?*. Das ergibt Syntaxfehler.

Nachfolgende Liste zeigt die wichtigsten Datentypen, in welchen Typ sie konvertiert werden können und die dafür verwendeten Befehle:

Datentyp	**konvertieren in Alphanumerisch**	**konvertieren in Numerisch**	**konvertieren in Datum**	**konvertieren in Zeit**
Alphanumerisch (*)		**Num**	**Date**	**Time**
Numerisch (**)	**String**			
Datum	**String**			
Zeit	**String**			
Boolean		**Num**		
Objekt	**JSON Stringify**			
Collection	**JSON Stringify**			

(*) In JSON formatierte Strings lassen sich über die Funktion **JSON Parse** in skalare Daten, Objekte oder Collections konvertieren.
(**) Werte vom Typ Zeit lassen sich wie Zahlen behandeln.

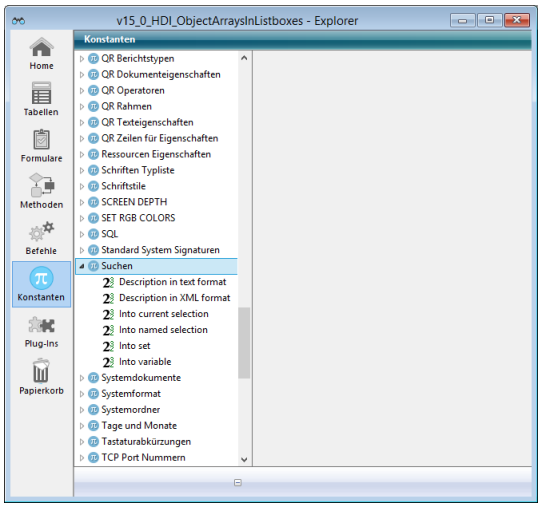
Hinweis: Durch Kombinieren von Operatoren und anderen Befehlen erreichen Sie auch komplexere Datenkonvertierungen.

Konstanten

Eine Konstante ist ein Ausdruck mit einem festen Wert. Es gibt zwei Arten: **vordefinierte Konstanten**, die Sie nach dem Namen auswählen, und **selbstdefinierte Konstanten**, für die Sie den aktuellen Wert eingeben.

Vordefinierte Konstanten

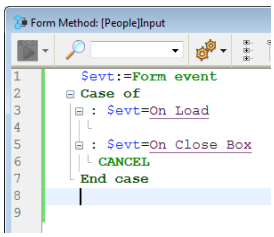
4D bietet eine ganze Reihe **vordefinierter Konstanten**. Sie erscheinen im Explorer-Fenster nach Themen sortiert:



Sie können eine vordefinierte Konstante auf zwei Arten in den **Methodeneditor** setzen:

- Markieren Sie die Konstante im Explorer und ziehen Sie diese bei gedrückter Maustaste in den Methodeneditor.
- Tippen Sie den entsprechenden Namen direkt im Methodeneditor ein. Sie müssen nur die ersten Buchstaben eingeben, der type-ahead Mechanismus schlägt dann passende Namen vor.

Vordefinierte Konstanten erscheinen im Methodeneditor und Debugger-Fenster standardmäßig unterstrichen. Hier ein Beispiel mit den vordefinierten Konstanten On Load und On Close Box:



Selbstdefinierte Konstanten

Es gibt folgende Arten:

- Alphanumerisch
- Numerisch
- Datum
- Zeit

Alphanumerische Konstanten

Eine alphanumerische Konstante steht zwischen geraden Anführungszeichen ("..."). Hier ein paar Beispiele:

"Datensätze hinzufügen"
"Keine Datensätze vorhanden."
"Rechnung"

Eine leere Zeichenkette legen Sie mit geraden Anführungszeichen ohne Inhalt an ("").

Numerische Konstanten

Eine numerische Konstante wird als Zahl geschrieben. Hier ein paar Beispiele:

27
123.76
0.0076

Negative Zahlen werden mit einem Minuszeichen gekennzeichnet (–), also:

–27
–123.76
–0.0076

Hinweis: Ab 4D v15 ist der Dezimaltrenner standardmäßig ein Punkt (.), unabhängig von der Sprache des Betriebssystems. Haben Sie die Option "Verwende regionale Systemeinstellungen" markiert (siehe **Seite Methoden**), müssen Sie den Trenner verwenden, der in Ihrem System definiert ist.

Datumskonstanten

Eine Datumskonstante steht zwischen Ausrufezeichen (!…!). Ab 4Dv15 muss ein Datum das ISO Format (!YYYY-MM-DD!) haben. Hier ein paar Beispiele für Konstanten:

!1976-1-1!
!2004-5-4!
!1997-12-25!

Verwenden Sie für Nulldatum das Format *!00-00-00!*

Tipp: Der Methodeneditor hat ein Tastaturkürzel für Nulldatum: Tippen Sie ein Ausrufezeichen (!) und drücken die Eingabetaste.

Hinweise:

- Zur Wahrung der Kompatibilität akzeptiert 4D die zweistellige Jahreseingabe. Eine Jahreszahl mit zwei Stellen betrifft standardmäßig das 20. oder 21. Jahrhundert, je nachdem, ob es größer oder kleiner 30 ist. Diese Standardeinstellung lässt sich mit dem Befehl **SET DEFAULT CENTURY** verändern.
- Haben Sie die Option "Verwende regionale Systemeinstellungen" (siehe **Seite Methoden**) markiert, müssen Sie das Datumsformat Ihres Systems verwenden. Im allgemeinen gilt in einer deutschen Umgebung für Datum das Format Tag.Monat.Jahr, getrennt durch Punkte; in der US-Version das Format Monat/Tag/Jahr, getrennt durch Schrägstriche (/).

Zeitkonstanten

Eine Zeitkonstante steht zwischen Fragezeichen (?...?).

In der deutschen und internationalen Version gelten für Zeit das Format Stunde:Minute:Sekunde, getrennt durch Doppelpunkt (:). Zeiten erscheinen im 24-Stunden Format.

Hier ein paar Beispiele:

?00:00:00? ` Mitternacht
?09:30:00? ` 9:30 am
?13:01:59? ` 13 Stunden, 1 Minute und 59 Sekunden

Verwenden Sie für Nullzeit das Format ?00:00:00?

Tipp: Der Methodeneditor hat ein Tastaturkürzel für Nullzeit. Tippen Sie ein Fragezeichen (?) und drücken die **Eingabetaste**.

Variablen

Eine Variable ist eine Speicheradresse. Sie sprechen sie mit ihrem Namen an und weisen ihr einen Wert zu.
Die einem Datenfeld zugewiesenen Werte werden auf der Festplatte gespeichert, die Werte einer Variablen werden dagegen nur für die Laufzeit im Speicher gehalten. Diese sind nicht Teil eines Datensatzes oder einer bestimmten Tabelle.

Beispiel: Sie haben zwei Datenfelder: Nettobetrag ohne MwSt. und MwSt.-Betrag. Sie benötigen nun kein drittes Datenfeld mehr, um den Bruttobetrag mit Mehrwertsteuer zu speichern. Es genügt eine Variable, in der der Brutto-Betrag immer neu berechnet wird.

Ein Variable kann von folgendem Datenfeldtyp sein:

- String (*) oder Text: Alphanumerischer String bis max. 2 GB Text
- Ganzzahl von -32 768 bis 32 767
- Lange Ganzzahl von -2 147 483 647 bis 2 147 483 648
- Zahl von ±1,7e±308 (13 signifikante Stellen)
- Datum von 1.1.100 bis 31.12.32767
- Zeit von 00:00:00 bis 596000:00:00 (Sekunden ab Mitternacht)
- Boolean: Wahr oder Falsch
- Bild: Windows- oder Macintosh-Bild
- Objekt: Ein Satz mit "Eigenschaft/Wert" Paaren in einem Format vom Typ JSON
- Collection: Eine sortierte Liste mit Werten unterschiedlichen Typs
- BLOB (Binary Large OBject): Reihe von Bytes bis zu 2 GB Größe
- Zeiger auf eine Tabelle, Datenfeld, Variable, Array oder Array-Element

(*) Im Unicode Modus sind Variablen vom Typ String oder Text identisch. Im ASCII-Kompatibilitätsmodus ist ein String eine feste alphanumerische Zeichenkette bis zu 255 Zeichen.

Sie können Variablen (außer Zeiger oder BLOB) auf dem Bildschirm darstellen, Daten darin eingeben und sie in Berichten drucken. Variablen mit eingebbarem oder nicht-eingebbarem Bereich verhalten sich wie Datenfelder. So sind auch dieselben Eingabekontrollen verfügbar:

- Anzeigeformate
- Datenbestätigung, wie Eingabefilter und Standardwerte
- Zeichenfilter
- Auswahllisten (hierarchische Listen)
- Eingebbare bzw. nicht eingebbare Werte

Variablen können auch folgende Aktionen ausführen:

- Schaltflächen steuern (Schaltflächen, Kontrollfelder, Optionsfelder, 3D Schaltflächen, usw.)
- Schieberegler steuern (Thermometer, Lineale und Halbkreisskalen)
- Listboxen, rollbare Bereiche, PopUp-Menüs und DropDown-Listen steuern
- Hierarchische Listen und hierarchische PopUp-Menüs steuern
- Schaltflächenraster, Registerkarten, Bildschaltflächen, usw. steuern
- Ergebnisse aus Berechnungen anzeigen, die nicht gesichert werden müssen.

Variable anlegen

Sie können eine Variable einfach durch Benutzen erzeugen. Sie müssen sie nicht formal wie bei Datenfeldern anlegen. Beispiel: Für eine Variable, die das aktuelle Datum plus 30 Tage angibt, schreiben Sie:

```
vDate:=Current date+30
```

4D erstellt *vDate* und gibt das entsprechende Datum an. Der Code liest "vDate gibt das aktuelle Datum plus 30 Tage an". Sie können diese Variable in Ihrer Datenbank beliebig verwenden, also z.B. in einem Datenfeld vom gleichen Typ speichern:

```
[MyTable]MyField:=vDate
```

In einigen Fällen benötigen Sie Variablen, die als ganz bestimmter Typ definiert sind. Weitere Informationen zu Variablen in einer Datenbank finden Sie im Kapitel **Compiler**.

Daten Variablen zuordnen

Sie können Daten in Variablen einsetzen oder herauskopieren. Im ersten Fall weisen Sie Daten einer Variablen zu. Dafür verwenden Sie den Zuordnungsoperator (*:=*). Mit diesem Operator weisen Sie auch Daten einem Datenfeld zu.

Schreiben Sie den Namen der zu erstellenden Variablen auf die linke Seite des Zuordnungsoperators. Die Anweisung

```
vNumber:=3
```

erstellt die Variable *vNumber* und setzt die Zahl *3* ein. Existiert die Variable *vNumber* bereits, wird nur die Zahl *3* eingetragen.

Natürlich sollten Sie aus Variablen auch Daten entnehmen können. Wieder verwenden Sie den Zuordnungsoperator. Wollen Sie den Wert von *vNumber* in das Datenfeld *[Products]Size*, einsetzen, schreiben Sie die Variable *vNumber* auf die rechte Seite des Zuordnungsoperators:

[Products]Size:=*vNumber*

In diesem Fall ist *[Products]Size* gleich 3. Das Beispiel ist recht einfach, zeigt jedoch, wie diese Information über die Programmiersprache von einem Platz zu einem anderen übertragen wird.

Wichtig: Verwechseln Sie nicht den Zuordnungsoperator (*:=*) mit dem Vergleichsoperator (=). Zuordnen und Vergleichen sind ganz unterschiedliche Vorgänge. Weitere Informationen dazu finden Sie im Abschnitt **Operatoren**.

Variablenarten

Der Geltungsbereich einer Variablen bestimmt die Ebene(n), in denen ihr Wert sinnvoll ist. 4D kennt drei unterschiedliche Variablen:

- Lokale Variablen
- Prozessvariablen
- Interprozessvariablen

Lokale Variablen

Eine lokale Variable gilt nur für die Methode, in der sie benutzt wird.

Sie verwenden eine lokale Variable, um:

- Konflikte mit den Namen anderer Variablen zu vermeiden
- Daten vorübergehend zu nutzen
- Die Anzahl der Prozessvariablen zu verringern

Der Name der lokalen Variablen beginnt immer mit einem $-Zeichen und kann bis zu 30 Zeichen lang sein. Längere Namen werden entsprechend gekürzt.

In einer Datenbank mit vielen Methoden und Variablen benötigen Sie eine Variable oft nur in der gerade benutzten Methode. Sie können eine lokale Variable in der Methode erstellen und einsetzen, ohne prüfen zu müssen, ob Sie denselben Variablennamen schon anderweitig verwendet haben.

Ein Benutzer benötigt in einer Datenbank oft nur eine ganz bestimmte Information. Das ist z.B. über den Befehl **Request** möglich. Dieser zeigt eine Meldung an, die der Benutzer beantworten muss. Der Befehl gibt dann die vom Benutzer eingegebene Information zurück. Diese Information muss normalerweise nicht sehr lange in der Methode verbleiben. Das ist eine typische Anwendung für eine lokale Variable. Hier ein Beispiel:

```
$vsID:=Request("Geben Sie Ihre Kennung ein:")
If(OK=1)
   QUERY([People];[People]ID =$vsID)
End if
```

Diese Methode fordert den Benutzer auf, eine Nummer einzugeben, setzt die Antwort in eine lokale Variable, *$vsID* und sucht dann nach der eingegebenen Nummer. Die lokale Variable *$vsID* wird aus dem Speicher entfernt, sobald die Methode endet. Das ist korrekt, denn die Variable wird nur einmal und nur in dieser Methode verwendet.

Prozessvariablen

Eine Prozessvariable ist nur dem Prozess bekannt, in dem sie geschrieben wurde. Sie ist für die Prozessmethode und alle Methoden verfügbar, die von diesem Prozess aus aufgerufen werden.

Eine Prozessvariable hat kein Zeichen vor dem Namen und kann bis zu 30 Zeichen lang sein.

Im interpretierten Modus werden Variablen dynamisch verwaltet, d.h. sie werden im Speicher "on the fly" erstellt und wieder entfernt. Im kompilierten Modus teilen sich alle erstellten Prozesse (Benutzerprozesse) dieselbe Definition der Prozessvariablen, wobei jeder Prozess für jede Variable eine andere Instanz hat. So ist die Variable *myVar* eine Variable im Prozeß *P_1* und eine andere im Prozess *P_2*.

Ein Prozess kann mit den Befehlen **GET PROCESS VARIABLE** und **SET PROCESS VARIABLE** Prozessvariablen aus einem anderen Prozess "peek and poke". Verwenden Sie diese Befehle beim Programmieren nur für die von 4D dafür vorgesehenen Situationen:

- Interprozesskommunikation an spezifischen Stellen Ihres Codes.
- Drag und Drop auf Interprozessebene anwenden
- Im Client/Server-Betrieb Kommunikation zwischen Prozessen auf Client-Rechnern und Serverprozeduren auf Server-Rechnern.

Weitere Informationen finden Sie im Kapitel **Prozesse** und in der Beschreibung dieser Befehle.

Interprozessvariablen

Interprozessvariablen sind allen Prozessen der Datenbank mit ihrem Wert bekannt. Sie dienen hauptsächlich zum Informationsaustausch zwischen den einzelnen Prozessen.

Ihr Name beginnt mit den Symbol <>, ein "kleiner als" Zeichen gefolgt von einem "größer als" Zeichen. Er kann bis zu 30 Zeichen lang sein.

Hinweis: Diese Syntax gilt für Windows und Macintosh. Auf Macintosh kann auch das Zeichen ◇ benutzt werden, dieses erhalten Sie mit der Tastenkombination **Umschalt-/Wahltaste + v**.

Im Client/Server-Betrieb greifen alle Rechner (Client und Server) auf dieselbe Definition der Interprozessvariablen zu, aber jeder Rechner hat für jede Variable eine andere Instanz.

Variablen für Formularobjekte

Im Formulareditor wird jedes aktive Objekt — Schaltfläche, Optionsfeld, Kontrollkästchen, rollbarer Bereich usw. — über einen Objektnamen identifiziert und automatisch eine Variable oder Ausdruck zugeordnet. Standardmäßig wird die Variable nicht beim Erstellen des Objekts definiert, sondern dynamisch beim Laden des Formulars angelegt (siehe unten). Bei Bedarf können Sie die

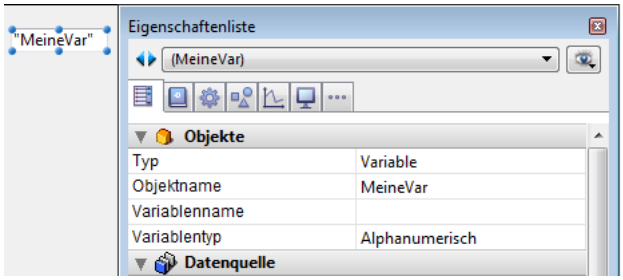
Variable in der Eigenschaftenliste benennen, um sie zu erstellen. Erstellen Sie z.B. eine Schaltfläche mit dem Namen *MyButton*, können Sie eine Variable mit dem gleichen Namen *MyButton* zuweisen.

Mit Variablen für Formularobjekte steuern und verwalten Sie die Objekte. Wurde z.B. die Schaltfläche angeklickt, wird die dazugehörige Variable auf 1 gesetzt; ansonsten hat sie den Wert 0. Über eine einem Thermometer zugeordnete Variable können Sie die aktuellen Einstellungen ablesen und ändern. Ziehen Sie z.B. ein Thermometer auf eine neue Einstellung, ändert sich der Wert der Variablen, um die neue Einstellung anzuzeigen. Genauso wird, wenn eine Methode den Wert der Variablen ändert, das Thermometer neu gezeichnet, um den neuen Wert anzuzeigen.

Weitere Informationen zu Variablen und Formularen finden Sie im Handbuch *4D Designmodus* und im Kapitel **Formularereignisse**.

Dynamische Variablen

Sie können es 4D überlassen, Variablen zu Ihren Formularobjekten (Schaltflächen, eingebbare Variablen, Optionsfelder, etc.) dynamisch und gemäß Ihren Bedürfnissen zu erstellen. Dafür lassen Sie in der Eigenschaftenliste das Feld "Variablenname" für das Objekt leer:



Ist beim Laden des Formulars eine Variable nicht benannt, erstellt 4D eine neue Variable für das Objekt, mit einem berechneten Namen, der im Bereich der Prozessvariablen des Interpreters einmalig ist (d.h. diese Funktionsweise ist sogar im kompilierten Modus verwendbar). Diese temporäre Variable wird beim Schließen des Formulars zerstört.
Damit dieses Prinzip im kompilierten Modus funktioniert, müssen dynamische Variablen explizit typisiert werden. Dafür gibt es zwei Möglichkeiten:

- Sie können den Typ über das Menü "Variablentyp" der Eigenschaftenliste setzen
  Hinweis: Das Menü "Variablentyp" typisiert eine benannte Variable nicht, sondern ermöglicht nur, die Optionen der Eigenschaftenliste zu aktualisieren (Bildvariablen ausgenommen). Zur Typisierung einer benannten Variablen müssen Sie die Befehle im Kapitel **Compiler** verwenden.
- Sie können beim Laden des Formulars einen spezifischen Initialisierungscode verwenden, z.B. den Befehl **VARIABLE TO VARIABLE**:

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

Hinweis: Definieren Sie eine dynamische Variable, d.h. eine Variable ohne Namen und wählen im Menü "Variablentyp" den Wert **Nichts** aus, gibt der Compiler einen Typisierungsfehler zurück.

Im 4D Code ist der Zugriff auf dynamische Variablen durch einen Zeiger möglich, der über den Befehl **OBJECT Get pointer** aufgerufen wird. Zum Beispiel:

```
// Der Variablen für das Objekt "tstart" die Zeit 12:00:00 zuweisen
$p :=OBJECT Get pointer(Object named;"tstart")
$p->:=?12:00:00?
```

Diese Funktionsweise hat zwei Vorteile:

- Sie erlaubt einerseits, Komponenten vom Typ "Unterformular" zu entwickeln, die sich im selben Host-Formular mehrmals verwenden lassen. Nehmen wir z.B. ein Unterformular *datepicker* , das zweimal in das Host-Formular eingefügt wird, um ein Anfangs- und Enddatum zu setzen. Dieses Unterformular verwendet Objekte, um das Datum des Monats und des Jahres zu wählen. Dafür müssen für das Anfangs- bzw. das Enddatum verschiedene Variablen verwendet werden. Eine Möglichkeit zur Lösung dieses Problems ist, 4D ihre Variable mit einem einmaligen Namen erstellen zu lassen.
- Sie erlaubt andererseits, die Speichernutzung zu begrenzen. Formularobjekte arbeiten nur mit Prozess- oder Interprozessvariablen. Im kompilierten Modus wird jedoch in allen Prozessen, einschließlich Serverprozessen, für jede Prozessvariable eine Instanz erstellt. Diese Instanz beansprucht Speicherplatz, selbst wenn das Formular während der Sitzung nicht verwendet wird. Deshalb kann Speicherplatz eingespart werden, wenn 4D die Variablen beim Laden der Formulare dynamisch erstellt.

Hinweis: Gibt es keinen Variablennamen, wird der Objektname in im Formulareditor in Anführungszeichen angezeigt (wenn das Objekt standardmäßig einen Variablennamen anzeigt).

Systemvariablen

Einige Variablen legt 4D selbst an. Sie dürfen nicht neu definiert oder neu typisiert werden. Sie sind beim ersten Öffnen der Datenbank noch nicht angelegt. Sie werden erst beim ersten Aufruf der Aktion, die die Variable erzeugt, von 4D definiert.

Die wichtigste Systemvariable ist **OK**. Diese Variable aktualisiert 4D laufend. Mit ihr fragen Sie ab, ob eine Eingabe bestätigt oder abgebrochen wurde, also ob der Datensatz gesichert wurde oder ob der Importvorgang abgeschlossen wurde. Wurde ein Vorgang erfolgreich abgeschlossen, hat die Systemvariable OK den Wert 1, ansonsten den Wert 0.

Weitere Informationen über Systemvariablen finden Sie im Abschnitt **Systemvariablen**.

Systemvariablen

4D verwaltet **Systemvariablen**, mit denen Sie die Ausführung verschiedener Operationen steuern können. Alle Systemvariablen sind Prozessvariablen, auf die Sie nur innerhalb eines Prozesses zugreifen können. Im folgenden werden die Systemvariablen von 4D beschrieben.

Weitere Informationen dazu finden Sie im Abschnitt **Richtlinien zur Typisierung**.

OK

Dies ist die gebräuchlichste Systemvariable. Sie hat den Wert 1, wenn die Operation erfolgreich ausgeführt wird, sonst den Wert 0 (Null). Viele 4D Befehle verändern den Wert der Systemvariablen *OK*. Ob dies der Fall ist, finden Sie in der Beschreibung der einzelnen Befehle.

In dieser Dokumentation zeigt das Pictogramm 

, dass der Befehl den Wert der Systemvariable OK ändert. Klicken Sie darauf, können Sie die Liste aller betroffenen Befehle generieren.

Document

Document enthält den vollständigen Namen (Zugriffspfad+Name) oder den Namen der Datei, die zuletzt mit folgenden Befehlen geöffnet oder erstellt wurde:

Append document	**BUILD APPLICATION**
Create document	**_o_Create resource file**
EXPORT DATA	**EXPORT DIF**
EXPORT SYLK	**EXPORT TEXT**
GET DOCUMENT ICON	**EXPORT DIF**
IMPORT DIF	**IMPORT SYLK**
IMPORT TEXT	**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 enthält den ASCII Code, mit dem Felder beim Export oder Import voneinander getrennt werden. *FldDelimit* hat standardmäßig den Wert 9. Das ist der ASCII Code für die Tabulartortaste. Wollen Sie einen anderes Trennzeichen verwenden, weisen Sie *FldDelimit* einen neuen Wert zu.

RecDelimit

RecDelimit enthält den ASCII Code, mit dem Datensätze beim Export oder Import voneinander getrennt werden. *RecDelimit* hat standardmäßig den Wert 13. Das ist der ASCII Code für die Zeilenschaltung. Wollen Sie ein anderes Trennzeichen verwenden, weisen Sie *RecDelimit* einen neuen Wert zu.

Error, Error method, Error line, Error formula

Diese Variablen sind nur in einer Methode verwendbar, die mit dem Befehl **ON ERR CALL** aufgerufen wurde. Sollen sie in der Methode verfügbar sein, die den Fehler verursacht hat, kopieren Sie den jeweiligen Wert in Ihre eigene Prozessvariable.

- *Error:* Systemvariable vom Typ Lange Ganzzahl. Diese Variable enthält die Fehlernummer. Die Fehler von 4D sowie Systemfehler sind im Kapitel **Fehlermeldungen** aufgelistet.
- *Error method:* Systemvariable vom Typ Text. Diese Variable enthält den kompletten Namen der Methode, die den Fehler ausgelöst hat.
- *Error line:* Systemvariable vom Typ lange Ganzzahl. Diese Variable enthält die Zeilennummer, wo der Fehler in dieser Methode auftritt.
- *Error formula*: Systemvariable vom Typ Text. Diese Variable enthält die Formel des 4D Code (Volltext), wo der Fehler auftritt. Der Formeltext erscheint in der aktuellen Sprache des 4D Code.
  Wird der für den Fehler verantwortliche Quellcode nicht gefunden, enthält **Error formula** einen leeren String. Das kann passieren, wenn:
  - Der Quellcode über den 4D Application Builder aus der kompilierten Struktur entfernt wurde.
  - Der Quellcode verfügbar ist, die Anwendung jedoch ohne die Option **Range Checking** kompiliert wurde.

MouseDown, MouseX, MouseY, KeyCode, Modifiers und MouseProc

Diese Systemvariablen lassen sich nur in einer Methode verwenden, die mit dem Befehl **ON EVENT CALL** aufgerufen wurde (außer *MouseX* und *MouseY* in einigen Fällen).

- *MouseDown* hat den Wert 1, wenn die Maustaste gedrückt wurde, sonst den Wert 0 (Null).
- Ist das Ereignis ein Mausklick (*MouseDown*=1), geben die Systemvariablen *MouseX* und *MouseY* die vertikale und horizontale Koordinate der Position des letzten Mausklicks an. Beide Werte werden in Pixel ausgedrückt und arbeiten mit dem lokalen Koordinatensystem des Fensters.
- Bei Klick auf ein Datenfeld vom Typ Bild oder Variable geben *MouseX* und *MouseY* die lokalen Koordinaten des Mausklicks in den Formularereignissen *On Clicked* oder *On Double clicked* bzw. in *On Mouse Enter* und *On Mouse Move* zurück. Die Koordinaten werden in Pixel angegeben, ausgehend von der oberen linken Ecke des Bildes (0,0). Weitere Informationen dazu finden Sie im Abschnitt **Überblick über Bilder** und unter dem Befehl **SVG Find element ID by coordinates**

- *KeyCode* ergibt den Zeichencode der zuletzt gedrückten Taste auf der Tastatur. Handelt es sich um eine Funktionstaste, ergibt *KeyCode* einen speziellen Code. Die Liste für Zeichencodes und Funktionstasten finden Sie im Abschnitt **EXPORT TEXT**.
- *Modifiers* hat nur einen sinnvollen Wert in einer Methode, die mit dem Befehl **ON EVENT CALL** aufgerufen wurde. Sie ergibt den Code für die jeweils gedrückte Sondertaste (**Ctrl/Befehlstaste**, **Alt/Wahltaste**, **Umschalttaste**, **Feststelltaste**).
- *MouseProc* gibt die Nummer des Prozesses an, in dem der letzte Mausklick ausgeführt wurde.

Zeiger

Beschreibung

Mit Hilfe der Zeiger können Sie Ihre Methoden strukturunabhängig schreiben und dadurch Ihre Programme übertragbarer gestalten.

Sie können mit einem Zeiger Objekte wie Tabellen, Variablen lesen oder ändern, ohne das Objekt genau anzugeben, bzw. ohne genau zu wissen, welches Objekt es ist.

Hinter einem Zeiger steht folgendes Konzept: Einen gesuchten Rechner erkennen Sie normalerweise an seiner Seriennummer. Dies ist aber nicht der einzige Weg. Sie können ihn auch über seinen Besitzer identifizieren. Sprechen Sie den Rechner über die Seriennummer an, nennt man das "direktes" Ansprechen; sprechen Sie ihn über den Besitzer an, z. B. "Meier's PC", so nennt man das "indirektes" Ansprechen. Diese Methode hat einige Vorteile: Meier kann seinen Rechner beliebig auswechseln. Sie erkennen ihn immer an seinem Besitzer. Das Objekt wird also über eine Art Vermittler, der auf das Objekt zeigt, angesprochen.

Mit Zeigern können Sie auf Tabellen, Datenfelder, Variablen, Arrays und Tabellenelemente verweisen. Folgende Tabelle zeigt ein Beispiel für jeden Datentyp:

Objekt	Referenz	Verwenden	Zuweisen
Tabelle	vpTable:=->[Table]	DEFAULT TABLE(vpTable->)	---
Datenfeld	vpField:=->[Table]Field	ALERT(vpField->)	vpField->:="John"
Variable	vpVar:=->Variable	ALERT(vpVar->)	vpVar->:="John"
Array	vpArr:=->Array	SORT ARRAY(vpArr->;>)	COPY ARRAY (Arr;vpArr->)
Array Element	vpElem:=->Array{1}	ALERT (vpElem->)	vpElem->:="John"

Zeiger einsetzen

Wir erläutern die Verwendung von Zeigern an einem Beispiel. Wir erstellen zuerst eine Variable:

```
MyVar:="Hello"
```

MyVar ist nun eine Variable mit der Zeichenkette *"Hello"*. Wir erstellen nun den Zeiger für *MyVar*:

```
MyPointer:=->MyVar
```

Das Symbol -> bedeutet "setze Zeiger auf." Dieses Symbol ist die Kombination aus einem Bindestrich und dem Zeichen "größer als". In diesem Fall zeigt der Zeiger auf *MyVar*. Der Zeiger wird mit dem Zuordnungsoperator dem Zeiger *MyPointer* zugewiesen.

MyPointer ist nun eine Variable mit einem Zeiger auf *MyVar*. *MyPointer* enthält nicht *"Hello"*, den Wert von *MyVar*, Sie erhalten diesen Wert über *MyPointer*. Folgender Ausdruck gibt den Wert in *MyVar* zurück:

```
MyPointer->
```

In diesem Fall gibt er die Zeichenkette *"Hello"* zurück. Das Symbol -> nach dem Zeiger verweist auf das Objekt, auf das gezeigt wird. Das nennt man **Dereferenzierung**.

Sie können den Ausdruck *MyPointer->* überall verwenden, wo Sie auch die urspüngliche Variable *MyVar* einsetzen können.

Folgende Programmierzeile zeigt eine Warnung mit dem Wort Hello:

```
ALERT(MyPointer->)
```

Über *MyPointer* können Sie auch den Inhalt von *MyVar* ändern. Folgendes Statement speichert die Zeichenkette *"Goodbye"* in der Variablen *MyVar*:

```
MyPointer->:="Goodbye"
```

Wie Sie sehen, verhält sich der Ausdruck *MyPointer->* wie die Variable *MyVar*. Die beiden nachfolgenden Zeilen führen dieselbe Aktion aus—beide zeigen eine Warnung mit dem aktuellen Wert der Variablen *MyVar*:

```
ALERT(MyPointer->)
ALERT(MyVar)
```

Die beiden nachfolgenden Zeilen führen dieselbe Aktion aus—beide weisen der Variablen *MyVar* die Zeichenkette *"Goodbye"* zu:

```
MyPointer->:="Goodbye"
MyVar:="Goodbye"
```

Zeiger auf Schaltflächen

Dieser Abschnitt beschreibt die Verwendung von Zeigern am Beispiel Schaltflächen. Die Beschreibung gilt für alle Variablen, die über einen Zeiger angesprochen werden können.

Wir gehen von einer Anzahl Schaltflächen in Ihren Formularen aus, die aktiviert oder deaktiviert werden sollen. Jeder Schaltfläche ist eine Bedingung zugewiesen, die *TRUE* oder *FALSE* ist. Die Bedingung gibt an, ob die Schaltfläche aktiviert oder deaktiviert wird. Sie können den Status über folgenden Code prüfen:

```
If(Condition)  ` Ist die Bedingung TRUE…
   OBJECT SET ENABLED(MyButton;True)  ` aktiviere Schaltfläche
Else  ` Sonst…
   OBJECT SET ENABLED(MyButton;False)  ` deaktiviere Schaltfläche
End if
```

Denselben Test benötigen Sie für alle weiteren Schaltflächen, lediglich der Name der Schaltfläche ändert sich. Hier bietet sich an, die Schaltfläche über einen Zeiger anzusprechen und für den eigentlichen Test eine Subroutine zu verwenden.

Bei Subroutinen müssen Sie Zeiger einsetzen, da Sie nur so die Variablen der Schaltflächen ansprechen können. Ihre Projektmethode könnte folgendermaßen aussehen:

```
  ` Projektmethode SET BUTTON
  ` SET BUTTON ( Pointer ; Boolean )
  ` SET BUTTON ( -> Button ; aktiviere oder deaktiviere )
  ` $1 – Zeiger auf eine Schaltfläche
  ` $2 – Boolean. Bei TRUE aktiviere die Schaltfläche. Bei FALSE deaktiviere die Schaltfläche

If($2)  ` Ist die Bedingung TRUE…
   OBJECT SET ENABLED($1->;True)  ` aktiviere die Schaltfläche
Else  ` Sonst…
   OBJECT SET ENABLED($1->;False)  ` deaktiviere die Schaltfläche
End if
```

Die Projektmethode *SET BUTTON* aufrufen:

```
  ` …
SET BUTTON(->bValidate;True)
  ` …
SET BUTTON(->bValidate;False)
  ` …
SET BUTTON(->bValidate;([Employee]Last Name#"")
  ` …
For($vlRadioButton;1;20)
   $vpRadioButton:=Get pointer("r"+String($vlRadioButton))
   SET BUTTON($vpRadioButton;False)
End for
```

Zeiger auf eine Tabelle

Sie können auch einen Zeiger erstellen, der auf eine Tabelle zeigt. Sie definieren den Zeiger folgendermaßen:

```
TablePtr:=->[anyTable]
```

Sie können einen Zeiger über die Funktion **Table** einrichten:

```
TablePtr:=Table(20)
```

oder den dereferenzierten Zeiger in Befehlen verwenden:

```
DEFAULT TABLE(TablePtr->)
```

Zeiger auf ein Datenfeld

Sie können einen Zeiger verwenden, der das Datenfeld anspricht. Sie definieren den Zeiger folgendermaßen:

```
FieldPtr:=->[Table1]Field
```

Sie können einen Zeiger über die Funktion **Field** einrichten:

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

oder den dereferenzierten Zeiger in Befehlen verwenden:

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

Zeiger auf Variablen

Das Beispiel zu Beginn dieses Abschnitts erläutert die Verwendung eines Zeigers auf eine Variable:

```
MyVar:="Hello"
MyPointer:=->MyVar
```

Sie können Zeiger auf Interprozess- und Prozessvariablen, ab Version 2004.1 auch auf lokale Variablen verwenden.

Bei Verwendung auf Prozess- oder lokale Variablen müssen Sie sicherstellen, dass die Variable, auf die gezeigt wird, bereits eingerichtet ist. Beachten Sie folgendes: Lokale Variablen werden gelöscht, wenn die Ausführung der Methode, welche diese erstellt hat, beendet ist; Prozessvariablen werden mit Beenden des Prozesses gelöscht, der sie erstellt hat. Ruft ein Zeiger eine Variable auf, die nicht mehr existiert, verursacht das im interpretierten Modus einen Syntaxfehler (Variable nicht definiert), im kompilierten Modus kann dies hingegen zu einem ernsteren Fehler führen.

Hinweis zu lokalen Variablen: Über Zeiger auf lokale Variablen lassen sich in vielen Fällen Prozessvariablen sichern. Zeiger auf lokale Variablen lassen sich nur im gleichen Prozess verwenden.
Im Debugger erscheint der ursprüngliche Methodenname nach dem Zeiger in Klammern, wenn Sie einen Zeiger auf eine lokale Variable anzeigen, die in einer anderen Methode deklariert wurde. Schreiben Sie zum Beispiel in ***Method1***:

```
$MyVar:="Hello world"
Method2(->$MyVar)
```

Zeigt der Debugger in ***Method2*** *$1* wie folgt an:

```
$1   ->$MyVar (Method1)
```

Der Wert von $1 lautet:

```
$MyVar (Method1)   "Hello world"
```

Zeiger auf Array Elemente

Sie können einen Zeiger auf ein Array Element erstellen. Im folgenden Beispiel wird ein Array erstellt und dem ersten Element des Array ein Zeiger auf die Variable ElemPtr zugewiesen:

```
ARRAY REAL(anArray;10)  `Array anlegen
ElemPtr:=->anArray{1}  `Zeiger auf das Array Element erstellen
```

Sie können den dereferenzierten Zeiger verwenden, um dem Element einen Wert zuzuweisen, z.B.:

```
ElemPtr->.=8
```

Zeiger auf Arrays

Sie können auch einen Zeiger erstellen, der auf ein Array zeigt. Da Sie einer Methode kein komplettes Array als Parameter übergeben können, ist diese Möglichkeit besonders wichtig. Sie übergeben statt des Array einfach einen Zeiger und verwenden diesen in der Unterroutine, um so das Array anzusprechen.
Folgende Methode erstellt ein Array und weist dem Array einen Zeiger auf die Variable *ArrPtr* zu:

```
ARRAY REAL(Array;10)  ` Erstelle Array
ArrPtr:=->Array  ` Erstelle Zeiger auf Array
```

Beachten Sie, dass der Zeiger das Array anspricht und nicht ein Element dieses Arrays. Sie können den soeben angelegten Zeiger folgendermaßen einsetzen:

```
SORT ARRAY(ArrPtr->;>)  ` Sortiere Array
```

oder über den Zeiger das 4. Element des Arrays ansprechen:

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

Zeiger-Array

Mit einem Zeiger-Array können Sie eine Gruppe verknüpfter Objekte ansprechen.

Ein Beispiel dafür ist ein Variablenraster in einem Formular. Die Variablen im Raster sind durchnumeriert, also *Var1*, *Var2*, …, *Var10*. Oft müssen Sie diese Variablen indirekt mit einer Nummer ansprechen. Dafür erstellen Sie einen Zeiger-Array, und initialisieren die Zeiger, so dass jede Variable angesprochen wird. Sie schreiben folgenden Code:

```
ARRAY POINTER(apPointer;10)  ` Erstelle ein Array für 10 Zeiger
For($i;1;10)  ` Durchlaufe einmal für jede Variable
   apPointer{$i}:=Get pointer("Var"+String($i))  ` Initialisiere das Array Element
End for
```

Die Funktion **Get pointer** gibt den Zeiger für das genannte Objekt zurück.

Sie sprechen eine beliebige Variable über die Elemente des Array an. Um z.B. Variablen vom Typ Datum mit den nächsten 10 Kalenderdaten zu füllen, schreiben Sie folgenden Code:

```
For($i;1;10)  ` Durchlaufe einmal für jede Variable
   apPointer{$i}->:=Current date+$i   ` Weise die Kalenderdaten zu
End for
```

Schaltflächen über Zeiger setzen

Über Zeiger können Sie rasch eine Gruppe verknüpfter Optionsfelder setzen. Die Schaltflächen sind durchnumeriert, also *Button1*, *Button2*, …, *Button5*.

In einer Gruppe von Optionsfeldern ist immer nur ein Optionsfeld an. Die Zahl dieses Optionsfeldes lässt sich in einem numerischen Datenfeld speichern. Beispiel: Enthält das Datenfeld *[Preferences]Setting* die Zahl *3*, wird *Button3* ausgewählt. Die Formularmethode zum Setzen der Schaltfläche lautet:

```
Case of
   :(Form event=On Load)
   ` ...
      Case of
         :([Preferences]Setting=1)
            Button1:=1
         :([Preferences]Setting=2)
            Button2:=1
         :([Preferences]Setting=3)
            Button3:=1
         :([Preferences]Setting=4)
            Button4:=1
         :([Preferences]Setting=5)
            Button5:=1
      End case
   ` ...
End case
```

Jedes Optionsfeld muss in einer eigenen Abfrage getestet werden. Das ergäbe bei vielen Optionsfeldern im Formular eine sehr lange Methode. Dafür verwenden Sie Zeiger. Mit der Funktion **Get pointer** können Sie einem Optionsfeld einen Zeiger zuordnen. Schreiben Sie dafür folgenden Code:

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

Die Zahl der gesetzten Schaltfläche muss im Datenfeld *[Preferences]Setting* gespeichert werden. Dafür verwenden Sie die Formularmethode für das Ereignis *On Clicked*:

```
[Preferences]Setting:=Button1+(Button2*2)+(Button3*3)+(Button4*4)+(Button5*5)
```

Zeiger in Methoden

Sie können einen Zeiger als Parameter in einer Methode übergeben. Die Methode ***TAKE TWO*** enthält zwei Parameter in Form von Zeigern. Das erste angesprochene Objekt wird in Großbuchstaben umgewandelt, das zweite in Kleinbuchstaben. Die Methode sieht folgendermaßen aus:

```
` Projektmethode TAKE TWO
` $1 – Zeiger auf alphanumerisches Feld oder Variable. Wandle dieses um in Großbuchstaben.
` $2 – Zeiger auf alphanumerisches Feld oder Variable. Wandle dieses um in Kleinbuchstaben.
$1->:=Uppercase($1->)
$2->:=Lowercase($2->)
```

Nachfolgende Zeile wandelt über die Methode ***TAKE TWO*** ein Datenfeld in Großbuchstaben um und eine Variable in Kleinbuchstaben:

```
TAKE TWO(->[MyTable]MyField;->MyVar)
```

Das Datenfeld *[MyTable]MyField* mit dem alphanumerischen Inhalt *"meier",* wird umgewandelt in *"MEIER"*. Die Variable *MyVar* mit dem alphanumerischen Inhalt *"hello"* wird umgewandelt in *"Hello"*.

Achten Sie darauf, dass in der Methode ***TAKE TWO*** und immer, wenn Sie Zeiger verwenden das angesprochene Objekt vom richtigen Datentyp ist. Im vorigen Beispiel muss der Zeiger ein Objekt vom Typ alphanumerisch oder Text ansprechen.

Zeiger auf Zeiger

Wenn Sie die Dinge komplexer machen wollen, können Sie auch Zeiger verwenden, die auf andere Zeiger verweisen. Beispiel:

```
MyVar:="Hello"
PointerOne:=->MyVar
PointerTwo:=->PointerOne
(PointerTwo->)->:="Goodbye"
ALERT((Point Two->)->)
```

Es zeigt eine Warnung mit dem Inhalt “Goodbye”.
Im folgenden werden die einzelnen Zeilen erläutert:

- MyVar:="Hello"
  --> Diese Zeile setzt die Zeichenkette "Hello" in die Variable *MyVar*.
- PointerOne:=->MyVar
  --> *PointerOne* enthält nun den Zeiger auf *MyVar*.
- PointerTwo:=->PointerOne
  --> *PointerTwo* (eine neue Variable) enthält einen Zeiger auf *PointerOne*, der auf *MyVar* zeigt.
- (PointerTwo->)->:="Goodbye"
  --> *PointerTwo->* verweist auf den Inhalt von *PointerOne*, der wiederum auf *MyVar* verweist.
  (*PointerTwo->)->* verweist also auf den Inhalt von *MyVar*. In diesem Beispiel ist also der Variablen *MyVar* "Goodbye" zugewiesen.
- ALERT ((PointerTwo->)->)
  --> Hier gilt das gleiche: *PointerTwo->*PointerTwo-> verweist auf den Inhalt von *PointerOne*, der wiederum auf *MyVar* verweist. (*PointerTwo->)->* verweist also auf den Inhalt von *MyVar*. In diesem Beispiel zeigt die Warnung den Inhalt von *MyVar* an.

Die folgende Zeile setzt "Hello" in *MyVar*:

```
(PointerTwo->)->:="Hello"
```

Die folgende Zeile nimmt "Hello" aus *MyVar* und setzt es in *NewVar*:

```
NewVar:=(PointerTwo->)->
```

Wichtig: Bei mehrfacher Dereferenzierung müssen Sie mit Klammern arbeiten.

Namenskonventionen

Dieser Abschnitt beschreibt die Konventionen, die Sie beim Benennen von Objekten in der 4D Programmiersprache beachten müssen. Folgende Regeln gelten für alle Objekte:

- Ein Name muss mit einem Buchstaben oder einem Unterstrich beginnen.
- Der Name kann Buchstaben, Zahlen, Leerzeichen und den Unterstrich enthalten.
- Punkte, Schrägstriche, Anführungszeichen und Kommas sind nicht erlaubt.
- In Tabellen-, Feld- und Methodennamen sind Punkte (".") bzw. eckige Klammern ("[ ]") nicht erlaubt, das gilt für Variablennamen, wenn Objektnotation aktiviert ist. (siehe **Seite Kompatibilität**)
- Zeichen für Operatoren, wie z.B. * und + sind nicht erlaubt.
- 4D ignoriert alle führenden Leerzeichen.
- Der Name kann Umlaute wie ä, ö, ü enthalten, sie werden jedoch nicht berücksichtigt: mühle = muhle

Hinweis: Bei Objekten, die über SQL verwaltet werden, müssen Sie zusätzliche Regeln beachten. Zugelassen sind nur die Zeichen _0123456789abcdefghijklmnopqrstuvwxyz, der Name darf keine SQL Keywords, also Befehle, Attribute u.ä. enthalten. Der SQL Bereich des Inspektors im Strukturedítor zeigt automatisch alle nicht erlaubten Zeichen bei Tabellen- oder Feldnamen an.

Befehle werden in einer speziellen Schrift und in Großbuchstaben geschrieben, zum Beispiel: **ADD RECORD**. 4D Funktionen werden mit großem Anfangsbuchstaben geschrieben, zum Beispiel: **Records in selection**.

Befehle, Funktionen und Anweisungen in Methoden oder Objektmethoden werden fett geschrieben. Befehle oder Funktionen von Plug-Ins haben eine spezifische Vorsilbe und werden zusätzlich kursiv geschrieben, um sie von den 4D Befehlen und Funktionen zu unterscheiden:

```
QUERY([Vorlagen];[Vorlagen]ID=vNummer)  ` 4D Befehl
If(Records in selection([Vorlagen])=1)
   WR PICTURE TO AREA(Bereich;[Vorlagen]Dok)  ` 4D Write Befehl
End if
```

In einigen Beispielen dieses Handbuchs passen Code-Zeilen nicht in eine Zeile, sie müssen in einer zweiten, manchmal einer dritten Zeile fortgesetzt werden. Wenn Sie diese Beispiele übernehmen, nehmen Sie diese Zeilen als eine Einheit. Verwenden Sie hier keine Zeilenschaltung, denn das unterbricht den Programmierfluss.

Tabellen

Eine Tabelle kennzeichnen Sie durch eckige Klammern: *[...]*. Der Tabellenname kann max. 31 Zeichen lang sein.

Beispiele:

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

Felder

Ein Datenfeld kennzeichnen Sie durch die vorangestellte Tabelle, zu der dieses Datenfeld gehört. Der Datenfeldname folgt unmittelbar auf die Tabelle. Der Datenfeldname kann max. 31 Zeichen lang sein.

Beispiele:

```
[Orders]Total:=Sum([Line]Amount)
QUERY([Clients];[Clients]Name="Meier")
[Letters]Text:=Capitalize text([Letters]Text)
```

Interprozessvariablen

Eine Interprozessvariable kennzeichnen Sie mit dem vorangestellten Symbol (<>) — die Zeichen "kleiner als" und "größer als".

Der Name einer Interprozessvariablen kann ohne die vorangestellten Symbole bis zu 255 Zeichen lang sein.
Die Länge ist auf 31 Zeichen beschränkt, wenn die Option für Kompatibilität "Sichere Methode in Unicode" nicht markiert ist (siehe **Seite Kompatibilität**).

Beispiele:

```
<>vlProcessID:=Current process
<>vsKey:=Char(KeyCode)
If(<>vtName#"")
```

Prozessvariablen

Eine Prozessvariable kennzeichnen Sie durch ihren Namen. (Er kann nicht mit den Symbolen <> oder *$* beginnen). Der Name der Prozessvariablen kann bis zu 255 Zeichen lang sein.
Die Länge ist auf 31 Zeichen beschränkt, wenn die Option für Kompatibilität "Sichere Methode in Unicode" nicht markiert ist (siehe **Seite Kompatibilität**).

Beispiele:

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

Lokale Variablen

Eine lokale Variable kennzeichnen Sie durch das vorangestellte Dollarzeichen (*$*). Der Name einer lokalen Variable kann ohne Dollarzeichen bis zu 255 Zeichen lang sein.
Die Länge ist auf 31 Zeichen beschränkt, wenn die Option für Kompatibilität "Sichere Methode in Unicode" nicht markiert ist (siehe **Seite Kompatibilität**).

Beispiele

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

Arrays

Ein Array kennzeichnen Sie durch seinen Namen. Das ist der Name, den Sie beim Erstellen des Array übergeben haben (z.B. **ARRAY LONGINT**). Arrays sind Variablen, von daher gibt es auch drei Arten:

- Interprozess-Arrays
- Prozess-Arrays
- Lokale Arrays

Interprozess-Arrays
Dem Namen des Interprozess-Arrays ist das Symbol <> vorangestellt— die Zeichen "kleiner als" und "größer als".

Hinweis: Dieses Symbol gilt für Windows und Macintosh. Auf Macintosh können Sie auch das Zeichen ◇ verwenden (Die Kombination **Wahl-/Umschalttaste + v**).

Der Name eines Interprozess-Arrays kann ohne die vorangestellten Symbole bis zu 255 Zeichen lang sein. (*)

Beispiele:

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

Prozess-Arrays
Ein Prozess-Array kennzeichnen Sie durch seinen Namen. (Er kann nicht mit den Symbolen <> oder *$* beginnen). Der Name des Prozess-Array kann bis zu 255 Zeichen lang sein. (*)

Beispiele:

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

Lokale Arrays
Dem Namen des lokalen Array ist das Dollarzeichen (*$*) vorangestellt. Ein lokales Array kann ohne Dollarzeichen bis zu 255 Zeichen lang sein. (*)

Beispiele:

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

(*) Die Länge ist auf 31 Zeichen beschränkt, wenn die Option für Kompatibilität "Sichere Methode in Unicode" nicht markiert ist (siehe **Seite Kompatibilität**).

Elemente von Arrays
Sie verweisen auf ein Element eines Interprozess-, Prozess- oder lokalen Array mit geschweiften Klammern ({…}). Das angesprochene Element ist ein numerischer Ausdruck.

Beispiele:

```
` Element eines Interprozess-Array ansprechen
If(<>asKeywords{1}="Stop")
   <>atSubjects{$vlElem}:=[Topics]Subject
   $viNextValue:=<>aiBigArray{Size of array(<>aiBigArray)}
```

```
` Element eines Prozess-Array ansprechen
If(asKeywords{1}="Stop")
    atSubjects{$vlElem}:=[Topics]Subject
    $viNextValue:=aiBigArray{Size of array(aiBigArray)}

` Element eines lokalen Array ansprechen
    If($asKeywords{1}="Stop")
        $atSubjects{$vlElem}:=[Topics]Subject
        $viNextValue:=$aiBigArray{Size of array($aiBigArray)}
```

Elemente von zweidimensionalen Arrays
Sie verweisen auf ein Element eines Interprozess-, Prozess- oder lokalen Array mit doppelten geschweiften Klammern ({…}). Das angesprochene Element besteht aus zwei numerischen Ausdrücken in geschweiften Klammern.
Beispiele:

```
` Element eines zweidimensionalen Interprozess-Array ansprechen
If(<>asKeywords{$vlNextRow}{1}="Stop")
    <>atSubjects{10}{$vlElem}:=[Topics]Subject
    $viNextValue:=<>aiBigArray{$vlSet}{Size of array(<>aiBigArray{$vlSet})}

` Element eines zweidimensionalen Prozess-Array ansprechen
    If(asKeywords{$vlNextRow}{1}="Stop")
        atSubjects{10}{$vlElem}:=[Topics]Subject
        $viNextValue:=aiBigArray{$vlSet}{Size of array(aiBigArray{$vlSet})}

` Element eines zweidimensionalen lokalen Array ansprechen
    If($asKeywords{$vlNextRow}{1}="Stop")
        $atSubjects{10}{$vlElem}:=[Topics]Subject
        $viNextValue:=$aiBigArray{$vlSet}{Size of array($aiBigArray{$vlSet})}
```

Objektattribute

Ist Objektnotation aktiviert (siehe **Seite Kompatibilität**), kennzeichnen Sie ein Objektattribut (auch Objekteigenschaft genannt) durch einen Punkt (".") zwischen dem Namen des Objekts (oder Attributs) und dem Attributsnamen. Ein Attributsname kann bis zu 255 Zeichen lang sein und unterscheidet zwischen Groß- und Kleinschreibung.
Beispiele:

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

Hinweis: Für Namen von Objektattributen gelten zusätzlich die Schreibregeln für JavaScript (ECMA Script standard). Weitere Informationen dazu finden Sie im Abschnitt **Objektnotation verwenden**.

Formulare

Ein Formular kennzeichnen Sie durch einen alphanumerischen Ausdruck, der dessen Namen darstellt. Ein Formularname kann max. 31 Zeichen lang sein.
Beispiele:

```
INPUT FORM([People];"Input")
FORM SET OUTPUT([People];"Output")
DIALOG([Storage];"Note box"+String($vlStage))
```

Formularobjekte

Sie geben ein Formularobjekt über einen Namen als String mit vorangestelltem Parameter * an. Ein Objektname kann bis zu 255 bytes enthalten.
Beispiel:

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

Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.
Hinweis: Verwechseln Sie nicht Formularobjekte (Schaltflächen, Listboxen, eingebbare Variablen, etc.) und Objekte in der 4D Programmiersprache. Diese werden über Objekt Notation oder spezifische Befehle erstellt und verwaltet (siehe **Objekte (Sprache)**).

Methoden

Eine Methode (Prozedur und Funktion) kennzeichnen Sie durch ihren Namen. Ein Methodenname kann max. 31 Zeichen lang sein.

Hinweis: Eine Methode, die kein Ergebnis zurückgibt, heißt auch **Prozedur**. Eine Methode, die ein Ergebnis zurückgibt, heißt auch **Funktion**.

Beispiele:

```
If(New client)
   DELETE DUPLICATED VALUES
   APPLY TO SELECTION([Employees];INCREASE SALARIES)
```

Tipp: Verwenden Sie zur besseren Übersicht dieselbe Namenskonvention wie bei den 4D Befehlen: Großschreibung für Methoden; Kleinschreibung mit großem Anfangsbuchstaben für Funktionen. Wenn Sie nun eine Datenbank nach einigen Monaten für eine Wartung öffnen, erkennen Sie im Explorer Fenster bereits an der Schreibweise des Namens, ob die Methode ein Ergebnis zurückgibt.

Hinweis: Wollen Sie eine Methode aufrufen, tippen Sie lediglich ihren Namen ein. Einige in 4D integrierten Befehle wie z.B. **ON EVENT CALL**, sowie alle Plug-In Befehle erwarten dagegen den Namen der Methode als Zeichenkette, wenn ein Parameter für die Methode übergeben wurde.

Beispiele:

```
   ` Dieser Befehl erwartet eine Methode (Funktion) oder Formel
QUERY BY FORMULA([aTable];Special query)
   ` Dieser Befehl erwartet eine Methode (Prozedur) oder Anweisung
APPLY TO SELECTION([Employees];INCREASE SALARIES)
   ` Dieser Befehl erwartet einen Methodennamen
ON EVENT CALL("HANDLE EVENTS")
```

Methoden akzeptieren Parameter (Argumente). Parameter stehen in Klammern hinter dem Methodennamen. Sie sind durch Semikolon (*;*) voneinander getrennt. Die Parameter sind innerhalb der aufgerufenen Methode als durchnumerierte lokale Variablen verfügbar: *$1*, *$2*,..., *$n*. Mehrere aufeinanderfolgende (und letzte) Parameter können Sie mit der Syntax *${n}* ansprechen, wobei n, der numerische Audruck, die Zahl des Parameters ist.

In einer Funktion enthält die lokale Variable *$0* den zurückzugebenden Wert.

Beispiele:

```
   ` In DROP SPACES ist $1 ein Zeiger auf das Datenfeld [Employees]Name
DROP SPACES(->[Employees]Name)

   ` In Calc Creator:
   ` - $1 ist numerisch und gleich 1
   ` - $2 ist numerisch und gleich 5
   ` - $3 ist Text oder Zeichenkette und gleich "Nice"
   ` - Der resultierende Wert wird $0 zugewiesen
$vsResult:=Calc creator(1;5;"Nice")

   ` In Text Creator:
   ` - Die drei Parameter sind Text oder Zeichenkette
   ` - Sie können als $1, $2 oder $3 angesprochen werden
   ` - Sie können aber z.B. auch als ${$vlParam} angesprochen werden, wobei $vlParam 1, 2 oder 3 ist
   ` - Der resultierende Wert wird $0 zugewiesen
vtClone:=Text Creator("ist";"der";"es")
```

Plug-In Befehle (Externe Prozeduren, Funktionen und Bereiche)

Einen Plug-In Befehl kennzeichnen Sie durch den Namen, den das Plug-In definiert. Ein Plug-In kann max. 31 Zeichen lang sein.

Beispiel:

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

Mengen

Es gibt zwei Arten:

- Interprozessmengen
- Prozessmengen

4D Server enthält auch:

- Client-Mengen

Interprozessmengen

Dem Namen einer Interprozessmenge wird das Symbol <> vorangestellt — die Zeichen "kleiner als" und "größer als".

Hinweis: Dieses Symbol gilt für Windows und Macintosh. Auf Macintosh können Sie auch das Zeichen ◇ verwenden (Die Kombination **Wahl-/Umschalttaste + v**).

Der Name einer Interprozessmenge kann ohne das Symbol <> bis zu 255 Zeichen lang sein.

Prozessmengen
Eine Prozessmenge kennzeichnen Sie durch einen alphanumerischen Wert, der ihren Namen darstellt. (Er kann nicht mit den Symbolen <> oder *$* beginnen). Der Name der Prozessmenge kann bis zu 255 Zeichen lang sein.

Client-Mengen
Dem Namen einer Client-Menge ist das Dollarzeichen (*$*) vorangestellt. Der Name der Client-Menge kann ohne das Dollarzeichen bis zu 255 Zeichen lang sein.

Hinweis: Mengen werden auf dem Server-Rechner gehalten. Benötigen Sie in bestimmten Fällen Mengen lokal auf dem Client-Rechner, arbeiten Sie mit Client-Mengen.

Beispiele

```
  ` Interprozessmengen
USE SET("<>Deleted Records")
CREATE SET([Customers];"<>Customer Orders")
If(Records in set("<>Selection"+String($i))>0)
  ` Prozessmengen
   USE SET("Deleted Records")
   CREATE SET([Customers];"Customer Orders")
   If(Records in set("<>Selection"+String($i))>0)
  ` Client-Mengen
      USE SET("$Deleted Records")
      CREATE SET([Customers];"$Customer Orders")
      If(Records in set("$Selection"+String($i))>0)
```

Temporäre Auswahlen

Es gibt zwei Arten:

- Temporäre Interprozessauswahlen
- Temporäre Prozessauswahlen

Temporäre Interprozessauswahlen
Dem Namen von temporären Interprozessauswahlen ist das Symbol <> vorangestellt — die Zeichen "kleiner als" und "größer als".

Hinweis: Dieses Symbol gilt für Windows und Macintosh. Auf Macintosh können Sie auch das Zeichen ◇ verwenden (Die Kombination **Wahl-/Umschalttaste + v**).

Der Name einer temporären Interprozessauswahl kann ohne das Symbol <> bis zu 255 Zeichen lang sein.

Temporäre Prozessauswahlen
Eine temporäre Prozessauswahl kennzeichnen Sie durch einen alphanumerischen Wert, der ihren Namen darstellt. (Er kann nicht mit den Symbolen <> oder *$* beginnen). Der Name der temporären Prozessauswahl kann bis zu 255 Zeichen lang sein.

Beispiele:

```
  ` Temporäre Interprozessauswahl
USE NAMED SELECTION([Customers];"<>ByZipcode")
  ` Temporäre Prozessauswahl
USE NAMED SELECTION([Customers];"<>ByZipcode")
```

Prozesse

In der Einzelplatzversion bzw. auf der Client-Seite im Client/Server-Betrieb gibt es zwei Arten:

- Globale Prozesse
- Lokale Prozesse

Globale Prozesse
Sie kennzeichnen einen globalen Prozess durch einen alphanumerischen Wert, der seinen Namen darstellt. (Er kann nicht mit dem Dollarzeichen beginnen). Der Name des globalen Prozesses kann bis zu 255 Zeichen lang sein.

Lokale Prozesse
Dem Namen eines lokalen Prozesses ist das Dollarzeichen (*$*) vorangestellt. Der Name des lokalen Prozesses kann ohne das Dollarzeichen bis zu 255 Zeichen lang sein.

Beispiele

```
  ` Globalen Prozess "Add Customers" starten
$vlProcessID:=New process("P_ADD_CUSTOMERS";48*1024;"Add Customers")
  ` Lokalen Prozess "$Follow Mouse Moves" starten
$vlProcessID:=New process("P_MOUSE_SNIFFER";16*1024;"$Follow Mouse Moves")
```

Zusammenfassung der Namenskonventionen

Typ	Max. Länge	Beispiel
Tabelle	31	[Rechnungen]
Datenfeld	31	[Angestellte]Nachname
Interprozessvariable	<> + 255 (*)	<>vlNächsteProzessID
Prozessvariable	255 (*)	vsAktuellerName
Lokale Variable	$ + 255 (*)	$vlLokalerZähler
Formular	31	"Meine Web Eingabe"
Formularobjekt	255	"MeineSchaltfläche"
Interprozess-Array	<> + 31	<>apTabellen
Prozess-Array	31	asGender
Lokales Array	$ + 31	$atWerte
Methode	31	M_ADD_CUSTOMERS
Plug-in Routine	31	WR INSERT TEXT
Interprozessmenge	<> + 255	"<>Datensätze zum Archivieren"
Prozessmenge	255	"Aktuell ausgewählte Datensätze"
Client-Menge	$ + 255	"$VorigeSubjekte"
Temporäre Auswahl	255	"Angestellte A bis Z"
Temporäre Interprozessauswahl	<> + 255	"<>Angestellte Z bis A"
Lokaler Prozess	$ + 255	"$Ereignisse verfolgen"
Globaler Prozess	255	"P_INVOICES_MODULE"
Semaphore	255	"meineSemaphore"

(*) Die Länge ist auf 31 Zeichen beschränkt, wenn die Option für Kompatibilität "Sichere Methode in Unicode" nicht markiert ist (siehe **Seite Kompatibilität**).

Hinweis: Bei Verwenden nicht-romanischer Zeichen in Namen kann die max. Länge kürzer sein.

Priorität bei Namenskonflikten

Achten Sie darauf, dass die Namen für Objekte in Ihrer Anwendung einmalig sind. Hat ein Objekt denselben Namen wie ein anderes Objekt (beispielsweise ein Datenfeld und eine Variable mit demselben Namen *Person*), verwendet 4D ein Prioritätensystem.

4D identifiziert Namen in Prozeduren in folgender Reihenfolge:

1. Datenfelder
2. Befehle
3. Methoden
4. Plug-In Routinen
5. Vordefinierte Konstanten
6. Variablen

Beispiel: In 4D gibt es den Befehl mit Namen **Date**. Hat eine Methode denselben Namen ***Date***, wertet 4D diese als den integrierten Befehl **Date** und nicht als Methode. Die Methode wird nicht aufgerufen. Haben Sie dagegen ein Datenfeld mit dem Namen *"Date"*, angelegt, versucht 4D, dieses Datenfeld anstatt des Befehls **Date** zu benutzen.

Ablaufsteuerung

In Methoden, ganz gleich, ob sie einfach oder komplex sind, verwenden Sie immer eine oder mehrere der drei Programmierstrukturen. Diese Strukturen steuern die Ausführungsabfolge der Statements in einer Methode. Es gibt folgende Arten:

- Sequentiell
- Abfragen
- Schleifen

Sequentielle Struktur

Die sequentielle Struktur ist eine einfache lineare Struktur. Alle Anweisungen werden nacheinander gemäß ihrer Reihenfolge ausgeführt. Beispiel:

```
OUTPUT FORM([People];"Listing")
ALL RECORDS([People])
DISPLAY SELECTION([People])
```

Ein typisches Beispiel ist die einzeilige Routine, die häufig für Objektmethoden verwendet wird, dies ist die einfachste Form einer sequentiellen Struktur. Beispiel:

```
[People]Last Name:=Uppercase([People]Last Name)
```

Hinweis: Mit den Anweisungen *Begin SQL /End SQL* können Sie sequentielle Strukturen auf die Ausführung mit der SQL Engine von 4D beschränken.

Abfragen

Bei Abfragen werden Befehlsfolgen bedingt ausgeführt. Die Bedingung ist ein boolean Ausdruck, der TRUE oder FALSE ermittelt. In der Struktur *If...Else...End if* wird eine bestimmte Bedingung abgefragt. Es gibt zwei Möglichkeiten. Trifft die Bedingung zu, d.h., ist sie wahr, führt sie den folgenden Befehl aus. Trifft die Bedingung nicht zu, d.h., ist sie falsch, wird der Befehl nicht ausgeführt, eventuell aber ein anderer. In der Struktur *Case of...Else...End case* gibt es mehrere Möglichkeiten, von den letztendlich eine ausgeführt wird.

Schleifen

Schleifen führen eine Anweisung oder eine Folge von Anweisungen wiederholt aus. Die Anweisungen müssen nicht neu geschrieben werden. 4D kennt drei Arten von Schleifen:

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

Schleifen werden entweder solange durchlaufen, bis eine Bedingung gefunden wird oder sie laufen ein bis n-Male ab. Die beiden Möglichkeiten lassen sich zwar für alle Arten von Schleifen einrichten. While- und Repeat-Schleifen eignen sich jedoch besser für das Durchlaufen, bis eine Bedingung gefunden wird, For-Schleifen für den zahlenmäßig begrenzten Durchlauf. Mit **For each...End for each** können Sie beide Arten mischen und damit Objekte und Collections durchlaufen.

Hinweis: 4D ermöglicht Programmierstrukturen (If/While/For/Case of/Repeat) bis zu einer Tiefe von 512 Ebenen einzubinden.

If...Else...End if

Die Syntax für die Abfragefolge **If...Else...End if** lautet:

```
If(Boolean_Expression)
    statements(s)
Else
    statement(s)
End if
```

Die zweite Folge, beginnend mit *Else* ist optional; Sie können auch nur schreiben:

```
If(Boolean_Expression)
    statements(s)
End if
```

Mit der Struktur **If...Else...End if** kann Ihre Methode zwischen zwei Aktionen wählen, je nachdem, ob die Abfrage (ein Boolean Ausdruck) TRUE oder FALSE ist.

Ist der Boolean Ausdruck TRUE, wird die unmittelbar darauffolgende Befehlsfolge ausgeführt. Ist der Boolean Ausdruck FALSE, wird die Befehlsfolge ausgeführt, die nach *Else* kommt. Die zweite Folge, beginnend mit *Else*, ist nicht unbedingt notwendig. Eine Abfrage kann eine Bedingung auch nur ausführen, wenn sie wahr ist.

Beachten Sie, dass der Boolean Ausdruck immer voll gewertet wird. Siehe hierzu folgenden Test:

```
If(MethodeA & MethodeB)
    ...
End if
```

Der Ausdruck ist nur WAHR, wenn beide Methoden WAHR sind. Aber auch wenn ***MethodeA*** FALSCH zurückgibt, bewertet 4D weiter ***MethodeB***, was unnötig Zeit verbrauchen kann. In diesem Fall ist folgende Struktur besser geeignet:

```
If(MethodeA)
    If(MethodeB)
        ...
    End if
End if
```

Das Ergebnis ist gleich und ***MethodeB*** wird nur bei Bedarf bewertet.

Beispiel

```
  ` Der Benutzer soll seinen Namen eingeben
$Find:=Request(Tippe einen Namen)
If(OK=1)
    QUERY([People];[People]LastName=$Find)
Else
    ALERT("Sie haben keinen Namen eingegeben.")
End if
```

Tipp: Sie können auch Schleifen ohne Befehlsfolge einrichten. Wenn Sie einen Algorithmus oder eine spezifische Anwendung entwickeln, können Sie auch schreiben:

```
If(Boolean_Expression)
Else
    statement(s)
End if
```

oder:

```
If(Boolean_Expression)
    statements(s)
Else
End if
```

Case of...Else...End case

Die Syntax für die Abfragefolge **Case of...Else...End case** lautet:

```
Case of
   :(Boolean_Expression)
      statement(s)
   :(Boolean_Expression)
      statement(s)
      .
      .
      .

   :(Boolean_Expression)
      statement(s)
   Else
      statement(s)
End case
```

Die zweite Folge, beginnend mit **Else** ist optional; Sie können auch nur schreiben:

```
Case of
   :(Boolean_Expression)
      statement(s)
   :(Boolean_Expression)
      statement(s)
      .
      .
      .

   :(Boolean_Expression)
      statement(s)
End case
```

Mit der Struktur **Case of...Else...End case** kann Ihre Methode wie bei der Struktur **If...Else...End if** zwischen alternativen Aktionen wählen. Im Gegensatz zur Struktur **If...Else...End if** prüft die Struktur **Case of...Else...End case** eine unbegrenzte Anzahl von Boolean Ausdrücken und führt bei einer wahren Bedingung eine Aktion aus.

Vor jedem Boolean Ausdruck steht ein Doppelpunkt (*:*). Die Kombination aus Doppelpunkt und Boolean Ausdruck kennzeichnet eine Bedingung. Beispiel:

```
:(bValidate=1)
```

Nur die Anweisungsfolge wird ausgeführt, die bei der Abfrage als erstes mit der Bedingung übereinstimmt. Trifft keine der Bedingungen zu, wird keine Anweisungsfolge ausgeführt.

Sie können nach der letzten Befehlsfolge eine *Else* Anweisung integrieren. Sind dann alle Bedingungen falsch, wird die Anweisungsfolge nach *Else* ausgeführt.

Beispiel

Dieses Beispiel prüft eine numerische Variable und zeigt eine Warnung mit einem Wort an:

```
Case of
   :(vResult=1)  ` Prüfe, ob die Nummer gleich 1 ist
      ALERT("Eins.")  ` Wenn ja, zeige eine Warnung
   :(vResult=2)  ` Prüfe, ob die Nummer gleich 2 ist
      ALERT("Zwei.")  ` Wenn ja, zeige eine Warnung
   :(vResult=3)  ` Prüfe, ob die Nummer gleich 3 ist
      ALERT("Drei.")  ` Wenn ja, zeige eine Warnung
   Else  ` Ist es weder 1, 2 oder 3, zeige eine Warnung
      ALERT("Es war weder eins, zwei oder drei.")
End case
```

Zum Vergleich folgt hier dieselbe Methode mit der Abfrage **If...Else...End if**:

```
If(vResult=1)  ` Prüfe, ob die Nummer gleich 1 ist
   ALERT("Eins.")  ` Wenn ja, zeige eine Warnung
```

```
Else
   If(vResult=2)  ` Prüfe, ob die Nummer gleich 2 ist
      ALERT("Zwei.")  ` Wenn ja, zeige eine Warnung
   Else
      If(vResult=3)  ` Prüfe, ob die Nummer gleich 3 ist
         ALERT("Drei.")  `Wenn ja, zeige eine Warnung
      Else  ` Ist es weder 1, 2 oder 3, zeige eine Warnung
         ALERT("Es war weder eins, zwei oder drei.")
      End if
   End if
End if
```

Beachten Sie, dass bei der Struktur **Case of...Else...End case** nur die erste Bedingung, die WAHR ist, ausgeführt wird. Selbst wenn zwei oder mehr Bedingungen WAHR sind, werden nur die Befehlsfolgen **nach** der ersten wahren Bedingung ausgeführt. Wollen Sie demnach in derselben Methode einfache und komplexe Bedingungen prüfen, müssen Sie die komplexen vor die einfachen Bedingungen setzen, da diese sonst nie ausgeführt werden könnten. Schreiben Sie z.B.:

```
Case of
   :(vResult=1)
      ... `Instruction(s)
   :((vResult=1) & (vRequest#2))
 `<-- Die nachfolgenden Anweisungen werden NIE ausgeführt
      ... `Instruction(s)<gen9>End case</gen9>
```

werden die der komplexen Bedingung zugeordneten Anweisungen nie ausgeführt. Denn dieser Fall ist nur WAHR, wenn die beiden booleschen Bedingungen wahr sind. Nun ist jedoch die erste Bedingung die vorangestellte einfache Bedingung. Wenn sie WAHR ist, wird die einfache Bedingung ausgeführt und 4D verlässt die Schleife, ohne die komplexe Bedingung zu bewerten. Das geschieht nur, wenn Sie Ihre Methode folgendermaßen strukturieren:

```
Case of
   :(vResult=1) & (vRequest#2)
 `<-- Komplexe Bedingungen müssen immer an erster Stelle stehen
      ... `Instruction(s)
   :(vResult=1)
      ... `Instruction(s)
End case
```

Tipp: Sie können auch Schleifen ohne Befehlsfolge einrichten. Wenn Sie einen Algorithmus oder eine spezifische Anwendung entwickeln, können Sie auch schreiben:

```
Case of
   :(Boolean_Expression)
   :(Boolean_Expression)

      .
      .
   :(Boolean_Expression)
      statement(s)
   Else
      statement(s)
End case
```

oder:

```
Case of
   :(Boolean_Expression)
   :(Boolean_Expression)
      statement(s)
      .
      .

   :(Boolean_Expression)
      statement(s)
   Else
End case
```

oder:

```
Case of
   Else
      statement(s)
End case
```

While...End while

Die Syntax für die Abfragefolge **While...End while** lautet:

```
While(Boolean_Expression)
    statement(s)
End while
```

Die Bedingung wird am Schleifenanfang abgefragt: Ist sie FALSE, wird die anschließende Anweisungsfolge nicht ausgeführt. Ist die Anfangsbedingung TRUE, führt das Programm die Anweisungsschleife **While...End while** aus, bis die Bedingung FALSE ist.

Sie müssen die Eintrittsbedingung unmittelbar vor der Schleife initialisieren, d.h. einen geeigneten Wert festsetzen, so dass der Boolean Ausdruck TRUE ist und die Abfragefolge **While...End while** abläuft.

Sie müssen dem Boolean Ausdruck in der Schleife einen sinnvollen Wert zuweisen, da sonst die Schleife endlos läuft. Nachfolgende Schleife läuft endlos, da *NeverStop* immer TRUE ist:

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

Im Falle einer unkontrolliert ablaufenden Methode können Sie die Schleife über den Schrittmodus stoppen und das Problem beheben. Weitere Informationen dazu finden Sie im Kapitel **Debugging**.

Beispiel

```
CONFIRM("Neuen Datensatz hinzufügen?") ` Will der Benutzer einen Datensatz hinzufügen?
While(OK=1) ` Durchlaufe, solange der Benutzer
    ADD RECORD([aTable]) ` Neuen Datensatz hinzufügen will
End while ` Die Schleife endet immer mit End while
```

In diesem Beispiel wird die Systemvariable OK über den Befehl **CONFIRM** vor Beginn der Schleife gesetzt. Klickt der Benutzer im Dialogfenster Bestätigen auf die Schaltfläche OK, wird die Systemvariable OK auf 1 gesetzt, die Schleife startet. Ansonsten hat die Systemvariable OK den Wert 0, die Schleife wird übersprungen. Sobald die Schleife läuft, sorgt der Befehl **ADD RECORD** dafür, dass sie weiterläuft, da er die Systemvariable OK auf 1 setzt, wenn der Benutzer den Datensatz sichert. Annulliert der Benutzer den letzten Datensatz, d.h. er sichert ihn nicht, wird die Systemvariable OK auf 0 gesetzt, die Schleife stoppt.

Repeat...Until

Die Syntax für die Abfragefolge **Repeat...Until** lautet:

```
Repeat
    statement(s)
Until(Boolean_Expression)
```

Die Schleife **Repeat...Until** arbeitet ähnlich wie die Schleife **While...End while**, mit dem Unterschied, dass die Bedingung am Schleifenende abgefragt wird. Dadurch wird die Anweisungsfolge mindestens einmal ausgeführt.
Außerdem läuft die Schleife **Repeat...Until** weiter bis die Bedingung TRUE ist.

Beispiel

Vergleichen Sie folgendes Beispiel mit dem Beispiel für die Schleife **While...End while**. Beachten Sie, dass Sie keine Eintrittbedingung initialisieren müssen — Sie müssen OK vorher nicht auf 1 setzen.

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

For...End for

Die Syntax für die Abfragefolge **For...End for** lautet:

```
For(Counter_Variable;Start_Expression;End_Expression{;Increment_Expression})
   statement(s)
End for
```

Die Schleife **For...End for** wird über eine Zählervariable gesteuert:

- *Counter_Variable* ist eine numerische Variable vom Typ Zahl, Ganzzahl oder Lange Ganzzahl. Sie wird von der Schleife auf den in *Start_Expression* festgelegten Wert initialisiert.
- Die Methode führt die Anweisungsfolge vom Zählbeginn bis Ende aus und zählt jeweils um einen Schritt weiter.
  Die Zählervariable wird nach jedem Durchlaufen der Schleife um den in *Increment_Expression* festgelegten Wert erhöht. *Increment_Expression* ist optional, ist hier kein Wert eingetragen, wird die Zählervariable standardmäßig um Eins (*1*) erhöht.
- Sobald die Zählervariable *End_Expression* durchläuft, stoppt die Schleife.

Wichtig: Die numerischen Ausdrücke *Start_Expression*, *End_Expression* und *Increment_Expression* werden einmal am Schleifenbeginn festgelegt. Sind diese Ausdrücke Variablen, und ändern Sie eine dieser Variablen **innerhalb** der Schleife, hat das **keine** Auswirkung auf die Schleife.

Tipp: Für besondere Zwecke können Sie den Wert der Zählervariablen *Counter_Variable* **innerhalb** der Schleife ändern, das wirkt sich dann auch auf die Schleife aus.

- Im Normalfall ist *Start_Expression* kleiner als *End_Expression*.
- Sind *Start_Expression* und *End_Expression* gleich, wird die Schleife nur einmal ausgeführt.
- Ist *Start_Expression* größer als *End_Expression*, wird die Schleife nicht ausgeführt, außer *Increment_Expression* hat einen negativen Wert.

Allgemeine Beispiele

1. Folgendes Beispiel führt 100 Durchläufe aus:

```
For(vCounter;1;100)
  ` Führe etwas aus
End for
```

2. Folgendes Beispiel durchläuft alle Elemente der Tabelle *anArray*:

```
For($vlElem;1;Size of array(anArray))
  ` Führe etwas mit dem Element aus
   anArray{$vlElem}:=...
End for
```

3. Folgendes Beispiel durchläuft alle Zeichen des Textes *vtSomeText*:

```
For($vlChar;1;Length(vtSomeText))
  ` Führe etwas mit dem Zeichen aus, wenn es ein TAB ist
   If(Character code(vtSomeText[[$vlChar]])=(Tab))
  ` ...
   End if
End for
```

4. Folgendes Beispiel durchläuft die ausgewählten Datensätze für die Tabelle *[aTable]*:

```
FIRST RECORD([aTable])
For($vlRecord;1;Records in selection([aTable]))
  ` Führe etwas mit dem Datensatz aus
   SEND RECORD([aTable])
  ` ...
  ` Gehe zum nächsten Datensatz
   NEXT RECORD([aTable])
End for
```

Die meisten der Schleifen **For...End for** in Ihrer Datenbank sehen wie in den oben aufgeführten Beispielen aus.

Zähler verringern

Manchmal benötigen Sie in einer Schleife eine absteigende Zählervariable. Dazu muss *Start_Expression* größer als *End_Expression* sein und *Increment_Expression* einen negativen Wert haben. Folgende Beispiele führen dieselben Aktionen wie oben aus, sie zählen jedoch in absteigender Reihenfolge:

5. Folgendes Beispiel führt 100 Durchläufe aus:

```
For(vCounter;100;1;-1)
  ` Führe etwas aus
End for
```

6. Folgendes Beispiel durchläuft alle Elemente der Tabelle *anArray*:

```
For($vlElem;Size of array(anArray);1;-1)
  ` Führe etwas mit dem Element aus
    anArray{$vlElem}:=...
End for
```

7. Folgendes Beispiel durchläuft alle Zeichen des Textes *vtSomeText*:

```
For($vlChar;Length(vtSomeText);1;-1)
  ` Führe etwas mit dem Zeichen aus, wenn es ein TAB ist
    If(Character code(vtSomeText[[$vlChar]])=(Tab)
  ` ...
    End if
End for
```

8. Folgendes Beispiel durchläuft die ausgewählten Datensätze für die Tabelle *[aTable]*:

```
LAST RECORD([aTable])
For($vlRecord;Records in selection([aTable]);1;-1)
  ` Führe etwas mit dem Datensatz aus
    SEND RECORD([aTable])
  ` ...
  ` Gehe zum nächsten Datensatz
    PREVIOUS RECORD([aTable])
End for
```

Die Zählervariable um mehr als Eins erhöhen

Bei Bedarf können Sie *Increment_Expression* (positiv oder negativ) für absolute Werte größer als Eins einsetzen.

9. Folgende Schleife ordnet der Tabelle anArray nur die geraden Elemente zu:

```
For($vlElem;2;((Size of array(anArray);2)
  ` Führe etwas mit dem Element #2,#4...#2n aus
    anArray{$vlElem}:=...
End for
```

Die Zählervariable verändern

In manchen Fällen soll eine Schleife n-Male ausgeführt, jedoch verlassen werden, wenn eine andere Bedingung TRUE wird. Dazu fragen Sie diese Bedingung in der Schleife ab. Wenn sie TRUE ist, setzen Sie die Zählervariable explizit auf einen Wert, der höher ist als der Endausdruck.

10. Folgendes Beispiel durchläuft eine Datensatzauswahl bis zum Ende oder bis die Interprozessvariable *<>vbWeStop*, die anfangs auf FALSE gesetzt ist, TRUE wird. Sie unterbrechen die Operation über die Projektmethode **ON EVENT CALL**:

```
<>vbWeStop:=False
ON EVENT CALL("HANDLE STOP")
  ` HANDLE STOP setzt <>vbWeStop auf TRUE, wenn Sie unter Windows ctrl-Punkt,
  ` Auf Macintosh Befehl-Punkt drücken
$vlNbRecords:=Records in selection([aTable])
FIRST RECORD([aTable])
For($vlRecord;1;$vlNbRecords)
  ` Führe etwas mit dem Datensatz aus
    SEND RECORD([aTable])
  ` ...
  ` Gehe zum nächsten Datensatz
    If(<>vbWeStop)
        $vlRecord:=$vlNbRecords+1
  ` Veranlasse die Zählervariable, die Schleife zu verlassen
    Else
        NEXT RECORD([aTable])
```

```
    End if
End for
ON EVENT CALL("")
If(<>vbWeStop)
    ALERT("Die Operation wurde unterbrochen.")
Else
    ALERT("Die Operation wurde erfolgreich abgeschlossen.")
End if
```

Schleifen vergleichen

Gehen wir zurück zum ersten Beispiel in **For...End for**:
Folgendes Beispiel führt 100 Durchläufe aus:

```
For(vCounter;1;100)
  ` Führe etwas aus
End for
```

Vergleichen Sie dieselbe Aktion, einmal mit der Schleife **While...End while**, einmal mit der Schleife **Repeat...Until**.
Ausführung mit der Schleife **While...End while**:

```
$i :=1  ` Initialisiere den Zähler
While($i<=100)  ` Durchlaufe 100 Mal
  `Führe etwas aus
    $i :=$i +1  ` Der Zähler muss erhöht werden
End while
```

Ausführung mit der Schleife **Repeat...Until**:

```
$i :=1  ` Initialisiere den Zähler
Repeat
  ` Führe etwas aus
    $i :=$i +1  ` Der Zähler muss erhöht werden
Until($i=100)  ` Durchlaufe 100 Mal
```

Tipp: Die Schleife **For...End for** ist normalerweise schneller als die Schleifen **While...End while** und **Repeat...Until**, da 4D die Bedingung für jeden Schleifenzyklus intern abfragt und dann den Zähler erhöht. Verwenden Sie deshalb möglichst die Schleife **For...End for**.

Die Ausführung der Schleife For...End for optimieren

Sie können als Zähler sowohl Variablen vom Typ Zahl, Ganzzahl und Lange Ganzzahl als auch Interprozess-, Prozess und lokale Variablen verwenden. Verwenden Sie für lange sich wiederholende Schleifen, besonders im kompilierten Modus, lokale Variablen vom Typ Lange Ganzzahl.
11. Beispiel:

```
C_LONGINT($vlCounter)  ` Verwende lokale Variablen vom Typ lange Ganzzahl
For($vlCounter;1;10000)
  ` Führe etwas aus
End for
```

In Schleifen eingebaute For...End for Abfragen

Sie können in Schleifen beliebig viele Abfragen **For...End for** einbauen. Um Fehler zu vermeiden, verwenden Sie für jede Schleifenstruktur eine andere Zählervariable.
12. Folgendes Beispiel durchläuft alle Elemente einer zweidimensionalen Tabelle:

```
For($vlElem;1;Size of array(anArray))
  ` ...
  ` Führe etwas mit der Spalte aus
  ` ...
    For($vlSubElem;1;Size of array(anArray{$vlElem}))
  ` Führe etwas mit dem Element aus
        anArray{$vlElem}{$vlSubElem}:=...
    End for
End for
```

13. Folgendes Beispiel erstellt eine Zeigertabelle mit allen Datenfeldern Ihrer Datenbank:

```
ARRAY POINTER($apDateFields;0)
$vlElem:=0
For($vlTable;1;Get last table number)
   If(Is table number valid($vlTable))
      For($vlField;1;Get last field number($vlTable))
         If(Is field number valid($vlTable;$vlField))
            $vpField:=Field($vlTable;$vlField)
            If(Type($vpField->)=ls date)
               $vlElem:=$vlElem+1
               INSERT IN ARRAY($apDateFields;$vlElem)
               $apDateFields{$vlElem}:=$vpField
            End if
         End if
      End for
   End if
End for
```

For each...End for each

Die Syntax einer Schleife mit **For each...End for each** sieht folgendermaßen aus:

```
For each(Current_Item;Expression{;begin{;end}}){UntilWhile}(Boolean_Expression)}
   Anweisung(en)
End for each
```

Die Struktur **For each...End for each** führt die Anweisung(en) für jeden aktuellen Eintrag (*Current_item*) von *Expression* aus. Der Typ des aktuellen Eintrags richtet sich nach der Art von *Expression*. Die Schleife **For each...End for each** kann drei Arten durchlaufen:

- **Collections**: Schleife für jedes Element der Collection
- **Entity-Selections**: Schleife für jede Entity
- **Objekte**: Schleife für jede Objekteigenschaft

Folgende Tabelle vergleicht drei Typen von **For each...End for each**:

	Schleife durch Collections	**Schleife durch Entity-Selections**	**Schleife durch Objekte**
Typ *Current_Item*	Variable vom gleichen Typ wie Collection Elemente	Entity	Textvariable
Typ *Expression*	Collection (mit Elementen vom gleichenTyp)	Entity-Selection	Objekt
Anzahl Schleifen (standardmäßig)	Anzahl der Collection Elemente	Anzahl Entities in der Selection	Anzahl der Objekteigenschaften
Unterstützung von Parameter begin / end	Ja	Ja	Nein

- Die Anzahl Schleifen wird beim Starten berechnet und ändert sich nicht während der Durchführung. Einträge während der Schleife hinzufügen oder entfernen wird in der Regel nicht empfohlen, da dies zu fehlenden oder überflüssigen Wiederholungen führen kann.
- Standardmäßig werden die angegebenen *Anweisungen* für jeden Wert in *Expression* ausgeführt. Es ist jedoch möglich, die Schleife durch Testen einer Bedingung entweder am Anfang (**While**) oder am Ende der Schleife (**Until**) zu verlassen.
- Über die optionalen Parameter *begin* und *end* lassen sich in Collections und Entity-Selections Grenzen für die Schleife definieren.

Schleife durch Collections

Beim Verwenden von **For each...End for each** mit einer *Expression* vom Typ *Collection* ist der Parameter *Current_Item* eine Variable vom gleichen Typ wie die Collection Elemente. Die Anzahl Schleifen basiert standardmäßig auf der Anzahl Einträge in der Collection.

Die Collection darf nur Elemente vom gleichen Typ enthalten, sonst wird ein Fehler zurückgegeben, sobald die Variable *Current_Item* dem ersten Wert mit einem unpassenden Typ zugewiesen wird.

Bei jeder Wiederholung der Schleife wird die Variable *Current_Item* automatisch mit dem passenden Element der Collection gefüllt. Dabei müssen Sie folgende Punkte berücksichtigen:

- Ist die Variable *Current_Item* vom Typ Objekt oder Collection *(z.B.* wenn *Expression* eine Collection von Objekten oder von Collections ist), wird durch Ändern dieser Variablen automatisch das zutreffende Element der Collection geändert (weil Objekte und Collections beide dieselbe Referenz nutzen). Bei einer Variablen mit einem skalaren Typ wird nur die Variable geändert.
- Die Variable *Current_Item* muss vom gleichen Typ wie die Elemente der Collection sein. Ist ein Element davon nicht vom gleichen Typ wie die Variable, wird ein Fehler generiert und die Schleife stoppt.
- Enthält die Collection Elemente mit einem Wert **Null**, wird ein Fehler generiert, wenn der Variablentyp von *Current_Item* **Null** Werte nicht unterstützt, wie z.B. der Typ Lange Ganzzahl.

Beispiel

Für eine Collection mit Zahlen ein paar Statistiken berechnen:

```
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
```

Schleife durch Entity-Selections

Beim Verwenden von **For each...End for each** mit einer *Expression* vom Typ *Entity selection* ist der Parameter *Current_Item* die Entity, die gerade bearbeitet wird.

Die Anzahl Schleifen basiert auf der Anzahl Entities in der Entity-Selection. Bei jeder Wiederholung der Schleife wird der Parameter *Current_Item* automatisch mit der Entity der Entity-Selection gefüllt, die gerade bearbeitet wird.

Hinweis: Enthält die Entity-Selection eine Entity, die zwischenzeitlich durch einen anderen Prozess entfernt wurde, wird diese während der Schleife automatisch übersprungen.

Beachten Sie, dass jede Änderung in der aktuellen Entity explizit mit **entity.save( )** gesichert werden muss.

Beispiel

Das Gehalt aller britischen Angestellten in einer Entity-Selection erhöhen:

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

Schleife durch Objekteigenschaften

Beim Verwenden von **For each...End for each** mit einer *Expression* vom Typ *Objekt* ist der Parameter *Current_Item* eine Textvariable, die automatisch mit dem Namen der gerade bearbeiteten Eigenschaft gefüllt wird.

Die Eigenschaften des Objekts werden in der Reihenfolge ihrer Erstellung bearbeitet. Während der Schleife lassen sich Eigenschaften im Objekt hinzufügen oder daraus entfernen. Das verändert nicht die Anzahl Schleifen, diese basiert weiterhin auf der ursprünglichen Anzahl Eigenschaften für das Objekt.

Beispiel

Die Namen in folgendem Objekt auf Großschreibung umstellen:

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

Sie schreiben:

```
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 }
```

Parameter begin / end

Mit den optionalen Parametern *begin* und *end* können Sie Grenzen für die Wiederholung der Schleife definieren.

Hinweis: Die Parameter *begin* und *end* sind nur für Schleifen in Collections und Entity-Selections möglich, in Objekteigenschaften werden sie ignoriert.

- Im Parameter *begin* übergeben Sie die Position des Elements in *Expression*, bei der der Durchlauf startet (inkl. *begin*).
- Im Parameter *end* übergeben Sie die Position des Elements in *Expression*, bei der Durchlauf stoppt (exkl. *end*).

Wird *end* weggelassen oder ist *end* größer als die Anzahl Elemente in *Expression*, werden Elemente ab *begin* bis zum letzten Element einschließlich durchlaufen.

Sind die Parameter *begin* und *end* positive Werte, geben sie die aktuellen Positionen der Elemente in *Expression* an.

Ist *begin* ein negativer Wert, wird er als *begin:=begin+Expression Größe* berechnet (=Versatz vom Ende der *Expression*). Ist der berechnete Wert negativ, wird *begin* auf 0 gesetzt.
Hinweis: Auch wenn *begin* negativ ist, erfolgt der Durchlauf in der standardmäßigen Reihenfolge.
Ist *end* ein negativer Wert, wird er berechnet als *end:=end+Expression Größe*

Hierzu ein Beispiel:

- Eine Collection enthält 10 Elemente (von 0 bis 9)
- begin=-4 -> begin=-4+10=6 -> der Durchlauf startet mit dem 6. Element (#5)
- end=-2 -> end=-2+10=8 -> der Durchlauf stoppt vor dem 8. Element (#7), z.B. beim 7. Element.

Beispiel

```
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"]
```

Bedingungen Until und While

Sie können die Ausführung von **For each...End for each** durch Einfügen einer Bedingung **Until** oder **While** in der Schleife steuern. Ist eine Anweisung **Until**(*condition*) oder **While**(*condition*) in der Schleife vorhanden, stoppt der Durchlauf, sobald *condition* mit **wahr** gewertet wird.

Sie können je nach Bedarf das eine oder andere Schlüsselwort verwenden:

- Die Bedingung **Until** wird am Ende jedes Durchlaufs getestet, d.h. wenn *Expression* nicht leer oder null ist, wird die Schleife mindestens einmal ausgeführt.
- Die Bedingung **While** wird am Anfang jedes Durchlaufs getestet, d.h. je nach Ergebnis der Bedingung wird die Schleife u.U. gar nicht ausgeführt.

Beispiel

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

$total:=0
For each($num;$colNum)While($total<30)  //getestet am Anfang
   $total:=$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)  //getestet am Ende
   $total:=$total+$num
End for each
ALERT(String($total))  //$total = 1001 (1000+1)
```

Shared Objects und Shared Collections

Die Schleife **For each...End for each** lässt sich auch in *shared objects* und *shared collections* verwenden. Weitere Informationen dazu finden Sie im Abschnitt **Shared Objects oder Collections verwenden**.

Muss Ihr Code ein oder mehrere Elemente der Collection oder Objekteigenschaften ändern, müssen Sie die Schlüsselwörter **Use...End use** verwenden. Sie können diese je nach Bedarf wie folgt aufrufen:

- Vor Eintreten in die Schleife, wenn Einträge zur Wahrung der Integrität zusammen geändert werden sollen oder
- Innerhalb der Schleife, wenn nur ein paar Elemente/Eigenschaften geändert werden müssen und das Verwalten der Integrität nicht erforderlich ist.

Use...End use

Die Syntax der Struktur **Use...End use** lautet:

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

Die Struktur **Use...End use** definiert eine Folge von Anweisungen, die unter dem Schutz einer internen Semaphore Aufgaben für den Parameter *Shared_object_or_Shared_collection* ausführen. *Shared_object_or_Shared_collection* kann jedes gültige *shared object* bzw. *shared collection* sein.

Shared objects und *shared collections* wurden zur Kommunikation zwischen Prozessen eingerichtet, insbesondere **Preemptive 4D Prozesse**. Sie lassen sich per Referenz als Parameter von einem Prozess zu einem anderen übergeben. Weitere Informationen dazu finden Sie auf der Seite **Shared Objects und Shared Collections**. Änderungen in *shared objects/collections* müssen zwingend in die Struktur **Use...End use** eingebettet werden, um konkurrierenden Zugriff zwischen Prozessen zu verhindern.

- Bei erfolgreicher Ausführung der Zeile **Use** werden alle *Shared_object_or_Shared_collection* Eigenschaften/Elemente für alle anderen Prozesse im Schreibmodus gesperrt, bis die dazugehörige Zeile **End use** ausgeführt ist.
- Der dazwischenliegende Teil *statement(s)* kann Änderungen in den Eigenschaften/Elementen von *Shared_object_or_Shared_collection* ohne das Risiko konkurrierender Zugriffe ausführen.
- Werden ein anderes Objekt oder eine Collection als Eigenschaft des Parameters *Shared_object_or_Shared_collection* hinzugefügt, werden sie mit derselben gemeinsam genutzten Gruppe verbunden (siehe **Shared Objects oder Collections verwenden**).
- Versucht ein anderer Prozess, auf eine der Eigenschaften bzw. der verbundenen Eigenschaften von *Shared_object_or_Shared_collection* zuzugreifen, während eine Sequenz **Use...End use** ausgeführt wird, wird er automatisch angehalten und wartet, bis die aktuelle Sequenz abgeschlossen ist.
- Die Zeile **End use** entsperrt die Eigenschaften *Shared_object_or_Shared_collection* und alle Objekte, die sich den gleichen Sperrschlüssel (*locking identifier*) teilen.
- Im 4D Code können auch mehrere Strukturen **Use...End use** eingebunden sein. In diesem Fall werden alle Sperrungen gestapelt und Eigenschaften/Elemente erst nach Ausführen der letzten Zeile **End use** freigegeben.

Hinweis: Verändert eine *Member* Funktion einer Collection (siehe **Member Funktionen**) eine *shared collection*, wird für die *shared collection* automatisch ein internes **Use** aufgerufen, während die Funktion ausgeführt wird.

Methoden

Damit Befehle, Operatoren sowie andere Teile der Programmiersprache arbeiten, setzen Sie diese in Methoden. Es gibt verschiedene Arten: Objektmethoden, Formularmethoden, Tabellenmethoden (Trigger), Projektmethoden und Datenbankmethoden. Dieser Abschnitt beschreibt die Features, die für alle Methodenarten gelten.

Eine Methode besteht aus **Anweisungen**, jede Anweisung ist eine Zeile in der Methode. Eine Anweisung führt eine Aktion aus, die einfach oder komplex sein kann. Obwohl eine Anweisung immer in einer Zeile steht, kann diese Zeile so lang wie erforderlich sein (bis zu 32.000 Zeichen, was für die meisten Fälle ausreichen dürfte).

Folgende Zeile ist eine Anweisung, die der Tabelle [People] einen neuen Datensatz hinzufügt:

```
ADD RECORD([People])
```

Eine Methode enthält auch **Abfragen** und **Schleifen** die den Ablauf der Ausführung steuern. Weitere Informationen dazu finden Sie im Abschnitt **Ablaufsteuerung**.

Hinweis: Eine Methode kann bis zu 2 GB groß sein oder 32.000 Code-Zeilen enthalten. Bei Überschreiten dieser Grenzen erscheint eine Meldung, dass die zusätzlichen Zeilen nicht angezeigt werden.

Methodenarten

In 4D gibt es fünf Methodenarten:

- **Objektmethoden**: Eine Objektmethode ist normalerweise eine kurze Methode, die einem aktiven Formularobjekt zugeordnet ist. Sie wird automatisch ausgeführt, wenn der Anwender das Objekt ändert, beispielsweise mit einem Klick auf eine Schaltfläche. Objektmethoden in definierten Eingabe- und Ausgabeformularen können auch ausgeführt werden beim Öffnen von Formularen, beim Druck, beim Import und Export von Daten.
- **Formularmethoden**: Eine Formularmethode ist einem Formular zugeordnet. Sie wird automatisch ausgeführt, wenn das Formular als Eingabe- oder Ausgabeformular genutzt wird. Eine Formularmethode ist dann sinnvoll, wenn sie für mehrere Formularobjekte gilt, beispielsweise das Errechnen der Gesamtsumme aus verschiedenen Datenfeldern oder wenn sich das Objekt nicht in diesem Formular befindet.

Weitere Informationen zu Objekt- und Formularmethoden finden Sie im Handbuch *4D Designmodus* und im Kapitel **Formularereignisse**.

- **Tabellenmethoden (Trigger)**: Ein Trigger ist eine Eigenschaft der Tabelle, Sie rufen Trigger nicht auf. Sie werden vielmehr automatisch von der 4D Datenbank-Engine aufgerufen, jedes Mal wenn Sie die Datensätze einer Tabelle bearbeiten. (Hinzufügen, Löschen, Ändern und Laden). Trigger sind Methoden, die "illegale" Operationen mit Datensätzen in Ihrer Datenbank verhindern. Ein Trigger sorgt zum Beispiel in einem Rechnungssystem dafür, dass der Benutzer eine Rechnung nur hinzufügen kann, wenn er auch den Kunden einträgt, an den die Rechnung gestellt wird. Trigger sind ein leistungsstarkes Tool: Sie beschränken Operationen in einer Tabelle und verhindern, dass Daten versehentlich verloren gehen oder beschädigt werden. Sie können ganz einfache Trigger schreiben, und diese dann nach und nach erweitern.

Weitere Informationen dazu finden Sie im Abschnitt **Einführung in Trigger**.

- **Projektmethoden:** Im Gegensatz zu Objektmethoden, Formularmethoden und Trigger, die einem bestimmten Objekt, Formular bzw. einer Tabelle zugeordnet sind, gelten Projektmethoden für die gesamte Datenbank. Sie sind wiederverwendbar und stehen für den Einsatz in jeder anderen Methode zur Verfügung. Soll ein Task wiederholt werden, müssen Sie nicht für jeden Fall die gleiche Methode schreiben. Sie können Projektmethoden immer da aufrufen, wo sie gebraucht werden—aus anderen Projektmethoden oder aus Objekt- bzw. Formularmethoden. Rufen Sie eine Projektmethode auf, läuft sie ab, als ob Sie die Methode an der aufgerufenen Stelle geschrieben hätten. Projektmethoden, die von anderen Methoden aufgerufen werden, werden oft auch "Unterroutinen" genannt. Eine Projektmethode, die ein Ergebnis zurückgibt, kann auch **Funktion** heißen.

Sie können Projektmethoden auch Menübefehlen zuordnen. Die Methode wird dann ausgeführt, wenn das Menü aufgerufen wird. Der Menübefehl ruft sozusagen die Projektmethode auf.

Weitere Informationen dazu finden Sie im Abschnitt **Projektmethoden**.

- **Datenbankmethoden:** Analog zu Objekt- und Formularmethoden, die aufgerufen werden, wenn Ereignisse im Formular auftreten, gibt es Methoden, die der Datenbank zugeordnet sind, die aufgerufen werden, wenn ein Sitzungsereignis eintritt. Sie wollen zum Beispiel bei jedem Öffnen der Datenbank einige Variablen initialisieren, die während der ganzen Sitzung verwendet werden. Dafür setzen Sie die **Datenbankmethode On Startup** ein. 4D ruft diese Methode beim Öffnen der Datenbank automatisch auf.

Weitere Informationen dazu finden Sie im Kapitel **Datenbankmethoden**.

Beispiel für eine Projektmethode

Alle Methoden funktionieren nach demselben Prinzip—sie starten mit der ersten Zeile und arbeiten jede Anweisung bis zur letzten Zeile ab. Hier ein Beispiel:

```
QUERY([People])  ` Zeige den Sucheditor
If(OK=1)  ` Der Benutzer hat auf OK, nicht auf Abbrechen geklickt
   If(Records in selection([People])=0)  ` Wurde kein Datensatz gefunden…
```

```
        ADD RECORD([People]) ` Soll der Benutzer einen neuen Datensatz hinzufügen
    End if
End if ` Ende
```

Jede Zeile des Beispiels ist eine **Anweisung** oder eine Programmierzeile. Alles, was Sie mit Hilfe der Programmiersprache schreiben, wird im allgemeinen als Code bezeichnet. Code wird ausgeführt oder läuft ab; d.h. 4D führt das durch Code festgelegte Task aus.

Wir analysieren die erste Zeile ausführlich und gehen dann rascher voran:

```
QUERY([People]) ` Zeige den Sucheditor
```

Das erste Element **QUERY** ist ein Befehl. Ein Befehl ist Teil der 4D Programmiersprache — er führt ein Task aus. In diesem Fall zeigt **QUERY** den Sucheditor an. Dasselbe geschieht, wenn Sie in der Designumgebung im Menü **Datensätze** den Menübefehl **Suchen** aufrufen.

Das zweite Element in runden Klammern ist ein Argument für den Befehl **QUERY**. Der Befehl benötigt ein Argument bzw **Parameter**, um ein Task auszuführen. In diesem Fall ist *[People]* der Name einer Tabelle. Tabellennamen stehen immer in eckigen Klammern (*[…]*). In unserem Beispiel ist die Tabelle People ein Argument für den Befehl **QUERY**. Ein Befehl kann auch mehrere Parameter haben.

Das dritte Element ist ein **Kommentar** am Ende der Zeile. Ein Kommentar erläutert Ihnen bzw. jedem anderen, der Ihren Code liest, was im Code passiert. Er ist durch Apostroph (`) gekennzeichnet. Alles, was nach dem Kommentarzeichen steht, wird bei der Code Ausführung ignoriert. Sie können einen Kommentar in eine eigene Zeile oder, wie im Beispiel neben die Programmierzeile setzen. Verwenden Sie ausführliche Kommentare, denn das erleichert das Lesen und Verstehen von Code für Sie und andere.

Hinweis: Ein Kommentar kann bis zu 32 000 Zeichen lang sein.

Die nächste Zeile prüft, ob Datensätze gefunden wurden:

```
If(Records in selection([People])=0) ` Wurde kein Datensatz gefunden···
```

Die If-Anweisung ist eine **Anweisung zur Ablaufsteuerung**—sie steuert die Ausführung Ihrer Methode Schritt für Schritt. Die If-Anweisung fragt eine Bedingung ab. Trifft sie zu, d.h. ist sie wahr, werden die nachfolgenden Zeilen ausgeführt. **Records in selection** ist eine Funktion, d.h. ein Befehl, der einen Wert zurückgibt. Hier gibt **Records in selection** die Anzahl der Datensätze in der aktuellen Auswahl für die Tabelle zurück, die als Argument übergeben wurde.

Hinweis: Beachten Sie, dass nur der Anfangsbuchstabe der Funktion großgeschrieben ist. Das ist die Namenskonvention von 4D für Funktionen.

Sicher kennen Sie bereits eine aktuelle Auswahl—Es ist eine Gruppe Datensätze, an der Sie gleichzeitig arbeiten. Ist die Anzahl der Datensätze gleich 0 (mit anderen Worten, wurde kein Datensatz gefunden), wird die folgende Zeile ausgeführt:

```
ADD RECORD([People]) ` Soll der Benutzer einen neuen Datensatz hinzufügen
```

Der Befehl **ADD RECORD** zeigt ein Formular an, so dass der Benutzer einen neuen Datensatz eingeben kann. 4D formatiert Ihren Code automatisch; beachten Sie, dass diese Zeile eingerückt ist. Damit wird angezeigt, dass sie zur If-Anweisung gehört.

```
End if ` Ende
```

Die Anweisung *End if* schließt die If-Anweisung. Für jede Anweisung benötigen Sie eine Anweisung, die der Programmiersprache angibt, wo die Steuerung aufhört.

Sie sollten mit den hier erläuterten Begriffen gut vertraut sein, bevor Sie weitergehen. Andernfalls lesen Sie den Abschnitt lieber noch ein zweites Mal.

Wie geht es weiter?

Weitere Informationen zu:

- Objektmethoden und Formularmethoden finden Sie im Kapitel **Formularereignisse** und im Handbuch *4D Designmodus*
- Triggern finden Sie im Abschnitt **Trigger**
- Projektmethoden finden Sie im Abschnitt **Projektmethoden**
- Datenbankmethoden finden Sie im Kapitel **Datenbankmethoden**

Projektmethoden

Projektmethoden werden je nach Situation benannt. Sie sind im Gegensatz zu Formular- und Objektmethoden überall verfügbar. Sie sind nicht an spezifische Objekte in der Datenbank gebunden. Je nach Ausführung und Verwendung kann eine Projektmethode folgende Rolle haben:

- Menümethode
- Unterroutine und Funktion
- Prozessmethode
- Ereignisbezogene Methode
- Fehlerbezogene Methode

Projektmethoden sind an keine vordefinierte Situation gebunden. Es bleibt Ihnen als Programmierer überlassen, die notwendigen Elemente zu bestimmen:

Eine **Menümethode** wird in einem eigenen Menü aufgerufen. Sie steuert den Ablauf Ihrer Anwendung und nimmt bei Bedarf eine bestimmte Richtung. Die Menümethode zeigt Formulare an, erstellt Berichte und verwaltet ganz allgemein Ihre Datenbank.

Eine **Unterroutine** ist eine Projektmethode, die als Diener fungiert. Sie führt die Aufgaben aus, die ihr von anderen Methoden aufgetragen werden. Eine Funktion ist eine Unterroutine, die der Methode, die sie aufruft, einen Wert zurückgibt.

Eine **Prozessmethode** wird aufgerufen, wenn ein Prozess startet. Der Prozess dauert nur solange, wie die Prozessmethode ausgeführt wird. Weitere Informationen zu Prozessen finden Sie im Kapitel **Prozesse**. Eine Menümethode, die einem Menübefehl mit der Eigenschaft **Starte neuen Prozeß** zugeordnet ist, ist gleichzeitig die Prozessmethode für den neu gestarteten Prozess.

Eine **ereignisbezogene Methode** läuft in einem eigenen Prozess, der nach Ereignissen sucht. Normalerweise verwaltet 4D die meisten Ereignisse automatisch für Sie. Beispielsweise bei der Dateneingabe nimmt 4D Tastaturkürzel und Klicks wahr, ruft die entsprechenden Objekt- und Formularmethoden auf, so dass Sie von diesen Methoden aus auf die Ereignisse entsprechend antworten können. Unter bestimmten Voraussetzungen möchten Sie jedoch Ereignisse direkt verwalten. Sie möchten bei länger andauernden Operationen (wie die Schleife **For...End for** zum Durchlaufen von Datensätzen) z.B. die Möglichkeit haben, die Operation zu unterbrechen, und zwar unter Windows mit der Tastenkombination **Strg + Punkt**, auf Macintosh **Befehl + Punkt**. Dafür verwenden Sie eine ereignisbezogene Methode. Weitere Informationen dazu finden Sie in der Beschreibung des Befehls **ON EVENT CALL**.

Eine **fehlerbezogene Methode** ist eine unterbrechende Projektmethode. Immer wenn ein Fehler oder eine Ausnahme auftreten, läuft diese Methode in dem Prozess ab, in welchem sie installiert ist. Weitere Informationen dazu finden Sie in der Beschreibung des Befehls **ON ERR CALL**.

Menümethoden

Eine Menümethode wird in der Anwendungsumgebung aufgerufen, wenn Sie den dazugehörigen Menübefehl auswählen. Sie weisen die Methode dem Menübefehl im Methodeneditor zu. Dies ist einer der Hauptaspekte bei der eigenen Gestaltung von Datenbanken. Durch Einrichten eigener Menüs mit dazugehörigen Menümethoden, die bestimmte Aktionen ausführen, personalisieren Sie Ihre Datenbank. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Mit eigenen Menübefehlen können eine oder mehrere Aktivitäten ausgelöst werden. Ein Menübefehl für die Eingabe von Datensätzen kann beispielsweise zwei Tasks ausführen: Das entsprechende Eingabeformular anzeigen und den Befehl **ADD RECORD** aufrufen, bis der Benutzer die Eingabe von Daten beendet.

Das automatische Ablaufen von mehreren Aktivitäten ist eine Leistungsstärke der Programmiersprache. Über eigene Menüs automatisieren Sie mehrere Tasks, die sonst manuell in der Anwendungsumgebung durchgeführt werden müssten. Benutzer der Datenbank werden so besser geführt, sie müssen sich keine Gedanken über die nächste Aktion machen.

Unterroutinen

Eine Projektmethode wird Teil der Programmiersprache für die Datenbank, in der Sie diese erstellt haben. Sie können die Projektmethode dann wie einen der 4D Befehle aufrufen. Eine so verwendete Projektmethode heißt Unterroutine.

Sie verwenden Unterroutinen, um:

- Sich wiederholendes Programmieren zu reduzieren
- Ihre Methoden klarer zu gliedern
- Ihre Methoden schnellerzu ändern
- Ihren Code modular aufzuteilen.

Wir gehen beispielsweise von einer Datenbank Kunden aus. Bei der individuellen Gestaltung stellen Sie fest, dass sich einige Vorgänge wiederholen, wie einen Kunden finden oder einen Datensatz ändern. Der Code dafür könnte folgendermaßen aussehen:

```
  ` Suche nach Kunde
QUERY BY EXAMPLE([Customers])
  ` Wähle das Eingabeformular
FORM SET INPUT([Customers];"Data Entry")
  ` Ändere den Kundendatensatz
MODIFY RECORD([Customers])
```

Arbeiten Sie ohne Unterroutinen, müssen Sie den Code immer schreiben, wenn Sie einen Kundendatensatz ändern wollen. Passiert das in Ihrer Datenbank an zehn Stellen, müssen Sie den Code zehnmal schreiben. Mit Unterroutinen schreiben Sie den Code nur einmal, sie können dadurch Programmierzeilen einsparen.

Wir legen den oben beschriebenen Code in einer Methode mit Namen *MODIFY CUSTOMER* an. Soll diese Methode in einer anderen Methode ausgeführt werden, müssen Sie nur den Namen einsetzen. Wollen Sie beispielsweise einen Kundendatensatz ändern und dann den Datensatz drucken, schreiben Sie folgende Methode:

```
MODIFY CUSTOMER
PRINT SELECTION([Customers])
```

Ihre Methode vereinfacht sich entscheidend. Sie müssen nicht wissen, wie die Methode *MODIFY CUSTOMER* arbeitet, sondern nur was sie tut. Dies ist ein Vorteil von Unterroutinen: Ihre Methoden werden klarer. Sie erweitern sozusagen die 4D Programmiersprache.

Wollen Sie in dieser Beispieldatenbank die Methode zum Auffinden von Kunden ändern, müssen Sie nur eine und nicht zehn Methoden ändern. Ein weiterer Vorteil von Unterroutinen: Methoden sind schnell geändert.

Mit Unterroutinen bauen Sie Ihren Code modular auf. Das bedeutet, sie unterteilen Ihren Code in Module (Unterroutinen), die jeweil ein logisches Task ausführen. Betrachten Sie folgenden Code aus einer Datenbank für Kontoführung:

```
FIND CLEARED CHECKS  ` Finde alle bezahlten Schecks
RECONCILE ACCOUNT  ` Gleiche Konto aus
PRINT CHECK BOOK REPORT  ` Drucke Scheckbuch Bericht
```

Auch für jemanden, der die Datenbank nicht kennt, ist klar, was der Code ausführt. Er muss nicht jede Unterroutine prüfen, die evtl. aus vielen Zeilen besteht und komplexe Operationen ausführt. Wichtig ist, dass die Tasks ausgeführt werden.

Wir empfehlen, den Code, immer wenn möglich, in logische Tasks oder Module aufzuteilen.

Parameter in Methoden

Bei größeren Programmen müssen Sie in Ihren Methoden oft auch Daten übergeben. Dazu setzen Sie Parameter ein.

Eine Methode benötigt **Parameter** (oder Argumente) zum Ausführen der Tasks. Dieses Handbuch verwendet beide Begriffe. Parameter werden in den 4D Befehlen übergeben. In diesem Beispiel ist die Zeichenkette "Hello" ein Argument des Befehls **ALERT**:

```
ALERT("Hello")
```

Auf dieselbe Weise werden auch Parameter in Methoden übergeben. Hier lässt die Methode *DO SOMETHING* drei Parameter zu:

```
DO SOMETHING(WithThis;AndThat;ThisWay)
```

Parameter sind durch Strichpunkt voneinander getrennt (*;*).

In der Unterroutine (die aufgerufene Methode) wird der Wert jedes Parameters automatisch in durchnummerierte lokale Variablen kopiert: *$1*, *$2*, *$3*, etc.. Diese geben die Reihenfolge der Parameter an.

```
 //Code der Methode DO SOMETHING
 //Alle Parameter sind vom Typ Text
C_TEXT($1;$2;$3)
ALERT("I received "+$1+" and "+$2+" and also "+$3)
 //$1 enthält den Parameter WithThis
 //$2 enthält den Parameter AndThat
 //$3 enthält den Parameter ThisWay
```

Sie können die Parameter *$1*, *$2...* in der Unterroutine genauso wie jede andere lokale Variable verwenden. Jedoch lassen sich bei Befehlen, welche den Wert der als Parameter übergebenen Variablen verändern, z.B. **Find in field**, die Parameter *$1*, *$2*, etc. nicht direkt verwenden. Sie müssen sie zuerst in lokale Standardvariablen kopieren (z.B. *$myvar.=$1*).

Weiterführender Hinweis: 4D Projektmethoden akzeptieren eine variable Anzahl von Parametern gleichen Typs, beginnend von rechts. Zum Deklarieren dieser Parameter verwenden Sie eine Compiler Direktive, in der Sie {${N} als Parameter übergeben, wobei N den ersten Parameter angibt. Zum Beispiel gibt die Deklaration **C_LONGINT**(${5}) 4D und dem Compiler an, dass die Methode ab dem fünften Parameter eine variable Anzahl Parameter vom Typ Lange Ganzzahl empfangen kann. Mit der Funktion **Count parameters** haben Sie über eine *For* Schleife und die Parameter Indirektion Syntax Zugriff auf diese Parameter. Weitere Informationen dazu finden Sie im 2. Beispiel unter **Count parameters**.

Parameter werden je nach Typ **als Kopie** oder **als Referenz** übergeben:

- Bei Übergabe **als Kopie** sind die lokalen Variablen/Parameter nicht die aktuellen Felder, Variablen oder Ausdrücke, übergeben von der **aufrufenden Methode**; sie enthalten nur die übergebenen Werte. Da ihre Reichweite lokal ist, wird ein in der Unterroutine geänderte Wert in der aufrufenden Methode nicht verändert.
- Bei Übergabe **als Referenz** enthalten die lokalen Variablen/Parameter Referenzen auf die aktuellen Ausgangsfelder, -variablen oder -ausdrücke, übergeben von der **aufrufenden Methode**;Wird nun der Wert des lokalen Parameters geändert, ändert das auch den Ausgangswert.

Nachfolgende Tabelle zeigt, wie sich die verschiedenen Elementtypen übergeben lassen:

Parametertyp	Übergeben als	Kommentar
Feld, Variable oder Ausdruck eines skalaren Typs (Zahl, Text, Datum...)	Wert	Lässt sich durch einen Zeiger als Referenz übergeben, siehe unten
Feld, Variable oder Ausdruck vom Typ Objekt	Referenz	Siehe Beispiel unten
Variable oder Ausdruck vom Typ Collection	Referenz	
Variable oder Ausdruck vom Typ Zeiger	Referenz	Siehe **Zeiger in Methoden**
Array	Lässt sich nicht direkt als Parameter übergeben	Lässt sich durch einen Zeiger als Referenz übergeben, siehe **Arrays und Zeiger**
Tabelle	Lässt sich nicht direkt als Parameter übergeben	Lässt sich durch einen Zeiger als Referenz übergeben, siehe **Zeiger**

Parameter als Wert übergeben

Bei Feldern, Variablen und Ausdrücken vom skalaren Typ werden als Parameter für Methoden nur Kopien der Werte übergeben.

Da $1, $2 lokale Variablen sind, sind sie nur innerhalb der Unterroutine verfügbar und werden am Ende der Unterroutine gelöscht. Von daher kann eine Unterroutine nicht den Wert der aktuellen Datenfelder oder Variablen ändern, die auf Methodenebene als Parameter übergeben werden. Beispiel:

```
  ` Ausschnitt aus dem Code der Methode MY METHOD
  ` ...
DO SOMETHING([People]Last Name)  ` [People]Last Name sei gleich "weber"
ALERT([People]Last Name)

  ` Code der Methode DO SOMETHING
$1:=Uppercase($1)
ALERT($1)
```

Die von ***DO SOMETHING*** angezeigte Meldung liest "WEBER", die von ***MY METHOD*** angezeigte Meldung liest "weber". Die Methode hat den Wert des Parameters *$1* lokal geändert, das verändert jedoch nicht den Wert des Datenfeldes *[People]Last Name*, der von der Methode ***MY METHOD*** als Parameter übergeben wurde.

Es gibt zwei Möglichkeiten, den Wert des Datenfeldes in der Methode ***DO SOMETHING*** zu ändern:

1. Sie übergeben in der Methode nicht das Datenfeld, sondern einen Zeiger darauf:

```
  ` Ausschnitt aus dem Code der Methode MY METHOD
  ` ...
DO SOMETHING(->[People]Last Name)  ` [People]Last Name sei gleich "weber"
ALERT([People]Last Name)

  ` Code der Methode DO SOMETHING
$1->:=Uppercase($1->)
ALERT($1->)
```

Der Parameter ist hier nicht das Datenfeld selbst, sondern ein Zeiger darauf. Das von *$1* **referenzierte** Objekt (*$1->*) ist das aktuelle Datenfeld. Wird nun das referenzierte Objekt geändert, geht das über die Reichweite der Unterroutine hinaus und wirkt sich auf das aktuelle Datenfeld aus. In diesem Beispiel lesen beide Meldedialoge "WEBER".

Weitere Informationen dazu finden Sie im Abschnitt **WA GET EXTERNAL LINKS FILTERS**.

2. Sie können die Methode ***DO SOMETHING*** so umschreiben, dass sie nicht etwas ausführt, sondern einen Wert zurückgibt:

```
  ` Ausschnitt aus dem Code der Methode MY METHOD
  ` ...
[People]Last Name:=DO SOMETHING([People]Last Name)
  ` [People]Last Name sei gleich "weber"
ALERT([People]Last Name)

  ` Code der Methode DO SOMETHING
$0:=Uppercase($1)
ALERT($0)
```

Diese Technik heißt auch "eine Funktion verwenden". Sie wird im nächsten Abschnitt beschrieben.

Parameter als Referenz übergeben

Bei Verwendung von Variablen, Ausdrücken oder Feldern vom Typ Objekt oder Collection als Parameter von Methoden werden Referenzen auf aktuelle Ausgangswerte übergeben. In diesem Fall enthält *$1, $2...* keine Werte sondern Referenzen. Wird der Wert von *$1, $2...* in der Unterroutine geändert, wirkt sich die Änderung überall aus, wo das Quellobjekt oder die Collection verwendet werden. Das ist das gleiche Prinzip wie für Zeiger, mit dem Unterschied, dass die Parameter *$1, $2...* in der Unterroutine nicht dereferenziert werden müssen.

Zum Beispiel:

```
  //Die Methode CreatePerson erstellt ein Objekt und sendet es als Parameter
C_OBJECT($person)
$person:=New object("Name";"Smith";"Age";40)
ChangeAge($person)
ALERT(String(OB get($person;"Age")))
```

```
//Die Methode ChangeAge fügt 10 zum Attribut Age des empfangenen Objekts hinzu
C_OBJECT($1)
OB SET($1;"Age";OB Get($1;"Age")+10)
ALERT(String(OB get($1;"Age")))
```

Bei Ausführen der Methode ***CreatePerson*** lesen beide Meldungsboxen "50", da beide Methoden dieselbe Referenz verwalten.

4D Server: Werden Parameter zwischen Methoden übergeben, die nicht auf demselben Rechner ausgeführt werden (z.B. die Option **Execute on Server**, siehe **Eigenschaften für Projektmethoden**), lassen sich Referenzen nicht verwenden. In diesen Fällen werden anstelle von Referenzen Kopien von Objekt und Collection Parametern gesendet.

Funktionen: Projektmethoden können einen Wert zurückgeben

Daten können auch von Methoden zurückgeben werden. Eine Methode, die einen Wert zurückgibt, heißt **Funktion**.

4D oder 4D Plug-in Befehle, die einen Wert zurückgeben, heißen ebenfalls **Funktionen**.

Folgende Programmierzeile verwendet die vordefinierte Funktion **Length**, sie gibt die Länge einer Zeichenkette zurück. Der von **Length** zurückgegebene Wert wird in eine Variable mit Namen *MyLength* gesetzt:

```
MyLength:=Length("How did I get here?")
```

Jede Unterroutine kann einen Wert zurückgeben. Der zurückzugebende Wert wird in die lokale Variable *$0* gesetzt.

Folgende Funktion mit Namen *Uppercase4* gibt eine Zeichenkette mit den ersten vier Zeichen in Großbuchstaben zurück:

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

Dieses Beispiel verwendet die Funktion *Uppercase4*:

```
NewPhrase:=Uppercase4("This is good.")
```

Die Variable *NewPhrase* erhält “THIS is good.”

Das **Ergebnis der Funktion** *$0* ist eine lokale Variable in der Unterroutine. Sie kann als solche in der Unterroutine verwendet werden. Im vorigen Beispiel ***DO SOMETHING*** wurde *$0* zuerst der Wert von *$1* zugewiesen, dann war sie Parameter für den Befehl **ALERT**. Sie können *$0* in der Unterroutine genauso wie jede andere lokale Variable verwenden. 4D übergibt der aufgerufenen Methode den Wert von *$0* (so als ob die Unterroutine endet).

Rekursive Projektmethoden

Projektmethoden können sich auch selbst aufrufen, d.h. sie sind rekursiv. Beispiel:

- Die Methode A ruft die Methode B auf, die A aufruft, so ruft A wieder B auf, usw..
- Eine Methode kann sich selbst aufrufen.

Die 4D Programmiersprache unterstützt **Rekursivität**.

Hier ein Beispiel. Wir haben eine Tabelle *[Friends and Relatives],* die extrem vereinfacht, so aussieht:
- *[Friends and Relatives]Name*
- *[Friends and Relatives]Children'Name*

Wir gehen davon aus, dass die Werte der Datenfelder einmalig sind, d.h. es gibt nicht zwei Personen mit demselben Namen. Sie wollen für einen gegebenen Namen folgenden Satz erstellen: “Hans, mein Freund, der das Kind ist von Paul, der das Kind ist von Susi, die das Kind ist von Robert, der das Kind ist von Elisabeth, tut dies für sein Leben gern!”:

1. Sie können den Satz folgendermaßen anlegen:

```
$vsName:=Request("Gib den Namen ein:";"Hans")
If(OK=1)
   QUERY([Friends and Relatives];[Friends and Relatives]Name=$vsName)
   If(Records in selection([Friends and Relatives])>0)
      $vtTheWholeStory:="Mein Freund, "+$vsName
      Repeat
         QUERY([Friends and Relatives];[Friends and Relatives]Children'Name=$vsName)
         $vlQueryResult:=Records in selection([Friends and Relatives])
         If($vlQueryResult>0)
            $vtTheWholeStory:=$vtTheWholeStory+" der das Kind ist von
            "+[Friends and Relatives]Name
            $vsName:=[Friends and Relatives]Name
         End if
      Until($vlQueryResult=0)
      $vtTheWholeStory:=$vtTheWholeStory+", tut dies für sein Leben gern!"
      ALERT($vtTheWholeStory)
   End if
End if
```

2. oder folgendermaßen:

```
$vsName:=Request("Gib den Namen ein:";"Hans")
If(OK=1)
   QUERY([Friends and Relatives];[Friends and Relatives]Name=$vsName)
   If(Records in selection([Friends and Relatives])>0)
      ALERT("Mein Freund, "+Genealogy of($vsName)+", tut dies für sein Leben gern!")
   End if
End if
```

mit der rekursiven Methode ***Genealogy of***:

```
` Projektmethode Genealogy of
` Genealogy of ( String ) -> Text
` Genealogy of ( Name ) -> Satzteil

$0:=$1
QUERY([Friends and Relatives];[Friends and Relatives]Children'Name=$1)
If(Records in selection([Friends and Relatives])>0)
   $0:=$0+" der das Kind ist von "+Genealogy of([Friends and Relatives]Name)
End if
```

Beachten Sie die Methode ***Genealogy of***, die sich selbst aufruft.

Die erste Möglichkeit ist ein **iterativer Algorithmus**, die zweite ein **rekursiver Algorithmus**.

Hinweis: Sie können bei Programmierungen wie im oben aufgeführten Beispiel sowohl iterative als auch rekursive Methoden schreiben. Rekursivität macht die Programmierung im allgemeinen präziser, leichter zu lesen und zu warten, sie ist jedoch nicht zwingend.

Einige typische Verwendungen für Rekursivität in 4D sind:

- Datensätze in Tabellen bearbeiten, die wie im obigen Beispiel miteinander verknüpft sind.
- Dokumente und Ordner auf Ihrer Festplatte mit den Befehlen **FOLDER LIST** und **DOCUMENT LIST** durchlaufen. Ein Ordner kann Ordner und Dokumente enthalten, die Unterordner selbst können Ordner und Dokumente enthalten, usw..

Wichtig: Rekursive Abfragen sollten immer an einem bestimmten Punkt enden. Im Beispiel ruft sich die Methode ***Genealogy of*** nicht mehr selbst auf, wenn die Suche keinen Datensatz zurückgibt. Ohne Abfragen dieser Bedingung würde sich die Methode endlos aufrufen; 4D gibt dann evtl. eine Fehlermeldung "Speicher voll" zurück, da es keinen Platze mehr hat zum Stapeln der Aufrufe (so wie bei Parametern und lokalen Variablen in den Methoden).

Debugging

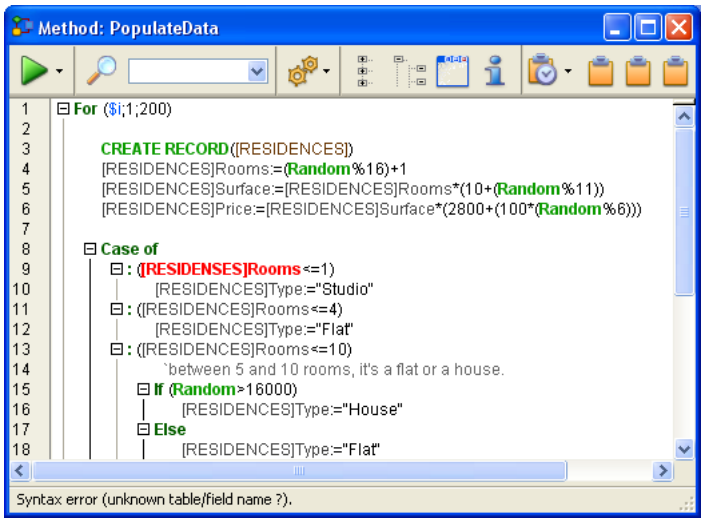
Wozu dient ein Debugger?

Beim Entwickeln und Testen Ihrer Methoden ist es wichtig, dass Sie evtl. darin enthaltene Fehler finden und beheben können.

Beim Programmieren gibt es verschiedene Fehlerarten: Tippfehler, Syntax- oder Ablauffehler, Strukturfehler oder logische Fehler und Runtime Fehler.

Tippfehler

Tippfehler findet der **Methodeneditor**.
Sie erscheinen in Rot. Im Bereich unter dem Methodenfenster erscheint eine Meldung. Nachfolgendes Fenster zeigt einen markierten Tippfehler:



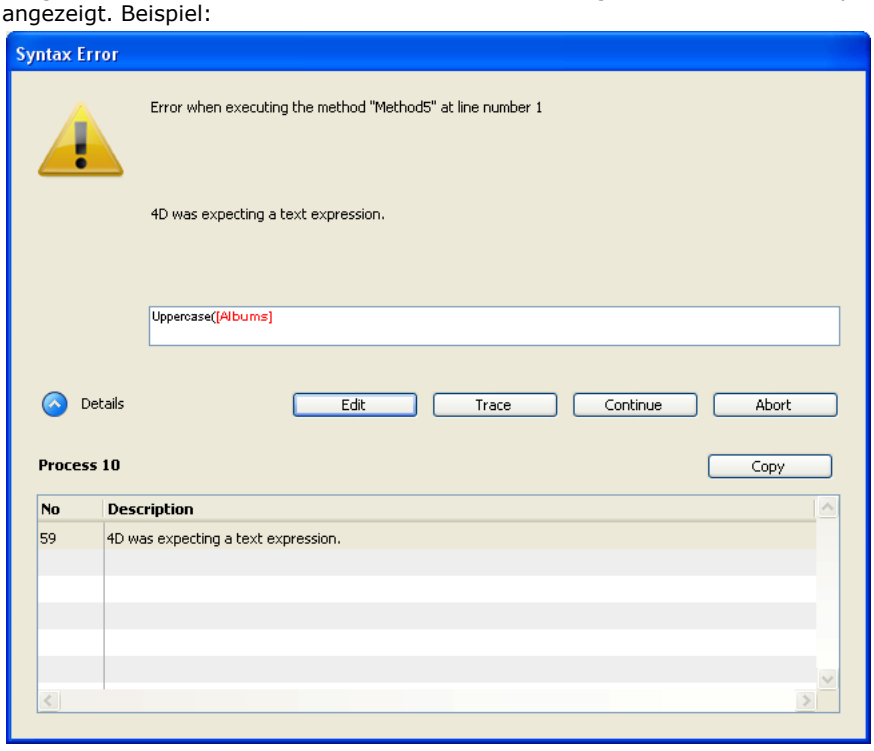
Hinweis: Die Kommentare wurden nachträglich per Hand eingefügt. 4D ändert nur die Farbe an der Stelle, wo der Fehler auftritt.

Solche Tippfehler rufen normalerweise Syntaxfehler hervor. Hier ist der Tabellenname unbekannt. Im Informationsbereich wird der Fehler beschrieben, wenn Sie die Programmierzeile bestätigen.

In diesem Fall korrigieren Sie den Tippfehler und bestätigen die Korrektur mit der Eingabetaste auf dem Zahlenblock. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

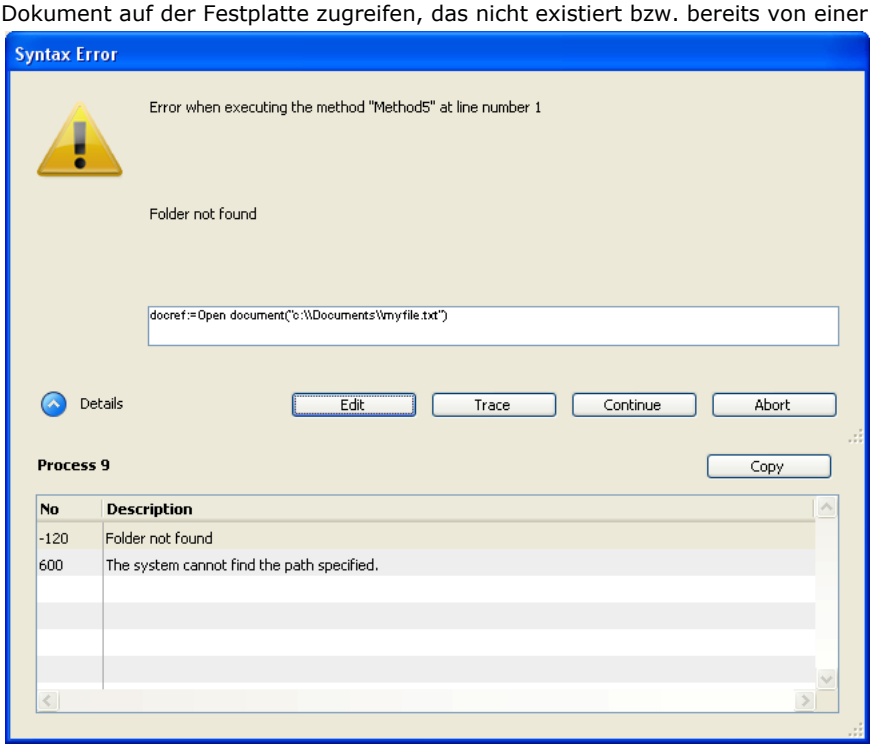
Syntax oder Ablauffehler

Einige Fehler werden nur beim Ausführen der Methode gefunden. Bei einem Syntaxfehler wird das **Fenster Syntaxfehler** angezeigt. Beispiel:



In diesem Fenster wurde für den Befehl **Uppercase** ein Tabellenname übergeben, der Befehl erwartet jedoch einen Ausdruck vom Typ Text. Der Bereich "Details" ist erweitert, um den letzten Fehler und seine Nummer anzuzeigen.

Es kann beispielsweise nicht genügend Speicher zum Erstellen eines Array oder BLOB vorhanden sein. Oder Sie wollen auf ein Dokument auf der Festplatte zugreifen, das nicht existiert bzw. bereits von einer anderen Anwendung geöffnet ist.



Diese Fehler hängen nicht direkt mit Ihrer Programmierung zusammen; sie treten auf, weil manchmal irgendetwas nicht stimmt. In den meisten Fällen lassen sich solche Fehler mit einer Fehlersuchmethode beheben, die den Befehl **ON ERR CALL** einsetzt.

Strukturfehler oder logische Fehler

Solche Fehler sind im allgemeinen schwer zu finden—verwenden Sie dafür den **Debugger**. In gewissem Maß fallen die oben beschriebenen Fehlertypen - mit Ausnahme der Tippfehler - auch unter die Rubrik Strukturfehler oder logische Fehler. Zum Beispiel:

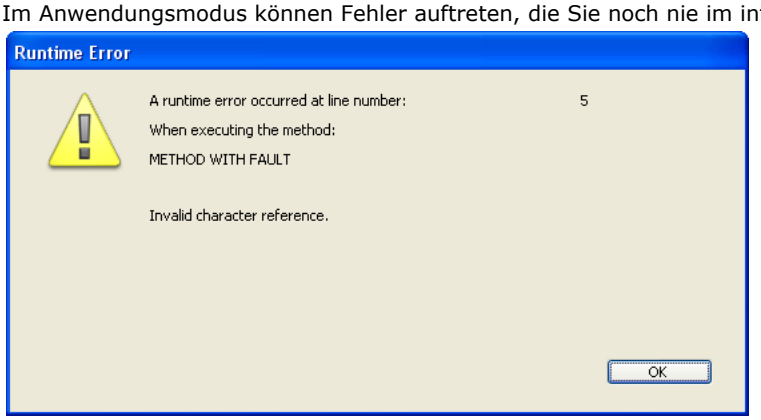
- Ein Syntaxfehler tritt ein, wenn Sie eine noch nicht initialisierte Variable verwenden wollen.
- Ein Ablauffehler tritt ein, wenn Sie ein Dokument öffnen wollen, dessen Name von einer Unterroutine empfangen wird, die im Parameter nicht den richtigen Wert erhält. Beachten Sie, dass sich in diesem Beispiel der abbrechende Codeteil unterscheiden kann von dem Code, der das Problem hervorruft.

Unter Strukturfehler oder logische Fehler fallen auch folgende Situationen:

- Ein Datensatz wird nicht korrekt aktualisiert, da Sie vor Aufrufen von **SAVE RECORD** nicht geprüft haben, ob der Datensatz gesperrt ist.
- Eine Methode läuft nicht korrekt ab, da Sie nicht geprüft haben, ob ein optionaler Parameter vorhanden ist.

Runtime Fehler

Im Anwendungsmodus können Fehler auftreten, die Sie noch nie im interpretierten Modus gesehen haben. Beispiel:



Diese Meldung sagt aus: "Sie versuchen auf ein Zeichen zuzugreifen, das außerhalb der definierten Länge für die Zeichenkette liegt." Um die Ursache des Problems schnell zu finden, notieren Sie den Methodennamen und die Zeilennummer, öffnen Sie erneut die interpretierte Version der Strukturdatei und gehen Sie in der Methode in die entsprechende Zeile.

Vorgehensweise bei Fehlern

Fehler sind nichts Außergewöhnliches. Es wäre unnormal, hunderte von Programmierzeilen ohne irgendeinen Fehler zu schreiben. So ist auch die Fehlersuche und -korrektur ein ganz normaler Vorgang.

Durch die Multitasking-Fähigkeiten von 4D können Sie Methoden schnell bearbeiten und durchlaufen, da Sie zwischen den einzelnen Fenstern wechseln können. Sie müssen nicht jedes Mal die ganze Programmierung durchlaufen. Auch der **Debugger** beschleunigt die Fehlerverwaltung.

Ein typischer Anfangsfehler ist, bei der Fehlersuche im Fenster Syntaxfehler auf die Schaltfläche Abbrechen zu klicken, in den Methodeneditor zurückzugehen und den Code zu durchforsten, um herauszufinden, was nicht stimmt. Tun Sie das nicht! Sie sparen viel Zeit und Energie, wenn Sie **immer** mit dem **Debugger** arbeiten.

- Tritt ein unerwarteter Syntaxfehler auf, verwenden Sie den **Debugger**.
- Tritt ein Ablauffehler auf, verwenden Sie den **Debugger**.
- Tritt irgendein anderer Fehlertyp auf, verwenden Sie den **Debugger**.

In 99 % der Fälle liefert der **Debugger** genau die Information, um zu verstehen, warum ein Fehler aufgetreten ist. Mit dieser Information können Sie dann auch den Fehler beheben.

Tipp: Befassen und experimentieren Sie ein paar Stunden mit dem **Debugger**. Denn so können Sie in Zukunft beim Suchen und Korrigieren von Fehlern Tage und Wochen einsparen.

Verwenden Sie den **Debugger** auch zum Entwickeln von Code. Sicher schreiben Sie manchmal einen recht umfangreichen Code. Auch wenn Sie noch so genau sind, können Sie erst bei der Ausführung ganz sicher sein, ob Ihr Code funktioniert. Anstatt ihn blind laufen zu lassen, setzen Sie den Befehl **TRACE** an den Anfang Ihres Codes. Sie können ihn dann Schritt für Schritt ausführen und kontrollieren, was passiert. Ein Purist mag gegen diese Methode sein, aber manchmal zahlt sich Pragmatismus doch aus. Wie auch immer ... verwenden Sie den **Debugger**.

Fazit

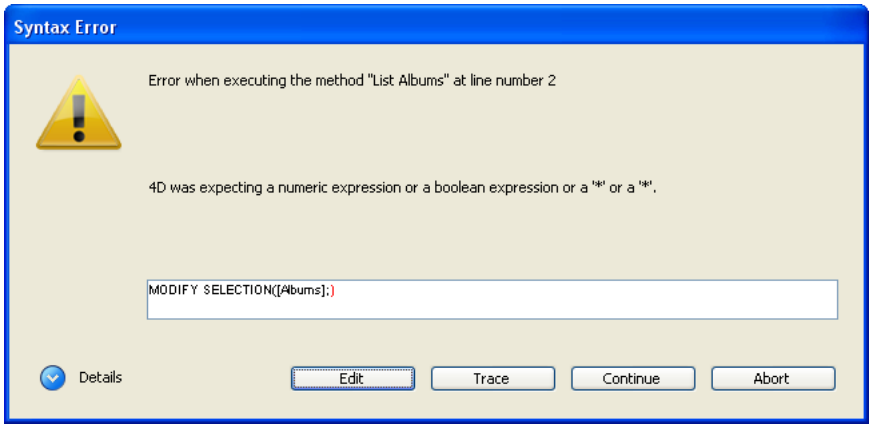
Verwenden Sie den **Debugger**.

Fenster Syntaxfehler

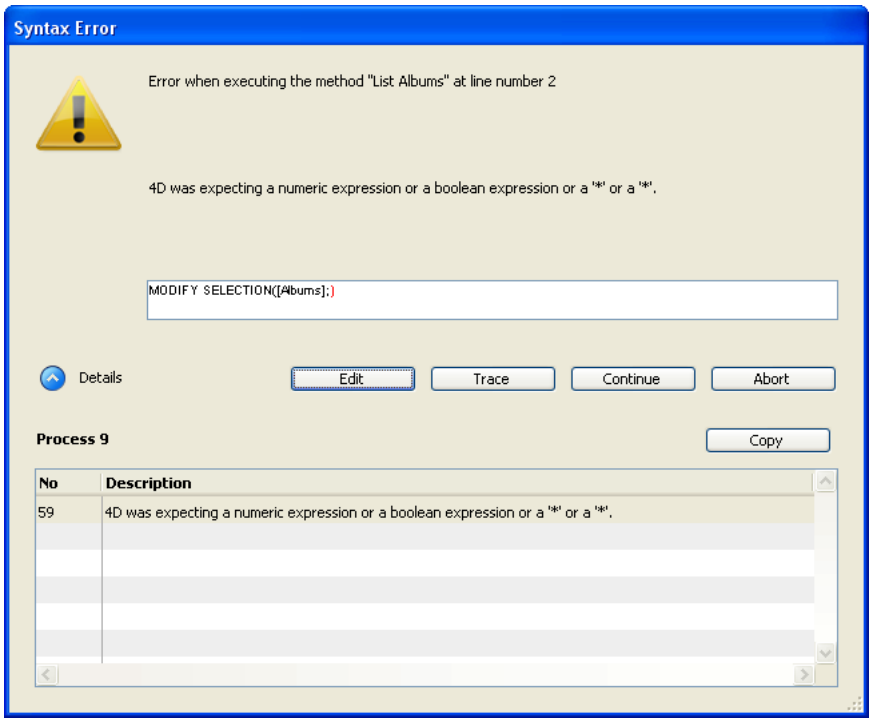
Das Fenster Syntaxfehler erscheint, wenn die Ausführung der Methode gestoppt wird. Dafür gibt es folgende Gründe:

- 4D stoppt die Ausführung, weil ein Fehler die weitere Ausführung verhindert.
- Die Methode erzeugt eine falsche Assertion (siehe Befehl **ASSERT**)

Das Fenster Syntax-Fehler sieht aus wie folgt:



Im oberen Textbereich erscheint eine Meldung, die den Fehler beschreibt. Der untere Textbereich zeigt die Programmierzeile, bei der der Fehler auftritt; der Fehlerbereich ist markiert.
Über die Schaltfläche **Details** können Sie den unteren Bereich aufklappen, der den Fehler genauer beschreibt.



Am unteren Rand des Fensters befinden sich die Schaltflächen **Abbrechen**, **Schritt**, **Weiter**, **Bearbeiten** und bei erweitertem Fenster **Kopieren**.

- **Abbrechen**: Die Methode wird angehalten, sie kehren zur Stelle vor Ausführung der Methode zurück. Wird eine Formular- oder Objektmethode gestoppt, die als Folge eines Ereignisses ausgeführt wird, kehren Sie zum Formular zurück. Läuft die Methode in der Anwendungsumgebung ab, kehren Sie zu dieser Umgebung zurück.
- **Schritt**: Sie geben den Schrittmodus ein, das **_o_During** wird angezeigt. Wurde die aktuelle Zeile nur teilweise ausgeführt, müssen Sie mehrmals auf die Schaltfläche **Schritt** klicken. Sobald die Zeile endet, öffnet sich wieder das **_o_During**.
- **Weiter**: Die Ausführung läuft weiter. Je nachdem, wo der Fehler liegt, ist die fehlerhafte Zeile u.U. nur teilweise ausgeführt. Von daher ist Vorsicht geboten — das kann nämlich dazu führen, dass die restliche Methode nicht ordnungsgemäß ausgeführt wird. Betätigen Sie die Schaltfläche **Weiter** nur bei trivialen Fehlern. Der Befehl **SET WINDOW TITLE** beispielsweise hat keine Auswirkung auf das Ausführen und Testen des restlichen Codes. So können Sie sich auf wichtige Code-Teile konzentrieren und kleinere Fehler später beheben.
  Hinweis: Klicken Sie mit gedrückter **Alt-Taste** unter Windows, bzw. **Wahltaste** auf Mac OS auf die Schaltfläche **Weiter**, wechselt er in **Ignorieren**. D.h., das Fenster erscheint nicht, wenn der gleiche Fehler, ausgelöst von der gleichen Methode in in der gleichen Zeile erneut auftritt. Dieses Tastenkürzel ist hilfreich, wenn ein Fehler wiederholt auftritt, z.B. in einer Schleife. In diesem Fall läuft alles so ab, als ob der Benutzer jedes Mal auf die Schaltfläche **Weiter** klicken würde.
- **Bearbeiten**: Die gesamte Ausführung der Methode wird angehalten. 4D wechselt in die Designumgebung. Die fehlerhafte Methode wird im Methodeneditor geöffnet, wo Sie den Fehler korrigieren können. Verwenden Sie diese Schaltfläche, wenn

Sie den Fehler sofort erkennen und ohne weitere Überprüfung beheben können.

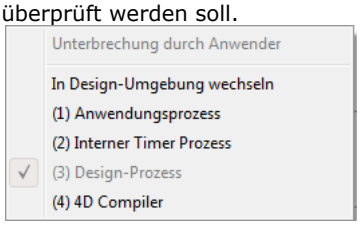
- **Kopieren**: Die Debugging Information wird in die Zwischenablage kopiert. Sie beschreibt die interne Umgebung des Fehlers, z.B. Fehlernummer, interne Komponente. Sie wird als Text mit Tabs formatiert. Sie können dann den Inhalt aus der Zwischenablage in eine Textdatei, ein E-Mail, o.ä. setzen und zur Fehleranalyse weiterleiten.

Fenster Debugger

Der Begriff Debugger kommt von Bug. Ein Bug in einer Methode ist ein Fehler, den Sie beheben wollen. Sie verwenden den Debugger, wenn ein Fehler aufgetreten ist oder Sie die Ausführung Ihrer Methoden überwachen müssen. Der Debugger unterstützt Sie bei der Fehlersuche, Sie können damit Ihre Methoden langsam durchlaufen und den Inhalt überprüfen. Dieser Prozess heißt **Schrittmodus**.

Der Debugger zeigt Methoden auf folgende Weise an bzw. durchläuft sie im Schrittmodus:

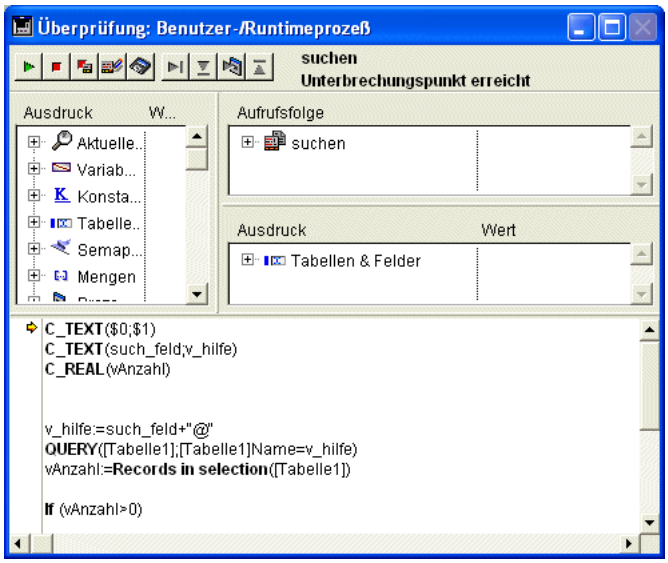
- Sie klicken im **Fenster Syntaxfehler** auf die Schaltfläche **Schritt**
- Sie benutzen den Befehl **TRACE**
- Sie klicken im Fenster Methode ausführen auf die Schaltfläche **Schritt.**
- Sie betätigen während der Methodenausführung die Kombination **Alt+Umschalttaste+Rechte Maustaste**, auf Macintosh die Kombination **Ctrl+Wahl-+Befehlstaste+Mausklick**, und dann im PopUp-Menü den Prozess, der im Schrittmodus überprüft werden soll.



- Sie klicken bei ausgewähltem Prozess im Runtime Explorer auf der Seite Prozess auf die Schaltfläche **Schritt**.
- Sie erstellen oder bearbeiten einen Unterbrechungspunkt im Fenster Methodeneditor oder in den Registerkarten **Unterbrechung** und **Unterbrechungspunkt** des **Runtime Explorer**.

Hinweis: Ist die Datenbank mit einem Kennwort versehen, können nur Designer und Benutzer mit Zugriffsberechtigung auf die Struktur den Schrittmodus auf Methoden anwenden.

Das Fenster Debugger sieht folgendermaßen aus:

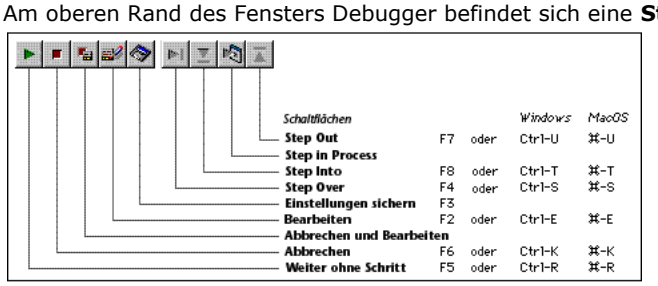


Sie können dieses Fenster bewegen sowie die darin enthaltenen Bereiche bei Bedarf vergrößern. Ein neues Debugger-Fenster wird mit derselben Konfiguration wie das zuletzt geöffnete Fenster derselben Sitzung angezeigt (Größe und Position des Fensters, Anordnung der Trennungslinien und Inhalt des Bereichs, der die Ausdrücke bewertet.)

Da 4D multitasking-fähig ist, können Sie auch mehrere Prozesse voneinander unabhängig im Schrittmodus durchlaufen. Sie können für jeden Prozess ein Fenster Debugger öffnen.

Schaltflächen der Steuerleiste

Am oberen Rand des Fensters Debugger befindet sich eine **Steuerleiste** mit neun Schaltflächen:



Schaltfläche Weiter ohne Schritt

Der Schrittmodus wird angehalten, die Methodenausführung wird fortgesetzt.

Hinweis: **Großschreibtaste+F5** oder **Großschreibtaste+Klick** fasst die Ausführung zusammen. Dadurch werden für den aktuellen Prozess auch alle folgenden Aufrufe von **Schritt** deaktiviert.
ALT+F5 unter Windows, bzw. **Wahl-/Befehlstaste+R** auf Macintosh setzt die Ausführung fort.

Schaltfläche Abbrechen

Die Methode wird angehalten, Sie kehren zur Stelle vor Ausführung der Methode zurück. Wird eine Formular- oder Objektmethode gestoppt, die als Folge eines Ereignisses ausgeführt wird, kehren Sie zum Formular zurück. Läuft die Methode in der Runtime-Umgebung ab, kehren Sie zur Anwendungsumgebung zurück.

Schaltfläche Abbrechen und Bearbeiten

Die Methode wird angehalten, als ob Sie auf Abbrechen geklickt hätten. 4D öffnet den Methodeneditor für die Methode, die bei Anklicken der Schaltfläche **Abbrechen** und **Bearbeiten** ausgeführt wurde.

Tipp: Verwenden Sie diese Schaltfläche, wenn Sie wissen, welche Änderungen in Ihrem Code erforderlich sind und Sie diese Änderungen für das weitere Testen Ihrer Methoden benötigen. Führen Sie die Änderungen aus und starten Sie erneut die Methode.

Schaltfläche Bearbeiten

Klicken Sie auf die Schaltfläche **Bearbeiten**, wird die aktuelle Ausführung im Unterschied zur Schaltfläche **Abbrechen und Bearbeiten** nicht abgebrochen. Die Methodenausführung pausiert an dieser Stelle. 4D öffnet dann den Methodeneditor für die Methode, die bei Anklicken der Schaltfläche **Bearbeiten** ausgeführt wurde.

Wichtig: Sie können diese Methode ändern; die Änderungen erscheinen jedoch nicht unmittelbar in der Methode, die gerade im Debugger-Fenster durchlaufen wird, sondern erst bei der nächsten Ausführung der Methode. D.h. die Änderungen werden erst berücksichtigt, wenn die Methode abgebrochen oder erfolgreich abgeschlossen und erneut geladen wurde.

Tipp: Verwenden Sie diese Schaltfläche, wenn Sie wissen, welche Änderungen in Ihrem Code erforderlich sind und diese nicht mit dem Rest des auszuführenden Code interferieren.

Tipp: Objektmethoden werden für jedes Ereignis erneut geladen. Durchlaufen Sie eine Objektmethode im Schrittmodus (z.B. als Folge eines Klicks auf eine Schaltfläche), müssen Sie das Formular nicht verlassen. Sie können die Objektmethode bearbeiten, die Änderungen sichern und dann zurückwechseln zum Formular und die Methode erneut testen. Wollen Sie Formularmethoden ändern bzw. im Schrittmodus durchlaufen, müssen Sie das Formular verlassen und erneut öffnen, um das Formular wieder zu laden. Bei extensiver Fehlersuche für ein Formular gibt es einen Trick. Setzen Sie den Code, der im Schrittmodus getestet wird, in eine Projektmethode, die Sie als Unterroutine in einer Formularmethode verwenden. Auf diese Weise können Sie im Formular bleiben, wenn Sie es im Schrittmodus durchlaufen, bearbeiten und erneut testen. Denn die Unterroutine wird jedes Mal erneut geladen, wenn sie von einer Formularmethode aufgerufen wird.

Schaltfläche Einstellungen sichern

Sichert die aktuelle Konfiguration des Debugger Fensters (Größe und Position des Fensters, Anordnung der Trennungslinien und Inhalt des Bereichs zur Bewertung der Ausdrücke. Diese Einstellungen werden dann bei jedem Öffnen der Datenbank standardmäßig verwendet. Sie werden in der Strukturdatei der Datenbank gespeichert.

Schaltfläche Step Over

Die aktuelle Methodenzeile wird ausgeführt (Sie ist mit einem gelben Pfeil, dem **Programmzähler**, markiert), der Debugger wechselt in die nächste Zeile. Die Schaltfläche **Step Over** springt nicht in Unterroutinen und Funktionen; sie bleibt auf der Ebene der Methode, die gerade durchlaufen wird. Die Unterroutine wird in einem Stück abgearbeitet, ohne dass der Debugger dies anzeigt.

Schaltfläche Step Into

Beim Ausführen einer Zeile, die eine andere Methode aufruft (Unterroutine oder Funktion), wird diese Methode im Debugger Fenster angezeigt. Sie können nun diese Methode im Schrittmodus durchlaufen. Diese Methode wird im **Bereich Aufruffolge** des Debugger Fensters zur aktuellen Methode. Wird bei Ausführen einer Methode keine andere Methode aufgerufen, funktioniert diese Schaltfläche wie die Schaltfläche **Step Over**.

Schaltfläche Step in Process

Beim Ausführen einer Zeile, die einen neuen Prozess erzeugt (z.B. Aufruf des Befehls **New process**) öffnet diese Schaltfläche ein neues Debugger Fenster. Darin können Sie die Prozessmethode des neu erzeugten Prozesses im Schrittmodus durchlaufen. Wird bei Ausführen einer Methode kein neuer Prozess erzeugt, funktioniert diese Schaltfläche wie die Schaltfläche **Step Over**.

Schaltfläche Step Out

Klicken Sie beim Durchlaufen von Unterroutinen und Funktionen auf diese Schaltfläche, können Sie die gesamte Methode, die gerade im Schrittmodus durchlaufen wird, ausführen und zur aufrufenden Methode zurückkehren. Das Debugger Fenster wird auf die vorherige Methode der Aufrufsfolge zurückgesetzt. Ist die aktuelle Methode die letzte Methode in der Aufrufsfolge, wird das Debugger Fenster geschlossen.

Hinweis: Durchlaufen Sie einen Prozess, der Code ausführt, erscheint sofort das Debugger-Fenster. Durchlaufen Sie dagegen einen Prozess, der läuft, aber keinen Code ausführt (schläft, wartet auf Ereignis...), speichert 4D die Anforderung des Debugger-Fensters. Es erscheint, sobald der Prozess die Code-Ausführung abgeschlossen hat.

Information zur Steuerleiste

Rechts neben der Steuerleiste zeigt der Debugger folgende Information an:

- Den Namen der Methode, die gerade im Schrittmodus durchlaufen wird. Er erscheint in schwarz.
- Das Problem, welches das Erscheinen des Debugger Fensters hervorruft. Es erscheint in rot.

Angewendet auf das oben angezeigte Beispielfenster erscheinen folgende Informationen:

- Die Methode ***DE_DebugDemo*** wird im Schrittmodus durchlaufen.
- Das Debugger Fenster erscheint, weil es einen Aufruf des Befehl **C_DATE** festgestellt hat und dieser Befehl ein Überwachungsbefehl ist.

Nachfolgende Liste zeigt die möglichen Gründe für das Erscheinen des Debuggers und der Meldung. Sie erscheint in rot:

- **Befehl TRACE**: Ein Aufrufen von **TRACE** wurde ausgelöst.
- **Erreichen eines Unterbrechungspunktes**: Ein Unterbrechungspunkt wurde gefunden.
- **Benutzerunterbrechung**: Sie haben die Tastenkombination **ALT+Großschreibtaste+Klick** unter Windows, bzw. **ctrl+Wahltaste+Befehl+Klick** auf Macintosh gedrückt oder in der Designumgebung im Runtime Explorer auf der Seite **Prozess** auf die Schaltfläche **Schritt** geklickt.
- **Aufrufen eines Überwachungsbefehls: Befehlsname**: Ein 4D Überwachungsbefehl wurde aufgerufen.
- **Wechseln in einen neuen Prozess**: Sie haben auf die Schaltfläche **Step Into Process** geklickt. Diese Meldung wird vom für den neu erzeugten Prozess geöffneten Debugger Fenster angezeigt.

Die Bereiche des Debugger Fensters

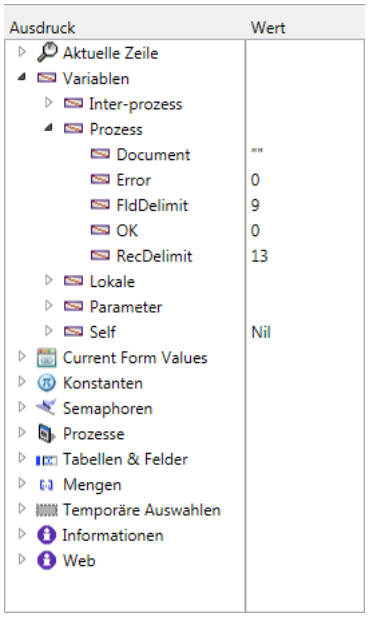
Das Debugger Fenster enthält die bereits beschriebene Steuerfläche sowie vier Bereiche, die Sie in der Größe verändern können:

- **Bereich Überprüfung**
- **Bereich Aufruffolge**
- **Bereich individuelle Überprüfung**
- **Bereich Source Code**

Die drei ersten Bereiche enthalten hierarchische Listen zum Navigieren und Anzeigen beständiger Debugging Informationen. Der vierte **Bereich Source Code** zeigt den Source Code der Methode, die im Schrittmodus durchlaufen wird. Jeder Bereich unterstützt Sie bei der Fehlersuche mit spezifischen Informationen. Mit der Maus können Sie das gesamte Debugger Fenster sowie jeden Bereich einzeln vertikal und horizontal vergrößern. Die drei oberen Bereiche sind in zwei Spalten mit Trennungslinie unterteilt, die Sie mit der Maus beliebig verschieben können.

Bereich Überprüfung

Der Überprüfungsbereich erscheint unterhalb der Steuerleiste auf der linken Seite des Debugger-Fensters. Es könnte zum Beispiel so aussehen:



Der Überprüfungsbereich zeigt nützliche Informationen über das System, die 4D Umgebung und die Ausführungsumgebung.
Die Spalte **Ausdruck** zeigt die Namen von Objekten oder Ausdrücken. Die Spalte **Wert** zeigt den dazugehörigen aktuellen Wert.
Sie können den Wert eines Objekts durch Anklicken in der Spalte Wert verändern, sofern das Objekt änderbar ist.
Die mehrstufigen hierarchischen Listen sind nach Themen gegliedert:

- Aktuelle Zeile
- Variablen
- Current Form Values
- Konstanten
- Semaphoren
- Prozesse
- Tabellen & Felder
- Mengen
- Temporäre Auswahlen
- Information
- Web

Jedes Thema hat eine bzw. mehrere Unterebenen. Das Kästchen links neben dem Themennamen öffnet bzw. schließt die Unterebenen. Hat ein Thema weitere Unterebenen, klicken Sie auf das jeweils links davor vorhandene Kästchen, um die gesamte Information für das Thema anzuzeigen.
Sie können jederzeit Themen, dazugehörige Unterlisten oder Unterthemen per Drag & Drop in den **Bereich individuelle Überprüfung** setzen.

Aktuelle Zeile

Dieses Thema zeigt Werte von Objekten oder Ausdrücken an, die:

- In der auszuführenden Programmierzeile verwendet werden, das ist die Zeile, die mit dem Programmzähler, dem gelben Pfeil im **Bereich Source Code** markiert ist oder
- In der vorigen Programmierzeile verwendet werden.

Das Thema *Aktuelle Zeile* zeigt die Objekte oder Ausdrücke vor und nach der gerade ausgeführten Programmierzeile. Sie führen z.B. folgende Methode aus:

```
TRACE
a:=1
b:=a+1
c:=a+b
` ...
```

1. Sie setzen im Debugger-Fenster den Programmzähler im **Bereich Source Code** auf die Zeile *a:=1*. Unter Aktuelle Zeile erscheint:

*a:***Undefined**
Die Variable *a* wird angezeigt, da sie in der auszuführenden Zeile verwendet wird, jedoch noch nicht initialisiert ist.

2. Sie gehen eine Zeile weiter. Der Programmzähler markiert nun die Zeile *b:=a+1*. Unter Aktuelle Zeile erscheint:

a:1
*b:***Undefined**

Die Variable *a* wird angezeigt, da sie in der soeben ausgeführten Zeile verwendet und der numerische Wert 1 zugewiesen wurde. Sie wird auch angezeigt, da sie in der auszuführenden Zeile verwendet wird und der Variablen *b* zugewiesen wird. Die Variable *b* wird angezeigt, da sie in der auszuführenden Zeile verwendet wird, jedoch noch nicht initialisiert ist.

3. Sie gehen erneut eine Zeile weiter. Der Programmzähler markiert nun die Zeile *c:=a+b*. Unter Aktuelle Zeile erscheint:

*c:***Undefined**
a:1
b:2

Die Variable *c* wird angezeigt, da sie in der auszuführenden Zeile verwendet wird, jedoch noch nicht initialisiert ist. Die Variablen *a* und *b* werden angezeigt, da sie in den vorigen Zeilen und der auszuführenden Zeile verwendet werden, ...

Das Thema Aktuelle Zeile ist äußerst hilfreich—denn Sie müssen nur die hier angezeigten Werte überprüfen und nicht jedes Mal, wenn Sie eine Zeile ausführen, einen Ausdruck in den **Bereich individuelle Überprüfung** eingeben.

Variablen

Dieses Thema enthält folgende Unterthemen:

- **Interprozess**: Zeigt die Liste der gerade verwendeten Interprozessvariablen. Gibt es keine Interprozessvariablen, ist diese Liste leer. Die Werte der Interprozessvariablen können Sie ändern.
- **Prozess**: Zeigt die Liste der im aktuellen Prozess verwendeten Prozessvariablen. Diese Liste ist selten leer. Die Werte der Prozessvariablen können Sie ändern.
- **Lokal**: Zeigt die Liste der lokalen Variablen in der Methode, die im Schrittmodus durchlaufen wird. Diese Methode wird im **Bereich Source Code** angezeigt. Gibt es keine lokalen Variablen oder sind noch keine angelegt, ist diese Liste leer. Die Werte der lokalen Variablen können Sie ändern.
- **Parameter**: Zeigt die Liste der Parameter, die die Methode empfängt, die im Schrittmodus durchlaufen wird. Diese Methode wird im **Bereich Source Code** angezeigt. Wurden dieser Methode keine Parameter übergeben, ist diese Liste leer. Die Werte der Parameter können Sie ändern.
- **Self Pointer**: Zeigt einen Zeiger auf das aktuelle Objekt, wenn Sie eine Objektmethode im Schrittmodus durchlaufen. Diesen Wert können Sie nicht ändern.

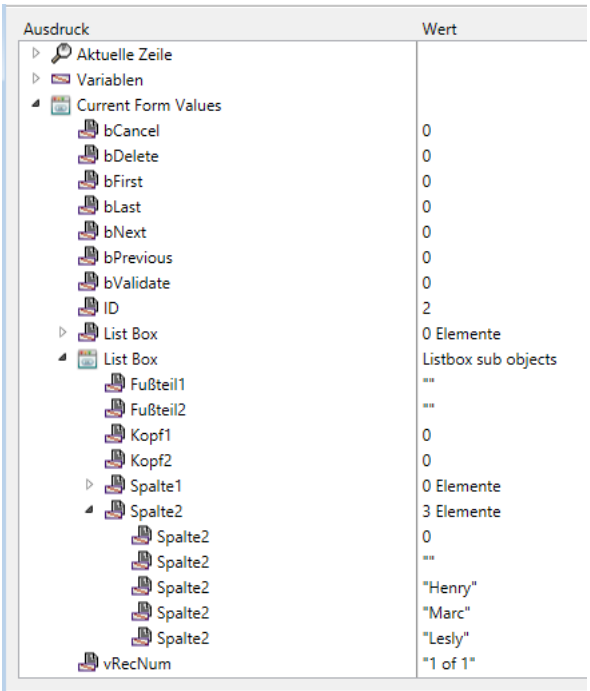
Hinweis: Sie können Variablen vom Typ Alphanumerisch, Text, Numerisch, Datum und Zeit ändern, das sind Variablen, deren Wert Sie über die Tastatur eingeben können.

Arrays sowie andere Variablen erscheinen je nach Geltungsbereich in den Unterthemen Interprozess, Prozess und Lokal. Der Debugger zeigt jedes Array mit einer zusätzlichen hierarchischen Ebene; so können Sie – sofern vorhanden- Werte der Array-Elemente erhalten oder ändern. Der Debugger zeigt die ersten 100 Elemente inkl. dem Element Null. Die Spalte Wert zeigt die Größe des Array in Bezug auf seinen Namen. Haben Sie das Array eínmal eingesetzt, zeigt der erste Unterpunkt die aktuell ausgewählte Elementnummer, dann das Element Null, dann die anderen Elemente (bis zu 100). Sie können Arrays vom Typ Alphanumerisch, Text, Numerisch und Datum ändern. Sie können auch die ausgewählte Elementnummer, das Element Null und die anderen Elemente ändern (bis zu 100), jedoch nicht die Größe des Array.

Zur Erinnerung: Sie können jederzeit per Drag & Drop einen Eintrag aus dem Überprüfungsbereich in den **Bereich individuelle Überprüfung** setzen, inkl. ein individuelles Array-Element.

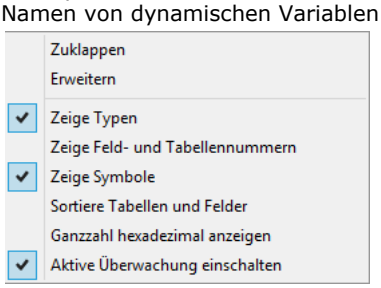
Liste Current Form Values

Im Debugger und Runtime Explorer gibt es die Liste **Current Form Values**. Sie enthält die Namen der dynamischen Objekte im aktuellen Formular, sowie den Wert der zugeordneten Variable:

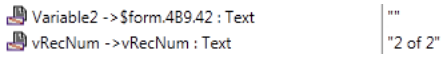


Einige Objekte, wie z.B. Listbox Arrays erscheinen mit zwei unterschiedlichen Einträgen: Die Variable des Objekts selbst und ihre Datenquelle.

Diese Liste ist besonders hilfreich bei Formularen mit mehreren dynamischen Variablen: Über die Formularobjektnamen lassen sich dynamische Variablen leicht identifizieren. Über den Eintrag **Zeige Typen** des Kontextmenüs können Sie den internen Namen von dynamischen Variablen anzeigen:



Dynamische Variablennamen haben die Form "$form.4B9.42":



Konstanten

Zeigt die vordefinierten Konstanten von 4D, ähnlich wie die Seite Konstanten im Fenster Explorer. Hier angezeigte Ausdrücke können Sie nicht ändern.

Tabellen und Datenfelder

Zeigt die Tabellen und Datenfelder der Datenbank; Unterdatenfelder werden nicht angezeigt. Die Spalte Wert zeigt für jede Tabelle die Größe der aktuellen Auswahl für den aktuellen Prozess, bei erweiterter Tabelle auch die Anzahl der gesperrten Datensätze. Die Spalte Wert zeigt für jedes Datenfeld den Wert für den aktuellen Datensatz an, sofern vorhanden. Davon ausgenommen sind Bilder, Untertabellen und BLOBs. Hier angezeigte Datenfeldwerte können Sie ändern, jedoch nicht rückgängig machen. Hier angezeigte Tabelleninformation können Sie dagegen nicht ändern.

Semaphoren

Zeigt die aktuell gesetzten lokalen Semaphoren. Die Spalte Wert zeigt den Namen des Prozesses, der die Semaphore setzt. Gibt es keine Semaphoren, ist diese Liste leer. Hier angezeigte Ausdrücke können Sie nicht ändern. Globale Semaphoren werden nicht angezeigt.

Mengen

Zeigt die im aktuellen Prozess definierten Mengen auf Prozess- und Interprozessebene. Die Spalte Wert zeigt für jede Menge die Anzahl der Datensätze und den Tabellennamen an. Gibt es keine Mengen, ist diese Liste leer. Hier angezeigte Ausdrücke können Sie nicht ändern.

Prozesse

Zeigt die seit Beginn der Arbeitssitzung gestarteten Prozesse. Die Spalte Wert zeigt die benötigte Zeit und den aktuellen Status für jeden Prozeß, z.B. Ausführung, Schläft, etc.. Hier angezeigte Ausdrücke können Sie nicht ändern.

Temporäre Auswahlen

Zeigt die im aktuellen Prozess definierten temporären Auswahlen auf Prozess- und Interprozessebene. Die Spalte Wert zeigt für jede temporäre Auswahl die Anzahl der Datensätze und den Tabellennamen an. Gibt es keine temporären Auswahlen, ist diese Liste leer. Hier angezeigte Ausdrücke können Sie nicht ändern.

Information

Zeigt allgemeine Informationen zu Datenbankoperationen, wie z.B. die aktuelle Standardtabelle (sofern vorhanden), benutzter physikalischer oder virtueller Speicher, Query Destination, etc. Mit Hilfe dieser Informationen können Sie die Funktionsweise der Datenbank überprüfen.

Web

Zeigt Informationen zum Web Server der Anwendung (ist nur bei aktivem Web Server verfügbar):

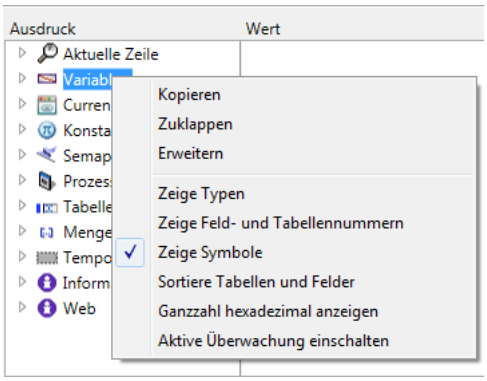
- Zu sendende Web Datei: Name der Web Datei, die auf Senden wartet (sofern vorhanden)
- Web Cache Benutzung: Anzahl der Seiten im Web Cache und ihre prozentuale Verwendung
- Web Server verbrauchte Zeit: Verwendungsdauer des Web Server im Format Stunden:Minuten:Sekunden
- Web Hits Count: Gesamtanzahl der HTTP Anfragen, die seit dem Start des Web Server empfangen wurden, sowie momentane Anzahl der Anfragen pro Sekunde
- Anzahl aktive Web Prozesse: Anzahl der aktiven Web Prozesse und Web Prozesse insgesamt.

Hier enthaltene Ausdrücke können Sie nicht verändern.

Kontextmenü

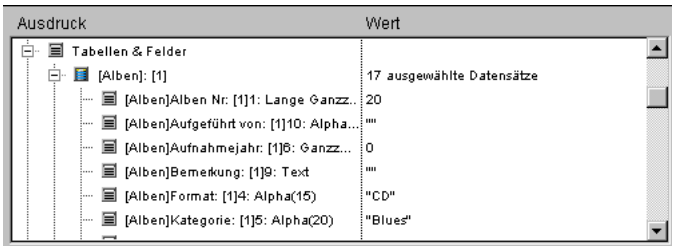
Das Kontextmenü im Bereich Überprüfung bietet zusätzliche Optionen. Um es anzuzeigen:

- Klicken Sie unter Windows mit der **rechten** Maustaste auf eine beliebige Stelle im Bereich Überprüfung.
- Klicken Sie auf Macintosh mit der **ctrl-Taste+Klick** auf eine beliebige Stelle im Bereich Überprüfung. Auf Mac OS 8.0 oder neuer wählen Sie die Tastenkombination **Alt+ctrl+Klick**



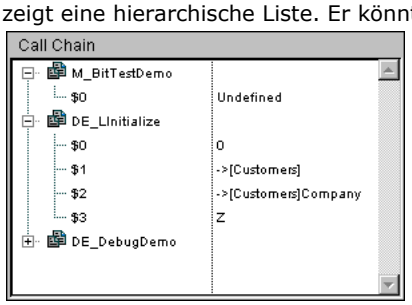
- **Zuklappen**: Blendet alle Ebenen der hierarchischen Liste im Bereich Überprüfung aus.
- **Erweitern**: Blendet alle Ebenen der hierarchischen Liste im Bereich Überprüfung ein.
- **Zeige Typen**: Zeigt für jedes Objekt die entsprechenden Objekttypen.
- **Zeige Feld- und Tabellennummern**: Zeigt die Nummer für jede Tabelle oder jedes Datenfeld von **Felder**. Diese Option ist sehr hilfreich, wenn Sie mit Tabellen oder Datenfeldnummern bzw. mit Zeigern arbeiten, die Funktionen wie **Table** oder **Field** benutzen.
- **Zeige Symbole**: Zeigt den Objekttyp für jedes Objekt als Icon an. Sie können diese Option ausschalten, wenn die Anzeige schneller erfolgen soll oder Sie nur die Option **Zeige Typen** verwenden wollen.
- **Sortiere Tabellen und Felder**: Zeigt die Tabellen und Datenfelder in den jeweiligen Listen in alphabetischer Reihenfolge an.
- **Ganzzahl hexadezimal anzeigen**: Zahlen werden normalerweise in Dezimalschreibweise angezeigt. Diese Option zeigt Zahlen hexadezimal an.
  Hinweis: Um einen Zahlenwert hexadezimal einzugeben, tippen Sie 0x (Null + x) und die hexadezimalen Stellen.
- **Aktive Überwachung einschalten**: Aktiviert die Überwachung der internen Aktivität der Applikation und zeigt die Informationen in den zusätzlichen Bereichen **Web**, **Planer** und **Netzwerk**.

Nachfolgend sehen den Bereich Überprüfung mit allen aufgeklappten Optionen:



Bereich Aufruffolge

Eine Methode kann andere Methoden aufrufen, die wiederum andere Methoden aufrufen. Deshalb ist es hilfreich, während dem Debugging Prozess die **Aufruffolge** der Methoden zu sehen. Der Bereich Aufruffolge liegt im Debugger-Fenster rechts oben. Er zeigt eine hierarchische Liste. Er könnte z.B. so aussehen:



- Die Hauptebene zeigt den Methodennamen. An oberster Stelle steht die Methode, die Sie gerade im Schrittmodus durchlaufen, darauf folgt die aufrufende Methode, also die Methode, die die Methode im Schrittmodus aufgerufen hat, dann folgt die Methode, die die vorige Methode aufgerufen hat, usw. Im oben angezeigten Beispiel wird die Methode ***M_BitTestDemo*** im Schrittmodus durchlaufen, diese wurde von der Methode ***DE_LInitialize*** aufgerufen, die wiederum von der Methode ***DE_DebugDemo*** aufgerufen wurde.

- Durch Doppelklick auf den Methodennamen gelangen Sie zurück zur aufrufenden Methode, gleichzeitig wird ihr Source Code im **Bereich Source Code** angezeigt. Dadurch sehen Sie schnell, wie diese Methode den Aufruf an die aufgerufene Methode tätigt. Auf diesem Weg können Sie jedes Stadium der Aufruffolge überprüfen.

- Durch Anklicken des Kästchens links neben dem Methodennamen können Sie die Parameter ($1, $2...) und die Liste der optionalen Funktionsergebnisse ($0) für die Methode ein- oder ausblenden. Die Werte erscheinen in der rechten Spalte. Sie können jeden beliebigen Parameter bzw. jedes Funktionsergebnis durch Anklicken verändern. Für obiges Beispiel ergibt sich folgendes:
  1. ***M_BitTestDemo*** hat keinen Parameter.
  2. ***M_BitTestDemo***'s *$0* ist derzeit undefiniert, da die Methode *$0* keinen Wert zugewiesen hat. (Die Zuweisung ist entweder noch nicht erfolgt oder die Methode ist eine Unterroutine und keine Funktion.)
  3. ***DE_LInitialize*** hat von *DE_DebugDemo* drei Parameter erhalten. *$1* ist ein Zeiger auf die Tabelle *[Customers*], *$2* ist ein Zeiger auf das Datenfeld *[Customers]Company, $3* ist ein alphanumerischer Parameter mit dem Wert *"Z"*.

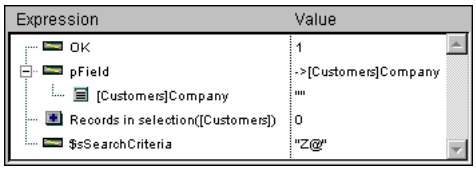
- Haben Sie die Parameterliste für eine Methode einmal eingesetzt, können Sie Parameter und Funktionsergebnisse per Drag&Drop in den **Bereich individuelle Überprüfung** setzen.

Bereich individuelle Überprüfung

Direkt unterhalb vom **Bereich Aufruffolge** liegt der **Bereich individuelle Überprüfung**. Hier bewerten Sie Ausdrücke. Sie können jede Art von Ausdruck bewerten. Dazu gehören Datenfelder, Variablen, Zeiger, Berechnungen, integrierte Funktionen, Ihre eigenen Funktionen, sowie alles, was einen Wert zurückgibt.

Sie können jeden Ausdruck bewerten, der sich in Textform anzeigen lässt. Davon ausgenommen sind Bilder, BLOB Felder oder Variablen. Der Debugger zeigt dagegen Arrays und Zeiger in hierarchischen Listen, die bereits eingesetzt wurden. Der Inhalt von BLOBs lässt sich mit BLOB Befehlen wie z.B. **BLOB to text** anzeigen.

Folgendes Beispiel zeigt verschiedene Arten an: Zwei Variablen, eine Variable vom Typ Feldzeiger und das Ergebnis einer integrierten Funktion sowie eine Berechnung.



Neuen Ausdruck einfügen

Einen Ausdruck, den Sie bewerten wollen, fügen Sie folgendermaßen in den Bereich individuelle Überprüfung ein:

- Sie bewegen ein Objekt oder Ausdruck per Drag & Drop aus dem Debugger
- Sie bewegen ein Objekt oder Ausdruck per Drag & Drop aus dem **Bereich Aufruffolge**
- Sie klicken im **Bereich Source Code** auf einen Ausdruck, der bewertbar ist

Wollen Sie einen leeren Ausdruck erstellen, doppelklicken Sie auf eine leere Stelle im Bereich individuelle Überprüfung. Ein Ausdruck *Neuer Ausdruck* wird hinzugefügt, Sie gelangen in den Bearbeitungsmodus. Hier können Sie jede 4D Formel eingeben, die ein Ergebnis zurückgibt.

Bestätigen Sie Ihre Eingabe mit der **Eingabetaste** oder der Zeilenschaltung oder klicken Sie an eine beliebige Stelle innerhalb des Bereichs.

Wollen Sie den Ausdruck ändern, wählen Sie diesen durch Anklicken aus. Durch erneutes Klicken bzw. durch Drücken der **Eingabetaste** auf dem Zahlenblock gelangen Sie in den Bearbeitungsmodus.

Benötigen Sie den Ausdruck nicht mehr, wählen Sie diesen durch Anklicken aus und drücken Sie dann die **Rückschritt-** oder die **Löschen-Taste**.

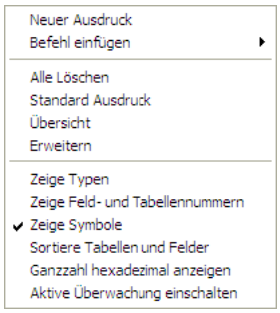
Kontextmenü

Über das Kontextmenü im Bereich individuelle Überprüfung gelangen Sie in den 4D Formeleditor, um einen Ausdruck einzugeben und zu bearbeiten. Dieses Menü bietet darüberhinaus weitere Optionen.

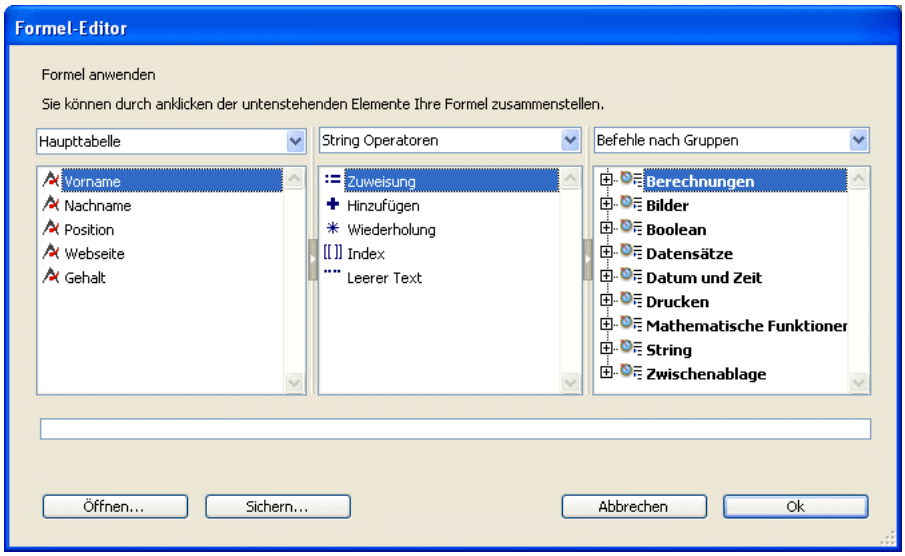
Um dieses Menü anzuzeigen:

- Klicken Sie unter Windows mit der **rechten** Maustaste auf eine beliebige Stelle im Bereich Überprüfung.
- Klicken Sie auf Macintosh mit der **ctrl-Taste+Klick** auf eine beliebige Stelle im Bereich Überprüfung. Auf Mac OS 8.0 oder neuer wählen Sie die Tastenkombination **Alt+ctrl+Klick**.

Das Kontextmenü erscheint:



- **Neuer Ausdruck:** Dieser Befehl fügt einen neuen Ausdruck ein und öffnet den 4D Formeleditor für seine Bearbeitung.



Weitere Informationen zum Formeleditor finden Sie im Handbuch *4D Designmodus*.

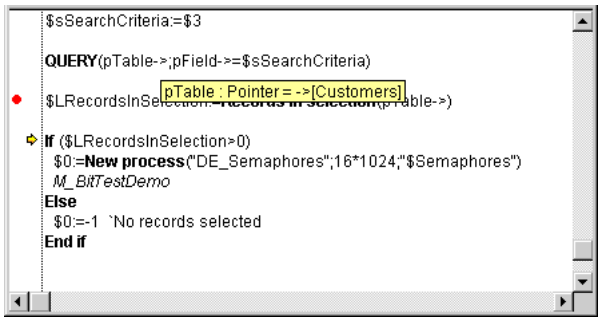
- **Befehl einfügen:** Damit geben Sie einen neuen Befehl direkt als neuen Ausdruck ein. Sie müssen nicht über den Formeleditor gehen.
- **Alle löschen:** Löscht alle aktuell angezeigten Ausdrücke.
- **Übersicht**: Blendet alle Ebenen der hierarchischen Liste im Bereich Überprüfung aus.
- **Erweitern**: Blendet alle Ebenen der hierarchischen Liste im Bereich Überprüfung ein.
- **Zeige Typen**: Zeigt für jedes Objekt die entsprechenden Objekttypen.
- **Zeige Feld- und Tabellennummern**: Zeigt die Nummer für jede Tabelle oder jedes Datenfeld von **Felder**. Diese Option ist sehr hilfreich, wenn Sie mit Tabellen oder Datenfeldnummern bzw. mit Zeigern arbeiten, die Funktionen wie **Table** oder **Field** benutzen.
- **Zeige Symbole**: Zeigt den Objekttyp für jedes Objekt als Icon an. Sie können diese Option ausschalten, wenn die Anzeige schneller erfolgen soll oder Sie nur die Option **Zeige Typen** verwenden wollen.
- **Sortiere Tabellen und Felder**: Zeigt die Tabellen und Datenfelder in den jeweiligen Listen in alphabetischer Reihenfolge an.
- **Ganzzahl hexadezimal anzeigen:** Zahlen werden normalerweise in Dezimalschreibweise angezeigt. Diese Option zeigt Zahlen hexadezimal an. Hinweis: Wollen Sie einen Zahlenwert hexadezimal eingeben, tippen Sie 0x (Null + x) und die hexadezimalen Stellen.
- **Aktive Überwachung einschalten:** Zeigt zusätzliche Information zu Zeitplan und Kommunikation über Netzwerk. Hier können Sie die interne Aktivität der Anwendung steuern. Bedenken Sie jedoch, dass diese Option den Ablauf von Prozessen verlangsamt.

Bereich Source Code

Der **Bereich Source Code** zeigt den Source Code für die Methode, die im Schrittmodus durchlaufen wird.

- Ist die Methode zu lang für den angezeigten Textbereich, können Sie nach oben und unten scrollen.
- Bewegen Sie den Mauszeiger auf einen bewertbaren Ausdruck (Datenfeld, Variable, Zeiger, Array,...), erscheint ein Hilfetext (**Tool Tip**), der den aktuellen Wert des Objekts oder Ausdrucks und den zugewiesenen Typ anzeigt.

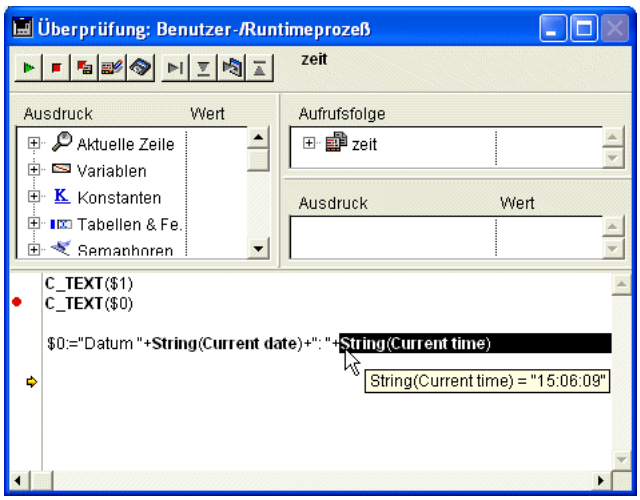
Hier ein Beispiel des **Bereich Source Code**:



Ein Hilfetext erscheint, da der Mauszeiger Über die Variable *pTable* gezogen wurde. Er zeigt an, dass die Variable ein Zeiger auf die Tabelle *[Customers]* ist.

Tipp: Klicken Sie im **Bereich Source Code** auf einen bewertbaren Ausdruck, wird der Ausdruck oder das Objekt in den **Bereich individuelle Überprüfung** kopiert.

- Sie können im Bereich, der den Code in der Ausführung anzeigt, auch einen Teil des Textes auswählen. Setzen Sie den Cursor über den ausgewählten Text, zeigt ein Tipp den Wert des ausgewählten Objekts an:



Klicken Sie auf einen Variablen- oder Feldnamen, wird er automatisch ausgewählt.

Tipp: Sie können auch einen beliebigen ausgewählten Ausdruck - der bewertbar ist - aus dem **Bereich Source Code** in den **Bereich individuelle Überprüfung** kopieren. Es gibt folgende Möglichkeiten:

- Sie verwenden Drag-and-Drop (Klicken Sie auf den ausgewählten Text und ziehen ihn in den Bewertungsbereich)
- Sie betätigen unter Windows die Tastenkombination **ctrl+D**, auf Macintosh die Tastenkombination **Befehlstaste+D** .

Programmzähler

Ein gelber Pfeil am linken Rand des Bereichs Source Code (siehe oben) markiert die Zeile, die als nächstes ausgeführt wird. Das ist der **Programmzähler**. Er gibt immer an, welche Zeile als nächste ausgeführt wird.

HInweis: Standardmäßig wird die Zeile des Programmzählers (auch *laufende Zeile* genannt) im Debugger hervorgehoben. In den 4D Einstellungen auf der **Seite Methoden** können Sie die Farbe zum Hervorheben anpassen.

Sie können den Programmzähler bei der Fehlersuche für die Methode verschieben, die gerade ausgeführt wird, also auf der obersten Ebene der Aufrufsfolge liegt.

WARNUNG: Nutzen Sie dieses Feature mit Vorsicht!

Das Vorwärtsstellen des Programmzählers bedeutet NICHT, dass der Debugger die übersprungenen Zeilen schneller ausführt. Genausowenig bedeutet das Zurücksetzen des Programmzählers, dass der Debugger die Auswirkung der bereits ausgeführten Zeilen rückgängig macht.

Das Bewegen des Programmzählers bedeutet lediglich, dass der Debugger ab hier mit dem Schrittmodus fortfahren soll. Alle aktuellen Einstellungen, Datenfelder, Variablen usw. sind davon nicht betroffen.

Hier ein Beispiel. Wir gehen von folgendem Code aus:

```
  ` ...
If(This condition)
   DO SOMETHING
Else
   DO SOMETHING ELSE
End if
  ` ...
```

Der Programmzähler wird vor die Zeile **If (This condition)** gesetzt. Sie gehen einen Schritt, der Programmzähler wird nun vor die Zeile **DO SOMETHING ELSE** gesetzt. Das ist unglücklich, denn Sie wollten auch die andere Alternative ausführen. Führt der Ausdruck **This condition** keine Operationen aus, die die nächsten Schritte Ihrer Überprüfung beeinflussen, können Sie den Programmierzähler auf die Zeile **DO SOMETHING** zurücksetzen, um diesen Teil im Schrittmodus zu durchlaufen.

Unterbrechungspunkte setzen

Während der Fehlersuche müssen Sie manchmal den Schrittmodus für einige Teile des Code überspringen. Der Debugger bietet mehrere Möglichkeiten, den Code **bis zu einem gewissen Punkt** auszuführen:

- Sie können beim Durchlaufen im Schrittmodus auf die Schaltfläche **Step Over** anstatt auf **Step Into** klicken. Das ist sinnvoll, wenn Sie in evtl. vorhandene Unterroutinen oder Funkionen gehen wollen, die innerhalb einer Programmierzeile aufgerufen werden.
- Sind Sie versehentlich in eine Unterroutine gegangen, können Sie diese ausführen und gelangen durch Anklicken der Schaltfläche **Step Out** direkt zurück in die aufrufende Methode.
- Haben Sie an irgendeiner Stelle einen Aufruf **TRACE** eingebaut und Sie klicken auf die Schaltfläche **Weiter**, wird die Ausführung bis zu diesem Aufruf fortgeführt.

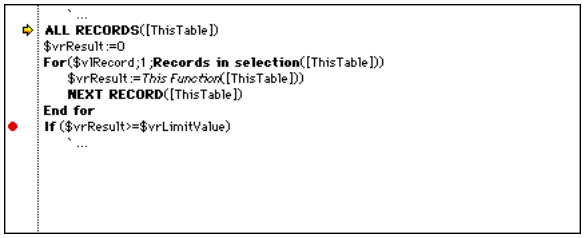
Sie führen beispielsweise folgenden Code aus. Der Programmzähler steht vor der Zeile **ALL RECORDS([ThisTable])**:

```
  ` ...
ALL RECORDS([ThisTable])
$vrResult:=0
For($vlRecord;1;Records in selection([ThisTable]))
   $vrResult:=This Function([ThisTable])
   NEXT RECORD([ThisTable])
End for
If($vrResult>=$vrLimitValue)
  ` ...
```

Sie möchten den Wert von *$vrResult* prüfen, nachdem die For-Schleife abgeschlossen ist. Da es jedoch eine geraume Zeit dauert, bis Sie diesen Punkt in Ihrem Code erreichen, wollen Sie die akuelle Ausführung abbrechen, um in der Methode vor der Zeile *If ($vrResult...* einen Aufruf **TRACE** einzufügen.

Eine Möglichkeit wäre, die Schleife zu durchlaufen. Enthält die Tabelle *[ThisTable]* jedoch hunderte von Datensätzen, dauert das einen ganzen Tag. Für solche Fälle bietet der Debugger **Unterbrechungspunkte**. Sie setzen Unterbrechungspunkte, wenn Sie in den linken Rand des Bereichs Source Code klicken.

Zum Beispiel:
Sie klicken in Höhe der Zeile *If ($vrResult...* in den linken Rand des Bereichs Source Code:



Neben dieser Zeile wird ein Unterbrechungspunkt gesetzt. Er erscheint als roter Punkt. Klicken Sie auf die Schaltfläche **Weiter**.

Die normale Ausführung läuft nun bis zu diesem Unterbrechungspunkt. Die markierte Zeile wird nicht ausgeführt—Sie sind wieder im Schrittmodus. In diesem Beispiel wurde folglich die gesamte Schleife normal ausgeführt. Sobald der Unterbrechungspunkt erreicht wird, müssen Sie lediglich die Maustaste über *$vrResult* ziehen, um den Wert am Endpunkt der Schleife zu prüfen.

Setzen Sie einen Unterbrechungspunkt außerhalb des Programmzählers und klicken Sie auf die Schaltfläche **Weiter**, können Sie Teile der Methode überspringen, die im Schrittmodus durchlaufen wird.

Hinweis: Sie können Unterbrechungspunkte auch direkt im 4D Methodeneditor setzen. Weitere Informationen dazu finden Sie im Abschnitt **Unterbrechungspunkte**.

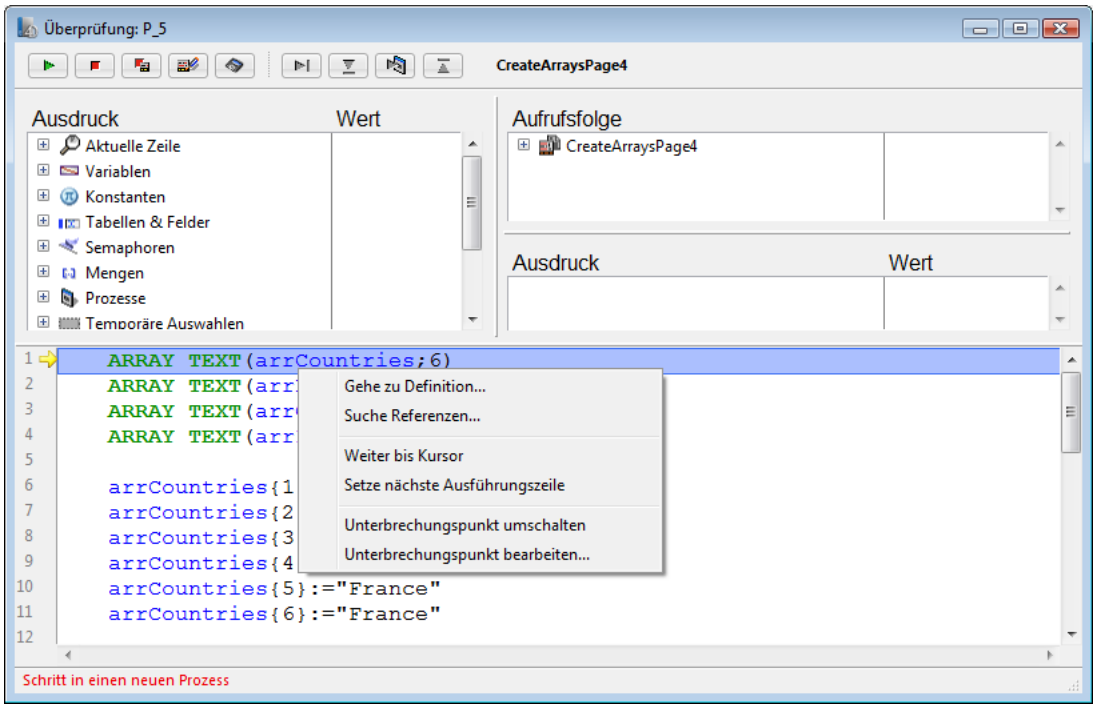
Ein **roter Unterbrechungspunkt** ist ein **ständiger** Unterbrechungspunkt. Er bleibt, sobald er angelegt wurde und das selbst dann, wenn Sie die Datenbank verlassen und später wieder öffnen.

Es gibt zwei Möglichkeiten, ständige Unterbrechungspunkte zu entfernen:

- Haben Sie den Unterbrechungspunkt bearbeitet, klicken Sie auf den roten Punkt, um ihn zu entfernen — der Unterbrechungspunkt verschwindet.
- Haben Sie den Unterbrechungspunkt nicht vollständig bearbeitet, wollen ihn also noch weiter behalten, können Sie ihn vorübergehend deaktivieren. Weitere Informationen dazu finden Sie im Abschnitt **Unterbrechungspunkte**.

Kontextmenü des Bereichs Quellcode

Das Kontextmenü des **Bereich Source Code** bietet Zugriff zu verschiedenen Funktionen, die beim Ausführen von Methoden im Schrittmodus hilfreich sind:



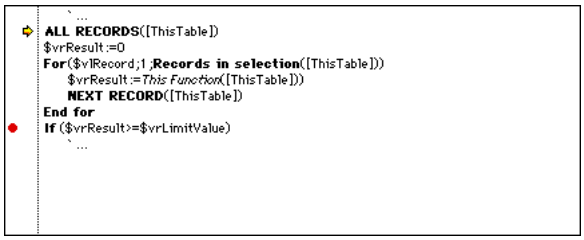
- **Gehe zu Definition:** Geht an die Stelle, wo das ausgewählte Objekt definiert ist. Dieser Befehl ist für folgende Objekte verfügbar:
  - *Projektmethoden:* Zeigt den Methodeninhalt in einem neuen Fenster des Methodeneditors
  - *Felder:* Zeigt Feldeigenschaften im Inspektor des Strukturfensters
  - *Tabellen:* Zeigt Tabelleneigenschaften im Inspektor des Strukturfensters
  - *Formulare:* Zeigt Formulare im Formulareditor
  - *Variablen (lokal, Prozess, Interprozess oder Parameter $n):* Zeigt die Zeile in der aktuellen Methode oder in den Compiler-Methoden, wo die Variable deklariert wurde.
- **Suche Referenzen** (auch im Methodeneditor verfügbar): Sucht alle Datenbankobjekte (Methoden und Formulare), auf die sich das aktuelle Element der Methode bezieht. Das aktuelle Element ist das markierte bzw. das, wo der Cursor steht. Es kann der Name eines Feldes, Befehls, Strings, einer Variablen, etc. sein. Suchergebnisse erscheinen in einem neuen Standard Ergebnisfenster.
- **Weiter bis Cursor:** Führt Anweisungen aus, die zwischen dem Programmzähler (gelber Pfeil) und der ausgewählten Linie der Methode liegen, d.h. wo der Cursor steht.
- **Setze nächste Ausführungszeile:** Bewegt den Programmzähler zur ausgewählten Zeile, ohne diese oder dazwischenliegende Zeilen auszuführen. Die angegebene Zeile wird nur ausgeführt, wenn der Benutzer auf eine der Ausführungsschaltflächen klickt.
- **Unterbrechungspunkt umschalten** (auch im Methodeneditor verfügbar): Fügt den Unterbrechungspunkt für die ausgewählte Zeile hinzu oder entfernt ihn. Dies ändert den Unterbrechungspunkt dauerhaft: Entfernen Sie z.B. einen Unterbrechungspunkt im Debugger, erscheint er nicht mehr in der ursprünglichen Methode.
- **Unterbrechungspunkt bearbeiten** (auch im Methodeneditor verfügbar): Öffnet das Dialogfenster Unterbrechungspunkt Eigenschaften. Alle hier gewählte Optionen verändern den Unterbrechungspunkt dauerhaft.

Unterbrechungspunkte

Wie bereits im **Bereich Source Code** beschrieben, setzen Sie einen Unterbrechungspunkt, wenn Sie im Bereich Source Code oder im Fenster Methodeneditor in Höhe der entsprechenden Programmierzeile an den linken Rand klicken.

Hinweis: Da Sie Unterbrechungspunkte sowohl im **Bereich Source Code** als auch direkt im Methodeneditor einfügen, ändern oder löschen können, gibt es einen dynamischen Austausch zwischen Methodeneditor, Debugger und Runtime Explorer. Temporäre Unterbrechungspunkte können Sie jedoch nur im Debugger setzen (siehe unten).

Im nachfolgenden Beispiel wurde im Debugger ein Unterbrechungspunkt vor die Zeile *If($vrResult>=$vrLimitValue)* gesetzt:



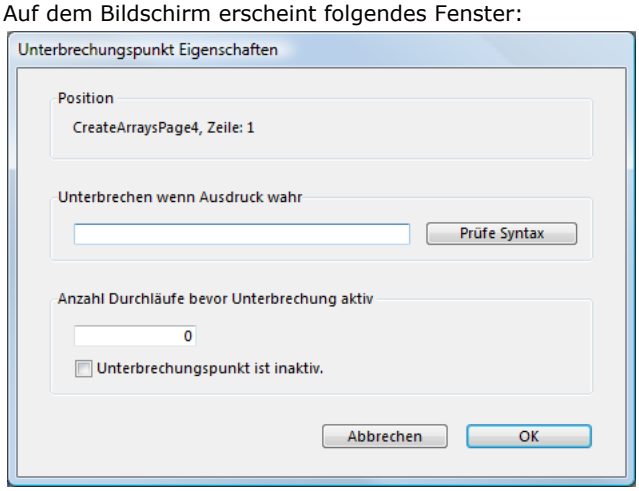
Wollen Sie den Unterbrechungspunkt wieder löschen, klicken Sie erneut darauf.

Unterbrechungspunkt bearbeiten

Drücken Sie für eine Programmierzeile am linken Rand vom **Bereich Source Code** oder im Fenster Methodeneditor die Kombination **Alt-Klick** unter Windows, **Wahltaste-Klick** auf Macintosh, erscheint das Fenster **Eigenschaften Unterbrechungspunkte**.

- Klicken Sie auf einen bestehenden Unterbrechungspunkt, erscheint das Fenster für diesen Unterbrechungspunkt.
- Klicken Sie auf eine Zeile ohne Unterbrechungspunkt, wird er gesetzt und das Fenster für diesen neuen Unterbrechungspunkt angezeigt.

Auf dem Bildschirm erscheint folgendes Fenster:



Es ist in folgende Bereiche gegliedert:

Position: Hier erscheint der Name der Methode und die Zeilennummer, wo der Unterbrechungspunkt gesetzt wurde. Sie können diese Information nicht ändern.

Unterbrechen wenn Ausdruck wahr: Sie können auch bedingte Unterbrechungspunkte erstellen. Geben Sie dazu eine 4D Formel ein, die Wahr oder Falsch zurückgibt. Wollen Sie zum Beispiel in einer Zeile nur unterbrechen, wenn gilt *Records in selection([aTable])=0*, geben Sie diese Formel ein. Stößt der Debugger nun auf eine Zeile mit Unterbrechungspunkt, wird sie nur unterbrochen, wenn für die Tabelle *[aTable]* keine Datensätze ausgewählt wurden. Mit der Schaltfläche **Prüfe Syntax** können Sie die Syntax Ihrer Formel überprüfen.

Anzahl Durchläufe bevor Unterbrechung aktiv: Sie können einen Unterbrechungspunkt auch in einer Programmierzeile setzen, die innerhalb einer Schleife mit While, Repeat oder For, innerhalb einer Unterroutine oder einer Funktion liegt, die von einer Schleife aus aufgerufen wird. Sie wissen zum Beispiel, dass das Problem erst ab dem 200. Durchlaufen der Schleife auftritt. Folglich geben Sie 200 ein, damit der Unterbrechungspunkt ab dem 201. Durchlauf aktiviert wird.

Unterbrechungspunkt ist inaktiv: Benötigen Sie einen dauerhaften Unterbrechungspunkt gerade nicht, können Sie ihn temporär ausblenden. Er erscheint dann im **Bereich Source Code** des Debugger-Fensters, im Methodeneditor und in der Registerkarte **Unterbrechung** des Runtime Explorers als Bindestrich (-) anstatt als Punkt (•).

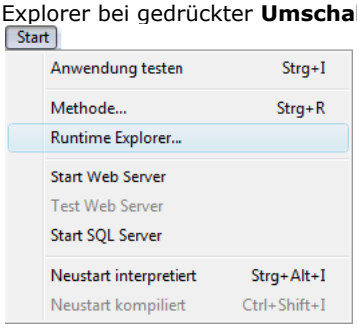
Sie erstellen und bearbeiten Unterbrechungspunkte vom Fenster Debugger oder dem Methodeneditor aus. Bestehende Unterbrechungspunkte können Sie auch in der Registerkarte **Unterbrechung** des Runtime Explorers bearbeiten. Weitere Informationen dazu finden Sie im folgenden Abschnitt **Unterbrechungsliste**.

Unterbrechungsliste

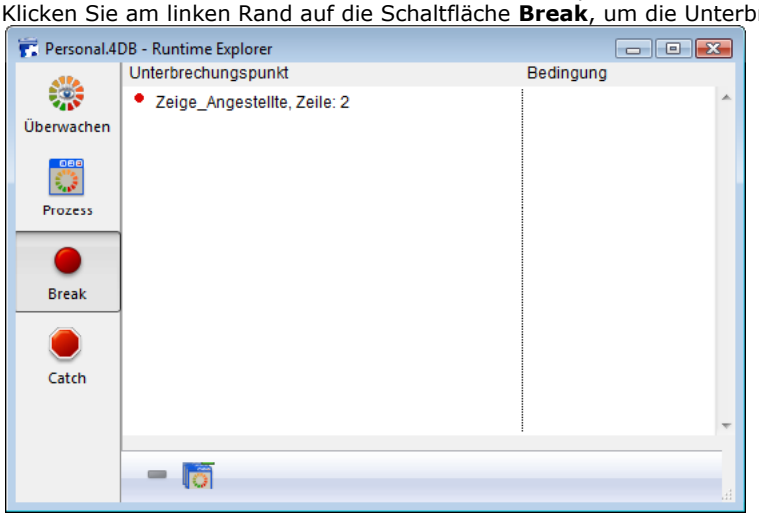
Unterbrechungsliste ist eine Seite des Runtimer Explorer, um dauerhafte Unterbrechungspunkte zu verwalten, die im Debugger-Fenster oder im Methodeneditor erstellt wurden.

Um die Seite Unterbrechungsliste zu öffnen:

1. Wählen Sie im Menü **Start** den Befehl **Runtime Explorer**.
   Er lässt sich als Palettenfenster anzeigen und bleibt dann immer im Vordergrund erhalten. Wählen Sie dazu den Runtime Explorer bei gedrückter **Umschalttaste** aus. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.



   Auf dem Bildschirm erscheint das Fenster Runtime Explorer.
2. Klicken Sie am linken Rand auf die Schaltfläche **Break**, um die Unterbrechungsliste anzuzeigen:



Die Unterbrechungsliste besteht aus zwei Spalten:

- Die linke Spalte zeigt den Status Aktivieren/Inaktivieren des Unterbrechungspunktes, den Methodennamen und die Zeilennummer, wo der Unterbrechungspunkt im Debugger-Fenster bzw. Methodeneditor gesetzt wurde.
- Die rechte Spalte zeigt -sofern vorhanden- die Bedingung, die diesem Unterbrechungspunkt zugewiesen wurde.

In diesem Fenster können Sie folgendes ausführen:

- Eine Bedingung für einen Unterbrechungspunkt setzen
- Jeden Unterbrechungspunkt aktivieren, deaktivieren oder löschen
- Mit Doppelklick auf einen Unterbrechungspunkt die dazugehörige Methode im Methodeneditor öffnen.

In diesem Fenster können Sie dagegen keinen neuen dauerhaften Unterbrechungspunkt hinzufügen. Das ist nur im Debugger-Fenster oder im Methodeneditor möglich.

Bedingung für Unterbrechungspunkt setzen

1. Klicken Sie in der rechten Spalte in die Eingabe.
2. Geben Sie eine 4D Formel ein (Ausdruck, Befehlsaufruf oder Projektmethode) ein, die einen Boolean Wert zurückgibt.

Hinweis: Wollen Sie eine Bedingung entfernen, löschen Sie die entsprechende Formel.

Unterbrechungspunkt aktivieren/deaktivieren

1. Markieren Sie die Eingabe oder navigieren Sie mit den Pfeiltasten durch die Liste, wenn die aktuell ausgewählte Eingabe noch nicht im Eingabemodus ist.
2. Wählen Sie im Menü Unterbrechungsliste bzw. im Kontextmenü den Befehl **Aktivieren/Deaktivieren**.

Kürzel: Sie können jede Eingabe in der Liste durch Anklicken des Punktes (•) aktivieren/deaktivieren. Eine inaktive Eingabe wird mit Bindestrich (–) angezeigt.

Unterbrechungspunkt löschen

1. Wählen Sie den Unterbrechungspunkt durch Anklicken oder durch Navigieren mit den Pfeiltasten in der Liste aus (wenn die aktuell ausgewählte Eingabe noch nicht im Eingabemodus ist).
2. Wählen Sie die Taste **Löschen** oder **Rückschritt** oder klicken Sie unter der Liste auf die Schaltfläche **Löschen**.

Hinweis: Wollen Sie alle Unterbrechungspunkte löschen, klicken Sie unter der Liste auf die Schaltfläche **Alle Löschen** oder wählen Sie im Menü Kontext den Befehl **Alle Löschen**.

Befehle unterbrechen

Mit der Seite **Catch** (Erkannte Befehle) des Runtime Explorers können Sie für Ihren Code zusätzliche Unterbrechungen setzen durch Unterbrechen der aufgerufenen 4D Befehle.

Unterbrechen Sie einen Befehl, können Sie die Ausführung jedes Prozesses im Schrittmodus durchlaufen, sobald dieser Prozess einen Befehl aufruft. Das Abfangen eines Befehls betrifft alle Prozesse, die 4D Code ausführen und diesen Befehl aufrufen. Ein Unterbrechungspunkt liegt dagegen in einer bestimmten Projektmethode und löst bei Unstimmigkeiten im Code den Schrittmodus nur aus, wenn dieser Punkt erreicht wird.

Das Unterbrechen eines Befehls ist zum Durchlaufen langer Code-Teile besser geeignet als an willkürlichen Stellen gesetzte Unterbrechungspunkte. Wird z.B. ein Datensatz, der nicht gelöscht werden soll, nach Ausführen eines oder mehrerer Prozesse gelöscht, können Sie versuchen, die Fehlersuche einzugrenzen, wenn Sie Befehle wie **DELETE RECORD** und **DELETE SELECTION** aufrufen. Immer wenn diese Befehle aufgerufen werden, können Sie prüfen, ob der betreffende Datensatz gelöscht wurde. So können Sie schnell den fehlerhaften Teil des Code isolieren.

Mit etwas Erfahrung können Sie Unterbrechungspunkte und Befehle unterbrechen kombinieren.

Um die Seite **Catch** zu öffnen:

1. Wählen Sie im Menü **Start** den Eintrag **Runtime Explorer**.
   Der Runtime Explorer lässt sich in einem Palettenfenster anzeigen. Wählen Sie dazu den Eintrag **Runtime Explorer** bei gedrückter **Umschalttaste**. Weitere Informationen dazu finden Sie im Kapitel **Runtime Explorer** des Handbuchs *4D Designmodus*.

   Auf dem Bildschirm erscheint das Fenster Runtime Explorer.
2. Klicken Sie auf die Registerkarte **Catch.**
   Sie erhalten die Liste **Erkannte Befehle**:

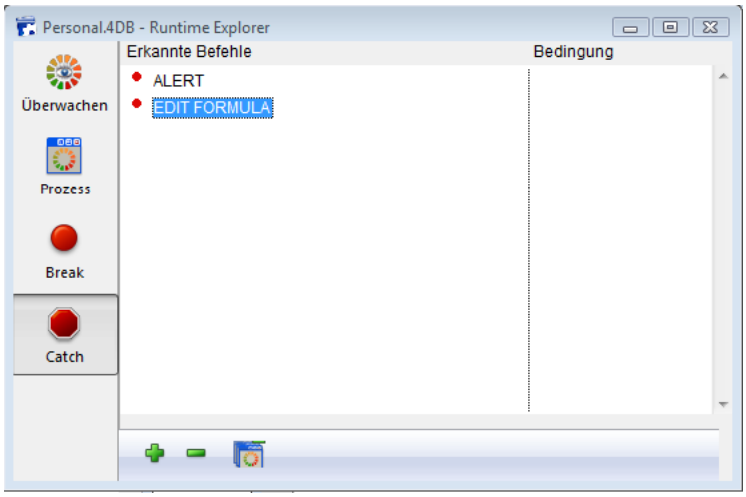
Diese Seite zeigt die Befehle, die während der Ausführung unterbrochen werden. Sie besteht aus zwei Spalten:

- Die linke Spalte zeigt den Status Aktiviert/Deaktiviert des gefundenen Befehls und den Befehlsnamen.
- Die rechte Spalte zeigt -sofern vorhanden- die Bedingung, die diesem Befehl zugewiesen ist.

Neuen Befehl hinzufügen, der unterbrochen werden soll

Um einen neuen Befehl hinzuzufügen:

Klicken Sie am unteren Rand auf das Pluszeichen. In der Liste wird ein neuer Eintrag hinzugefügt. Der Befehl **ALERT** ist standardmäßig vorgegeben.



Klicken Sie auf den Eintrag **ALERT** und geben den Befehlsnamen ein, den Sie unterbrechen wollen. Betätigen Sie die **Eingabetaste** oder die **Zeilenschaltung**, um die Eingabe zu bestätigen.

Name des unterbrochenen Befehls bearbeiten

Um den Namen eines unterbrochenen Befehls zu editieren

1. Wählen Sie die Eingabe aus. Dazu klicken Sie entweder auf den Eintrag oder navigieren mit den Pfeiltasten durch die Liste, wenn der gerade ausgewählte Eintrag noch nicht editierbar ist.
2. Um einen Eintrag vom Modus Bearbeiten in den Modus Auswählen zu bringen, drücken Sie die **Eingabetaste** oder die **Zeilenschaltung**.
3. Geben Sie den Namen des Befehls ein oder ändern ihn.
4. Betätigen Sie die **Eingabetaste** oder die **Zeilenschaltung**, um die Änderungen zu bestätigen.

Unterbrochenen Befehl aktivieren/deaktivieren

Um einen unterbrochenen Befehl zu deaktivieren oder zu aktivieren:

Klicken Sie auf den Punkt vor dem Eintrag.
Auf diese Weise können Sie den Unterbrechungspunkt jeweils aktivieren oder deaktivieren

- rot = aktiviert
- orange = deaktiviert

Hinweis: Deaktivieren eines unterbrochenen Befehls ist fast dasselbe wie Löschen. Während der Ausführung benötigt der Debugger kaum Zeit bei der Eingabe. Deaktivieren eines Eintrags hat den Vorteil, dass sie ihn nicht neu erstellen müssen, falls sie ihn wieder benötigen.

Unterbrochenen Befehl deaktivieren

Um einen unterbrochenen Befehl zu löschen:

1. Wählen Sie die Eingabe aus. Dazu klicken Sie entweder auf den Eintrag oder navigieren mit den Pfeiltasten durch die Liste, wenn der gerade ausgewählte Eintrag noch nicht im Modus Bearbeiten ist.
2. Betätigen Sie die **Rückschritttaste** oder die **Löschtaste** oder klicken Sie am unteren Rand auf die Schaltfläche **Löschen**.
3. Um alle unterbrochenen Befehle zu löschen, klicken Sie am unteren Rand auf die Schaltfläche **Alle Löschen**.

Unterbrochenem Befehl eine Bedingung zuweisen

Um eine Bedingung zum Unterbrechen eines Befehls zu setzen:

1. Klicken Sie in der rechten Spalte in die Eingabe.
   Ein Cursor zur Eingabe erscheint
2. Geben Sie eine 4D Formel (Ausdruck, Befehlsaufruf oder Projektmethode) ein, die einen Boolean Wert zurückgibt.

Hinweis: Wollen Sie eine Bedingung entfernen, löschen Sie die dazugehörige Formel.

Über eine Bedingung erreichen Sie, dass die Ausführung des Befehls nur gestoppt wird, wenn die Bedingung zutrifft. Setzen Sie z.B. die Bedingung "**Records in selection**([Emp]>10)" mit dem Unterbrechungspunkt auf den Befehl **DELETE SELECTION**, wird der Code während der Ausführung von **DELETE SELECTION** nicht gestoppt, wenn die aktuelle Auswahl der Tabelle [Emp] nur 9 Datensätze oder weniger enthält.

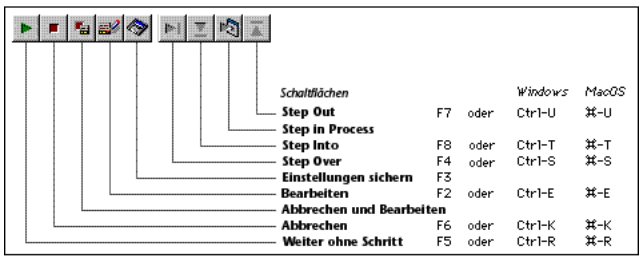
Unterbrochene Befehle mit einer Bedingung verlangsamen die Ausführung, da die Bedingung bei jedem Ausnahmefall geprüft werden muss. Eine Bedingung kann die Ausführung aber auch beschleunigen, da 4D alle Vorkommen, die nicht zur Bedingung passen, automatisch überspringt.

Tastenkürzel des Debuggers

Dieser Abschnitt erläutert die Tastenkürzel des Debugger-Fensters.

Steuerleiste

Nachfolgendes Schema zeigt die Tastenkürzel für die neun Schaltflächen in der oberen linken Ecke des Debugger-Fensters:



Klicken Sie mit **Umschalttaste+F5** unter Windows, **Umschalttaste+Klick** auf Macintosh auf die Schaltfläche **Weiter ohne Schritt**, wird die Ausführung zusammengefasst. Das deaktiviert auch alle weiteren Aufrufe von **TRACE** für den aktuellen Prozess.

Bereich Überprüfung

- Drücken Sie im **Bereich Überprüfung** unter Windows die rechte Maustaste, auf Macintosh die Kombination **ctrl-Taste+Klick**, erscheint das dazugehörige Kontextmenü.
- Doppelklicken Sie im **Bereich Überprüfung** auf einen Eintrag, wird er in den **Bereich individuelle Überprüfung** kopiert.

Bereich Aufruffolge

Doppelklicken Sie im **Bereich Aufruffolge** auf einen Methodennamen, erscheint die Methode im **Bereich individuelle Überprüfung** in der Zeile, die diese Aufruffolge aufruft.

Bereich individuelle Überprüfung

- Drücken Sie im **Bereich individuelle Überprüfung** unter Windows die rechte Maustaste, auf Macintosh die Kombination **ctrl-Taste+Klick**, erscheint das dazugehörige Kontextmenü.
- Doppelklicken Sie im **Bereich individuelle Überprüfung**, erscheint ein neuer Bereich.

Bereich Source Code

- Klicken Sie in den linken Rand, werden ständige Unterbrechungspunkte gesetzt bzw. entfernt.
- Mit der Kombination **ALT-Shift-Klick** unter Windows, **Wahltaste-Shift-Klick** auf Macintosh setzen Sie temporäre Unterbrechungspunkte.
- **Alt-Klick** unter Windows, **Wahltaste-Klick** auf Macintosh zeigt das Fenster **Unterbrechung bearbeiten** für einen neuen oder bestehenden Unterbrechungspunkt.
- Eine ausgewählter Ausdruck bzw. ein Objekt lässt sich per Drag-and-Drop in den **Bereich individuelle Überprüfung** kopieren.
- Die Tastenkombination **Ctrl+D** unter Windows, **Befehl+D** auf Macintosh kopiert den ausgewählten Text in den **Bereich individuelle Überprüfung**.

Alle Bereiche

- Mit **Strg + *** unter Windows, **Befehlstaste + *** auf Mac OS wird der **Bereich Überprüfung** aktualisiert.
- Ist in keinem der Bereiche ein Eintrag ausgewählt und Sie drücken die **Eingabetaste**, wird die Zeile Schritt für Schritt durchlaufen.
- Ist ein Eintrag ausgewählt, navigieren Sie mit den Pfeiltasten durch die Liste.
- Wurde ein Eintrag bearbeitet, bewegen Sie den Cursor mit den Pfeiltasten; drücken Sie unter Windows die Kombination **Strg-Taste + A/X/C/V**, auf Macintosh die Kombination **Befehlstaste + A/X/C/V** für die Befehle **Alles auswählen**/**Ausschneiden**/**Kopieren**/**Einsetzen** des Menüs **Bearbeiten**.

4D Umgebung

- 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 -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	String	Kompletter Name der ausführbaren 4D Datei oder des 4D Programms.

Beschreibung

Die Funktion **Application file** gibt den kompletten Namen des festgelegten 4D Programms zurück.

Windows
Arbeiten Sie z.B. mit 4D aus dem Volume E unter Programme\4D, gibt der Befehl zurück: E:\Programme\4D\4D.exe.

Macintosh
Arbeiten Sie z.B. mit 4D aus dem Ordner Programme auf der Festplatte Macintosh HD, gibt der Befehl zurück: Macintosh HD:Programme:4D .app.

Beispiel

Beim Starten von Windows müssen Sie prüfen, ob DLL Library auf derselben Ebene wie die programmierbare 4D Datei liegt. Sie schreiben in der **Datenbankmethode On Startup** für Ihre Anwendung:

```
If(On Windows & (Application type#4D Server))
   If(Test path name(Long name to path name(Application file)+"DOFAX.DLL")
      #Is a document)
   ` Zeige Meldung, dass die Library DOFAX.DLL fehlt.
   ` Deshalb sind die Faxfähigkeiten nicht verfügbar.
   End if
End if
```

Hinweis: Weitere Informationen zu den Projektmethoden ***On Windows*** und ***Long name to path name*** finden Sie im Abschnitt **Einführung in Systemdokumente**.

Application type

Application type -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl	➲	Numerischer Wert, der die Art der Anwendung anzeigt

Beschreibung

Die Funktion **Application type** gibt einen numerischen Wert zurück, der die Art der eingesetzten 4D Umgebung zeigt. 4D bietet dafür folgende vordefinierten Konstanten:

Konstante	Typ	Wert
4D Desktop	Lange Ganzzahl	3
4D Local mode	Lange Ganzzahl	0
4D Remote mode	Lange Ganzzahl	4
4D Server	Lange Ganzzahl	5
4D Volume desktop	Lange Ganzzahl	1

Hinweis: *4D Desktop* fügt bestimmte Deployment Optionen ein, wie z.B. "4D SQL Desktop".

Beispiel

Sie müssen an einer Stelle Ihres Code außerhalb der **Datenbankmethode On Server Startup** prüfen, ob 4D Server läuft. Sie schreiben:

```
If(Application type=4D Server)
   ` Führe entsprechende Aktionen aus
End if
```

Application version

Application version {(ErstellungsNr {; *})} -> Funktionsergebnis

Parameter	Typ		Beschreibung
ErstellungsNr	Lange Ganzzahl	⇐	Nummer der Erstellung
*	Operator	⇒	Mit *: Lange Versionsnummer Ohne *: Kurze Versionsnummer
Funktionsergebnis	String	⇒	Zeichenkette mit codierter Versionsnummer

Beschreibung

Die Funktion **Application version** gibt einen codierten String Wert zurück, der die Versionsnummer der eingesetzten 4D Umgebung anzeigt.

- Übergeben Sie keinen optionalen * Parameter, wird eine vierstellige Zeichenkette mit folgender Formatierung zurückgegeben:

Zeichen	Beschreibung
1-2	Versionsnummer
3	"R" Nummer
4	Revision Nummer

- Übergeben Sie einen optionalen * Parameter, wird eine achtstellige Zeichenkette mit folgender Formatierung zurückgegeben:

Zeichen	Beschreibung
1	"F" kennzeichnet eine Final Version "B" kennzeichnet eine Beta Version Andere Zeichen kennzeichnen eine 4D-interne Version
2-3-4	Interne 4D Kompilierungsnummer
5-6	Versionsnummer
7	"R" Nummer
8	Revision Nummer

Hinweis zur Kompatibilität (4D v14)
Die Nummerierung der Version hat sich mit 4D Version 14 geändert:

- **Die "R" Nummer** ist die Nummer der "R" Version von 4D (enthält 0 für eine Bugfix Version),
- **Die Revision Nummer** ist die Nummer der Bugfix Version von 4D (enthält 0 für eine "R" Version).

In bisherigen Versionen von 4D war die Nummer der "R" Version die Update Nummer; sie bezeichnete die Revision und die Revision Nummer selbst war immer 0.
Beispiele für kurze Versionsnummer:

Version	zurückgegebener Wert	
4D v13.1	"1310"	*Bisheriges Nummerierungssystem*
4D v14 R2	"1420"	4D v14 Release R2
4D v14 R3	"1430"	4D v14 Release R3
4D v14.1	"1401"	Erste Bugfix Revision von 4D v14
4D v14.2	"1402"	Zweite Bugfix Revision von 4D v14

Beispiele für lange Versionsnummer:

Version	zurückgegebener Wert
4D v12.5 beta	"B0011250"
4D v14 beta R2	"B0011420"
4D v14 final R3	"F0011430"
4D v14.1 beta	"B0011401"

Die Funktion **Application version** kann über den optionalen Parameter *ErstellungsNr* zusätzliche Information zurückgeben: Die Erstellungsnummer (Build number) der aktuellen Version der 4D Anwendung. Dies ist eine interne Kompilierungsnummer, die zur Festlegung der Version oder über die Abteilung 4D Technical Services vergeben werden kann.

Hinweis: Bei Anwendungen mit einkompilierter 4D Volume License ist die zurückgegebene Erstellungsnummer nicht signifikant. In diesem Kontext verwaltet der Entwickler die Informationen zur Version.

Beispiel 1

Dieses Beispiel zeigt die Versionsnummer der 4D Umgebung:

```
$vs4Dversion:=Application version
ALERT("Sie verwenden die Version "+String(Num(Substring($vs4Dversion;1;2)))
+"."+$vs4Dversion[[3]]+"."+$vs4Dversion[[4]])
```

Beispiel 2

Dieses Beispiel überprüft, ob Sie die Final Version verwenden:

```
If(Substring(Application version(*);1;1)#"F")
   ALERT("Stelle sicher, dass mit dieser Datenbank die Final Version von 4D verwendet wird!")
   QUIT 4D
End if
```

Beispiel 3

Folgender Code erstellt die Versionsnummer der Anwendung und ermöglicht die Unterscheidung zwischen "R" Releases und "Bug fix" Revisions:

```
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]] //Versionsnummer, z.B. 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 {(ProjektName)}

Parameter	Typ		Beschreibung
ProjektName	String	➡	Voller Zugriffspfad auf das zu verwendende Projekt

Beschreibung

Der Befehl **BUILD APPLICATION** startet den Prozess zum Erstellen der Anwendung. Er berücksichtigt Parameter, die im aktuellen oder dem in *ProjektName* angegebenen Anwendungsprojekt gesetzt sind.

Ein Anwendungsprojekt ist eine XML Datei, die alle Parameter enthält, die zum Erstellen einer Anwendung notwendig sind. Die meisten Parameter erscheinen im Dialogfenster „Build Application". Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus* im Abschnitt **Anwendung erstellen**. 4D erstellt standardmäßig für jede Datenbank ein Anwendungsprojekt mit Namen "buildapp.xml" und legt es im Ordner *Preferences* im Unterordner *BuildApp* ab.

Wurde die Datenbank noch nicht kompiliert oder ist der kompilierte Code veraltet, startet **BUILD APPLICATION** zuerst den Kompilierungsprozess. In diesem Fall erscheint das Kompilerfenster nicht, außer es tritt ein Fehler auf. Nur der Ablaufbalken wird angezeigt. Mit dem Befehl **MESSAGES OFF** können Sie den Ablaufbalken ausblenden.

Übergeben Sie nicht den optionalen Parameter *ProjektName*, zeigt der Befehl einen Standard Öffnen-Dialog, so dass Sie eine Projektdatei bestimmen können. Nach Bestätigen des Dialogs enthält die Variable *Document* den vollen Pfadnamen der offenen Projektdatei.

Übergeben Sie den Zugriffspfad und Namen einer XML Datei für ein gültiges Anwendungsprojekt (UTF-8 Codierung und Endung .xml), verwendet der Befehl die in dieser Datei definierten Parameter. Weitere Informationen zur Struktur und den verwendbaren Schlüsseln in der XML Datei eines Anwendungsprojekts finden Sie im Handbuch **4D XML Keys BuildApplication**.

Beispiel

Dieses Beispiel erstellt zwei Anwendungen in einer einzigen Methode:

```
BUILD APPLICATION("c:\\folder\\projects\\myproject1.xml")
If(OK=1)
   BUILD APPLICATION("c:\\folder\\projects\\myproject2.xml")
End if
```

Systemvariablen oder Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null). Die Systemvariable Document enthält den vollen Pfadnamen der offenen Projektdatei.

Fehler verwalten

Scheitert der Befehl, wird ein Fehler zurückgegeben. Dieser lässt sich über den Befehl **ON ERR CALL** abfangen.

CHANGE LICENSES

CHANGE LICENSES
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **CHANGE LICENSES** zeigt das Dialogfenster zur Aktualisierung der 4D Lizenzen.

Dieser Befehl lässt sich nur mit 4D Anwendungen im Einzelplatz verwenden und nicht von einer Komponente aufrufen. Sind Kennwörter angelegt, können nur der Designer oder Administrator diesen Befehl ausführen; wird er von Benutzern ohne entsprechende Zugriffsrechte aufgerufen, führt er nichts aus.

Im Dialogfenster zur Aktualisierung der 4D Lizenzen kann der Benutzer Plug-Ins und den Web Server auf dem Rechner aktivieren, wo er ausgeführt wird. In 4D erscheint das Dialogfenster, wenn Sie im Menü **Hilfe** den Befehl **Lizenz aktualisieren** aufrufen.

Mit **CHANGE LICENSES** lassen sich die Lizenzen aktivieren bzw. Expansion Pack Nummern in einer kompilierten 4D Anwendung im Einzelplatz hinzufügen, die beim Kunden im Einsatz sind. 4D Entwickler oder IS Manager setzen diesen Befehl ein, damit die Benutzer einer 4D Anwendung ihre Lizenzen selbst erweitern können und dafür nicht jedes Mal ein Update dieser Anwendung benötigen.

Weitere Informationen dazu finden Sie im Abschnitt **Installation und Aktivierung** des *4D Installationshandbuchs*.

Beispiel

In einer eigenen Konfiguration bzw. einem angepassten Dialogfenster Voreinstellungen fügen Sie eine Schaltfläche mit dieser Objektmethode ein:

```
// Objektmethode für Schaltfläche bLicense
CHANGE LICENSES
```

Auf diese Weise kann ein Benutzer Lizenzen aktivieren, ohne die Datenbank zu ändern.

Compact data file

Compact data file (StrukturPfad ; DatenPfad {; ArchivOrdner {; Option {; Methode}}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
StrukturPfad	Text	➡	Pfadname der Strukturdatei
DatenPfad	Text	➡	Pfadname der zu komprimierenden Datendatei
ArchivOrdner	Text	➡	Pfadname des Ordners, wo die Original Datendatei abgelegt wird
Option	Lange Ganzzahl	➡	Komprimierungsoptionen
Methode	Text	➡	Name der 4D Callback Methode
Funktionsergebnis	Text	↩	Kompletter Pfadname des Ordners mit den Original Datendateien

Beschreibung

Die Funktion **Compact data file** komprimiert die Datendatei, definiert im Parameter *DatenPfad*, die der Strukturdatei, definiert im Parameter *StrukturPfad*, zugeordnet ist. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Um die Kontinuität der Datenbankoperationen zu gewährleisten, ersetzt die neue komprimierte Datendatei automatisch die Originaldatei. Die Originaldatei wird zur Sicherheit nicht verändert und in einen speziellen Ordner mit Namen "Ersetzte Dateien (Komprimierung)" JJJJ-MM-DD HH-MM-SS" abgelegt, wobei JJJJ-MM-DD HH-MM-SS Datum und Zeit des Backup angeben. Zum Beispiel: "Ersetzte Dateien (Komprimierung) 2007-09-27 15-20-35".

Die Funktion gibt den vollständigen Dateinamen des Ordners zurück, der aktuell zum Speichern der Original Datendatei angelegt wird. Sie lässt sich nur in 4D im lokalen Modus oder in 4D Server (Serverprozeduren) ausführen. Die zu komprimierende Datendatei muss zur Strukturdatei passen, die im Parameter *StrukturPfad* definiert ist. Außerdem darf die Datendatei beim Ausführen der Funktion nicht geöffnet sein. Sonst wird ein Fehler erzeugt.
Tritt während der Komprimierung ein Fehler auf, bleiben die Originaldateien am ursprünglichen Ort bestehen. Ist der Datendatei eine Indexdatei (.4DIndx) zugeordnet, wird diese auch komprimiert. Ihre Originaldatei wird, wie die Datendatei, gesichert. Die neue komprimierte Version ersetzt die bisherige Datei.

- Im Parameter *StrukturPfad* übergeben Sie den vollständigen Pfadnamen der Strukturdatei, die der Datendatei, die Sie komprimieren wollen, zugeordnet ist. Diese Information ist für die Komprimierung erforderlich, der Pfad muss in der Schreibweise des Betriebssystems sein. Sie können auch einen leeren String übergeben; in diesem Fall erscheint der Standard-Öffnen Dialog, wo Sie die gewünschte Strukturdatei wählen können.
- Im Parameter *DatenPfad* können Sie einen leeren String, einen Dateinamen oder einen kompletten Pfadnamen in der Schreibweise des Betriebssystems angeben. Übergeben Sie einen leeren String; erscheint der Standard-Öffnen Dialog, wo der Benutzer die gewünschte Datendatei wählen kann. Sie muss zur Strukturdatei passen, die im Parameter *StrukturPfad* übergeben wurde. Übergeben Sie nur den Namen der Datendatei, sucht 4D danach auf derselben Ebene wie die Strukturdatei.
- Im optionalen Parameter *ArchivOrdner* geben Sie die Position des Ordners "Ersetzte Dateien (Komprimieren) DatumZeit" an, der für die Originalversion der Datendatei sowie evtl. Indexdateien vorgesehen ist. Der Befehl gibt den vollständigen Pfadnamen des aktuell erstellten Ordners an.
  - Geben Sie diesen Parameter nicht an, werden die Originaldateien automatisch in einen Ordner "Ersetzte Dateien (Komprimierung) DatumZeit" gelegt, der neben der Strukturdatei angelegt wird.
  - Übergeben Sie einen leeren String, erscheint der Standard-Öffnen Dialog, wo der Benutzer die Position des anzulegenden Ordners angeben kann.
  - Übergeben Sie einen Pfadnamen (in der Schreibweise des Betriebssystems), erstellt der Befehl den Ordner "Ersetzte Dateien (Komprimierung)" DatumZeit" an dieser Position.
- Mit dem optionalen Parameter *Option* können Sie verschiedene Einstellungen für die Komprimierung setzen. Dafür verwenden Sie die folgenden Konstanten unter dem Thema **Datendatei Wartung**. Sie können auch mehrere Optionen miteinander kombinieren:

Konstante	Typ	Wert	Kommentar
Compact address table	Lange Ganzzahl	131072	Neuschreiben der Adresstabellen der Datensätze erzwingen (verlangsamt die Komprimierung). Beachten Sie, dass in diesem Fall die Nummern der Datensätze neu geschrieben werden. Übergeben Sie nur diese Option, aktiviert 4D automatisch die Option 'Datensätze aktualisieren'.
Create process	Lange Ganzzahl	32768	Ist diese Option übergeben, erfolgt das Komprimieren asynchron und Sie müssen die Ergebnisse über die Callback Methode verwalten. 4D zeigt keinen Ablaufbalken an (das ist bei Verwenden der Callback Methode möglich). Die Systemvariable OK wird auf 1 gesetzt, wenn der Prozess korrekt gestartet wurde, in allen anderen Fällen auf 0. Ist diese Option nicht übergeben, wird die Systemvariable OK auf 1 gesetzt, wenn die Komprimierung korrekt ausgeführt wurde, sonst auf 0.
Do not create log file	Lange Ganzzahl	16384	Dieser Befehl erstellt generell ein Logbuch im XML Format. Mit dieser Option wird kein Logbuch angelegt.
Timestamp log file name	Lange Ganzzahl	262144	Ist diese Option übergeben, zeigt der Name des generierten Logbuchs Datum und Uhrzeit seiner Erstellung und ersetzt dann folglich nicht das vorige Logbuch. Standardmäßig hat der Name eines Logbuchs keinen Zeitstempel und das vorige Logbuch wird überschrieben.
Update records	Lange Ganzzahl	65536	Neuschreiben aller Datensätze gemäß der aktuellen Definition der Felder in der Struktur erzwingen.

- Mit dem Parameter *Methode* setzen Sie eine Callback Methode, die während der Komprimierung regelmäßig aufgerufen wird, wenn die Konstante Create process übergeben wurde. Andernfalls wird die Callback Methode nie aufgerufen. Weitere Informationen dazu finden Sie unter dem Befehl **VERIFY DATA FILE**.
Existiert die Callback Methode nicht in der Datenbank, wird ein Fehler generiert und die Systemvariable OK wird auf 0 (Null) gesetzt.

Die Funktion **Compact data file** erstellt standardmäßig ein Logbuch im XML Format. (Haben Sie die Option Do not Create log file übergeben, siehe Parameter *Option*.) Das Logbuch liegt im Ordner **Logs** der aktuellen Datenbank und übernimmt den Namen der Strukturdatei. Lautet diese z.B. "myDB.4dd", hat das Logbuch den Namen "myDB_Compact_Log.xml".

Haben Sie die Option Timestamp log file name übergeben, zeigt der Name des Logbuchs auch Datum und Uhrzeit der Erstellung im Format "YYYY-MM-DD HH-MM-SS", der Name im obigen Beispiel lautet dann: "myDB_Compact_Log_2015-09-27 15-20-35.xml". Das bedeutet, dass ein neues Logbuch nicht das vorige ersetzt und dann nicht benötigte Dateien per Hand entfernt werden müssen.
Unabhängig von der gewählten Option wird der Pfad des generierten Logbuchs nach Ausführung des Befehls in der Systemvariablen *Document* zurückgegeben.

Beispiel

Folgendes Beispiel (Windows) führt die Komprimierung einer Datendatei aus:

```
$structFile:=Structure file
$dataFile:="C:\Databases\Invoices\January\Invoices.4dd"
$origFile:="C:\Databases\Invoices\Archives\January\"
$archFolder:=Compact data file($structFile;$dataFile;$origFile;0)
```

Systemvariablen und Mengen

Bei korrekt ausgeführter Komprimierung wird die Systemvariable OK auf 1 gesetzt; sonst auf 0 (Null). Wurde ein Logbuch generiert, wird sein kompletter Pfadname in der Systemvariable *Document* zurückgegeben.

COMPONENT LIST

COMPONENT LIST (KomponenteArray)

Parameter	Typ		Beschreibung
KomponenteArray	Array Text	⇐	Namen der Komponenten

Beschreibung

COMPONENT LIST definiert und füllt das Array *KomponenteArray* mit den Namen der vom 4D Programm geladenen Komponenten für die aktuelle Host Datenbank.

Beim Öffnen einer Datenbank lädt 4D die gültigen Komponenten aus dem bzw. den Ordnern *Components*:

- Ordner *Components* neben der Strukturdatei (sofern vorhanden)
- Ordner *Components* neben der ausführbaren 4D Anwendungsdatei

Achtung: Ist die gleiche Komponente an beiden Stellen vorhanden, lädt 4D nur die Komponente neben der Strukturdatei.

Sie können diesen Befehl von der Host Datenbank oder einer Komponente aus aufrufen. Verwendet die Datenbank keine Komponenten, wird das Array *KomponenteArray* leer zurückgegeben.
Die Namen der Komponenten sind die Namen der Strukturdateien der Matrix Datenbanken (.4db, .4dc oder .4dbase). Mit diesem Befehl können Sie Architekturen und modulare Oberflächen einrichten, die je nach Komponente zusätzliche Funktionalitäten bieten.

Weitere Informationen zu 4D Komponenten finden Sie im Handbuch *4D Designmodus* im Kapitel **4D Komponenten entwickeln und installieren**.

CREATE DATA FILE

CREATE DATA FILE (Zugriffspfad)

Parameter	Typ		Beschreibung
Zugriffspfad	String	➡	Name oder kompletter Zugriffspfad der zu erstellenden Datendatei

Beschreibung

Der Befehl **CREATE DATA FILE** erstellt eine neue Datendatei auf der Festplatte und ersetzt die von der 4D Anwendung geöffnete Datendatei „on-the-fly".

Dieser Befehl funktioniert wie der Befehl **OPEN DATA FILE**. Einziger Unterschied ist, dass die neue im Parameter *Zugriffspfad* gesetzte Datendatei direkt nach dem Wiederöffnen der Struktur erstellt wird.

Vor dem Start der Operation prüft der Befehl, dass der angegebene Zugriffspfad nicht einer vorhandenden Datei entspricht.

4D Server: Ab Version 13 können Sie diesen Befehl mit 4D Server ausführen. In diesem Kontext ruft er vor dem Erstellen der angegebenen Datei intern **QUIT 4D** auf dem Server auf und zeigt auf jedem remote Rechner ein Dialogfenster mit der Meldung an, dass der Server gerade verlassen wird.

Data file

Data file {(Segment)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Segment	Lange Ganzzahl	⇒	überholt, nicht verwenden
Funktionsergebnis	String	⇐	Kompletter Name der Datendatei der Datenbank

Beschreibung

Die Funktion **Data file** gibt den kompletten Namen der Datendatei für die Datenbank zurück, mit der Sie gerade arbeiten.

Ab 4D Version 11 werden Datensegmente nicht mehr unterstützt. Der Parameter *Segment* wird ignoriert und sollte nicht mehr verwendet werden.

Windows
Arbeiten Sie zum Beispiel mit der Datenbank *MeineCDs* aus dem Volume G in *\DOCS\MeineCDs*, gibt die Funktion beim Aufrufen der Datendatei zurück: *G:\DOCS\MeineCDs\MeineCDs.4DD* (wenn Sie beim Erstellen der Datenbank die Standardposition und -benennung von 4D übernommen haben).

Macintosh
Arbeiten Sie zum Beispiel mit der Datenbank *MeineCDs* aus dem Ordner *Dokumente:MeineCDsƒ:* auf der Festplatte Macintosh HD, gibt die Funktion beim Aufrufen der Datendatei zurück: *Macintosh HD:Dokumente:MeineCDsƒ:MeineCDs.data* (wenn Sie beim Erstellen der Datenbank die Standardposition und -benennung von 4D übernommen haben).

WARNUNG: Der Befehl gibt in 4D im remote Modus nur den Namen der Datendatei zurück und nicht den langen Namen.

Get 4D file

Get 4D file (Datei {; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Datei	Lange Ganzzahl	➡	Dateityp
*	Operator	➡	Datei der Host Datenbank
Funktionsergebnis	String	⬅	Pfadname zur 4D Datei

Beschreibung

Die Funktion **Get 4D file** gibt den Pfadnamen der Datei 4D Umgebung zurück, angegeben im Parameter *Datei*. Der Pfad wird in der Syntax des Systems zurückgegeben.

Mit dieser Funktion erhalten Sie den aktuellen Pfadnamen bestimmter Dateien, deren Name oder Speicherort vom Kontext der Anwendung abhängen kann. Sie ist auch hilfreich zum Schreiben von generischem Code, der unabhängig von der 4D Version oder dem Betriebssystem ist.

In *Datei* übergeben Sie einen Wert für die Datei, deren vollen Pfadnamen Sie erhalten wollen. Sie können eine der folgenden Konstanten unter dem Thema **4D Umgebung** übergeben:

Konstante	Typ	Wert	Kommentar
Backup configuration file	Lange Ganzzahl	1	Datei Backup.xml, im Ordner Preferences/Backup neben der Strukturdatei der Anwendung gespeichert
Backup log file	Lange Ganzzahl	13	Logbuch des aktuellen Backup. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde kein Logbuch angelegt oder existiert es nicht, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Build application log file	Lange Ganzzahl	14	Aktuelles Logbuch im xml Format des Application Builder. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde kein Logbuch angelegt oder existiert es nicht, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Compacting log file	Lange Ganzzahl	6	Logbuch der letzten Komprimierung der Anwendung, die mit dem Befehl **Compact data file** oder über das Maintenance und Security-Center (MSC) ausgeführt wurde. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde kein Logbuch angelegt oder existiert es nicht, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Debug log file	Lange Ganzzahl	12	Mit dem Befehl **SET DATABASE PARAMETER**(Debug log recording) erstelltes Logbuch. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde kein Debug Loguch angelegt oder existiert es nicht, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Diagnostic log file	Lange Ganzzahl	11	Mit dem Befehl **SET DATABASE PARAMETER**(Diagnostic log recording) erstelltes Logbuch. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde keine Diagnose ausgeführt oder existiert kein Logbuch, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
HTTP debug log file	Lange Ganzzahl	9	Mit dem Befehl **WEB SET OPTION**(Web debug log) erstelltes Logbuch. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde kein Debug Logbuch angelegt oder existiert es nicht, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Last backup file	Lange Ganzzahl	2	Letzte Backup Datei mit Namen <baseName>[bkpNum].4BK, an einem eigenen Ort gespeichert
Repair log file	Lange Ganzzahl	7	Protokoll der ausgeführten Reparaturen der Datenbank im Maintenance und Security Center (MSC). Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde keine Reparatur ausgeführt oder existiert kein Logbuch, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Request log file	Lange Ganzzahl	10	Logbuch mit standardmäßigen Client-/Server Anfragen (ohne Web Anfragen). das mit **SET DATABASE PARAMETER**(4D Server log recording) oder **SET DATABASE PARAMETER**(Client log recording) erstellt wurde. Bei Ausführung auf dem Server wird das Server Log zurückgegeben (gespeichert im Ordner Logs auf dem Server). Bei Ausführung auf dem Client wird das Client Log zurückgegeben (gespeichert im lokalen Ordner Logs des Clients). Existiert kein Logbuch, wird ein leerer Pfad zurückgegeben.
User settings file	Lange Ganzzahl	3	Datei Settings.4DSettings für alle Datendateien, falls aktiviert, im Ordner Preferences neben der Strukturdatei der Anwendung gespeichert
User settings file for data	Lange Ganzzahl	4	Datei Settings.4DSettings für aktuelle Datendatei, im Ordner Preferences neben der Datendatei der Anwendung gespeichert
Verification log file	Lange Ganzzahl	5	Logbücher, die mit den Befehlen **VERIFY CURRENT DATA FILE** und **VERIFY DATA FILE** oder über das Maintenance und Security Center (MSC) erstellt wurden. Werden im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde keine Überprüfung ausgeführt oder existiert kein Logbuch, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Web request log file	Lange Ganzzahl	8	Mit dem Befehl **WEB SET OPTION**(Web log recording) erstelltes Logbuch. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde kein Logbuch angelegt oder existiert es nicht, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.

Wird diese Funktion in einer Komponente aufgerufen, übergeben Sie den optionalen Parameter *, um den Pfad *Datei* der Host Datenbank zu erhalten. Lassen Sie in diesem Fall den Parameter * weg, wird immer ein leerer String zurückgegeben.

Bei den Konstanten User settings file for data und User settings file wird der Pfad nur zurückgegeben, wenn im Dialogfenster Datenbank-Eigenschaften die Sicherheitsoption **Benutzer-Einstellungen in externer Datei** markiert ist. Weitere Informationen dazu finden Sie im Abschnitt **Modus Benutzer Einstellungen aktivieren**.

Beispiel

Den Pfad der letzten Backup Datei erhalten:

```
C_TEXT($path)
$path:=Get 4D file(Last backup file)
 // $path = zum Beispiel "C:\Backups\Countries\Countries[0025].4BK"
```

Get 4D folder

Get 4D folder {(Ordner {; *})} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Ordner	Lange Ganzzahl	➡	Ordnertyp (ohne Angabe = aktiver 4D Ordner)
*	Operator	➡	Gibt den Ordner der Host Datenbank zurück
Funktionsergebnis	String	⬅	Pfadname zu 4D Ordner

Beschreibung

Die Funktion **Get 4D folder** gibt den Pfadnamen zum aktiven 4D Ordner zurück, bzw. zum Ordner der 4D Umgebung, wenn der Parameter *Ordner* übergeben wurde. Über diese Funktion erhalten Sie den aktuellen Pfadnamen der Ordner, welche die 4D Anwendung verwendet. Mit dieser Funktion stellen Sie sicher, dass Ihr Code auf jeder Plattform funktioniert, die mit einem lokalisierten System läuft.

In *Ordner* übergeben Sie eine der folgenden Konstanten unter dem Thema **4D Umgebung**:

Konstante	Typ	Wert
4D Client database folder	Lange Ganzzahl	3
Active 4D Folder	Lange Ganzzahl	0
Current resources folder	Lange Ganzzahl	6
Data folder	Lange Ganzzahl	9
Database folder	Lange Ganzzahl	4
Database folder Unix syntax	Lange Ganzzahl	5
HTML Root folder	Lange Ganzzahl	8
Licenses folder	Lange Ganzzahl	1
Logs folder	Lange Ganzzahl	7

Es folgt die Beschreibung der einzelnen Ordner

Definitionen

- *{Festplatte}* ist die Platte, auf der das Betriebssystem installiert ist.
- *Benutzername* ist der Name des Benutzers, der die aktuelle Sitzung geöffnet hat.

Aktiver 4D Ordner

Die 4D Umgebung verwendet einen spezifischen Ordner zum Speichern folgender Informationen:

- Dateien Preferences der Anwendungen der 4D Umgebung
- Datei Shortcuts.xml (eigene Tastenkürzel)
- Ordner Macros v2 (Makro Befehle des Methodeneditors)
- Ordner Favorites v1x, z.B. Favorites v14 (Pfadnamen für lokale und remote Datenbanken, die geöffnet wurden)

Mit den 4D Hauptprogrammen (4D und 4D Server) lautet der aktive 4D Ordner **4D** und liegt standardmäßig an folgender Stelle:

- Unter Windows 7 und höher: *{Festplatte}\Users\<Benutzername>\AppData\Roaming\4D*
- Auf OS X: *{Festplatte}:Benutzer:<Benutzername>:Library:Application Support:4D*

Ab 4D v13 liegt der aktive 4D Ordner bei einer Anwendung mit einkompilierter 4D Volume Desktop an folgender Stelle:

- Unter Windows 7 und höher: *{Festplatte}\Users\<Benutzername>\AppData\Roaming\<Anwendungsname>*
- Auf OS X: *{Festplatte}:Benutzer:<Benutzername>:Library:Application Support:<Anwendungsname>*

Ordner Licenses

Der Ordner enthält die Lizenzdateien des Rechners. Er liegt an folgender Stelle:

- Unter Windows 7 und höher: {Festplatte}:\ProgramData\4D\Licenses
- Auf OS X: *{Festplatte}:Library:Application Support:4D:Licenses*

Hinweise:

- Bei einer Anwendung mit einkompilierter 4D Volume Desktop ist der Ordner Licenses im Paket der Anwendung enthalten.
- Kann der Ordner Licenses nicht im System erstellt werden, weil die Authorisation dazu fehlt, wird er an folgender Stelle angelegt:
  - Unter Windows 7 und höher: *{Festplatte}:\Users\<Benutzername>\AppData\Roaming\4D\Licenses*
  - Auf OS X: *{Festplatte}:Users:<Benutzername>:Library:Application Support:4D:Licenses*

Ordner Data

Pfad des Ordners mit der aktuellen Datendatei. Der Pfadname wird in der Standardsyntax der aktuellen Plattform angegeben.

Ordner 4D Client Database (Client Rechner)

4D Datenbank Ordner, der auf jedem Client Rechner angelegt wird, zum Speichern von Dateien und Ordnern, die zur Datenbank gehören (Ressourcen, Plug-Ins, Ordner Resources, etc.).
Der Ordner liegt auf dem Client-Rechner an folgender Stelle:

- Unter Windows 7 und höher: *{Festplatte}:\Users\<Benutzername>\AppData\Local\4D\<DatenbankName_Adresse>*
- Auf OS X: *{Festplatte}:Users:<Benutzername>:Library:Caches:4D:<DatenbankName_Adresse>*

Ordner Database

Ordner mit der Strukturdatei der Datenbank. Der Pfadname wird in der standardmäßigen Syntax der aktuellen Plattform dargestellt.
In der Client-Anwendung entspricht diese Konstante exakt der bisherigen Konstante 4D Client Database Folder: Sie gibt den Pfadnamen des lokal erstellten Ordners zurück.

Ordner Database Unix Syntax

Ordner mit der Strukturdatei der Datenbank. Diese Konstante bezeichnet denselben Ordner wie die vorige, der Pfadname wird jedoch in der Unix Syntax (Posix) vom Typ /Users/... dargestellt. Diese Syntax wird hauptsächlich mit dem Befehl **LAUNCH EXTERNAL PROCESS** unter OS X zurückgegeben.

Aktueller Ordner Resources

Ordner Resources der Datenbank. Dieser Ordner enthält zusätzliche Elemente (Bilder, Text) für die Datenbankoberfläche. Eine Komponente kann einen eigenen Ordner Resources haben. Der Ordner Resources liegt neben der Struktur der Datenbank.
Im Client/Server-Betrieb kann dieser Ordner zur Übertragung eigener Elemente zwischen Server- und Client-Rechner dienen. Der Inhalt dieses Ordners wird automatisch aktualisiert, wenn sich der Client-Rechner anmeldet. Im Client/Server-Betrieb werden alle diesem Ordner zugewiesenen Referenz-Mechanismen unterstützt(Ordner .lproj, XLIFF, Bilder, etc.) 4D bietet darüberhinaus verschiedene Tools, um diesen Ordner dynamisch zu verwalten und zu aktualisieren, insbesondere einen Ressourcen Explorer.

Hinweis: Existiert der Ordner Resources nicht für die Datenbank, wird er angelegt, wenn Sie die Funktion **Get 4D folder** mit der Konstanten CurrentResources Folder ausführen.

Ordner Logs

Ordner Logs der Datenbank mit den Logbüchern der aktuellen Datenbank. Er liegt auf derselben Ebene wie die Strukturdatei und enthält folgende Logbücher:

- Datenbankkonvertierung,
- Web Server Anfragen,
- Überprüfung und Reparatur der Daten,
- Überprüfung und Reparatur der Struktur,
- Journal zu Backup/Wiederherstellen Aktivitäten,
- Debugging der Befehle und Funktionen,
- 4D Server Anfragen (auf Client-Rechnern und auf dem Server generiert).

Hinweis: Existiert der Ordner *Logs* nicht für die Datenbank, wird er angelegt, wenn Sie die Funktion **Get 4D folder** mit der Konstanten Logs Folder ausführen.

Ordner HTML Root

Aktueller HTML Root Ordner der Datenbank zurück. Der zurückgegebene Pfadname wird in der standardmäßigen Syntax der aktuellen Plattform dargestellt. Es ist der Ordner, in dem der 4D Web Server nach den angefragten Web Dateien und Seiten sucht. Der Ordner heißt standardmäßig WebFolder und liegt neben der Strukturdatei (oder eine lokale Kopie bei 4D im remote Modus). Sie können diesen Speicherort in den Einstellungen der Datenbank auf der Seite Web>Konfiguration oder dynamisch über den Befehl **WEB SET ROOT FOLDER** verändern.
Wird die Funktion **Get 4D folder** über ein Remote 4D aufgerufen, wird der Pfad des Remote Rechners und nicht der von 4D Server zurückgegeben.

Der optionale Parameter * ist hilfreich beim Arbeiten mit Komponenten. Damit können Sie die Datenbank (Host oder Komponente) bestimmen, für welche Sie den Pfadnamen des Ordners erhalten möchten. Dieser Parameter gilt nur für Database Folder, Database Folder Unix Syntax und Current Resources folder. In allen anderen Fällen hat er keine Auswirkung.
Wird die Funktion von einer Komponente aus aufgerufen:

- Ist der Parameter * übergeben, gibt die Funktion den Pfadnamen der Host Datenbank zurück.
- Ist der Parameter * nicht übergeben, gibt die Funktion den Pfadnamen des Ordners *Components* zurück.
  Der zurückgegebene Datenbankordner (Database Folder und Database Folder Unix Syntax) ist unterschiedlich, je nach der Art der Komponentenarchitektur:
  - Bei einem Ordner .4dbase/package gibt die Funktion den Pfadnamen des Ordners .4dbase/package zurück,
  - Bei einer Datei .4db oder .4dc gibt die Funktion den Pfadnamen des Ordners *Components* zurück,
  - Bei einem Alias oder einer Verknüpfung gibt die Funktion den Pfadnamen des Ordners mit der Original Matrix Datenbank zurück. Das Ergebnis ist unterschiedlich, je nach dem Format dieser Datenbank (Ordner .4dbase /package oder Datei .4db/.4dc), wie oben beschrieben.
  Wird der Befehl von der Host Datenbank aus aufgerufen, gibt er immer den Pfadnamen des Host Datenbankordners zurück, unabhängig ob der Parameter * übergeben ist oder nicht.

Beispiel 1

Sie wollen während dem Starten einer Datenbank im Einzelplatzbetrieb in eine Datei aus dem 4D Ordner eigene Einstellungen laden oder erstellen. Ihr Code in der **Datenbankmethode On Startup** könnte folgendermaßen aussehen:

```
MAP FILE TYPES("PREF";"PRF";"Preferences file")
  ` Setze Dateityp PREF Mac  OS auf .PRF Windows Dateierweiterung
$vsPrefDocName:=Get 4D folder+"MyPrefs"
  ` Erstelle Pfadname zur Datei Voreinstellungen
  ` Prüfe, ob die Datei existiert
If(Test path name($vsPrefDocName+(".PRF"*Num(On Windows)))#Is a document)
    $vtPrefDocRef:=Create document($vsPrefDocName;"PREF")
  ` Falls nein, erstelle sie
Else
    $vtPrefDocRef:=Open document($vsPrefDocName;"PREF")  ` Falls ja, öffne sie
End if
```

```
If(OK=1)
  ` Inhalt der Prozessdokumente
   CLOSE DOCUMENT($vtPrefDocRef)
Else
  ` Verwalte Fehler
End if
```

Beispiel 2

Dieses Beispiel zeigt die Verwendung der Konstanten Database Folder Unix Syntax unter Mac OS, um den Inhalt des Datenbankordners anzuzeigen:

```
$posixpath:="\""+Get 4D folder(Database folder Unix syntax)+"\""
$myfolder:="ls -l "+$posixpath
$in:=""
$out:=""
$err:=""
LAUNCH EXTERNAL PROCESS($myfolder;$in;$out;$err)
```

Hinweis: Auf Mac OS müssen Sie die Pfadnamen in Anführungszeichen setzen, wenn sie Datei- oder Ordnernamen mit Leerzeichen enthalten. Über das Sonderzeichen "\"oder die Anweisung *Char(Double quote)* können Sie Anführungszeichen in den String einfügen.

Systemvariablen und Mengen

Ist der Parameter *Ordner* ungültig oder wird der Pfadname leer zurückgegeben, wird die Systemvariable *OK* auf 0 (Null) gesetzt.

Get database localization

Get database localization {(Sprache)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Sprache	Lange Ganzzahl	➡	Sprachtyp
Funktionsergebnis	String	⮌	Aktuelle Sprache der Datenbank

Beschreibung

Die Funktion **Get database localization** gibt die aktuelle Sprache der Datenbank zurück, angegeben in *Sprache* und definiert durch den Standard RFC 3066. Die Funktion gibt z.B. "en" für Englisch, "es" für Spanisch zurück. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus* im Abschnitt **Anhang C: XLIFF Architektur**.

Sie können in der Anwendung verschiedene Sprachkonfigurationen gleichzeitig verwenden. Zum Definieren der Einstellungen übergeben Sie im Parameter *Sprache* eine der folgenden Konstanten unter dem Thema **4D Umgebung**.

Konstante	Typ	Wert	Kommentar
Current localization	Lange Ganzzahl	1	Aktuelle Sprache der Anwendung: Standardsprache oder Sprache, die über den Befehl **SET DATABASE LOCALIZATION** gesetzt wird.
Default localization	Lange Ganzzahl	0	Sprache, die 4D automatisch beim Starten setzt gemäß dem Ordner Resources und der Systemumgebung (nicht änderbar).
Internal 4D localization	Lange Ganzzahl	3	Sprache, die 4D zum Sortieren und für Textvergleiche verwendet (definiert in den Einstellungen der Anwendung).
User system localization	Lange Ganzzahl	2	Sprache, die der aktuelle Systembenutzer setzt.

Lassen Sie diesen Parameter weg, gibt die Funktion die Standardsprache (0) zurück.

Die aktuelle Sprache der Datenbank lässt sich zum Setzen des Ordners .lproj verwenden. Dort sucht das Programm dann nach den lokalisierten Einträgen der Datenbank. 4D bestimmt automatisch die aktuelle Sprache beim Starten der Datenbank gemäß dem Inhalt des Ordners **Resources** und der Systemumgebung. Dazu lädt 4D zuerst den Ordner .lproj der Datenbank, der zu der Sprache passt. Es gilt folgende Priorität:

1. Sprache des Systems (auf Mac OS lassen sich verschiedene Sprachen nach Priorität setzen, 4D verwendet diese Einstellungen)
2. Sprache der 4D Anwendung.
3. Englisch
4. Erste Sprache, die im Ordner **Resources** gefunden wird

Hinweis: Hat die Anwendung keinen Ordner .lproj, 4D gilt folgende Priorität: 1. Sprache des Systems, 2. Englisch (wenn die Systemsprache nicht identifiziert werden kann.)

Get database measures

Get database measures {(Optionen)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Optionen	Objekt	➡	Optionen für zurückgegebene Information
Funktionsergebnis	Objekt	⮌	Objekt mit Datenbankinformation

Beschreibung

Die Funktion **Get database measures** gibt detaillierte Informationen über Ereignisse der 4D Datenbank Engine zurück. Dies umfasst Lese- und Schreibzugriffe auf die Festplatte oder den Memory Cache, sowie die Verwendung von Indizes, Suchen und Sortieren in der Datenbank.

Get database measures gibt ein einzelnes Objekt mit allen relevanten Messwerten zurück. Im Parameter *Optionen* können Sie die gewünschten Informationen genauer definieren.

Übersicht über das zurückgegebene Objekt

Das zurückgegebene Objekt enthält eine einzelne Eigenschaft mit Namen "DB" mit folgender Grundstruktur:

```
{
    "DB": {
        "diskReadBytes": {…},
        "cacheReadBytes": {…},
        "cacheMissBytes": {…},
        "diskWriteBytes": {…},

        "diskReadCount": {…},
        "cacheReadCount": {…},
        "cacheMissCount": {…},
        "diskWriteCount": {…},

        "dataSegment1": {…},
        "indexSegment": {…},

        "tables": {…},
        "indexes": {…}

    }
}
```

Dieses Objekt enthält bis zu acht elementare Eigenschaften mit Messwerten ("diskReadBytes", "cacheReadBytes", "cacheMissBytes", "diskWriteBytes", "diskReadCount", "cacheReadCount", "cacheMissCount", "diskWriteCount") gültig für die gesamte Datenbank, sowie zusätzlich eine weitere Unterteilung in "dataSegment1", "indexSegment", "tables", "indexes". Diese Eigenschaften haben wiederum selbst die elementaren Eigenschaften, jeweils gültig für die entsprechende Untergruppe. Diese fein abgestimmte Unterteilung erlaubt die Analyse der Datenzugriffe je nach Bedarf vom Gesamtsystem bis hin zu Zugriffen auf einzelne Tabellen oder Indizes.

Hinweis: Eine Eigenschaft erscheint nur im Objekt, wenn sie Inhalt enthält. Eine Eigenschaft ohne Inhalt wird nicht in das Objekt aufgenommen. Wurde die Anwendung z.B. im Nur-Lesen Modus geöffnet und keine Indizes verwendet, enthält das zurückgegebene Objekt nicht die Messwerte "diskWriteBytes", "diskWriteCount", "indexSegment" und "indexes".

Elementare Eigenschaften

Elementare Eigenschaften gibt es auf verschiedenen Ebenen im Objekt DB. Sie enthalten dieselbe Informationsart, aber für unterschiedliche Datenbankobjekte. Hier die Beschreibung der einzelnen Eigenschaften:

Name	zurückgegebene Information
diskReadBytes	von der Festplatte gelesene Bytes
cacheReadBytes	vom Cache gelesene Bytes
cacheMissBytes	im Cache nicht enthaltene Daten in Bytes
diskWriteBytes	auf die Festplatte geschriebene Bytes
diskReadCount	Anzahl der Lesezugriffe von der Festplatte
cacheReadCount	Anzahl der Lesezugriffe vom Cache
cacheMissCount	im Cache nicht enthaltene Daten; Anzahl Zugriffe
diskWriteCount	Schreibzugriffe auf die Festplatte

Alle acht elementaren Eigenschaften enthalten dieselbe Objektunterstruktur. Hier ein Beispiel:

```
"diskReadBytes": {      "value": 33486473620,      "history": [          // optional          {"value": 52564,"time": -1665},          {"value": 54202,"time": -1649},
            …       ] }
```

- **"value"** (Zahl): Die Eigenschaft "value" enthält eine Zahl, die entweder eine Anzahl Bytes oder eine Anzahl Zugriffe angibt. Dieser Wert ist im allgemeinen die Summe der Werte des Objekts "history" (auch wenn das Objekt "history" nicht vorhanden ist).

- "**history**" (Array der Objekte): Das Array "history" ist eine Zusammenstellung von Ereigniswerten, gruppiert nach Sekunden. Die Eigenschaft "history" gibt es nur, wenn sie über den Parameter *Optionen* angefordert wurde (siehe unten). Dieses Array enthält maximal 200 Einträge. Jedes Element des Array ist selbst ein Objekt mit den beiden Eigenschaften "value" und "time".
  - "value" (Zahl): Anzahl Bytes oder Zugriffe, die während der Zeitspanne, definiert in der zugeordneten Eigenschaft "time", verwaltet wird.
  - "time" (Zahl): Anzahl Sekunden, die seit Aufrufen der Funktion verstrichen ist. Im obigen Beispiel ("time": -1649) bedeutet die Zahl vor 1649 Sekunden, oder genauer zwischen den Sekunden 1649 und 1650. In dieser Zeitspanne von einer Sekunde wurden 54.202 Bytes auf die Festplatte gelesen.
    Das Array "history" enthält keine sequentiellen Werte (-1650,-1651,-1652, etc.). Der vorige Wert ist -1665, d.h., in den 15 Sekunden zwischen 1650 und 1665 wurde nichts von der Festplatte gelesen.
    Hinweis: Das Array enthält nur vorhandene Informationen. Werte mit Null sind nicht aufgeführt.
    Da die Größe des Array auf 200 begrenzt ist, ist "history" bei intensiver Nutzung der Datenbank (jede Sekunde wird etwas auf die Festplatte gelesen) 200 Sekunden lang. Passiert dagegen nur wenig, gibt es z.B. nur alle 3 Minuten ein Ereignis, kann "history" 600 Minuten (3*200) enthalten.
    Hierzu ein Beispiel:

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 und indexSegment

Die Eigenschaften "dataSegment1" und "indexSegment" enthalten bis zu vier der elementaren Eigenschaften (sofern verfügbar):

```
"dataSegment1": {
    "diskReadBytes": {…},
    "diskWriteBytes": {…},
    "diskReadCount": {…},
    "diskWriteCount": {…}
    },
"indexSegment": {
    "diskReadBytes": {…},
    "diskWriteBytes": {…},
    "diskReadCount": {…},
    "diskWriteCount": {…}
    }
```

Diese Eigenschaften enthalten dieselbe Information wie die elementaren Eigenschaften, jedoch aufgeschlüsselt nach Datei der Datenbank:

- "dataSegment1" ist die Datendatei .4dd auf der Festplatte
- "indexSegment" ist die Indexdatei .4dx auf der Festplatte

Ihr Objekt könnte z.B. folgendermaßen aussehen:

```
{ "DB": { "diskReadBytes": {  "value": 718260  }, "diskReadCount": {  "value": 229  }, "dataSegment1": {  "diskReadBytes": {  "value": 679092  },
    "diskReadCount": {  "value": 212  }  }, "indexSegment": {  "diskReadBytes": {  "value": 39168  },  "diskReadCount": {  "value": 17  } }
```

Die Werte werden so berechnet:

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

Tabellen

Die Eigenschaft "tables" enthält alle Tabellen, auf die seit Öffnen der Datenbank im Lese- oder Schreibmodus zugegriffen wurde. Der Name der Eigenschaft leitet sich vom jeweiligen Tabellennamen ab, zum Beispiel:

```
"tables": {  "Employees": {…)  "Companies": {…)  }
```

Jedes Objekt "table" enthält 10 Eigenschaften:

- Die ersten acht Eigenschaften sind die *elementaren Eigenschaften* (siehe oben) mit Werten zur betreffenden Tabelle.
- Die beiden anderen Eigenschaften "records" und "blobs" haben dieselben acht Eigenschaften, jedoch nur für bestimmte Feldtypen:
  - Die Eigenschaft "records" betrifft alle Feldtypen der Tabelle (Strings, Datum, Zahlenarten, etc.) mit Ausnahme von Text, Bild und Blobs
  - Die Eigenschaft "blobs" betrifft die Felder vom Typ Text, Bild und Blob der Tabelle.

Je nach Such- oder Sortierläufen in der Tabelle können zusätzliche Eigenschaften "fields" und "queries" vorhanden sein:

- Die Eigenschaft "fields" enthält soviel Attribute "field name" (die auch Unterobjekte sind), wie Felder für Suchen oder Sortieren verwendet werden.
  Jedes Objekt Feldname enthält:
  - Ein Objekt "queryCount" (mit oder ohne Chronik, abhängig vom Parameter *Optionen*), wenn eine Suche mit diesem Feld ausgeführt wurde.
  - und/oder ein Objekt "sortCount" (mit oder ohne Chronik, abhängig vom Parameter *Optionen*), wenn eine Sortierung mit diesem Feld ausgeführt wurde.

  Dieses Attribut basiert nicht auf der Verwendung von Indizes; alle Such- und Sortierläufe werden berücksichtigt.
  Beispiel: Seit Starten der Datenbank wurden mehrere Such- und Sortierläufe mit den Feldern *CompID*, *Name* und *FirstName* ausgeführt. Das zurückgegebene Objekt enthält das folgende Unterobjekt "fields" (*Optionen* sind mit Pfad und ohne Chronik):


```
{ "DB": { "tables": { "Employees": { "fields": { "CompID": { "queryCount": { "value": 3 } }, "Name": { "queryCount": { "value": 1 }, "sortCount": { "value": 3 } }, "FirstName": { "sortCount": { "value": 2 } } (...)
```


  Hinweis: Das Attribut "fields" wird nur erstellt, wenn in der Tabelle eine Suche oder Sortierung durchgeführt wurde; andernfalls ist dieses Attribut nicht vorhanden.
- "queries" ist ein Array mit Objekten, das jede in der Tabelle durchgeführte Suche beschreibt. Jedes Element des Array enthält drei Attribute:
  - "queryStatement" (String): Suchstring (enthält Feldnamen, aber keine Werte der Suchkriterien). Zum Beispiel: "(Companies.PK_ID != ?)"
  - "queryCount" (Objekt):
    - "value" (Zahl): wieviele Male die Suchanweisung ausgeführt wurde, unabhängig von den Werten der Suchkriterien.
    - "history" (Array mit Objekten) (wenn in *Optionen* angefordert): "value" und "time" standardmäßige Chronik-Eigenschaften
  - "duration" (Objekt) wenn "value" >0)
    - "value" (Zahl): Anzahl Millisekunden
    - "history" (Array mit Objekten) (wenn in *Optionen* angefordert): "value" und "time" standardmäßige Chronik-Eigenschaften

  Beispiel: Seit dem Start der Datenbank wurde in der Tabelle *Employees* eine Suche durchgeführt (*Optionen* sind mit Pfad und ohne Chronik):


```
{ "DB": { "tables": { "Employees": { "queries": [ { "queryStatement": "(Employees.Name == ?)", "queryCount": { "value": 1, "history": [ { "value": 1, "time": -2022 } ] }, "duration": { "value": 2, "history": [ { "value": 2, "time": -2022 } ] } }, (...)
```


  Hinweis: Das Attribut "queries" wird erstellt, wenn in der Tabelle mindestens eine Suche durchgeführt wurde.

Indizes

Dies ist das umfangreichste Objekt. Alle Tabellen, auf die über einen oder mehrere Indizes zugegriffen wurde, werden als Eigenschaften gespeichert, die verwendeten Indizes darin wieder als Eigenschaften aufgeführt. Volltextindexes erscheinen separat und an ihren Namen wird "*(Keyword)*" angehängt. Schließlich erscheint jede Eigenschaft Indexname jeweils wieder mit den acht elementaren Eigenschaften und, je nach Index-Verwendung seit Starten der Datenbank, mit bis zu vier Unterobjekten. Jedes dieser Unterobjekte existiert nur, wenn die dazugehörige Operation seit Starten der Datenbank an irgendeinem Punkt ausgeführt wurde.

Beispiel: Seit dem Starten der Datenbank wurden mehrere Indizes des Feldes [Employees]EmpLastName angefordert. Außerdem wurden in der Tabelle [Companies] 2 Datensätze erstellt und 16 gelöscht. Diese Tabelle hat das indizierte Feld "name". Sie wurde über dieses Feld auch durchsucht und sortiert. Als Ergebnis ergibt sich folgendes:


```
"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: {···} }
```


Parameter Optionen

Mit dem Parameter *Optionen* können Sie die zurückgegebene aktuelle Information anpassen. In *Optionen* übergeben Sie ein Objekt mit bis zu drei Eigenschaften: "withHistory", "historyLength" und "path".

Eigenschaft	Typ	Beschreibung
"withHistory"	Boolean	bei "true" erscheint die Chronik über die Funktion im zurückgegebenen Objekt; bei "false" enthält das zurückgegebene Objekt keine Chronik
"historyLength"	Zahl	Definiert die Größe des zurückgegebenen Array "history" in Sekunden (*).
"path"	string | string array	Kompletter Pfad einer bestimmten Eigenschaft oder Array mit kompletten Pfaden für alle spezifischen Eigenschaften, die Sie erhalten wollen. Übergeben Sie einen String, wird nur der entsprechende Wert im Objekt "DB" zurückgegeben (wenn der Pfad gültig ist). Beispiel "DB.tables.Employee.records.diskWriteBytes" (bei gültigen Pfaden). Übergeben Sie ein Array mit Strings, werden im Objekt "DB" alle entsprechenden Werte zurückgegeben. Beispiel: ["DB.tables.Employee.records.diskWriteBytes", "DB.tables.Employee.records.diskReadCount","DB.dataSegment1.diskReadBytes"]

(*) Wie oben beschrieben, wird der Verlauf nicht sekundenweise gespeichert, sondern nur nach relevanten Werten. Passiert ein paar Sekunden lang nichts, wird nichts gespeichert und im internen Verlauf des Array entsteht eine zeitliche Lücke. "time" kann z.B. -2, -4, -5, -10, -15, -30 mit den Werten 200, 300, 250, 400, 500,150 enthalten. Hat die Eigenschaft "historyLength" den Wert 600 (10 Minuten), enthält das zurückgegebene Array 0, -1, -2, -3 … -599 für Zeit, aber nur die Werte von -2, -4, -5, -10, -15, -30 werden gefüllt. Alle anderen Werte erhalten den Wert 0 (Null). Wie bereits beschrieben, ist die Begrenzung des internen Array die Größe (200) und nicht die Zeit. Folglich kann bei geringer Aktivität einer bestimmten Eigenschaft die älteste Zeit länger zurückliegen, z.B. -3600 für eine ganze Stunde. Das Array kann auch weniger als 200 Werte enthalten, wenn die Datenbank gerade gestartet wurde. Liegt dann die interne Zeitspanne der Chronik unter der angeforderten ODER erscheinen alle relevanten Werte bereits im zurückgegebenen Array, lautet der zurückgegebene Wert -1.

Beispiel: Die Datenbank wurde vor 20 Sekunden gestartet, die Chronik der Anfrage ist 60 Sekunden. Die zurückgegebenen Werte zwischen 0 und -20 werden mit Werten oder Nullen gefüllt, die anderen werden auf -1 gesetzt. Wird ein Wert "-1" zurückgegeben, bedeutet dies entweder, dass die Anfragezeit zu alt ist oder der Wert nicht mehr in der Chronik liegt, z.B. weil der 200. Eintrag erreicht ist bzw. ältere Werte entfernt wurden.

Hinweis: Das Filtern über "historyLength" verlangsamt deutlich die Abarbeitung des Befehls. Zur statistischen Auswertung belasteter Server ist es sinnvoller, das gesamte Objekt ohne Angabe von "historyLength" abzufragen und das Ergebnis offline auszuwerten.

Über Client/Server und Komponenten

Get database measures gibt ein gültiges Objekt mit relevanten Werten nur in folgendem Kontext zurück:

- bei 4D im lokalen Modus (bei Aufruf über eine Komponente wird Information über die Host Datenbank zurückgegeben)
- auf dem Server im Client/Server-Modus

Bei Aufruf über ein remote 4D bleibt das Objekt leer.
Benötigen Sie in remote 4D Information über die Datenbank auf dem Server, legen Sie einfach eine Methode an und aktivieren dafür die Option "auf Server ausführen".
Dieses Prinzip funktioniert auch für eine Komponente: Im lokalen Kontext von 4D gibt sie Information über die Host Datenbank zurück; im remote Kontext von 4D gibt sie Information über die Server Datenbank zurück.

Beispiel 1

Im zurückgegebenen Objekt die Chronik anzeigen:

```
C_OBJECT($param)
C_OBJECT($measures)
OB SET($param;"withHistory";True)
$measures:=Get database measures($param)
```

Beispiel 2

Nur die globale Anzahl der im Cache gelesenen Bytes anfordern ("cacheReadBytes"):

```
C_OBJECT($oStats)
C_OBJECT($oParams)
OB SET($oParams;"path";"DB.cacheReadBytes")
$oStats:=Get database measures($oParams)
```

Das zurückgegebene Objekt könnte folgendermaßen aussehen:

```
{    "DB": {        "cacheReadBytes": {            "value": 9516637        }    } }
```

Beispiel 3

Die Meßwerte für im Cache gelesene Bytes der letzten zwei Minuten anfordern:

```
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 ({Tabellenname ;} Selector {; StringWert}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Erhalten der Parameter, ohne Angabe Standardtabelle
Selector	Lange Ganzzahl	➡	Code des Parameters der Datenbank
StringWert	String	⬅	Stringwert des Parameters
Funktionsergebnis	Zahl	↩	Aktueller Wert des Parameters

Beschreibung

Die Funktion **Get database parameter** liest den aktuellen Wert der Parameter der 4D Datenbank. Ist der Wert eine Zeichenkette, wird er im Parameter *StringWert* zurückgegeben.

Der Parameter *Selector* bezeichnet den zu lesenden Parameter. 4D bietet unter dem Thema **Datenbankparameter** folgende vordefinierten Konstanten:

Konstante	Typ	Wert	Kommentar
Minimum Web process	Lange Ganzzahl	6	**Reichweite:** 4D lokal, 4D Server **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Werte: 0 -> 32 767 **Beschreibung:** Mindestanzahl der zu unterhaltenden Web Prozesse im nicht kontextuellen Modus mit 4D im lokalen Modus und 4D Server. Standardmäßig ist der Wert 0 (siehe unten).
Maximum Web process	Lange Ganzzahl	7	**Reichweite:** 4D lokal, 4D Server **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 -> 32,767 **Beschreibung:** Höchstanzahl der zu unterhaltenden Web Prozesse im nicht kontextuellen Modus mit 4D im lokalen Modus und 4D Server. Standardmäßig ist der Wert 10. Um den Web Server reaktivierbar zu machen, verzögert 4D im nicht-kontextuellen Modus die Web Prozesse um 5 Sekunden und verwendet sie wieder, um mögliche spätere HTTP Anfragen auszuführen. In Bezug auf die Performance ist dies vorteilhafter als pro Anfrage einen neuen Prozess zu erstellen. Sobald ein Web Prozess wieder verwendet wird, wird er erneut um 5 Sekunden verzögert, außer die Höchstanzahl der Web Prozesse ist bereits erreicht. Dann wird er abgebrochen. Erhält ein Web Prozess nicht binnen 5 Sekunden eine Anfrage, wird er abgebrochen. Mit diesen Parametern können Sie je nach Anzahl der Anfragen, nach verfügbarem Speicher, etc. einstellen, wie Ihr Web Server arbeitet.
_o_Web conversion mode	Lange Ganzzahl	8	**** *Selector deaktiviert* ****
_o_Database cache size	Lange Ganzzahl	9	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** - **Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen, stattdessen die Funktion **Get cache size** zu verwenden.
_o_4D Local mode scheduler	Lange Ganzzahl	10	**** *Dieser Selector ist überholt und darf nicht mehr verwendet werden.*
_o_4D Server scheduler	Lange Ganzzahl	11	**** *Dieser Selector ist überholt und darf nicht mehr verwendet werden.*
_o_4D Remote mode scheduler	Lange Ganzzahl	12	**** *Dieser Selector ist überholt und darf nicht mehr verwendet werden.*
4D Server timeout	Lange Ganzzahl	13	**Reichweite:** 4D Anwendung bei positivem *Wert* **Wird zwischen 2 Sitzungen beibehalten:** Ja bei positivem *Wert* **Mögliche Werte:** 0 -> 32 767 **Beschreibung:** Damit ändern Sie den Timeout-Wert des 4D Server. Der Standardwert für das Timeout des 4D Server wird auf dem Server Rechner in den Datenbank-Eigenschaften auf der Seite **Client-Server>IP Konfiguration** festgelegt. Mit dem Selector 4D Server Timeout legen Sie im dazugehörigen Parameter *Wert* ein neues Timeout in Minuten fest. Das ist besonders hilfreich bei Operationen auf dem Client, die den Rechner blockieren oder viel Zeit beanspruchen, z.B. wenn eine große Anzahl an Seiten gedruckt wird. Das kann bei einem kleinen Standardwert zu einem unerwartetem Timeout führen. Es gibt zwei Optionen: • Ein positiver Wert im Parameter *Wert* setzt ein globales und permanentes Timeout: Der neue Wert gilt für alle Prozesse und wird in den Voreinstellungen der 4D Anwendung gespeichert. Das entspricht einer Änderung in den Einstellungen der Datenbank. • Ein negativer Wert im Parameter *Wert* setzt ein lokales und temporäres Timeout: Der neue Wert gilt nur für den aufgerufenen Prozess. Die anderen Prozesse behalten den Standardwert bei. Der temporäre Wert wird zurückgesetzt, sobald der Server vom Client ein Signal für Aktivität erhält – zum Beispiel, wenn die Operation beendet ist. Diese Option eignet sich für lange Operationen, die über 4D Plug-Ins gestartet werden. Um die Option "Kein Timeout" zu setzen, übergeben Sie 0 (Null) in Wert (siehe 1. Beispiel).
4D Remote mode timeout	Lange Ganzzahl	14	**Reichweite:** 4D Anwendung bei positivem Wert **Wird zwischen 2 Sitzungen beibehalten:** Ja bei positivem Wert **Beschreibung:** Nur für ganz spezifische Fälle verwenden. Damit ändern Sie die Zeitspanne zum Abschalten des Rechners mit remote 4D vom 4D Server Rechner. Der Standardwert für das Timeout wird auf dem remote Rechner in den Einstellungen der Datenbank auf der Seite “Client/Server>Konfiguration” festgelegt. Der Selector 4D Remote Mode Timeout wird nur bei Verwenden der legacy Netzwerkschicht berücksichtigt. Er wird ignoriert, wenn *ServerNet* aktiviert ist: Diese Einstellung wird vom Selector 4D Server Timeout (13) verwaltet.

Konstante	Typ	Wert	Kommentar
Port ID	Lange Ganzzahl	15	**Reichweite**: 4D lokal, 4D Server **Wird zwischen zwei Sitzungen beibehalten**: Nein **Beschreibung**: ID des TCP Port, den der 4D Web Server mit 4D im lokalen Modus und 4D Server verwendet. Der Standardwert ist 80 und lässt sich in den Datenbank-Eigenschaften auf der Seite **Web/Konfiguration** setzen. Für den Parameter *Wert* können Sie eine Konstante unter dem Thema **TCP Port Nummern** verwenden. Der Selector Port ID ist hilfreich für 4D Web Server mit einkompilierter 4D Desktop, d.h. ohne Zugriff auf den Designmodus. Weitere Informationen dazu finden Sie im Abschnitt **Web Server, Einstellungen**.
_o_IP Address to listen	Lange Ganzzahl	16	**** *Selector ist deaktiviert, die Befehle **WEB SET OPTION** und **WEB GET OPTION** verwenden* ****
Character set	Lange Ganzzahl	17	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
Max concurrent Web processes	Lange Ganzzahl	18	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
Client minimum Web process	Lange Ganzzahl	19	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 6 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client maximum Web process	Lange Ganzzahl	20	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 17 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client Max Web requests size	Lange Ganzzahl	21	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 27 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client port ID	Lange Ganzzahl	22	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 15 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
_o_Client IP address to listen	Lange Ganzzahl	23	**** *Selector ist deaktiviert, die Befehle **WEB SET OPTION** und **WEB GET OPTION** verwenden* ****
Client character set	Lange Ganzzahl	24	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 17 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client max concurrent Web proc	Lange Ganzzahl	25	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 18 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Maximum Web requests size	Lange Ganzzahl	27	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.

Konstante	Typ	Wert	Kommentar
4D Server log recording	Lange Ganzzahl	28	**Reichweite:** 4D Server, remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 oder von 1 bis X (0 = nicht speichern, 1 bis X = sequentielle Nummer, an den Dateinamen angefügt. **Beschreibung:** Startet oder stoppt das Speichern von Standardanfragen, die von 4D Server empfangen werden (mit Ausnahme von Web Anfragen). Der Standardwert ist 0, d.h. Anfragen werden nicht gespeichert. 4D Server ermöglicht, jede Anfrage, die vom Server-Rechner empfangen wird, in einem Logbuch zu speichern. Ist das Speichern aktiviert, werden im Logbuch neben der Strukturdatei der Datenbank zwei Dateien mit den Namen *4DRequestsLog_X.txt* und *4DRequestsLog_ProcessInfo_X.txt* angelegt, wobei X die Sequenznummer des Logbuchs ist. Hat *4DRequestsLog_X.txt* die Größe von 10 MB, wird sie geschlossen und es wird ein neues Logbuch mit der nächsthöheren Sequenznummer angelegt. Ist bereits eine Datei mit diesem Namen vorhanden, wird sie direkt ersetzt. Mit dem Parameter *Wert* können Sie die Startnummer für die Sequenzen setzen. Diese Textdateien speichern verschiedene Informationen zu jeder Anfrage in einer einfachen Liste: Zeit, Prozessnummer und -dauer, Größe der Anfrage, etc. Weitere Informationen dazu finden Sie im **Anhang E: Beschreibung der Protokolldateien**.
_o_Web Log recording	Lange Ganzzahl	29	**Reichweite:** 4D lokal, 4D Server **Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
Client Web log recording	Lange Ganzzahl	30	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 = Nicht speichern (Standard), 1 = In CLF Format speichern, 2 = In DLF Format speichern, 3 = In ELF Format speichern, 4 = In WLF Format speichern. **Beschreibung:** Startet oder stoppt das Speichern von Web Anfragen, die vom Web Server aller Client-Rechner empfangen werden. Der Standardwert ist 0, d.h. Anfragen werden nicht gespeichert. Dieser Selector arbeitet genauso wie Selector 29; mit dem Unterschied, dass er für alle Client Rechner gilt, die als Web Server verwendet werden. Die Datei "logweb.txt" wird in diesem Fall automatisch in den Unterordner Logs des 4D Client Anwendungsordners (Cache Ordner) gelegt. Wollen Sie nur Werte für bestimmte Client-Rechner setzen, verwenden Sie die Voreinstellungen von remote 4D.
Table sequence number	Lange Ganzzahl	31	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Jeder Wert vom Typ Lange Ganzzahl. **Beschreibung:** Dieser Selector dient zum Ändern oder Erhalten der aktuellen einmaligen Nummer für Datensätze der Tabelle, die als Parameter übergeben ist. "Aktuelle Nummer" bedeutet "zuletzt verwendete Nummer": Verändern Sie diesen Wert über den Befehl **SET DATABASE PARAMETER**, erhält der nächste Datensatz die Nummer übergebener Wert + 1. Diese neue Nummer wird von der Funktion **Sequence number** sowie in einem Feld der Tabelle zurückgegeben, der die Eigenschaft "Autoincrement" im Inspektorfenster oder via SQL zugeordnet wurde. Diese einmalige Nummer wird standardmäßig von 4D angelegt und entspricht der Reihenfolge, in der die Datensätze erstellt werden. Weitere Informationen dazu finden Sie in der Beschreibung zur Funktion **Sequence number**.
_o_Real display precision	Lange Ganzzahl	32	**** Selector deaktiviert ****

Konstante	Typ	Wert	Kommentar

Debug log recording	Lange Ganzzahl	34	**Reichweite**: 4D Anwendung
			Wird zwischen 2 Sitzungen beibehalten: Nein
			Mögliche Werte: 0 oder 1
			Beschreibung: Startet oder stoppt das sequentielle Speichern von Ereignissen auf 4D Programmierebene in der Datei *4DDebugLog*, die automatisch in den Unterordner Logs neben der Strukturdatei gelegt wird. Ab 4D v14 wird im Logbuch für Ereignisse "4DDebugLog[_n].txt" ein neues kompakteres Textformat verwendet (_n ist die Segmentnummer der Datei).
			Werte: Lange Ganzzahl mit einem Bit-Feld: Wert = bit1(1)+bit2(2)+bit3(4)+bit4(8)+ …).
			- Bit 1 (Wert 1) aktiviert das Schreiben des Logbuchs. Beachten Sie, dass jeder andere Wert, der nicht Null ist, es ebenfalls aktiviert.
			- Bit 2 (Wert 2) nimmt Parameter von Methoden und Befehlen mit auf.
			- Bit 3 (Wert 4) aktiviert neue Formate mit Tabulatoren.
			- Bit 4 (Wert 8) deaktiviert sofortiges Schreiben jeder Operation auf die Festplatte (standardmäßig aktiviert). Sofortiges Schreiben ist langsamer, erlaubt aber, die Ursachen bei einem Absturz herauszufinden. Deaktivieren macht den Inhalt der Datei kompakter und generiert ihn schneller.
			- Bit 5 (Wert 16) deaktiviert das Protokollieren von Plug-In Aufrufen (standardmäßig aktiviert)
			Im älteren Format ohne Tabs werden Ausführungszeiten in Millisekunden angezeigt und der Wert "< ms" erscheint für Operationen, die weniger als eine Millisekunde dauern
			Im Format mit Tab werden Ausführungszeiten in Mikrosekunden angezeigt.
			Beispiele:
			SET DATABASE PARAMETER (34;1) // aktiviert das Logbuch im Modus v13 ohne Parameter und mit der Dauer
			SET DATABASE PARAMETER (34;2) // aktiviert das Logbuch im Modus v13 mit Parametern und Dauer
			SET DATABASE PARAMETER (34;2+4) // aktiviert das Logbuch im v14 Format, mit Parametern und Dauer
			SET DATABASE PARAMETER (34;0) // deaktiviert das Logbuch
			Damit eine Datei nicht zuviel Information protokolliert, können Sie mit dem Selektor 80 Log Command list die Liste der zu prüfenden 4D Befehle einschränken.
			Diese Option lässt sich für alle 4D Applikationen aktivieren (4D alle Modi, 4D Server, 4D Volume Desktop), im interpretierten oder kompiliertem Modus.
			Hinweis: Diese Option dient nur für Debugging-Zwecke und sollte nicht im laufenden Betrieb verwendet werden, da sie zu Einschränkungen der Performance und Überlastung der Festplatte führen kann. Weitere Informationen zu diesem Format und Einsatz der Datei *4DDebugLog[_n].txt* finden Sie im **Anhang E: Beschreibung der Protokolldateien**.

Client Server port ID	Lange Ganzzahl	35	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 bis 65535 **Beschreibung:** Mit diesem Parameter lässt sich per Programmierung die Nummer des TCP-Port bestimmen, in dem 4D Server die Datenbank veröffentlicht (Zugriff für 4D im lokalen Modus). Der Standardwert ist 19813. Durch individuelles Anpassen dieses Wertes können auf einem Rechner mit dem TCP Protokoll mehrere 4D Client/Server Anwendungen laufen. Dazu müssen Sie für jede Anwendung eine andere Port-Nummer angeben. Der Wert wird in der Strukturdatei der Datenbank gespeichert. Er lässt sich mit 4D im lokalen Modus setzen, wirkt sich jedoch nur im Client/Server-Betrieb aus. Verändern Sie diesen Wert, müssen Sie den Server-Rechner neu starten, damit er berücksichtigt wird.
Invert objects	Lange Ganzzahl	37	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0, 1 oder 2 (0 = Modus inaktiv, 1 = automatischer Modus, 2 = Modus aktiviert). **Beschreibung:** Einstellen des Modus "Objekt-Inversion", um Formulare, Objekte, Menüleisten, etc. im Anwendungsmodus umzukehren, wenn die Datenbank unter Windows in einer rechts-nach-links laufenden Sprache angezeigt wird. Dieser Modus lässt sich auch auf der Seite **Oberfläche** in den Datenbank-Eigenschaften einstellen: • Wert 0 gibt an, dass der Modus nie aktiviert wird, unabhängig von der Systemkonfiguration (entspricht der Einstellung **Nie** auf der Seite **Oberfläche**). • Wert 1 gibt an, dass der Modus je nach Systemkonfiguration aktiviert oder deaktiviert ist (entspricht der Einstellung **Automatisch** auf der Seite **Oberfläche**). • Wert 2 gibt an, dass der Modus aktiviert ist, unabhängig von der Systemkonfiguration (entspricht der Einstellung **Immer** auf der Seite **Oberfläche**). Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.
HTTPS Port ID	Lange Ganzzahl	39	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.

Konstante	Typ	Wert	Kommentar
Client HTTPS port ID	Lange Ganzzahl	40	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 bis 65535 **Beschreibung:** TCP Port Nummer, welche die Web Server der Client-Rechner für sichere Verbindungen via SSL (HTTPS Protokoll) verwenden. Der Wert ist standardmäßig 443. Mit diesem Selector können Sie per Programmierung die TCP Portnummer ändern, die Web Server von Client-Rechnern für sichere Verbindungen via SSL (HTTPS protocol) verwenden. Dieser Selector funktioniert auf dieselbe Weise wie der Selector 39; er wird jedoch auf alle Client-Rechner angewandt, die als Web Server dienen. Wollen Sie nur den Wert eines bestimmten Client-Rechners verändern, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Unicode mode	Lange Ganzzahl	41	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 (Modus Kompatibilität) oder 1 (Modus Unicode) **Beschreibung:** Aktueller Ausführungsmodus der Datenbank in Bezug auf den Zeichensatz. 4D unterstützt den Unicode Zeichensatz, kann aber auch im Modus "Kompatibilität" arbeiten, der auf dem Zeichensatz Mac ASCII basiert. Standardmäßig werden konvertierte Datenbanken im Modus Kompatibilität (0), in Version 11 oder höher erstellte Datenbanken im Modus Unicode ausgeführt. Der Ausführungsmodus lässt sich über eine Option in den Voreinstellungen steuern. Er lässt sich über diesen Selector auch lesen oder für Testzwecke verändern. Sie müssen die Datenbank neu starten, damit die Änderung berücksichtigt wird. Beachten Sie, dass Sie diesen Wert innerhalb einer Komponente nicht verändern, sondern ihn nur lesen können.
SQL Autocommit	Lange Ganzzahl	43	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 (deaktivieren) oder 1 (aktivieren) **Beschreibung:** Aktivierung oder Deaktivierung des Modus SQL auto-commit. Standardmäßig ist der Wert 0 (deaktiviert). Der Modus auto-commit sorgt für strengere referentielle Integrität der Datenbank. Ist er aktiv, werden alle Suchläufe mit **SELECT**, **INSERT**, **UPDATE** und **DELETE** (SIUD) automatisch in ad hoc Transaktionen gesetzt, wenn das noch nicht der Fall ist. Sie können diesen Modus auch in den Einstellungen der Datenbank setzen.
SQL Engine case sensitivity	Lange Ganzzahl	44	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 oder 1 (0 = Groß-/Kleinschreibung nicht beachten, 1 = Groß-/Kleinschreibung beachten) **Beschreibung:** Aktiviert oder deaktiviert die Berücksichtigung von Groß-/Kleinschreibung beim Vergleichen von Strings mit der SQL Engine. Der Wert ist standardmäßig 1, d.h. die SQL Engine unterscheidet beim Vergleichen von Strings zwischen Groß- und Kleinschreibung (Sortieren und Suchen) und Akzenten. Beispiel: "ABC"= "ABC" aber "ABC" # "Abc und "abc" # "âbc". In bestimmten Fällen, z.B. um die Funktionsweise der SQL Engine an die 4D Engine anzupassen, sollen beim Vergleichen von Strings die Groß- und Kleinschreibung, sowie Akzente nicht berücksichtigt werden ("ABC"="Abc"="âbc"). Sie können diese Option auch in den Datenbank-Eigenschaften auf der **Seite SQL** einstellen.
Client log recording	Lange Ganzzahl	45	**Reichweite:** Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 oder von 1 bis X (0 = nicht speichern, 1 bis X= sequentielle Nummer, an den Dateinamen angehängt). **Beschreibung:** Startet oder stoppt das Speichern der Standardanfragen, die vom 4D Client Rechner ausgehen, der den Befehl ausführt (außer Web Anfragen). Der Wert ist standardmäßig 0 (Null), d.h. die Anfragen werden nicht gespeichert. Sie können in 4D alle vom Client Rechner ausgeführten Anfragen in einem Logbuch speichern. Ist das Speichern aktiviert, werden auf dem Client Rechner im lokalen Ordner der Datenbank im Unterordner *Logs* zwei Dateien angelegt: *4DRequestsLogX.txt* und *4DRequestsLog_ProcessInfo_X.txt*, wobei X die fortlaufende Nummer des Logbuchs ist. Hat *4DRequestsLogX.txt* die Größe von 10 MB erreicht, wird es geschlossen und ein neues Logbuch mit der nächsthöheren Nummer erstellt. Gibt es bereits eine Datei mit demselben Namen, wird sie ersetzt. Über den Parameter *Wert* können Sie die Anfangsnummer setzen. Diese Textdateien speichern die Informationen zu jeder Anfrage in einem einfachen Format mit Tabs: Zeit, Prozessnummer, Größe der Anfrage, Bearbeitungsdauer, etc. Weitere Informationen dazu finden Sie im Abschnitt **Anhang E: Beschreibung der Protokolldateien**.

Konstante	Typ	Wert	Kommentar
Query by formula on server	Lange Ganzzahl	46	**Reichweite:** Aktuelle Tabelle und Prozess **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 (Konfiguration der Datenbank verwenden), 1 (auf Client ausführen) oder 2 (auf Server ausführen) **Beschreibung:** Ausführungsort der Befehle **QUERY BY FORMULA** und **QUERY SELECTION BY FORMULA** für die im Parameter übergebene Tabelle. Bei einer Datenbank im Client/Server-Betrieb können die Suchbefehle "nach Formel" entweder auf dem Server oder auf dem Client Rechner ausgeführt werden. • In einer mit 4D v11 SQL erstellten Datenbank werden die Befehle auf dem Server ausgeführt. • In konvertierten Datenbanken werden sie, wie in den bisherigen 4D Versionen, auf dem Client Rechner ausgeführt. • In konvertierten Datenbanken gibt es eine spezielle Voreinstellung (Seite Anwendung>Kompatibilität), um den Ausführungsort dieser Befehle global zu verändern. Der Ausführungsort beeinflusst nicht nur die Performance der Anwendung - die Ausführung auf dem Server ist in der Regel schneller - sondern auch die Programmierung. Der Wert der Komponenten der Formel, insbesondere bei Variablen, die über eine Methode aufgerufen werden, ändert sich je nach Ausführungsort. Über diesen Selektor können Sie die Arbeitsweise Ihrer Anwendung einstellen. Übergeben Sie 0 (Null) im Parameter Wert, richtet sich der Ausführungsort der Suchbefehle nach der Konfiguration der Datenbank: In mit 4D v11 SQL erstellten Datenbanken werden diese Befehle auf dem Server ausgeführt. In konvertierten Datenbanken werden sie, je nach den Datenbank-Eigenschaften, auf dem Client Rechner oder Server ausgeführt. Übergeben Sie 1 oder 2 in Wert, um die Ausführung dieser Befehle jeweils auf dem Client oder Server Rechner zu erzwingen. Siehe Beispiel 2. **Hinweis:** Wollen Sie auch die Möglichkeit haben, "SQL type" joins zu aktivieren (siehe Selector QUERY BY FORMULAR Joins, müssen Sie Formeln immer auf dem Server ausführen, damit diese auf die Datensätze zugreifen können. Bedenken Sie, dass die Formel in diesem Kontext keine Aufrufe einer Methode enthalten darf, da sie sonst automatisch auf den remote Rechner wechselt.
Order by formula on server	Lange Ganzzahl	47	**Reichweite:** Aktuelle Tabelle und Prozess **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 (Konfiguration der Datenbank verwenden), 1 (auf Client ausführen) oder 2 (auf Server ausführen) **Beschreibung:** Ausführungsort des Befehls **ORDER BY FORMULA** für die im Parameter übergebene Tabelle. Bei einer Datenbank im Client/Server-Betrieb lässt sich dieser Befehl entweder auf dem Server oder auf dem Client-Rechner ausführen. Über diesen Selector können Sie den Ausführungsort dieses Befehls angeben (Server oder Client). Sie können diesen Modus auch in dern Datenbank-Eigenschaften setzen. Weitere Informationen dazu finden Sie unter Selector 46, Query By Formula On Server. **Hinweis:** Wollen Sie auch die Möglichkeit haben, "SQL type" joins zu aktivieren (siehe Selector QUERY BY FORMULAR Joins, müssen Sie Formeln immer auf dem Server ausführen, damit diese auf die Datensätze zugreifen können. Bedenken Sie, dass die Formel in diesem Kontext keine Aufrufe einer Methode enthalten darf, da sie sonst automatisch auf den remote Rechner wechselt.
Auto synchro resources folder	Lange Ganzzahl	48	**Reichweite:** Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 (keine Synchronisation), 1 (auto Synchronisation) oder 2 (Fragen). **Beschreibung:** Dynamische Synchronisation des lokalen Ordners **Resources** auf dem Client-Rechner, der den Befehl mit dem Ordner **Resources** auf dem Server ausführt. Wurde der Inhalt des Ordners **Resources** auf dem Server geändert oder hat ein Benutzer die Synchronisation angefordert, z.B. über den Resourcen Explorer oder nach der Ausführung des Befehls **SET DATABASE LOCALIZATION**, benachrichtigt der Server die angemeldeten Client-Rechner. Auf der Client-Seite gibt es dann drei Möglichkeiten zur Synchronisation. Der Selektor Auto Synchro Resources Folder ermöglicht die Einstellung auf dem Client für die aktuelle Arbeitssitzung: • 0 (Standardwert) = keine dynamische Synchronisation (die Anfrage zur Synchronisation wird ignoriert) • 1 = automatische dynamische Synchronisation • 2 = Anzeige des Dialogfensters auf den Client-Rechnern, um die Synchronisation zu erlauben oder zu verweigern. Der Synchronisationsmodus lässt sich auch global in den Einstellungen der Anwendung definieren.

Konstante	Typ	Wert	Kommentar
Query by formula joins	Lange Ganzzahl	49	**Reichweite:** Aktueller Prozess **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 (Konfiguration der Datenbank verwenden), 1 (immer automatische Verknüpfungen) oder 2 (Wenn möglich, SQL joins verwenden). **Beschreibung:** Arbeitsweise der Befehle **QUERY BY FORMULA** und **QUERY SELECTION BY FORMULA** bei Verwendung von "SQL joins". In Datenbanken, die mit Version 11.2 von 4D v11 SQL erstellt wurden, führen diese Befehle Verknüpfungen nach dem SQL joins Modell aus. Auf diese Weise lässt sich die ausgewählte Tabelle verändern, wenn eine Suche in einer anderen Tabelle ausgeführt wird, auch wenn diese Tabellen nicht über eine automatische Verknüpfung miteinander verbunden sind. Diese automatische Verknüpfung war in bisherigen 4D Versionen erforderlich. Über den Selector QUERY BY FORMULA Joins legen Sie fest, wie die Suchbefehle im aktuellen Prozess arbeiten: • 0 (Standardwert) = Verwendet die aktuellen Einstellungen der Datenbank. In Datenbanken, die mit Version 11.2 von 4D v11 SQL erstellt wurden, sind für Suchen nach Formel immer "SQL joins" aktiviert. In konvertierten Datenbanken ist diese Arbeitsweise aus Kompatibilitätsgründen nicht aktiv, sie lässt sich aber über eine Voreinstellung einrichten. • 1 = Verwende immer automatische Verknüpfungen (= bisherige Funktionsweise in 4D). In diesem Modus ist eine Verknüpfung notwendig, damit bei Suche in einer anderen Tabelle die entsprechende Tabelle ausgewählt wird. 4D macht keine "SQL joins." • 2 = Verwende SQL joins, wenn möglich (= Standardverhalten in Datenbanken, die mit Version 11.2 von 4D v11 SQL und höher erstellt wurden). In diesem Modus erstellt 4D "SQL joins" für Suchen nach Formel, wenn die Formel dazu passt - bis auf zwei Ausnahmen. Weitere Informationen dazu finden Sie unter den Befehlen **QUERY BY FORMULA** oder **QUERY SELECTION BY FORMULA**. **Hinweis:** Mit 4D im remote Modus lassen sich "SQL joins" nur verwenden, wenn die Formeln auf dem Server ausgeführt werden, da diese auf die Datensätze zugreifen müssen. Wie Sie konfigurieren, wo Formeln ausgeführt werden, sehen Sie unter den Selectoren 46 und 47.
HTTP compression level	Lange Ganzzahl	50	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
HTTP compression threshold	Lange Ganzzahl	51	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
Server base process stack size	Lange Ganzzahl	53	**Reichweite:** 4D Server **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** Positive Lange Ganzzahl **Beschreibung:** Größe des Stapelspeichers, der jedem preemptiven Systemprozess auf dem Server zugewiesen wird, ausgedrückt in Bytes. Die Standardgröße wird vom System bestimmt. Preemptive Systemprozesse, d.h. Prozesse vom Typ 4D Client base process werden geladen, um die Hauptprozesse von 4D Client zu steuern. Die Größe, die dem Stapelspeicher jedes preemptiven Prozesses standardmäßig zugewiesen ist, sorgt für eine gut laufende Ausführung, kann aber eine wichtige Rolle spielen, wenn eine umfangreiche Anzahl, also mehrere hundert Prozesse erstellt werden. Diese Größe lässt sich für Optimierungszwecke beträchtlich verringern, wenn die von der Datenbank ausgeführten Operationen dies zulassen. Das ist z.B. der Fall, wenn die Datenbank keine Sortierung auf eine große Anzahl Datensätze ausführt. Möglich sind Werte von 512 oder auch 256 KB. Beachten Sie, dass eine zu geringe Stapelgröße kritisch werden kann, da sie die Operationen von 4D Server beeinträchtigt. Setzen Sie diesen Parameter mit Bedacht und unter Berücksichtigung der Datenbankbedingungen, wie z.B. Anzahl der Datensätze, Art der Operationen, etc. Damit dieser Parameter berücksichtigt wird, muss er auf dem Server-Rechner ausgeführt werden, z.B. in der **Datenbankmethode On Server Startup**.

Konstante	Typ	Wert	Kommentar
Idle connections timeout	Lange Ganzzahl	54	**Reichweite:** 4D Anwendung, außer Wert ist negativ **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** Ganzzahl für die Dauer in Sekunden. Der Wert kann positiv sein (neue Verbindungen) oder negativ (vorhandene Verbindungen). Standardmäßig ist der Wert 20. **Beschreibung:** Dieser Parameter ermöglicht auf der Server-Seite die maximale Zeitspanne an Inaktivität (timeout) für Verbindungen zur 4D Datenbank-Engine und der SQL Engine zu definieren. Erreicht eine pausierende Verbindung das Limit, wird sie automatisch auf standby gesetzt. Das friert die Client/Server Sitzung ein und schließt den Netzwerk Socket. Im Server Verwaltungsfenster lautet der Status des Benutzerprozesses "Zurückgestellt". Dieser Ablauf läuft für den Benutzer unsichtbar ab: Sobald bei der auf standby gesetzten Verbindung neue Aktivität auftritt, wird der Socket automatisch wieder geöffnet und die Client/Server Sitzung wiederhergestellt. Diese Einstellung lässt Sie einerseits Ressourcen auf dem Server einsparen: Verbindungen in standby schließen den Socket und geben einen Prozess auf dem Server frei. Andererseits können Sie so auch verhindern, dass Verbindungen verloren gehen, weil pausierende Sockets durch die Firewall geschlossen werden. Deshalb muss in diesem Fall der Wert für das Timeout von pausierenden Verbindungen niedriger als der für die Firewall sein. Übergeben Sie einen positiven Wert in *Wert*, gilt er für alle neuen Verbindungen in allen Prozessen. Übergeben Sie einen negativen Wert, gilt er für Verbindungen, die im aktuellen Prozess geöffnet sind. Übergeben Sie 0, unterliegen pausierende Verbindungen keinem Timeout. Dieser Parameter lässt sich auf Server und Client setzen. Geben Sie zwei unterschiedliche Zeiten an, wird der kleinere Wert berücksichtigt. Im Normalfall müssen Sie diesen Wert nicht ändern.
PHP interpreter IP address	Lange Ganzzahl	55	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: Formatierter String vom Typ IPv4 (zum Beispiel "127.0.0.1") oder IPv6 (zum Beispiel "2001:0db8:0000:0000:0000:ff00:0042:8329"). **Beschreibung**: von 4D lokal verwendete IP Adresse, um mit dem PHP Interpreter via FastCGI zu kommunizieren. Der Wert ist standardmäßig "127.0.0.1" (Adressen im IPv6 Format werden ab 4D v16R4 unterstützt). Diese Adresse muss dem Rechner entsprechen, auf dem 4D läuft. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner setzen. Weitere Informationen zum PHP Interpreter finden Sie im Handbuch *4D Designmodus*.
PHP interpreter port	Lange Ganzzahl	56	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: Positive Lange Ganzzahl. Standardmäßig ist der Wert 8002. **Beschreibung**: Nummer des TCP Port, den der PHP Interpreter von 4D verwendet. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner setzen. Weitere Informationen zum PHP Interpreter finden Sie im Handbuch *4D Designmodus*.
PHP number of children	Lange Ganzzahl	57	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: Positive Lange Ganzzahl. Standardmäßig ist der Wert 5. **Beschreibung**: Anzahl der Kindprozesse, die vom PHP Interpreter von 4D erstellt und lokal gewartet werden. Aus Optimierungsgründen erstellt und verwendet der PHP Interpreter zum Bearbeiten der Anfragen zur Skript-Ausführung einen Satz von Systemprozessen, genannt "Kindprozesse". Sie können die Anzahl der Kindprozesse je nach den Anforderungen Ihrer Anwendung variieren. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner setzen. Weitere Informationen zum PHP Interpreter finden Sie im Handbuch *4D Designmodus*. **Hinweis:** Auf Mac OS nutzen alle Kindprozesse den gleichen Port. Unter Windows verwendet jeder Kindprozess eine spezifische Port Nummer. Die erste Nummer ist die für den PHP Interpreter gesetzte; die anderen Kindprozesse erhöhen die erste Nummer jeweils. Ist zum Beispiel der Standardport 8002 und starten Sie 5 Kindprozesse, verwenden sie die Ports 8002 bis 8006.
PHP max requests	Lange Ganzzahl	58	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: Positive Lange Ganzzahl. Standardmäßig ist der Wert 500. **Beschreibung**: Maximale Anzahl der Anfragen, die der PHP Interpreter akzeptiert. Ist die maximale Anzahl erreicht, gibt der Interpreter Fehler vom Typ "Server ist überlastet" zurück. Sie können diesen Wert aus Sicherheits- oder Performance-Gründen verändern. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner verändern. Weitere Informationen dazu finden Sie in der Dokumentation zu FastCGI-PHP. **Hinweis:** Auf 4D Seite werden diese Parameter dynamisch angewandt; d.h. 4D muss nicht beendet werden, damit sie berücksichtigt werden. Ist dagegen der PHP Interpreter bereits gestartet, muss beendet und neu gestartet werden, damit diese Änderungen berücksichtigt werden.

Konstante	Typ	Wert	Kommentar
PHP use external interpreter	Lange Ganzzahl	60	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: 0 = Verwende internen Interpreter, 1 = Verwende externen Interpreter **Beschreibung**: Der Wert gibt an, ob PHP Anfragen in 4D an den von 4D intern gelieferten Interpreter oder an einen externen Interpreter gesendet werden. Standardmäßig ist der Wert 0, d.h. der interne Interpreter von 4D wird verwendet. Wollen Sie Ihren eigenen PHP Interpreter verwenden, um z.B. zusätzliche Module oder eine spezifische Konfiguration zu verwenden, übergeben Sie 1 in *Wert*. In diesem Fall startet 4D bei PHP Anfragen nicht den internen Interpreter. Eigene PHP Interpreter müssen in FastCGI kompiliert sein und auf demselben Rechner wie die 4D Engine liegen. Beachten Sie, dass Sie diese Interpreter komplett selbst verwalten müssen; sie werden von 4D weder gestartet noch gestoppt. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner setzen.
Maximum temporary memory size	Lange Ganzzahl	61	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Werte:** Positive Lange Ganzzahl **Beschreibung:** Maximale Größe des temporären Speichers, den 4D jedem Prozess zuordnen kann, ausgedrückt in MB. Der Wert ist standardmäßig 0 (keine maximale Größe). 4D verwendet einen speziellen temporären Speicher zum Indizieren und Sortieren der Operationen. Ziel dieses Speichers ist, den "standard" Cache Speicher bei massiven Operationen beizubehalten. Er wird nur bei Bedarf aktiviert. Die Größe des temporären Speichers wird nur durch die verfügbaren Ressourcen begrenzt. Das richtet sich nach der Konfiguration des Systemspeichers. Diese Vorgehensweise passt für die meisten Programme. In einigen spezifischen Fällen, insbesondere wenn eine Client-/Server Anwendung simultan eine große Anzahl sequentieller Sortierungen durchführt, kann der temporäre Speicher eine kritische Größe erreichen, die das System instabil macht. Für solche Fälle können Sie für den temporären Speicher eine maximale Größe einrichten, damit die Anwendung weiterhin einwandfrei laufen kann. Das kann jedoch wiederum die Ausführungsgeschwindigkeit beeinträchtigen: Ist die maximale Größe für einen Prozess erreicht, verwendet 4D Dateien der Festplatte, was den Vorgang verlangsamen kann. Für spezifische Anforderungen, wie z.B. oben beschrieben, ist im allgemeinen eine maximale Größe von ca. 50 MB ein guter Kompromiss. Der ideale Wert muss jedoch anhand der Eigenheiten der Anwendung bestimmt werden und ergibt sich aus Volumentests in Echtzeit.
SSL cipher list	Zeichenkette	64	**Reichweite**: 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: Folge von Strings, getrennt durch Doppelpunkt (zum Beispiel "RC4-MD5:RC4-64-MD5:....") **Beschreibung:** Cipher Liste, die 4D für das SSL Protokoll verwendet. Diese Liste verändert die Priorität der in 4D implementierten cipher Algorithmen. Sie können im Parameter *Wert* z.B. folgenden String übergeben. "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH". Eine ausführliche Beschreibung der Syntax für die Cipher Liste finden Sie auf der ciphers Seite der OpenSSL Site. Diese Einstellung gilt für die gesamte Anwendung (sie betrifft HTTP Server, SQL Server, Client/Server Verbindungen, sowie den HTTP Client und alle 4D Befehle, die das SSL Protokoll nutzen), aber sie ist temporär, d.h. sie bleibt zwischen Sitzungen nicht erhalten. Wurde die Cipher Liste geändert, müssen Sie den betroffenen Server neu starten, damit die neuen Einstellungen berücksichtigt werden. Um die Cipher Liste auf ihren Standardwert zurückzusetzen, der permanent in der SLI Datei gespeichert ist, rufen Sie den Befehl **SET DATABASE PARAMETER** auf und übergeben im Parameter *Wert* einen leeren String (""). **Hinweis:** Mit der Funktion **Get database parameter** wird die Cipher Liste im optionalen Parameter *StringWert* zurückgegeben und der Parameter ist immer 0.
Cache unload minimum size	Lange Ganzzahl	66	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Mögliche Werte**: Positive Lange Ganzzahl > 1. **Beschreibung**: Mindestgröße des Speichers, der vom Datenbank Cache freigegeben werden soll, wenn die Engine Platz schaffen muss, um dem Cache ein Objekt zuzuweisen (Wert in Bytes). Dieser Selektors dient dazu, für eine bessere Performance die Momente zu reduzieren, zu denen Daten aus dem Cache freigegeben werden. Sie können diese Einstellung je nach Größe des Cache bzw. nach Datenblock, der in Ihrer Datenbank bearbeitet wird, variieren. Ohne diesen Selektor entlädt 4D standardmäßig mindestens 10% des Cache, wenn Platz gebraucht wird.

Konstante	Typ	Wert	Kommentar
Direct2D status	Lange Ganzzahl	69	**Reichweite:** 4D Anwendung **Zwischen 2 Sitzungen beibehalten:** Nein **Beschreibung:** Aktivierungsmodus zum Integrieren von Direct2D unter Windows. **Mögliche Werte:** Eine der folgenden Konstanten (standardmäßig Modus 5): Direct2D Disabled (0): Direct2D Modus ist nicht aktiviert. Die Anwendung funktioniert wie im früheren Modus (GDI/GDIPlus). Direct2D Hardware (1): Direct2D als Grafik Hardware-Kontext für die gesamte 4D Anwendung verwenden. Falls nicht verfügbar, Direct2D Grafik Software-Kontext verwenden (außer unter Vista; hier wird zur besseren Performance der Modus GDI/GDIPlus verwendet. Direct2D Software (3):(Standardmodus) Ab Windows 7 den Software-Kontext Direct2D Grafik für die gesamte 4D Anwendung verwenden. Unter Vista wird zur besseren Performance der Modus GDI/GDIPlus verwendet. **Hinweis zur Kompatibilität:** Ab 4D v14 werden hybrid Modi deaktiviert und auf die verfügbaren Modi umgeleitet (der bisherige Modus 2 entspricht 1; die bisherigen Modi 4 und 5 entsprechen Modus 3).

Direct2D get active status	Lange Ganzzahl	74	**Hinweis:** Sie können diesen Selector nur mit der Funktion **Get database parameter** verwenden und keinen Wert setzen.
			Beschreibung: Gibt die aktuelle Integration von Direct2D unter Windows zurück.
			Mögliche Werte: 0, 1, 2, 3, 4 oder 5 (siehe Werte des Selector 69). Der zurückgegebene Wert richtet sich nach der Verfügbarkeit von Direct2D, sowie Hardware und der vom Betriebssystem unterstützten Qualität von Direct2D.
			Führen Sie zum Beispiel folgendes aus:
			`SET DATABASE PARAMETER(Direct2D status;Direct2D Hardware)`
			`$mode:=Get database parameter(Direct2D get active status)`
			
			- Ab Windows 7 und höher wird $mode auf 1 gesetzt, wenn das System Hardware findet, die mit Direct2D kompatibel ist; andernfalls wird $mode auf 3 gesetzt (Software-Kontext).
			- Unter Windows Vista wird *$mode* auf 1 gesetzt, wenn das System Hardware findet, die mit Direct2D; kompatibel ist; andernfalls wird *$mode* auf 0 gesetzt, d.h. Direct2D wird deaktiviert.
			- Unter Windows XP wird *$mode* immer auf 0 gesetzt, da diese Version nicht mit Direct2D kompatibel ist.

Diagnostic log recording	Lange Ganzzahl	79	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 oder 1 (0 = nicht speichern, 1 = speichern) **Beschreibung:** Startet oder stoppt das Protokollieren der 4D Diagnosedatei. Standardmäßig ist der Wert 0 (nicht protokollieren). 4D kann kontinuierlich eine Reihe Ereignisse, die bei der internen Operationsweise der Anwendung auftreten, in einer Diagnosedatei mitschreiben. Information in dieser Datei ist hilfreich beim Entwickeln von 4D Anwendungen und kann mit Hilfe des 4D Tech Support analysiert werden. Übergeben Sie 1 in diesem Selector, wird im Ordner **Logs** der Anwendung automatisch eine Diagnosedatei mit Namen *DatenbankName.txt* angelegt bzw. geöffnet. Bei einer Größe von 10 MB wird sie geschlossen und es wird eine neue Datei mit Namen *DatenbankName_N.txt* generiert mit der ansteigenden Nummer N. Über den Befehl **LOG EVENT** können Sie in dieser Datei eigene Informationen hinzufügen.
Log command list	Zeichenkette	80	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** String mit der Liste der 4D Befehlsnummern zum Protokollieren, getrennt durch Strichpunkte. Sie können "all" übergeben, um alle Befehle zu protokollieren oder "" (leerer String), um keine Befehle zu protokollieren. **Beschreibung***:* Liste der 4D Befehle, die in der Protokolldatei mitgeschrieben werden (siehe Selector 34, Debug Log Recording). Standardmäßig werden alle 4D Befehle protokolliert. Dieser Selector reduziert die Informationen, die in der Diagnosedatei gesichert wird, durch Einschränken der 4D Befehle, deren Ausführung Sie nicht protokollieren wollen. Sie können z.B. schreiben: `SET DATABASE PARAMETER(Log command list;"277;334") //Nur die Befehle QUERY und QUERY SELECTION protokollieren`
Spellchecker	Lange Ganzzahl	81	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: 0 (Standard) = native OS X Rechtschreibprüfung (Hunspell deaktiviert), 1 = Hunspell Rechtschreibprüfung aktiviert. **Beschreibung**: Aktiviert die Hunspell Rechtschreibprüfung auf OS X. Auf dieser Plattform wird standardmäßig die native Rechtschreibprüfung aktiviert. Sie können jedoch die Hunspell Rechtschreibprüfung aktivieren, z.B. um die Oberfläche Ihrer plattformunabhängigen Anwendungen einheitlich zu gestalten (unter Windows ist nur die Hunspell Rechtschreibprüfung verfügbar). Weitere Informationen dazu finden Sie im Abschnitt **Rechtschreibprüfung**.

Konstante	Typ	Wert	Kommentar
QuickTime support	Lange Ganzzahl	82	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Ja **Werte**: 0 (Standard) = QuickTime deaktiviert, 1 = QuickTime aktiviert **Beschreibung**: 4D Version 14 unterstützt QuickTime Codecs nicht mehr standardmäßig. Zur Wahrung der Kompatibilität können Sie die Unterstützung in Ihrer Anwendung über diesen Selector reaktivieren. Beachten Sie jedoch, dass die QuickTime Unterstützung in zukünftigen 4D Versionen komplett eingestellt wird.
Dates inside objects	Lange Ganzzahl	85	**Reichweite**: Aktueller Prozess **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: String type without time zone (0), String type with time zone (1), Date type (2) (Standard) **Beschreibung**: Definiert wie Datumsangaben innerhalb von Objekten gespeichert werden und sie in JSON importiert/exportiert werden. Mit Date type (Standardwert für Anwendungen, die mit 4Dv17 oder höher erstellt wurden) werden 4D Datumsangaben mit dem Datumstyp im Objekt gespeichert unter Berücksichtigung der lokalen Datumseinstellung. Beim Konvertieren in das JSON Format werden Datumsattribute in Strings ohne Zeitangabe konvertiert. (**Hinweis:** Dies kann über die Option "Verwende Datumstyp statt ISO Datumsformat in Objekten" in den Datenbank-Einstellungen auf der Seite **Seite Kompatibilität** gesetzt werden) Mit String type with time zone werden 4D Datumsangaben in ISO Strings konvertiert und berücksichtigen die lokale Zeitzone. Zum Beispiel erhalten Sie beim Konvertieren des Datums !23.08.2013! in JSON Format 2013-08-22T22:00:00Z, wenn die Operation in Deutschland zu einer Sicherungszeit bei Tageslicht (GMT+2) ausgeführt wird. Dieses Prinzip entspricht der Standardoperation von JavaScript. Das kann zu Fehlern führen, wenn Sie Datumswerte in JSON an jemanden in einer anderen Zeitzone senden wollen. Das ist z.B. der Fall, wenn Sie eine Tabelle über **Selection to JSON** in Deutschland exportieren, die dann in USA über **JSON TO SELECTION** importiert wird. Da Datumsangaben in jeder Zeitzone erneut interpretiert werden, sind die in der Anwendung gespeicherten Werte unterschiedlich. In diesem Fall können Sie den Konvertierungsmodus für Datum ändern. Damit die Zeitzone nicht berücksichtigt wird, übergeben Sie in diesem Selektor String type without time zone. Dann ergibt die Konvertierung des Datums !23.08.2013! in allen Fällen 2013-08-23T00:00:00Z.
Use legacy network layer	Lange Ganzzahl	87	**Reichweite**: 4D im lokalen Modus, 4D Server **Wird zwischen 2 Sitzungen beibehalten**: Ja **Beschreibung**: Setzt oder erhält den aktuellen Status der legacy Netzwerk-Schicht für Client/Server Verbindungen. Die legacy Netzwerk-Schicht ist ab 4D v14 R5 überholt und sollte in Ihren Anwendungen nach und nach durch die Netzwerk-Schicht *ServerNet* ersetzt werden. *ServerNet* ist in den nächsten 4D Releases erforderlich, um auch in Zukunft von Netzwerk Evolutionen zu profitieren. Zur Wahrung der Kompatibilität wird die bisherige Netzwerk-Schicht noch unterstützt, um den allmählichen Übergang für vorhandene Anwendungen zu ermöglichen. Sie wird in konvertierten Anwendungen aus einem Release vor v14 R5 standardmäßig genutzt. Übergeben Sie 1, um für Ihre Client/Server Verbindungen die bisherige Netzwerk-Schicht zu verwenden (und *ServerNet* zu deaktivieren); 0, um sie zu deaktivieren (und *ServerNet* zu verwenden). Diese Eigenschaft lässt sich auch über die Option "Verwende legacy Netzwerk Schicht" auf der **Seite Kompatibilität** der Datenbank-Eigenschaften setzen. Hier wird auch die Vorgehensweise zum Migrieren erläutert. Wir empfehlen, so bald wie möglich in all Ihren Anwendungen auf *ServerNet* umzustellen (siehe auch **Netzwerk und Client-Server Optionen**). Sie müssen die Anwendung neu starten, damit dieser Selektor berücksichtigt wird. **Mögliche Werte**: 0 oder 1 (0 = legacy Schicht nicht verwenden, 1 = legacy Schicht verwenden) **Standardwert:** 0 in Datenbanken, die mit 4D v14 R5 oder neuer erstellt wurden, 1 in Datenbanken, die aus 4D v14 R4 oder früher konvertiert wurden.
SQL Server Port ID	Lange Ganzzahl	88	**Reichweite**: 4D im lokalen Modus und 4D Server. **Wird zwischen 2 Sitzungen beibehalten**: Ja **Beschreibung**:Setzt oder erhält die Nummer des TCP Port, den der in 4D integrierte SQL Server mit 4D im lokalen Modus oder 4D Server verwendet. Der Wert ist standardmäßig 19812. Die TCP Port Nummer lässt sich auch auf der Seite "SQL" in den Datenbank-Eigenschaften setzen. Mit diesem Selector wird die Datenbank-Eigenschaft entsprechend aktualisiert und bleibt auch nach Beenden und Neustart erhalten. **Mögliche Werte**: 0 bis 65535 **Standardwert**: 19812
Circular log limitation	Lange Ganzzahl	90	**Reichweite**: 4D lokal, 4D Server. **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: Jeder Wert vom Typ Zahl, 0 = alle Journale behalten **Beschreibung**: Maximale Anzahl der beizubehaltenden Journale pro Typ. Standardmäßig werden alle Dateien beibehalten. Übergeben Sie einen Wert *X*, werden nur die *letzten X* Dateien behalten, wobei bei Erstellen eines neuen Journals das älteste automatisch entfernt wird. Diese Einstellung gilt jeweils pro Journal für folgende Typen: request logs (Selector 28 und 45), debug log (Selector 34), events log (Selector 79), sowie Web request logs und Web debug logs (Selector 29 und 84 des Befehls **WEB SET OPTION**).

Konstante	Typ	Wert	Kommentar
Number of formulas in cache	Lange Ganzzahl	92	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Mögliche Werte**: Positive Lange Ganzzahl **Standardwert**: 0 (kein Cache) **Beschreibung**: Setzt oder erhält die maximale Anzahl Formeln, die im vom Befehl [#cmd id="63"/] verwendeten Formel-Cache beibehalten werden sollen. Diese Begrenzung gilt für alle Prozesse, jedoch hat jeder Prozess seinen eigenen Formel-Cache. Formeln im Cache zu halten, beschleunigt die Ausführung von **EXECUTE FORMULA** im kompilierten Modus, da jede dieser Formeln nur einmal tokenisiert wird (weitere Info siehe [#title id="8567"/]). Ändern Sie den Cache Wert, wird der vorhandene Inhalt zurückgesetzt, selbst wenn die neue Größe die bisherige übersteigt. Ist die Anzahl Formeln im Cache erreicht, ersetzt die neu ausgeführte Formel die älteste im Cache (FIFO Modus). Dieser Parameter wird nur in kompilierten Anwendungen oder Komponenten berücksichtigt.
Cache flush periodicity	Lange Ganzzahl	95	**Reichweite**: 4D local, 4D Server **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: Lange Ganzzahl > 1 (Sekunden) **Beschreibung**: Erhält oder setzt das Zeitintervall zum Sichern des aktuellen Cache in Sekunden. Eine Änderung dieses Werts überschreibt die Option **Daten-Cache sichern alle X Sekunden** auf der Seite **Seite Datenbank/ Speicher** der Datenbank Eigenschaften für die Sitzung (wird nicht in den Datenbank Eigenschaften gespeichert).
Tips enabled	Lange Ganzzahl	101	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 = Tipps deaktiviert, 1 = Tipps aktiviert (Standard) **Beschreibung:** Setzt oder erhält den aktuellen Status der Anzeige von Tipps für die 4D Anwendung. Tipps sind standardmäßig aktiviert. Beachten Sie, dass dieser Parameter alle 4D Tipps setzt, z.B. Hilfemeldungen für Formulare und Tipps im Editor des Designmodus.
Tips delay	Lange Ganzzahl	102	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** Lange Ganzzahl >= 0 (Ticks) **Beschreibung:** Verzögerte Anzeige von Tipps, nachdem der Mauszeiger in Objekten mit zugewiesenen Hilfemeldungen gesetzt wurde. Wert wird in Ticks gerechnet (1/60stel Sekunde). Standardwert ist 45 Ticks (0.75 Sekunde).
Tips duration	Lange Ganzzahl	103	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** Lange Ganzzahl >= 60 (Ticks) **Beschreibung:** Maximale Anzeigedauer für einen Tipp. Wert wird in Ticks gerechnet (1/60stel Sekunde). Standardwert ist 720 Ticks (12 Sekunden).
Min TLS version	Lange Ganzzahl	105	**Reichweite**: 4D Server, 4D Web Server und 4D SQL Server **Wird zwischen 2 Sitzungen beibehalten**: Nein **Beschreibung**: Gibt die Ebene von Transport Layer Security (TLS) für die Datenverschlüsselung und Authentifizierung zwischen Anwendungen und Servern an. Die definierten Werte gelten global für die Netzwerkebene. Ein geänderter Wert wird erst nach Neustart des Servers verwendet. Das standardmäßige Mindestprotokoll ist TLSv1_2. **Mögliche Werte**: • TLSv1_0 - TLS 1.0, eingeführt in 1999 als Upgrade von SSL v3.0. TLS 1.0 und SSL 3.0 funktioniere n nicht zusammen. • TLSv1_1 - TLS 1.1, eingeführt in 2006 als Update von TLS 1.0. Verbesserungen sind u.a. bessere Sicherheit und Fehlerverwaltung. • TLSv1_2 - TLS 1.2, eingeführt in 2008 als Update von TLS 1.1. Verbesserungen sind u.a. erhöhte Flexibilität, zusätzliche Unterstützung für authentifizierte Verschlüsselungen. **Hinweis**: Das Plug-In 4D Internet Commands nutzt eine andere Netzwerkschicht. Von daher hat dieser Selector keine Auswirkung auf deren TLS Version.
Times inside objects	Lange Ganzzahl	109	**Reichweite**: 4D lokal, 4D Server (alle Prozesse) **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: Times in seconds (0) (Standard), Times in milliseconds (1) **Beschreibung**: Definert, wie Zeitangaben konvertiert und in Objekteigenschaften und Elementen in Collections gespeichert werden, und wie sie in JSON und in Web Areas importiert bzw. daraus exportiert werden. Ab 4D v17 werden Zeitangaben in Objekten standardmäßig in Anzahl von Sekunden konvertiert und gespeichert. In bisherigen Releases wurden Zeitwerte in diesem Kontext als Anzahl von Millisekunden konvertiert und gespeichert. Dieser Selector unterstützt Sie beim Migrieren Ihrer Anwendungen und geht bei Bedarf zurück auf die bisherige Einstellung. **Hinweis**: ORDA Methoden und SQL Engine ignorieren diese Einstellung, sie gehen immer davon aus, dass Zeitwerte die Anzahl von Sekunden sind.

Beispiel

Mit folgender Methode erhalten Sie die aktuellen Werte des 4D Planers:

```
C_LONGINT($ticksbtwcalls;$maxticks;$minticks;$lparams)
If(Application type=4DLocal Mode)  ` 4D lokaler Modus wird verwendet
   $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 -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Text	Pfadname des neuesten Update Log

Beschreibung

Die Funktion **Get last update log path** gibt den kompletten Pfadnamen des neuesten Update Logbuchs auf dem Rechner zurück, wo es aufgerufen wird.

Das Update Logbuch wird von 4D während der automatischen Update Prozesse angelegt. Es zeigt Informationen zu den durchgeführten Updates sowie evtl. aufgetretene Fehler.

Diese Funktion wird beim automatischen Update Prozess für doppelklickbare Anwendungen (Server oder Einzelplatz) verwendet. Weitere Informationen dazu finden Sie im Abschnitt **Eigenständige Anwendung erstellen und weitergeben** des Handbuchs *4D Design*.

Get license info

Get license info -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Objekt	Angaben zur aktiven Lizenz

Beschreibung

Die Funktion **Get license info** gibt ein Objekt mit ausführlichen Angaben zur aktiven Lizenz zurück.
Wird die Funktion in einer 4D Anwendung ausgeführt, die nicht direkt eine Lizenz verwendet, wie z.B. 4D remote, gibt sie ein Null Objekt zurück.
Das zurückgegebene Objekt enthält folgende Eigenschaften:

```
{
    "name": String
    "licenseNumber": String
    "version": String
    "attributes": optional, Array mit Strings
    "userName": String
    "userMail": String
    "companyName": String
    "platforms": Array mit Strings
    "expirationDate": optional, Objekt
    "renewalFailureCount": optional, Anzahl
    "products":[ //für jedes registrierte Expansion Produkt
        {
            "id": Zahl
            "name": String
            "usedCount": Zahl
            "allowedCount": Zahl
            "rights": [
                {
                    "count": Zahl
                    "expirationDate" optional, Objekt
                }
            ]
        }
    ]
}
```

Name der Eigenschaft	Beschreibung	Beispiel
name	Produktname	"4D Developer Professional v16"
licenseNumber	Lizenznummer	"4DDP16XXXXX1123456789"
version	Versionsnummer des Produkts	"16", "16R2"
attributes	Lizenztyp(en), falls zutreffend (optional)	["Anwendung","OEM"]
userName	Name des 4D Store Account	"John Smith"
userMail	Mail des 4D Store Account	"john.smith@gmail.com"
companyName	Firmenname des 4D Store Account	"Alpha Cie"
platforms	Plattform(en) der Lizenz	["macOS", "windows"]
expirationDate	Ablaufdatum (optional)	{"day":2, "month":6, "year":2018}
renewalFailureCount	Anzahl erfolgloser automatischer Versuche zum Erneuern für mindestens eine der Produktlizenzen (optional)	3
products	Beschreibung der Produktlizenzen (Array mit Objekten, ein Element pro Produktlizenz)	
id	Lizenznummer	Für verfügbare Werte, siehe Funktion **Is license available**
name	Lizenzname	"4D Write - 4D Write Pro"
usedCount	Anzahl der genutzten Verbindungen	8
allowedCount	Insgesamt zugelassene Verbindungen für das Produkt bis zum Ablaufdatum	15
rights	Rechte für das Produkt (Array mit Objekten, ein Element pro Ablaufdatum)	
count	Anzahl zugelassene Verbindungen	15 (32767 bedeutet unbegrenzt)
expirationDate	Ablaufdatum (gleiches Format wie oben)	{"day":1, "month":11, "year":2017}

Beispiel

Sie wollen Informationen über Ihre aktuelle 4D Server Lizenz erhalten:

```
C_OBJECT($obj)
$obj:=Get license info
```

$obj kann beispielsweise folgendes enthalten:

{ "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} //abgelaufen, wird nicht gezählt }], "usedCount": 12 }, {...}] }

GET SERIAL INFORMATION

GET SERIAL INFORMATION (Schlüssel ; Benutzer ; Firma ; Angemeldet ; MaxBenutzer)

Parameter	Typ		Beschreibung
Schlüssel	Lange Ganzzahl	⇐	Einmalige Produktnummer (verschlüsselt)
Benutzer	String	⇐	Registrierter Name
Firma	String	⇐	Registrierte Organisation
Angemeldet	Lange Ganzzahl	⇐	Anzahl der angemeldeten Benutzer
MaxBenutzer	Lange Ganzzahl	⇐	Max. Anzahl der angemeldeten Benutzer

Beschreibung

Der Befehl **GET SERIAL INFORMATION** gibt verschiedene Informationen über die Seriennummer der aktuellen 4D Version zurück.

- *Schlüssel*: Einmalige Kennummer des installierten Produkts. Der 4D Anwendung, die auf dem Rechner installiert ist, z.B. 4D Server, 4D im lokalen Modus, 4D Desktop, etc, wird eine einmalige Nummer zugeordnet, die natürlich verschlüsselt ist.
- *Benutzer*: Name des Benutzers der Anwendung, wie er bei der Installation definiert wurde.
- *Firma*: Firma oder Organisation des Benutzers, wie sie bei der Installation definiert wurde.
- *Angemeldet*: Anzahl der angemeldeten Benutzer beim Ausführen des Befehls
- *MaxBenutzer*: Maximale Anzahl der gleichzeitig angemeldeten Benutzer.

Die beiden letzten Parameter geben für den 4D Einzelplatz immer 1 zurück, mit Ausnahme der Demo-Versionen. Hier wird 0 zurückgegeben.

Der Befehl **GET SERIAL INFORMATION** ist Teil des allgemeinen Sicherungssystems für Komponenten. Entwickler von Komponenten können einer bestimmten installierten 4D Anwendung eine Kopie ihres Produkts zuordnen, um so illegales Kopieren zu verhindern.
Die Serialisierung funktioniert folgendermaßen: Ein Benutzer, der eine 4D Komponente erhalten möchte, sendet dem Entwickler seinen einmaligen Schlüssel, der mit dem Befehl **GET SERIAL INFORMATION** erstellt wurde. Das kann über ein Bestellformular erfolgen, das in einer Demo-Version der 4D Komponente enthalten ist. Der erzeugte Schlüssel ist einmalig und einem bestimmten 4D Produkt zugeordnet.
Der Entwickler der Komponente kann nun seine eigene Seriennummer erzeugen. Dazu kombiniert er den Schlüssel mit einer bestimmten Verschlüsselung. Die gelieferte Komponente enthält eine Funktion, die prüft, ob die Information, die über den Befehl **GET SERIAL INFORMATION** zurückgegeben wird, zu dieser Seriennummer passt. Andernfalls kann der Benutzer die Komponente nicht einsetzen.

Hinweis: Auch Entwickler von Plug-Ins können dieses Schutzsystem verwenden. Weitere Informationen dazu finden Sie im Handbuch *4D Plugin API Reference*.

Get table fragmentation

Get table fragmentation (Tabelle) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabelle	Tabelle	⇒	Tabelle, deren Fragmentierung zurückgegeben werden soll
Funktionsergebnis	Zahl	⇐	Prozentsatz der Fragmentierung

Beschreibung

Die Funktion **Get table fragmentation** gibt den Prozentsatz der logischen Fragmentierung für die Datensätze der Tabelle, definiert in *Tabelle*, zurück.

Die Rate der logischen Fragmentierung der Datensätze gibt an, ob die Datensätze in geordneter Weise in der Datendatei gespeichert sind. Wird die Fragmentierung zu hoch, kann das Sortierungen und sequentielles Suchen in der Tabelle erheblich verlangsamen. Eine Fragmentierungsrate von 0 % entspricht keiner Fragmentierung. Ab einer Rate von 20 % ist es vorteilhaft, die Datendatei zu komprimieren.

Beispiel

Mit der folgenden Wartungsmethode können Sie das Komprimieren der Datendatei anfordern, wenn in mindestens einer Tabelle der Anwendung eine beträchtliche Fragmentierung vorliegt:

```
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)
  // Setzt eine Marke, die Komprimierung anfordert
End if
```

Is compiled mode

Is compiled mode {(*)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Gibt Information über die Host Datenbank zurück
Funktionsergebnis	Boolean	➲	Kompiliert (Wahr), Interpretiert (Falsch)

Beschreibung

Die Funktion **Is compiled mode** überprüft, ob Sie im kompilierten Modus (Wahr) oder im interpretierten Modus (Falsch) arbeiten.

Der optionale Parameter * ist sinnvoll beim Arbeiten mit Komponenten:

- Wird die Funktion von einer Komponente aus aufgerufen:
  - Ist der Parameter * übergeben, gibt die Funktion, je nach Ausführungsmodus der Host Datenbank, Wahr oder Falsch zurück.
  - Ist der Parameter * nicht übergeben, gibt die Funktion, je nach Ausführungsmodus der Komponente, Wahr oder Falsch zurück
- Wird die Funktion über eine Methode einer Host Datenbank aufgerufen, gibt sie immer, je nach Ausführungsmodus der Host Datenbank, Wahr oder Falsch zurück.

Beispiel

In eine Ihrer Routinen fügen Sie einen Fehlercode ein, der nur von Bedeutung ist, wenn Sie im interpretierten Modus arbeiten. Verwenden Sie dafür einen Test, der die Funktion **Is compiled mode** aufruft:

```
` ...
If(Not(Is compiled mode))
` Include debugging code here
End if
` ...
```

Is data file locked

Is data file locked -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		Wahr = Datei/Segment gesperrt Falsch = Datei/Segment nicht gesperrt

Beschreibung

Die Funktion **Is data file locked** gibt Wahr zurück, wenn die Datendatei der geöffneten Datenbank oder mindestens ein Segment davon gesperrt ist — z.B. schreibgeschützt.

Liegt diese Funktion in der **Datenbankmethode On Startup**, verhindert sie, dass versehentlich eine gesperrte Datendatei geöffnet wird.

Beispiel

Diese Methode verhindert das Öffnen der Datenbank, wenn die Datendatei gesperrt ist:

```
If(Is data file locked)
   ALERT("Datendatei ist gesperrt. Datenbank kann nicht geöffnet werden.")
   QUIT 4D
End if
```

Is license available

Is license available {(Lizenz)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Lizenz	Lange Ganzzahl	➡	Produkt/Plug-In, für das die Gültigkeit der Lizenz geprüft werden soll.
Funktionsergebnis	Boolean	⮌	Wahr, wenn Produkt/Plug-In verfügbar ist, sonst Falsch

Beschreibung

Diese Funktion prüft, ob ein Produkt oder Plug-In zur Verfügung steht. Das ist zum Beispiel hilfreich, um Funktionen, für die ein Plug-In notwendig ist, ein- oder auszublenden.

Es gibt folgende Arten, die Funktion **Is license available** einzusetzen:

- Der Parameter *Lizenz* ist nicht angegeben: In diesem Fall gibt die Funktion *Falsch* zurück, wenn das 4D Programm im Demo-Modus läuft.
- Im Parameter *Lizenz* übergeben Sie eine der Konstanten unter dem Thema **Is license available**:

Konstante	Typ	Wert
4D Client SOAP license	Lange Ganzzahl	808465465
4D Client Web license	Lange Ganzzahl	808465209
4D for OCI license	Lange Ganzzahl	808465208
4D Mobile license	Lange Ganzzahl	808464439
4D Mobile Test license	Lange Ganzzahl	808465719
4D ODBC Pro license	Lange Ganzzahl	808464946
4D SOAP license	Lange Ganzzahl	808465464
4D View license	Lange Ganzzahl	808465207
4D Web license	Lange Ganzzahl	808464945
4D Write license	Lange Ganzzahl	808464697

  Die Funktion gibt *Wahr* zurück, wenn für das entsprechende Plug-In eine Lizenz verfügbar ist. Sie berücksichtigt alle Lizenznummern, die im Designmodus oder über den Befehl **SET PLUGIN ACCESS** zugewiesen wurden.
  Is license available gibt Falsch zurück, wenn das Plug-In im Demo-Modus arbeitet.
- Die Kennummer der Plug-In Ressource "4BNX" übergeben Sie direkt im Parameter *Lizenz*. Die Funktion arbeitet genauso wie oben beschrieben.

NOTIFY RESOURCES FOLDER MODIFICATION

NOTIFY RESOURCES FOLDER MODIFICATION
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **NOTIFY RESOURCES FOLDER MODIFICATION** weist 4D Server an, alle angemeldeten 4D Rechner zu benachrichtigen, dass der Ordner **Resources** der Datenbank geändert wurde, so dass diese ihre lokalen Ordner **Resources** anpassen können.

Dieser Befehl eignet sich besonders, um die Synchronisation des Ordners **Resources** auf anderen Rechnern zu verwalten, wenn der Ordner durch eine Serverprozedur auf dem Server geändert wurde. Weitere Informationen dazu finden Sie im Abschnitt **Serverprozeduren** des *4D Server Handbuchs*.

Es wird nur die Nachricht gesendet, dass sich der Inhalt des Ordners geändert hat. Jeder angemeldete Rechner reagiert darauf gemäß seinen aktuellen Einstellungen. Es gibt folgende Optionen:

- Keine Synchronisation des lokalen Ordners **Resources** während der Arbeitssitzung,
- Automatische Synchronisation des lokalen Ordners **Resources** während der Arbeitssitzung,
- Anzeigen einer Meldung, so dass der Benutzer die Synchronisation, falls gewünscht, ausführen kann.

Aktuelle Einstellungen werden jeweils definiert:

- allgemein über den Parameter **'Resources'-Ordner während einer Session aktualisieren** auf der Seite Client-Server der Datenbank-Eigenschaften. In diesen Fall gelten sie für alle remote Rechner;
- lokal über den auf dem remote Rechner ausgeführten Befehl **SET DATABASE PARAMETER** (Selector Auto Synchro Resources Folder). In diesen Fall überschreiben sie die Datenbank-Eigenschaften und gelten nur für die Dauer der Arbeitssitzung auf dem remote Rechner.

OPEN ADMINISTRATION WINDOW

OPEN ADMINISTRATION WINDOW
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **OPEN ADMINISTRATION WINDOW** zeigt das Server Verwaltungsfenster auf dem Rechner mit remote 4D, der auf den Server zugreift. Im 4D Server Verwaltungsfenster lassen sich die aktuellen Parameter ansehen und verschiedene Wartungsoperationen ausführen (siehe Handbuch *4D Server* im Abschnitt **Seite Wartung**). Dieses Fenster lässt sich ab Version 11 von 4D Server auch auf dem Client Rechner anzeigen.

Dieser Befehl muss im Rahmen eines 4D Programms aufgerufen werden, das an einen 4D Server angemeldet ist. Er führt nichts aus, wenn:

- Er in einem 4D Programm im lokalen Modus ausgeführt wird,
- Von einem Benutzer, der weder Designer oder Administrator ist, ausgeführt wird. In diesem Fall wird der Fehler -9991 erzeugt (siehe Abschnitt **Fehler der Datenbank (-10602 -> 4004)**).

Beispiel

Dieser Code lässt sich einer Verwaltungsschaltfläche zuweisen:

```
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
```

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, andernfalls auf 0 (Null).

OPEN DATA FILE

OPEN DATA FILE (Zugriffspfad)

Parameter	Typ		Beschreibung
Zugriffspfad	String	➡	Name oder kompletter Zugriff der zu öffnenden Datendatei

Beschreibung

Der Befehl **OPEN DATA FILE** ermöglicht, die von der 4D Anwendung geöffnete Datendatei „on-the-fly" zu wechseln.

Im Parameter *Zugriffspfad* übergeben Sie den Namen bzw. den kompletten Zugriffspfad der zu öffnenden Datendatei (Datei mit der Endung ".4DD"). Übergeben Sie nur den Namen der Datei, muss sie neben der Strukturdatei der Datenbank liegen.
Setzt der Zugriffspfad eine gültige Datendatei, verlässt 4D die gerade aktive Datenbank und öffnet sie erneut mit der angegebenen Datendatei. Die **Semaphore** und **Datenbankmethode On Startup** werden der Reihe nach aufgerufen.

Warnung: Da dieser Befehl die Anwendung beendet, ehe sie wieder mit der angegebenen Datendatei geöffnet wird, darf er nur mit Bedacht in der **Datenbankmethode On Startup** bzw. einer Methode verwendet werden, die von dieser Datenbankmethode aufgerufen wird, damit es nicht zu einer unendlichen Schleife kommt.

Der Befehl wird in asynchroner Weise ausgeführt, d.h. nach dem Aufruf führt 4D die verbleibende Methode aus. Dann verhält sich die Anwendung wie beim Befehl **Beenden** im Menü **Datei/Ablage**

Geöffnete Dialogfenster werden geschlossen, alle offenen Prozesse werden innerhalb von 10 Sekunden beendet, etc.
Vor dem Start der Operation prüft der Befehl, ob die angegebene Datei gültig ist. Auch wenn die Datei bereits offen ist, prüft der Befehl, ob sie zur aktuellen Strukturdatei passt.

Übergeben Sie in *Zugriffspfad* einen leeren String, öffnet **OPEN DATA FILE** die Datenbank erneut, ohne die Datendatei zu ändern.

4D Server: Ab Version 13 können Sie diesen Befehl mit 4D Server ausführen. In diesem Kontext ruft er vor dem Erstellen der angegebenen Datei intern **QUIT 4D** auf dem Server auf und zeigt auf jedem remote Rechner ein Dialogfenster mit der Meldung an, dass der Server gerade verlassen wird.

Beispiel

In einer doppelklickbaren Anwendung zur Weitergabe die Benutzerdatendatei in der **Datenbankmethode On Startup** öffnen oder anlegen. Dieses Beispiel verwendet die standardmäßige Datendatei (siehe **Standard Datendatei in doppelklickbaren Anwendungen verwalten**):

```
If(Data file="@default.4dd")
   If(Version type?? Merged application)
      If(Is data file locked)
         $dataPath:=Get 4D folder(Active 4D Folder)+"data.4dd"
 //Gibt es bereits eine lokale Datendatei
         If(Test path name($dataPath)=Is a document)
            OPEN DATA FILE($dataPath)  //Datei öffnen
         Else
            CREATE DATA FILE($dataPath)  //Datei anlegen
         End if
      End if
   End if
End if
```

OPEN DATABASE

OPEN DATABASE (DateiPfad)

Parameter	Typ		Beschreibung
DateiPfad	String	➡	Name oder kompletter Zugriffspfad der zu öffnenden Datenbankdatei (.4db, 4dc, .4dbase oder .4dlink)

Beschreibung

Der Befehl **OPEN DATABASE** schließt die aktuelle 4D Anwendung und öffnet direkt die im Parameter *DateiPfad* angegebene Anwendung. Dieser Befehl ist hilfreich für automatische Testabläufe oder, um eine Anwendung nach dem Kompilieren automatisch wieder zu öffnen.

Im Parameter *DateiPfad* übergeben Sie den Namen oder den kompletten Zugriffspfad der zu öffnenden Anwendung. Sie können Dateien mit einer der folgenden Endungen verwenden:

- .4db (interpretierte Strukturdatei)
- .4dc (kompilierte Strukturdatei)
- .4dbase (OS X Package)
- .4dlink (Datei mit Tastenkürzeln)

Übergeben Sie nur den Dateinamen, muss er auf derselben Ebene wie die Strukturdatei der aktuellen Anwendung liegen.

Ist der Zugriffspfad gültig, beendet 4D die laufende Anwendung und öffnet die angegebene Anwendung. Im Einzelplatz werden die Methoden **Semaphore** der geschlossenen Anwendung und **Datenbankmethode On Startup** der geöffneten Anwendungen nacheinander aufgerufen.

Warnung: Da dieser Befehl die Anwendung beendet, bevor sie mit der angegebenen Datei erneut geöffnet wird, raten wir davon ab, ihn in der Methode **Datenbankmethode On Startup** zu verwenden oder in einer Methode, die diese Datenbankmethode aufruft.

Der Befehl wird asynchron ausgeführt, d.h. nach dem Aufrufen führt 4D erst die restliche Methode aus. Dann verhält sich die Anwendung wie beim Schließen über den Befehl **Schließen** im Menü **Datei**: Geöffnete Dialogfenster werden geschlossen, geöffnete Prozesse beenden innerhalb von 10 Sekunden, etc.

Wird die Zielanwendung nicht gefunden oder ist sie ungültig, wird ein standardmäßiger Systemfehler zurückgegeben und 4D führt nichts aus.

Dieser Befehl lässt sich nur von einer nicht eingebundenen Anwendung ausführen. Beim Ausführen in einer Applikation mit einkompilierter Engine (Einzelplatz oder Server) wird der Fehler -10509 "Die Anwendung "" lässt sich nicht öffnen" zurückgegeben.

Beispiel

```
OPEN DATABASE("C:\\databases\\Invoices\\Invoices.4db")
```

OPEN SECURITY CENTER

OPEN SECURITY CENTER
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **OPEN SECURITY CENTER** zeigt das Dialogfenster „Maintenance und Security Center"(MSC) an.

Je nach den Zugriffsrechten des aktuellen Benutzers können einige Funktionen in diesem Fenster inaktiv sein.

Hinweis: Dieser Befehl funktioniert genauso wie Aufrufen von **DIALOG** mit dem Parameter *: Das MSC wird in einem Fenster angezeigt und der Befehl gibt sofort die Steuerung des 4D Code zurück. Endet der aktuelle Prozess, wird das Fenster automatisch durch Simulieren von **CANCEL** geschlossen. Folglich müssen Sie seine Anzeige über den Code des ausführenden Prozesses verwalten.

OPEN SETTINGS WINDOW

OPEN SETTINGS WINDOW (Selector {; Zugriff}{; EinstellungenTyp})

Parameter	Typ		Beschreibung
Selector	String	➡	Pfad eines Themas, einer Seite oder einer Parametergruppe in den Dialogfenstern Einstellungen oder Datenbank-Eigenschaften
Zugriff	Boolean	➡	Wahr=die anderen Seiten des Dialogfensters sperren, Falsch oder weggelassen=die anderen Seiten des Dialogfensters aktiv lassen.
EinstellungenTyp	Lange Ganzzahl	➡	0 oder weggelassen = Struktur-Einstellungen, 1 = Benutzer-Einstellungen

Beschreibung

Der Befehl **OPEN SETTINGS WINDOW** öffnet das Dialogfenster 4D Einstellungen oder die Datenbank-Eigenschaften der aktuellen 4D Anwendung und zeigt die Parameter oder die Seite, deren Schlüssel in *Selector* übergeben wurde.

Selector muss einen Schlüssel enthalten, der das Dialogfenster und die zu öffnende Seite angibt. Der Schlüssel setzt sich folgendermaßen zusammen: */Thema{/Seite{/Parametergruppe}}*. Thema gibt das entsprechende Dialogfenster an: Sie können "4D" (für Einstellungen) oder "Database" (für Datenbank-Eigenschaften) übergeben. Wollen Sie z.B. die Seite „Compiler" der Datenbank-Eigenschaften angeben, muss *Selector* enthalten *"/Database/Compiler"*. Die Liste der verwendbaren Schlüssel finden Sie unten. Wollen Sie direkt die erste Seite des Dialogfensters öffnen, übergeben Sie in *Selector* nur das Zeichen "/".

Mit dem Parameter *Zugriff* können Sie Benutzeraktionen im Dialogfenster 4D Einstellungen oder Datenbank-Eigenschaften steuern, indem Sie die anderen Seiten sperren. Ein typisches Beispiel ist, dass Sie für den Benutzer festlegen wollen, dass er bestimmte Parameter anpassen kann, andere dagegen nicht. Dazu übergeben Sie für *Zugriff* *Wahr*, denn dann ist nur die im Parameter *Selector* angegebene Seite aktiv und änderbar, während der Zugriff auf alle anderen Seiten gesperrt ist (Klicken auf die Schaltflächen in der Navigationsleiste bleibt ohne Wirkung). Übergeben Sie *Falsch* oder lassen den Parameter weg, sind alle Seiten des Dialogfensters ohne Einschränkung zugänglich.

Der Parameter *EinstellungenTyp* wird nur in Anwendungen berücksichtigt, für die der Modus "Benutzer Einstellungen" aktiviert wurde. In diesem Modus werden eigene Benutzereinstellungen oder Benutzereinstellungen für Datendatei in einer externen Datei erstellt und anstelle der Standardeinstellungen verwendet. Weitere Informationen dazu finden Sie im Kapitel **Einstellungen** des Handbuchs *4D Designmodus*. In diesem Kontext können Sie über den Parameter *EinstellungenTyp* angeben, ob Sie auf das Dialogfenster "Struktur-Einstellungen", "Benutzer-Einstellungen" oder "Benutzereinstellungen für Datendatei" zugreifen wollen. Sie können eine der foglenden Konstanten unter dem Thema **4D Umgebung** angeben:

Konstante	Typ	Wert	Kommentar
Structure settings	Lange Ganzzahl	0	Zugriff auf die "Struktur-Einstellungen" (Standardwert, wenn Parameter weggelassen). In diesem Modus sind die für *Selector* verwendeten Pfade identisch mit denen im Standardmodus.
User settings	Lange Ganzzahl	1	Zugriff auf "Benutzer-Einstellungen". In diesem Modus sind im Parameter *Selector* nur bestimmte Pfade möglich.
User settings for data	Lange Ganzzahl	2	Zugriff auf "Benutzereinstellungen für Datendatei", d.h. Benutzereinstellungen auf derselben Ebene wie die Datendatei gespeichert. In diesem Modus lassen sich im Parameter *Selector* nur bestimmte Pfade verwenden (gleiche Untermenge wie Benutzereinstellungen)

Übergeben Sie einen ungültigen Schlüssel, erscheint die erste Seite des Dialogfensters Datenbank-Eigenschaften.

Pfade (Standardmodus)

Im Parameter *Selector* sind standardmäßig folgende Pfade möglich, d.h. in den Struktureinstellungen. Der erste Teil gibt das jeweilige Dialogfenster an:
/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

Hinweis zur Kompatibilität: Der Befehl funktioniert weiterhin mit den Pfaden für 4D Versionen 11.x oder früher; 4D stellt automatisch die Entsprechung her. Wir empfehlen jedoch, ältere Aufrufe durch die oben aufgelisteten Pfade zu ersetzen.

Pfade (Benutzer-Einstellungen)

Im Modus "Benutzer-Einstellungen" können Sie in Selector einen der folgenden Pfade übergeben:

/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

Beispiel 1

Die Seite "Methoden" der 4D Einstellungen öffnen:

```
OPEN SETTINGS WINDOW("/4D/Method editor")
```

Beispiel 2

Die Parameter "Abkürzungen" in den Datenbank-Eigenschaften öffnen und die anderen Einstellungen sperren:

```
OPEN SETTINGS WINDOW("/Database/Interface/Shortcuts";True)
```

Beispiel 3

Die erste Seite der Datenbank-Eigenschaften öffnen:

```
OPEN SETTINGS WINDOW("/")
```

Beispiel 4

Im Modus "Benutzer-Einstellungen" auf die Seite Oberfläche der Datenbank-Eigenschaften zugreifen:

```
OPEN SETTINGS WINDOW("/Database/Interface";1)
```

Systemvariablen und Mengen

Wird das Dialogfenster Einstellungen bzw. Datenbank-Eigenschaften bestätigt, gibt die Systemvariable OK den Wert 1 zurück, sonst den Wert 0.

PLUGIN LIST

PLUGIN LIST (ArrayZahl ; ArrayName)

Parameter	Typ		Beschreibung
ArrayZahl	Array Lange Ganzzahl	⇐	Nummern der Plug-Ins
ArrayName	Array String	⇐	Namen der Plug-Ins

Beschreibung

Der Befehl **PLUGIN LIST** füllt die Arrays *ArrayZahl* und *ArrayName* mit der Nummer und den Namen der Plug-Ins, die von der 4D Anwendung geladen und verwendbar sind. Die beiden Arrays werden vom Befehl automatisch angepasst und synchronisiert.

Hinweis: Sie können die in *ArrayZahl* zurückgegebenen Werte mit den Konstanten unter dem Thema **Is license available** vergleichen.

PLUGIN LIST berücksichtigt alle Plug-Ins, inkl. der in 4D integrierten, wie z.B. 4D Chart, und Plug-Ins von Drittherstellern.

QUIT 4D

QUIT 4D {(Zeit)}

Parameter	Typ		Beschreibung
Zeit	Lange Ganzzahl	➡	Zeit (Sekunden) bis zum Beenden des Server

Beschreibung

QUIT 4D verlässt die aktuelle 4D Anwendung und kehrt zum Desktop zurück.
Der Befehl arbeitet unterschiedlich, je nachdem, ob er in 4D (lokal oder remote) oder 4D Server ausgeführt wird.

Mit 4D im lokalen und im remote Modus

Nach Aufrufen von **QUIT 4D** stoppt der aktuelle Prozess die Ausführung. 4D geht dann folgendermaßen vor:

- Gibt es eine **Semaphore**, startet 4D diese Methode in einem neuen lokalen Prozess. Mit dieser Datenbankmethode informieren Sie z.B. andere Prozesse per Interprozesskommunikation, zu beenden (Dateneingabe) oder die Ausführung von Operationen zu stoppen, die mit der **Datenbankmethode On Startup** gestartet wurden. (Anbindung von 4D an einen anderen Datenbank-Server). Beachten Sie, dass evtl. 4D beendet. Die **Semaphore** kann alle gewünschten Operationen zum Bereinigen oder Schließen ausführen, jedoch nicht das Beenden an sich verweigern. Sie wird also an einem bestimmten Punkt enden.
- Gibt es keine **Semaphore**, bricht 4D jeden laufenden Prozess der Reihe nach und ohne Unterscheidung ab.

Gibt der Benutzer gerade Daten ein, werden die Datensätze ohne Sicherung abgebrochen.
Soll der Benutzer die Möglichkeit haben, Änderungen bei der Dateneingabe im gerade offenen Fenster zu sichern, können Sie allen Benutzerprozessen per Interprozesskommunikation signalisieren, dass die Datenbank geschlossen wird. Sie haben folgende Möglichkeiten:

- Diese Operationen vor Aufrufen von **QUIT 4D** im aktuellen Prozess auszuführen.
- Diese Operationen über die **Semaphore** zu verwalten.

Eine weitere Möglichkeit wäre, vor Aufrufen von **QUIT 4D** zu prüfen, ob ein Fenster bestätigt werden muss. In diesem Fall fordern Sie den Benutzer auf, die Fenster zu bestätigen oder abzubrechen und dann erneut Beenden auszuwählen. Die beiden ersten Möglichkeiten sind jedoch benutzerfreundlicher.
Hinweis: Der Parameter *Zeit* lässt sich nicht mit 4D im lokalen oder remote Modus verwenden.

Mit 4D Server (Serverprozedur)

Der Befehl **QUIT 4D** lässt sich in einer Serverprozedur auf dem Server-Rechner ausführen. Er erlaubt den optionalen Parameter *Zeit*.
Mit dem Parameter *Zeit* setzen Sie ein Timeout zum Beenden des 4D Servers, damit die Client-Rechner sich abmelden können. Sie geben den Wert in Sekunden an.
Dieser Parameter wird nur bei Ausführung auf dem Server-Rechner berücksichtigt. Bei 4D im lokalen oder remote Modus wird er ignoriert.
Übergeben Sie in *Zeit* keinen Wert, wartet 4D Server ab, bis alle Client-Rechner abgemeldet sind.
Die Ausführung von **QUIT 4D** auf dem Server läuft im Gegensatz zu 4D asynchron, d.h. die Methode, in welcher der Befehl aufgerufen wird, wird nach der Ausführung nicht unterbrochen.
Die **Datenbankmethode On Server Shutdown** wird – sofern vorhanden – entweder nach dem im Parameter *Zeit* festgelegten Timeout oder nach Abmelden aller Clients ausgeführt.
Der Befehl **QUIT 4D** in einer Serverprozedur funktioniert wie der Befehl **Beenden** des Menüs **Datei/Ablage** von 4D Server: Auf jedem Client-Rechner erscheint ein Dialogfenster mit der Meldung, dass der Server abgeschaltet wird.

Beispiel

Nachfolgende Projektmethode ist dem Menübefehl **Beenden** aus dem Menü **Datei/Ablage** zugewiesen.

```
` Projektmethode M_FILE_QUIT

CONFIRM("Sie wollen wirklich beenden?")
If(OK=1)
   QUIT 4D
End if
```

RESTART 4D

RESTART 4D {(Zeit {; Meldung})}

Parameter	Typ		Beschreibung
Zeit	Lange Ganzzahl	➡	Zeitversatz (Sekunden) vor Neustart von 4D
Meldung	String	➡	Text zum Anzeigen auf Client Rechnern

Beschreibung

Der Befehl **RESTART 4D** startet erneut die aktuelle 4D Applikation.

Dieser Befehl dient hauptsächlich zum Einsatz für eine doppelklickbare Anwendung (Client/Server oder Einzelplatz mit einkompilierter 4D Volume Desktop) in Verbindung mit dem Befehl **SET UPDATE FOLDER**. In diesem Fall wird der automatische Update Prozess gestartet, d.h. die neue Version der Anwendung, die im Befehl **SET UPDATE FOLDER** angegeben ist, ersetzt automatisch die aktuelle Version beim Neustart über den Befehl **RESTART 4D**. Der Pfadname der Datendatei wird gesichert und automatisch verwendet.

Ist über den Befehl **SET UPDATE FOLDER** in der aktuellen Sitzung keine Update Information angegeben, startet der Befehl die 4D Anwendung einfach mit der aktuellen Struktur- und Datendatei.

Über den Parameter *Zeit* können Sie den Neustart der Anwendung verzögern, um den Client-Rechnern Zeit zum Abmelden zu geben. Sie müssen die Zeitverzögerung in Sekunden angeben. Ohne diesen Parameter wartet die Server Anwendung maximal 10 Minuten auf das Abmelden aller Client-Anwendungen. Danach werden alle Client-Anwendungen automatisch abgemeldet.

Hinweis: Die Parameter *Zeit* und *Meldung* werden nur für Server-Anwendungen berücksichtigt. In Einzelplatz- oder remote-Anwendungen werden sie ignoriert.

Der optionale Parameter *Meldung* zeigt eine eigene Meldung für die angemeldeten Client-Anwendungen.

Bei korrekt ausgeführtem Befehl wird die Systemvariable auf 1 gesetzt; andernfalls auf 0 und die Anwendung startet neu. Durch diesen Befehl generierte Fehler können Sie über eine Methode abfangen, die über den Befehl **ON ERR CALL** aufgerufen wird.

SET DATABASE LOCALIZATION

SET DATABASE LOCALIZATION (SprachCode {; *})

Parameter	Typ		Beschreibung
SprachCode	Text	➡	Sprach-Selector
*	Operator	➡	Reichweite des Befehls

Beschreibung

Der Befehl **SET DATABASE LOCALIZATION** ermöglicht, die aktuelle Sprache der Anwendung für die aktuelle Sitzung zu ändern.
Über die aktuelle Sprache der Anwendung wird der Ordner .lproj spezifiziert, in dem das Programm nach den lokalisierten Elementen der Anwendung, also Text und Bilder, sucht. Standardmäßig bestimmt 4D die Sprache automatisch gemäß dem Inhalt des Ordners **Resources** und der Systemumgebung. Weitere Informationen dazu finden Sie unter der Funktion **Get database localization**. Mit dem Befehl **SET DATABASE LOCALIZATION** können Sie die vorgegebene aktuelle Sprache ändern.

Der Befehl ändert nicht die Sprache von bereits geladenen Formularen. Nur Elemente, die nach Aufrufen des Befehls angezeigt werden, berücksichtigen die neue Konfiguration.

Im Parameter *SprachCode* übergeben Sie die Sprache für Ihre Anwendung mit den Sprach-ID für RFC 3066, ISO639 und ISO3166. Gängiger Standard ist "fr" für Französisch, "es" für Spanisch, "en-us" für amerikanisches Englisch. Weitere Informationen dazu finden Sie im Abschnitt **Anhang C: XLIFF Architektur** des Handbuchs *4D Designmodus*.

Der Befehl gilt standardmäßig für alle geöffneten Anwendungen und Komponenten in allen Prozessen. Ist der optionale Parameter * übergeben, gilt der Befehl nur für die Anwendung, die ihn aufgerufen hat. Über diesen Parameter lassen sich insbesondere die Sprache der Anwendung und die der Komponente separat angeben.

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, andernfalls auf 0.

Hinweis: Der Befehl verwendet in Übereinstimmung mit dem RFC den Bindestrich "-" zum Trennen zwischen Sprachcode und Regionscode, z.B. "fr-ca" oder "en-us". Dagegen verwenden die Unterordner "lproj" im Ordner **Resources** den Unterstrich "_" , zum Beispiel "fr_ca.lproj" oder "en_us.lproj".

4D Server: Mit 4D Server sind die Sprachen verfügbar, die auf dem remote Rechner liegen, der den Befehl aufgerufen hat. Sie müssen deshalb sicherstellen, dass die Ordner **Resources** aufeinander abgestimmt sind

Beispiel 1

Wir legen Französisch als Sprache der Oberfläche fest:

```
SET DATABASE LOCALIZATION("fr")
```

Beispiel 2

Die Oberfläche Ihrer Anwendung verwendet den statischen String ":xliff:shopping". Die XLIFF Dateien enthalten dann folgende Information:

- Ordner FR:

```
<trans-unit id="15" resname="Shopping"> <source>Shopping</source> <target>Faire les courses</target> </trans-unit>
```

- Ordner FR_CA:

```
<trans-unit id="15" resname="Shopping"> <source>Shopping</source> <target>Magasiner</target> </trans-unit>
```

```
SET DATABASE LOCALIZATION("fr")
  //the string ":xliff:shopping" zeigt "Faire les courses"
SET DATABASE LOCALIZATION("fr-ca")
  //the string ":xliff:shopping" zeigt "Magasiner"
```

SET DATABASE PARAMETER

SET DATABASE PARAMETER ({Tabellenname ;} Selector ; Wert)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Setzen der Parameter, ohne Angabe Standardtabelle
Selector	Lange Ganzzahl	⇒	Code des zu ändernden Parameters der Datenbank
Wert	Zahl, String	⇒	Wert des Parameters

Beschreibung

Der Befehl **SET DATABASE PARAMETER** ändert verschiedene interne Parameter der 4D Datenbank.

Selector bezeichnet den Code der zu ändernden Parameter der Datenbank. 4D bietet unter dem Thema **Datenbankparameter** vordefinierte Konstanten. Nachfolgende Liste zeigt die Konstanten, ihre Reichweite und gibt an, ob Änderungen zwischen zwei Sitzungen beibehalten werden:

Konstante	Typ	Wert	Kommentar
Minimum Web process	Lange Ganzzahl	6	**Reichweite:** 4D lokal, 4D Server **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Werte: 0 -> 32 767 **Beschreibung:** Mindestanzahl der zu unterhaltenden Web Prozesse im nicht kontextuellen Modus mit 4D im lokalen Modus und 4D Server. Standardmäßig ist der Wert 0 (siehe unten).
Maximum Web process	Lange Ganzzahl	7	**Reichweite:** 4D lokal, 4D Server **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 -> 32,767 **Beschreibung:** Höchstanzahl der zu unterhaltenden Web Prozesse im nicht kontextuellen Modus mit 4D im lokalen Modus und 4D Server. Standardmäßig ist der Wert 10. Um den Web Server reaktivierbar zu machen, verzögert 4D im nicht-kontextuellen Modus die Web Prozesse um 5 Sekunden und verwendet sie wieder, um mögliche spätere HTTP Anfragen auszuführen. In Bezug auf die Performance ist dies vorteilhafter als pro Anfrage einen neuen Prozess zu erstellen. Sobald ein Web Prozess wieder verwendet wird, wird er erneut um 5 Sekunden verzögert, außer die Höchstanzahl der Web Prozesse ist bereits erreicht. Dann wird er abgebrochen. Erhält ein Web Prozess nicht binnen 5 Sekunden eine Anfrage, wird er abgebrochen. Mit diesen Parametern können Sie je nach Anzahl der Anfragen, nach verfügbarem Speicher, etc. einstellen, wie Ihr Web Server arbeitet.
_o_Web conversion mode	Lange Ganzzahl	8	**** *Selector deaktiviert* ****
_o_Database cache size	Lange Ganzzahl	9	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** - **Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen, stattdessen die Funktion **Get cache size** zu verwenden.
_o_4D Local mode scheduler	Lange Ganzzahl	10	**** *Dieser Selector ist überholt und darf nicht mehr verwendet werden.*
_o_4D Server scheduler	Lange Ganzzahl	11	**** *Dieser Selector ist überholt und darf nicht mehr verwendet werden.*
_o_4D Remote mode scheduler	Lange Ganzzahl	12	**** *Dieser Selector ist überholt und darf nicht mehr verwendet werden.*
4D Server timeout	Lange Ganzzahl	13	**Reichweite:** 4D Anwendung bei positivem *Wert* **Wird zwischen 2 Sitzungen beibehalten:** Ja bei positivem *Wert* **Mögliche Werte:** 0 -> 32 767 **Beschreibung:** Damit ändern Sie den Timeout-Wert des 4D Server. Der Standardwert für das Timeout des 4D Server wird auf dem Server Rechner in den Datenbank-Eigenschaften auf der Seite **Client-Server>IP Konfiguration** festgelegt. Mit dem Selector 4D Server Timeout legen Sie im dazugehörigen Parameter *Wert* ein neues Timeout in Minuten fest. Das ist besonders hilfreich bei Operationen auf dem Client, die den Rechner blockieren oder viel Zeit beanspruchen, z.B. wenn eine große Anzahl an Seiten gedruckt wird. Das kann bei einem kleinen Standardwert zu einem unerwartetem Timeout führen. Es gibt zwei Optionen: • Ein positiver Wert im Parameter *Wert* setzt ein globales und permanentes Timeout: Der neue Wert gilt für alle Prozesse und wird in den Voreinstellungen der 4D Anwendung gespeichert. Das entspricht einer Änderung in den Einstellungen der Datenbank. • Ein negativer Wert im Parameter *Wert* setzt ein lokales und temporäres Timeout: Der neue Wert gilt nur für den aufgerufenen Prozess. Die anderen Prozesse behalten den Standardwert bei. Der temporäre Wert wird zurückgesetzt, sobald der Server vom Client ein Signal für Aktivität erhält – zum Beispiel, wenn die Operation beendet ist. Diese Option eignet sich für lange Operationen, die über 4D Plug-Ins gestartet werden. Um die Option "Kein Timeout" zu setzen, übergeben Sie 0 (Null) in Wert (siehe 1. Beispiel).
4D Remote mode timeout	Lange Ganzzahl	14	**Reichweite:** 4D Anwendung bei positivem Wert **Wird zwischen 2 Sitzungen beibehalten:** Ja bei positivem Wert **Beschreibung:** Nur für ganz spezifische Fälle verwenden. Damit ändern Sie die Zeitspanne zum Abschalten des Rechners mit remote 4D vom 4D Server Rechner. Der Standardwert für das Timeout wird auf dem remote Rechner in den Einstellungen der Datenbank auf der Seite “Client/Server>Konfiguration” festgelegt. Der Selector 4D Remote Mode Timeout wird nur bei Verwenden der legacy Netzwerkschicht berücksichtigt. Er wird ignoriert, wenn *ServerNet* aktiviert ist: Diese Einstellung wird vom Selector 4D Server Timeout (13) verwaltet.

Konstante	Typ	Wert	Kommentar
Port ID	Lange Ganzzahl	15	**Reichweite**: 4D lokal, 4D Server **Wird zwischen zwei Sitzungen beibehalten**: Nein **Beschreibung**: ID des TCP Port, den der 4D Web Server mit 4D im lokalen Modus und 4D Server verwendet. Der Standardwert ist 80 und lässt sich in den Datenbank-Eigenschaften auf der Seite **Web/Konfiguration** setzen. Für den Parameter *Wert* können Sie eine Konstante unter dem Thema **TCP Port Nummern** verwenden. Der Selector Port ID ist hilfreich für 4D Web Server mit einkompilierter 4D Desktop, d.h. ohne Zugriff auf den Designmodus. Weitere Informationen dazu finden Sie im Abschnitt **Web Server, Einstellungen**.
_o_IP Address to listen	Lange Ganzzahl	16	**** *Selector ist deaktiviert, die Befehle **WEB SET OPTION** und **WEB GET OPTION** verwenden* ****
Character set	Lange Ganzzahl	17	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
Max concurrent Web processes	Lange Ganzzahl	18	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
Client minimum Web process	Lange Ganzzahl	19	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 6 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client maximum Web process	Lange Ganzzahl	20	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 17 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client Max Web requests size	Lange Ganzzahl	21	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 27 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client port ID	Lange Ganzzahl	22	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 15 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
_o_Client IP address to listen	Lange Ganzzahl	23	**** *Selector ist deaktiviert, die Befehle **WEB SET OPTION** und **WEB GET OPTION** verwenden* ****
Client character set	Lange Ganzzahl	24	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 17 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client max concurrent Web proc	Lange Ganzzahl	25	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 18 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Maximum Web requests size	Lange Ganzzahl	27	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.

Konstante	Typ	Wert	Kommentar
4D Server log recording	Lange Ganzzahl	28	**Reichweite:** 4D Server, remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 oder von 1 bis X (0 = nicht speichern, 1 bis X = sequentielle Nummer, an den Dateinamen angefügt. **Beschreibung:** Startet oder stoppt das Speichern von Standardanfragen, die von 4D Server empfangen werden (mit Ausnahme von Web Anfragen). Der Standardwert ist 0, d.h. Anfragen werden nicht gespeichert. 4D Server ermöglicht, jede Anfrage, die vom Server-Rechner empfangen wird, in einem Logbuch zu speichern. Ist das Speichern aktiviert, werden im Logbuch neben der Strukturdatei der Datenbank zwei Dateien mit den Namen *4DRequestsLog_X.txt* und *4DRequestsLog_ProcessInfo_X.txt* angelegt, wobei X die Sequenznummer des Logbuchs ist. Hat *4DRequestsLog_X.txt* die Größe von 10 MB, wird sie geschlossen und es wird ein neues Logbuch mit der nächsthöheren Sequenznummer angelegt. Ist bereits eine Datei mit diesem Namen vorhanden, wird sie direkt ersetzt. Mit dem Parameter *Wert* können Sie die Startnummer für die Sequenzen setzen. Diese Textdateien speichern verschiedene Informationen zu jeder Anfrage in einer einfachen Liste: Zeit, Prozessnummer und -dauer, Größe der Anfrage, etc. Weitere Informationen dazu finden Sie im **Anhang E: Beschreibung der Protokolldateien**.
_o_Web Log recording	Lange Ganzzahl	29	**Reichweite:** 4D lokal, 4D Server **Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
Client Web log recording	Lange Ganzzahl	30	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 = Nicht speichern (Standard), 1 = In CLF Format speichern, 2 = In DLF Format speichern, 3 = In ELF Format speichern, 4 = In WLF Format speichern. **Beschreibung:** Startet oder stoppt das Speichern von Web Anfragen, die vom Web Server aller Client-Rechner empfangen werden. Der Standardwert ist 0, d.h. Anfragen werden nicht gespeichert. Dieser Selector arbeitet genauso wie Selector 29; mit dem Unterschied, dass er für alle Client Rechner gilt, die als Web Server verwendet werden. Die Datei "logweb.txt" wird in diesem Fall automatisch in den Unterordner Logs des 4D Client Anwendungsordners (Cache Ordner) gelegt. Wollen Sie nur Werte für bestimmte Client-Rechner setzen, verwenden Sie die Voreinstellungen von remote 4D.
Table sequence number	Lange Ganzzahl	31	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Jeder Wert vom Typ Lange Ganzzahl. **Beschreibung:** Dieser Selector dient zum Ändern oder Erhalten der aktuellen einmaligen Nummer für Datensätze der Tabelle, die als Parameter übergeben ist. "Aktuelle Nummer" bedeutet "zuletzt verwendete Nummer": Verändern Sie diesen Wert über den Befehl **SET DATABASE PARAMETER**, erhält der nächste Datensatz die Nummer übergebener Wert + 1. Diese neue Nummer wird von der Funktion **Sequence number** sowie in einem Feld der Tabelle zurückgegeben, der die Eigenschaft "Autoincrement" im Inspektorfenster oder via SQL zugeordnet wurde. Diese einmalige Nummer wird standardmäßig von 4D angelegt und entspricht der Reihenfolge, in der die Datensätze erstellt werden. Weitere Informationen dazu finden Sie in der Beschreibung zur Funktion **Sequence number**.
_o_Real display precision	Lange Ganzzahl	32	**** Selector deaktiviert ****

Konstante	Typ	Wert	Kommentar

Debug log recording	Lange Ganzzahl	34	**Reichweite**: 4D Anwendung
			Wird zwischen 2 Sitzungen beibehalten: Nein
			Mögliche Werte: 0 oder 1
			Beschreibung: Startet oder stoppt das sequentielle Speichern von Ereignissen auf 4D Programmierebene in der Datei *4DDebugLog*, die automatisch in den Unterordner Logs neben der Strukturdatei gelegt wird. Ab 4D v14 wird im Logbuch für Ereignisse "4DDebugLog[_n].txt" ein neues kompakteres Textformat verwendet (_n ist die Segmentnummer der Datei).
			Werte: Lange Ganzzahl mit einem Bit-Feld: Wert = bit1(1)+bit2(2)+bit3(4)+bit4(8)+ ...).
			- Bit 1 (Wert 1) aktiviert das Schreiben des Logbuchs. Beachten Sie, dass jeder andere Wert, der nicht Null ist, es ebenfalls aktiviert.
			- Bit 2 (Wert 2) nimmt Parameter von Methoden und Befehlen mit auf.
			- Bit 3 (Wert 4) aktiviert neue Formate mit Tabulatoren.
			- Bit 4 (Wert 8) deaktiviert sofortiges Schreiben jeder Operation auf die Festplatte (standardmäßig aktiviert). Sofortiges Schreiben ist langsamer, erlaubt aber, die Ursachen bei einem Absturz herauszufinden. Deaktivieren macht den Inhalt der Datei kompakter und generiert ihn schneller.
			- Bit 5 (Wert 16) deaktiviert das Protokollieren von Plug-In Aufrufen (standardmäßig aktiviert)
			Im älteren Format ohne Tabs werden Ausführungszeiten in Millisekunden angezeigt und der Wert "< ms" erscheint für Operationen, die weniger als eine Millisekunde dauern
			Im Format mit Tab werden Ausführungszeiten in Mikrosekunden angezeigt.
			Beispiele:
			SET DATABASE PARAMETER (34;1) // aktiviert das Logbuch im Modus v13 ohne Parameter und mit der Dauer
			SET DATABASE PARAMETER (34;2) // aktiviert das Logbuch im Modus v13 mit Parametern und Dauer
			SET DATABASE PARAMETER (34;2+4) // aktiviert das Logbuch im v14 Format, mit Parametern und Dauer
			SET DATABASE PARAMETER (34;0) // deaktiviert das Logbuch
			Damit eine Datei nicht zuviel Information protokolliert, können Sie mit dem Selektor 80 Log Command list die Liste der zu prüfenden 4D Befehle einschränken.
			Diese Option lässt sich für alle 4D Applikationen aktivieren (4D alle Modi, 4D Server, 4D Volume Desktop), im interpretierten oder kompiliertem Modus.
			Hinweis: Diese Option dient nur für Debugging-Zwecke und sollte nicht im laufenden Betrieb verwendet werden, da sie zu Einschränkungen der Performance und Überlastung der Festplatte führen kann. Weitere Informationen zu diesem Format und Einsatz der Datei *4DDebugLog[_n].txt* finden Sie im **Anhang E: Beschreibung der Protokolldateien**.

Client Server port ID	Lange Ganzzahl	35	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 bis 65535 **Beschreibung:** Mit diesem Parameter lässt sich per Programmierung die Nummer des TCP-Port bestimmen, in dem 4D Server die Datenbank veröffentlicht (Zugriff für 4D im lokalen Modus). Der Standardwert ist 19813. Durch individuelles Anpassen dieses Wertes können auf einem Rechner mit dem TCP Protokoll mehrere 4D Client/Server Anwendungen laufen. Dazu müssen Sie für jede Anwendung eine andere Port-Nummer angeben. Der Wert wird in der Strukturdatei der Datenbank gespeichert. Er lässt sich mit 4D im lokalen Modus setzen, wirkt sich jedoch nur im Client/Server-Betrieb aus. Verändern Sie diesen Wert, müssen Sie den Server-Rechner neu starten, damit er berücksichtigt wird.
Invert objects	Lange Ganzzahl	37	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0, 1 oder 2 (0 = Modus inaktiv, 1 = automatischer Modus, 2 = Modus aktiviert). **Beschreibung:** Einstellen des Modus "Objekt-Inversion", um Formulare, Objekte, Menüleisten, etc. im Anwendungsmodus umzukehren, wenn die Datenbank unter Windows in einer rechts-nach-links laufenden Sprache angezeigt wird. Dieser Modus lässt sich auch auf der Seite **Oberfläche** in den Datenbank-Eigenschaften einstellen: • Wert 0 gibt an, dass der Modus nie aktiviert wird, unabhängig von der Systemkonfiguration (entspricht der Einstellung **Nie** auf der Seite **Oberfläche**). • Wert 1 gibt an, dass der Modus je nach Systemkonfiguration aktiviert oder deaktiviert ist (entspricht der Einstellung **Automatisch** auf der Seite **Oberfläche**). • Wert 2 gibt an, dass der Modus aktiviert ist, unabhängig von der Systemkonfiguration (entspricht der Einstellung **Immer** auf der Seite **Oberfläche**). Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.
HTTPS Port ID	Lange Ganzzahl	39	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.

Konstante	Typ	Wert	Kommentar
Client HTTPS port ID	Lange Ganzzahl	40	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 bis 65535 **Beschreibung:** TCP Port Nummer, welche die Web Server der Client-Rechner für sichere Verbindungen via SSL (HTTPS Protokoll) verwenden. Der Wert ist standardmäßig 443. Mit diesem Selector können Sie per Programmierung die TCP Portnummer ändern, die Web Server von Client-Rechnern für sichere Verbindungen via SSL (HTTPS protocol) verwenden. Dieser Selector funktioniert auf dieselbe Weise wie der Selector 39; er wird jedoch auf alle Client-Rechner angewandt, die als Web Server dienen. Wollen Sie nur den Wert eines bestimmten Client-Rechners verändern, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Unicode mode	Lange Ganzzahl	41	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 (Modus Kompatibilität) oder 1 (Modus Unicode) **Beschreibung:** Aktueller Ausführungsmodus der Datenbank in Bezug auf den Zeichensatz. 4D unterstützt den Unicode Zeichensatz, kann aber auch im Modus "Kompatibilität" arbeiten, der auf dem Zeichensatz Mac ASCII basiert. Standardmäßig werden konvertierte Datenbanken im Modus Kompatibilität (0), in Version 11 oder höher erstellte Datenbanken im Modus Unicode ausgeführt. Der Ausführungsmodus lässt sich über eine Option in den Voreinstellungen steuern. Er lässt sich über diesen Selector auch lesen oder für Testzwecke verändern. Sie müssen die Datenbank neu starten, damit die Änderung berücksichtigt wird. Beachten Sie, dass Sie diesen Wert innerhalb einer Komponente nicht verändern, sondern ihn nur lesen können.
SQL Autocommit	Lange Ganzzahl	43	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 (deaktivieren) oder 1 (aktivieren) **Beschreibung:** Aktivierung oder Deaktivierung des Modus SQL auto-commit. Standardmäßig ist der Wert 0 (deaktiviert). Der Modus auto-commit sorgt für strengere referentielle Integrität der Datenbank. Ist er aktiv, werden alle Suchläufe mit **SELECT**, **INSERT**, **UPDATE** und **DELETE** (SIUD) automatisch in ad hoc Transaktionen gesetzt, wenn das noch nicht der Fall ist. Sie können diesen Modus auch in den Einstellungen der Datenbank setzen.
SQL Engine case sensitivity	Lange Ganzzahl	44	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 oder 1 (0 = Groß-/Kleinschreibung nicht beachten, 1 = Groß-/Kleinschreibung beachten) **Beschreibung:** Aktiviert oder deaktiviert die Berücksichtigung von Groß-/Kleinschreibung beim Vergleichen von Strings mit der SQL Engine. Der Wert ist standardmäßig 1, d.h. die SQL Engine unterscheidet beim Vergleichen von Strings zwischen Groß- und Kleinschreibung (Sortieren und Suchen) und Akzenten. Beispiel: "ABC"= "ABC" aber "ABC" # "Abc und "abc" # "âbc". In bestimmten Fällen, z.B. um die Funktionsweise der SQL Engine an die 4D Engine anzupassen, sollen beim Vergleichen von Strings die Groß- und Kleinschreibung, sowie Akzente nicht berücksichtigt werden ("ABC"="Abc"="âbc"). Sie können diese Option auch in den Datenbank-Eigenschaften auf der **Seite SQL** einstellen.
Client log recording	Lange Ganzzahl	45	**Reichweite:** Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 oder von 1 bis X (0 = nicht speichern, 1 bis X= sequentielle Nummer, an den Dateinamen angehängt). **Beschreibung:** Startet oder stoppt das Speichern der Standardanfragen, die vom 4D Client Rechner ausgehen, der den Befehl ausführt (außer Web Anfragen). Der Wert ist standardmäßig 0 (Null), d.h. die Anfragen werden nicht gespeichert. Sie können in 4D alle vom Client Rechner ausgeführten Anfragen in einem Logbuch speichern. Ist das Speichern aktiviert, werden auf dem Client Rechner im lokalen Ordner der Datenbank im Unterordner *Logs* zwei Dateien angelegt: *4DRequestsLogX.txt* und *4DRequestsLog_ProcessInfo_X.txt*, wobei X die fortlaufende Nummer des Logbuchs ist. Hat *4DRequestsLogX.txt* die Größe von 10 MB erreicht, wird es geschlossen und ein neues Logbuch mit der nächsthöheren Nummer erstellt. Gibt es bereits eine Datei mit demselben Namen, wird sie ersetzt. Über den Parameter *Wert* können Sie die Anfangsnummer setzen. Diese Textdateien speichern die Informationen zu jeder Anfrage in einem einfachen Format mit Tabs: Zeit, Prozessnummer, Größe der Anfrage, Bearbeitungsdauer, etc. Weitere Informationen dazu finden Sie im Abschnitt **Anhang E: Beschreibung der Protokolldateien**.

Konstante	Typ	Wert	Kommentar
Query by formula on server	Lange Ganzzahl	46	**Reichweite:** Aktuelle Tabelle und Prozess **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 (Konfiguration der Datenbank verwenden), 1 (auf Client ausführen) oder 2 (auf Server ausführen) **Beschreibung:** Ausführungsort der Befehle **QUERY BY FORMULA** und **QUERY SELECTION BY FORMULA** für die im Parameter übergebene Tabelle. Bei einer Datenbank im Client/Server-Betrieb können die Suchbefehle "nach Formel" entweder auf dem Server oder auf dem Client Rechner ausgeführt werden. • In einer mit 4D v11 SQL erstellten Datenbank werden die Befehle auf dem Server ausgeführt. • In konvertierten Datenbanken werden sie, wie in den bisherigen 4D Versionen, auf dem Client Rechner ausgeführt. • In konvertierten Datenbanken gibt es eine spezielle Voreinstellung (Seite Anwendung>Kompatibilität), um den Ausführungsort dieser Befehle global zu verändern. Der Ausführungsort beeinflusst nicht nur die Performance der Anwendung - die Ausführung auf dem Server ist in der Regel schneller - sondern auch die Programmierung. Der Wert der Komponenten der Formel, insbesondere bei Variablen, die über eine Methode aufgerufen werden, ändert sich je nach Ausführungsort. Über diesen Selektor können Sie die Arbeitsweise Ihrer Anwendung einstellen. Übergeben Sie 0 (Null) im Parameter Wert, richtet sich der Ausführungsort der Suchbefehle nach der Konfiguration der Datenbank: In mit 4D v11 SQL erstellten Datenbanken werden diese Befehle auf dem Server ausgeführt. In konvertierten Datenbanken werden sie, je nach den Datenbank-Eigenschaften, auf dem Client Rechner oder Server ausgeführt. Übergeben Sie 1 oder 2 in Wert, um die Ausführung dieser Befehle jeweils auf dem Client oder Server Rechner zu erzwingen. Siehe Beispiel 2. **Hinweis:** Wollen Sie auch die Möglichkeit haben, "SQL type" joins zu aktivieren (siehe Selector QUERY BY FORMULAR Joins, müssen Sie Formeln immer auf dem Server ausführen, damit diese auf die Datensätze zugreifen können. Bedenken Sie, dass die Formel in diesem Kontext keine Aufrufe einer Methode enthalten darf, da sie sonst automatisch auf den remote Rechner wechselt.
Order by formula on server	Lange Ganzzahl	47	**Reichweite:** Aktuelle Tabelle und Prozess **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 (Konfiguration der Datenbank verwenden), 1 (auf Client ausführen) oder 2 (auf Server ausführen) **Beschreibung:** Ausführungsort des Befehls **ORDER BY FORMULA** für die im Parameter übergebene Tabelle. Bei einer Datenbank im Client/Server-Betrieb lässt sich dieser Befehl entweder auf dem Server oder auf dem Client-Rechner ausführen. Über diesen Selector können Sie den Ausführungsort dieses Befehls angeben (Server oder Client). Sie können diesen Modus auch in dern Datenbank-Eigenschaften setzen. Weitere Informationen dazu finden Sie unter Selector 46, Query By Formula On Server. **Hinweis:** Wollen Sie auch die Möglichkeit haben, "SQL type" joins zu aktivieren (siehe Selector QUERY BY FORMULAR Joins, müssen Sie Formeln immer auf dem Server ausführen, damit diese auf die Datensätze zugreifen können. Bedenken Sie, dass die Formel in diesem Kontext keine Aufrufe einer Methode enthalten darf, da sie sonst automatisch auf den remote Rechner wechselt.
Auto synchro resources folder	Lange Ganzzahl	48	**Reichweite:** Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 (keine Synchronisation), 1 (auto Synchronisation) oder 2 (Fragen). **Beschreibung:** Dynamische Synchronisation des lokalen Ordners **Resources** auf dem Client-Rechner, der den Befehl mit dem Ordner **Resources** auf dem Server ausführt. Wurde der Inhalt des Ordners **Resources** auf dem Server geändert oder hat ein Benutzer die Synchronisation angefordert, z.B. über den Resourcen Explorer oder nach der Ausführung des Befehls **SET DATABASE LOCALIZATION**, benachrichtigt der Server die angemeldeten Client-Rechner. Auf der Client-Seite gibt es dann drei Möglichkeiten zur Synchronisation. Der Selektor Auto Synchro Resources Folder ermöglicht die Einstellung auf dem Client für die aktuelle Arbeitssitzung: • 0 (Standardwert) = keine dynamische Synchronisation (die Anfrage zur Synchronisation wird ignoriert) • 1 = automatische dynamische Synchronisation • 2 = Anzeige des Dialogfensters auf den Client-Rechnern, um die Synchronisation zu erlauben oder zu verweigern. Der Synchronisationsmodus lässt sich auch global in den Einstellungen der Anwendung definieren.

Konstante	Typ	Wert	Kommentar
Query by formula joins	Lange Ganzzahl	49	**Reichweite:** Aktueller Prozess **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 (Konfiguration der Datenbank verwenden), 1 (immer automatische Verknüpfungen) oder 2 (Wenn möglich, SQL joins verwenden). **Beschreibung:** Arbeitsweise der Befehle **QUERY BY FORMULA** und **QUERY SELECTION BY FORMULA** bei Verwendung von "SQL joins". In Datenbanken, die mit Version 11.2 von 4D v11 SQL erstellt wurden, führen diese Befehle Verknüpfungen nach dem SQL joins Modell aus. Auf diese Weise lässt sich die ausgewählte Tabelle verändern, wenn eine Suche in einer anderen Tabelle ausgeführt wird, auch wenn diese Tabellen nicht über eine automatische Verknüpfung miteinander verbunden sind. Diese automatische Verknüpfung war in bisherigen 4D Versionen erforderlich. Über den Selector QUERY BY FORMULA Joins legen Sie fest, wie die Suchbefehle im aktuellen Prozess arbeiten: • 0 (Standardwert) = Verwendet die aktuellen Einstellungen der Datenbank. In Datenbanken, die mit Version 11.2 von 4D v11 SQL erstellt wurden, sind für Suchen nach Formel immer "SQL joins" aktiviert. In konvertierten Datenbanken ist diese Arbeitsweise aus Kompatibilitätsgründen nicht aktiv, sie lässt sich aber über eine Voreinstellung einrichten. • 1 = Verwende immer automatische Verknüpfungen (= bisherige Funktionsweise in 4D). In diesem Modus ist eine Verknüpfung notwendig, damit bei Suche in einer anderen Tabelle die entsprechende Tabelle ausgewählt wird. 4D macht keine "SQL joins." • 2 = Verwende SQL joins, wenn möglich (= Standardverhalten in Datenbanken, die mit Version 11.2 von 4D v11 SQL und höher erstellt wurden). In diesem Modus erstellt 4D "SQL joins" für Suchen nach Formel, wenn die Formel dazu passt - bis auf zwei Ausnahmen. Weitere Informationen dazu finden Sie unter den Befehlen **QUERY BY FORMULA** oder **QUERY SELECTION BY FORMULA**. **Hinweis:** Mit 4D im remote Modus lassen sich "SQL joins" nur verwenden, wenn die Formeln auf dem Server ausgeführt werden, da diese auf die Datensätze zugreifen müssen. Wie Sie konfigurieren, wo Formeln ausgeführt werden, sehen Sie unter den Selectoren 46 und 47.
HTTP compression level	Lange Ganzzahl	50	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
HTTP compression threshold	Lange Ganzzahl	51	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
Server base process stack size	Lange Ganzzahl	53	**Reichweite:** 4D Server **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** Positive Lange Ganzzahl **Beschreibung:** Größe des Stapelspeichers, der jedem preemptiven Systemprozess auf dem Server zugewiesen wird, ausgedrückt in Bytes. Die Standardgröße wird vom System bestimmt. Preemptive Systemprozesse, d.h. Prozesse vom Typ 4D Client base process werden geladen, um die Hauptprozesse von 4D Client zu steuern. Die Größe, die dem Stapelspeicher jedes preemptiven Prozesses standardmäßig zugewiesen ist, sorgt für eine gut laufende Ausführung, kann aber eine wichtige Rolle spielen, wenn eine umfangreiche Anzahl, also mehrere hundert Prozesse erstellt werden. Diese Größe lässt sich für Optimierungszwecke beträchtlich verringern, wenn die von der Datenbank ausgeführten Operationen dies zulassen. Das ist z.B. der Fall, wenn die Datenbank keine Sortierung auf eine große Anzahl Datensätze ausführt. Möglich sind Werte von 512 oder auch 256 KB. Beachten Sie, dass eine zu geringe Stapelgröße kritisch werden kann, da sie die Operationen von 4D Server beeinträchtigt. Setzen Sie diesen Parameter mit Bedacht und unter Berücksichtigung der Datenbankbedingungen, wie z.B. Anzahl der Datensätze, Art der Operationen, etc. Damit dieser Parameter berücksichtigt wird, muss er auf dem Server-Rechner ausgeführt werden, z.B. in der **Datenbankmethode On Server Startup**.

Konstante	Typ	Wert	Kommentar
Idle connections timeout	Lange Ganzzahl	54	**Reichweite:** 4D Anwendung, außer Wert ist negativ **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** Ganzzahl für die Dauer in Sekunden. Der Wert kann positiv sein (neue Verbindungen) oder negativ (vorhandene Verbindungen). Standardmäßig ist der Wert 20. **Beschreibung:** Dieser Parameter ermöglicht auf der Server-Seite die maximale Zeitspanne an Inaktivität (timeout) für Verbindungen zur 4D Datenbank-Engine und der SQL Engine zu definieren. Erreicht eine pausierende Verbindung das Limit, wird sie automatisch auf standby gesetzt. Das friert die Client/Server Sitzung ein und schließt den Netzwerk Socket. Im Server Verwaltungsfenster lautet der Status des Benutzerprozesses "Zurückgestellt". Dieser Ablauf läuft für den Benutzer unsichtbar ab: Sobald bei der auf standby gesetzten Verbindung neue Aktivität auftritt, wird der Socket automatisch wieder geöffnet und die Client/Server Sitzung wiederhergestellt. Diese Einstellung lässt Sie einerseits Ressourcen auf dem Server einsparen: Verbindungen in standby schließen den Socket und geben einen Prozess auf dem Server frei. Andererseits können Sie so auch verhindern, dass Verbindungen verloren gehen, weil pausierende Sockets durch die Firewall geschlossen werden. Deshalb muss in diesem Fall der Wert für das Timeout von pausierenden Verbindungen niedriger als der für die Firewall sein. Übergeben Sie einen positiven Wert in *Wert*, gilt er für alle neuen Verbindungen in allen Prozessen. Übergeben Sie einen negativen Wert, gilt er für Verbindungen, die im aktuellen Prozess geöffnet sind. Übergeben Sie 0, unterliegen pausierende Verbindungen keinem Timeout. Dieser Parameter lässt sich auf Server und Client setzen. Geben Sie zwei unterschiedliche Zeiten an, wird der kleinere Wert berücksichtigt. Im Normalfall müssen Sie diesen Wert nicht ändern.
PHP interpreter IP address	Lange Ganzzahl	55	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: Formatierter String vom Typ IPv4 (zum Beispiel "127.0.0.1") oder IPv6 (zum Beispiel "2001:0db8:0000:0000:0000:ff00:0042:8329"). **Beschreibung**: von 4D lokal verwendete IP Adresse, um mit dem PHP Interpreter via FastCGI zu kommunizieren. Der Wert ist standardmäßig "127.0.0.1" (Adressen im IPv6 Format werden ab 4D v16R4 unterstützt). Diese Adresse muss dem Rechner entsprechen, auf dem 4D läuft. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner setzen. Weitere Informationen zum PHP Interpreter finden Sie im Handbuch *4D Designmodus*.
PHP interpreter port	Lange Ganzzahl	56	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: Positive Lange Ganzzahl. Standardmäßig ist der Wert 8002. **Beschreibung**: Nummer des TCP Port, den der PHP Interpreter von 4D verwendet. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner setzen. Weitere Informationen zum PHP Interpreter finden Sie im Handbuch *4D Designmodus*.
PHP number of children	Lange Ganzzahl	57	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: Positive Lange Ganzzahl. Standardmäßig ist der Wert 5. **Beschreibung**: Anzahl der Kindprozesse, die vom PHP Interpreter von 4D erstellt und lokal gewartet werden. Aus Optimierungsgründen erstellt und verwendet der PHP Interpreter zum Bearbeiten der Anfragen zur Skript-Ausführung einen Satz von Systemprozessen, genannt "Kindprozesse". Sie können die Anzahl der Kindprozesse je nach den Anforderungen Ihrer Anwendung variieren. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner setzen. Weitere Informationen zum PHP Interpreter finden Sie im Handbuch *4D Designmodus*. **Hinweis:** Auf Mac OS nutzen alle Kindprozesse den gleichen Port. Unter Windows verwendet jeder Kindprozess eine spezifische Port Nummer. Die erste Nummer ist die für den PHP Interpreter gesetzte; die anderen Kindprozesse erhöhen die erste Nummer jeweils. Ist zum Beispiel der Standardport 8002 und starten Sie 5 Kindprozesse, verwenden sie die Ports 8002 bis 8006.
PHP max requests	Lange Ganzzahl	58	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: Positive Lange Ganzzahl. Standardmäßig ist der Wert 500. **Beschreibung**: Maximale Anzahl der Anfragen, die der PHP Interpreter akzeptiert. Ist die maximale Anzahl erreicht, gibt der Interpreter Fehler vom Typ "Server ist überlastet" zurück. Sie können diesen Wert aus Sicherheits- oder Performance-Gründen verändern. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner verändern. Weitere Informationen dazu finden Sie in der Dokumentation zu FastCGI-PHP. **Hinweis:** Auf 4D Seite werden diese Parameter dynamisch angewandt; d.h. 4D muss nicht beendet werden, damit sie berücksichtigt werden. Ist dagegen der PHP Interpreter bereits gestartet, muss beendet und neu gestartet werden, damit diese Änderungen berücksichtigt werden.

Konstante	Typ	Wert	Kommentar
PHP use external interpreter	Lange Ganzzahl	60	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: 0 = Verwende internen Interpreter, 1 = Verwende externen Interpreter **Beschreibung**: Der Wert gibt an, ob PHP Anfragen in 4D an den von 4D intern gelieferten Interpreter oder an einen externen Interpreter gesendet werden. Standardmäßig ist der Wert 0, d.h. der interne Interpreter von 4D wird verwendet. Wollen Sie Ihren eigenen PHP Interpreter verwenden, um z.B. zusätzliche Module oder eine spezifische Konfiguration zu verwenden, übergeben Sie 1 in *Wert*. In diesem Fall startet 4D bei PHP Anfragen nicht den internen Interpreter. Eigene PHP Interpreter müssen in FastCGI kompiliert sein und auf demselben Rechner wie die 4D Engine liegen. Beachten Sie, dass Sie diese Interpreter komplett selbst verwalten müssen; sie werden von 4D weder gestartet noch gestoppt. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner setzen.
Maximum temporary memory size	Lange Ganzzahl	61	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Werte:** Positive Lange Ganzzahl **Beschreibung:** Maximale Größe des temporären Speichers, den 4D jedem Prozess zuordnen kann, ausgedrückt in MB. Der Wert ist standardmäßig 0 (keine maximale Größe). 4D verwendet einen speziellen temporären Speicher zum Indizieren und Sortieren der Operationen. Ziel dieses Speichers ist, den "standard" Cache Speicher bei massiven Operationen beizubehalten. Er wird nur bei Bedarf aktiviert. Die Größe des temporären Speichers wird nur durch die verfügbaren Ressourcen begrenzt. Das richtet sich nach der Konfiguration des Systemspeichers. Diese Vorgehensweise passt für die meisten Programme. In einigen spezifischen Fällen, insbesondere wenn eine Client-/Server Anwendung simultan eine große Anzahl sequentieller Sortierungen durchführt, kann der temporäre Speicher eine kritische Größe erreichen, die das System instabil macht. Für solche Fälle können Sie für den temporären Speicher eine maximale Größe einrichten, damit die Anwendung weiterhin einwandfrei laufen kann. Das kann jedoch wiederum die Ausführungsgeschwindigkeit beeinträchtigen: Ist die maximale Größe für einen Prozess erreicht, verwendet 4D Dateien der Festplatte, was den Vorgang verlangsamen kann. Für spezifische Anforderungen, wie z.B. oben beschrieben, ist im allgemeinen eine maximale Größe von ca. 50 MB ein guter Kompromiss. Der ideale Wert muss jedoch anhand der Eigenheiten der Anwendung bestimmt werden und ergibt sich aus Volumentests in Echtzeit.
SSL cipher list	Zeichenkette	64	**Reichweite**: 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: Folge von Strings, getrennt durch Doppelpunkt (zum Beispiel "RC4-MD5:RC4-64-MD5:....") **Beschreibung:** Cipher Liste, die 4D für das SSL Protokoll verwendet. Diese Liste verändert die Priorität der in 4D implementierten cipher Algorithmen. Sie können im Parameter *Wert* z.B. folgenden String übergeben. "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH". Eine ausführliche Beschreibung der Syntax für die Cipher Liste finden Sie auf der ciphers Seite der OpenSSL Site. Diese Einstellung gilt für die gesamte Anwendung (sie betrifft HTTP Server, SQL Server, Client/Server Verbindungen, sowie den HTTP Client und alle 4D Befehle, die das SSL Protokoll nutzen), aber sie ist temporär, d.h. sie bleibt zwischen Sitzungen nicht erhalten. Wurde die Cipher Liste geändert, müssen Sie den betroffenen Server neu starten, damit die neuen Einstellungen berücksichtigt werden. Um die Cipher Liste auf ihren Standardwert zurückzusetzen, der permanent in der SLI Datei gespeichert ist, rufen Sie den Befehl **SET DATABASE PARAMETER** auf und übergeben im Parameter *Wert* einen leeren String (""). **Hinweis:** Mit der Funktion **Get database parameter** wird die Cipher Liste im optionalen Parameter *StringWert* zurückgegeben und der Parameter ist immer 0.
Cache unload minimum size	Lange Ganzzahl	66	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Mögliche Werte**: Positive Lange Ganzzahl > 1. **Beschreibung**: Mindestgröße des Speichers, der vom Datenbank Cache freigegeben werden soll, wenn die Engine Platz schaffen muss, um dem Cache ein Objekt zuzuweisen (Wert in Bytes). Dieser Selektors dient dazu, für eine bessere Performance die Momente zu reduzieren, zu denen Daten aus dem Cache freigegeben werden. Sie können diese Einstellung je nach Größe des Cache bzw. nach Datenblock, der in Ihrer Datenbank bearbeitet wird, variieren. Ohne diesen Selektor entlädt 4D standardmäßig mindestens 10% des Cache, wenn Platz gebraucht wird.

Konstante	Typ	Wert	Kommentar
Direct2D status	Lange Ganzzahl	69	**Reichweite:** 4D Anwendung **Zwischen 2 Sitzungen beibehalten:** Nein **Beschreibung:** Aktivierungsmodus zum Integrieren von Direct2D unter Windows. **Mögliche Werte:** Eine der folgenden Konstanten (standardmäßig Modus 5): Direct2D Disabled (0): Direct2D Modus ist nicht aktiviert. Die Anwendung funktioniert wie im früheren Modus (GDI/GDIPlus). Direct2D Hardware (1): Direct2D als Grafik Hardware-Kontext für die gesamte 4D Anwendung verwenden. Falls nicht verfügbar, Direct2D Grafik Software-Kontext verwenden (außer unter Vista; hier wird zur besseren Performance der Modus GDI/GDIPlus verwendet. Direct2D Software (3):(Standardmodus) Ab Windows 7 den Software-Kontext Direct2D Grafik für die gesamte 4D Anwendung verwenden. Unter Vista wird zur besseren Performance der Modus GDI/GDIPlus verwendet. **Hinweis zur Kompatibilität:** Ab 4D v14 werden hybrid Modi deaktiviert und auf die verfügbaren Modi umgeleitet (der bisherige Modus 2 entspricht 1; die bisherigen Modi 4 und 5 entsprechen Modus 3).

Direct2D get active status	Lange Ganzzahl	74	**Hinweis:** Sie können diesen Selector nur mit der Funktion **Get database parameter** verwenden und keinen Wert setzen.
			Beschreibung: Gibt die aktuelle Integration von Direct2D unter Windows zurück.
			Mögliche Werte: 0, 1, 2, 3, 4 oder 5 (siehe Werte des Selector 69). Der zurückgegebene Wert richtet sich nach der Verfügbarkeit von Direct2D, sowie Hardware und der vom Betriebssystem unterstützten Qualität von Direct2D.
			Führen Sie zum Beispiel folgendes aus:
			`SET DATABASE PARAMETER(Direct2D status;Direct2D Hardware)`
			`$mode:=Get database parameter(Direct2D get active status)`
			
			- Ab Windows 7 und höher wird $mode auf 1 gesetzt, wenn das System Hardware findet, die mit Direct2D kompatibel ist; andernfalls wird $mode auf 3 gesetzt (Software-Kontext).
			- Unter Windows Vista wird *$mode* auf 1 gesetzt, wenn das System Hardware findet, die mit Direct2D; kompatibel ist; andernfalls wird *$mode* auf 0 gesetzt, d.h. Direct2D wird deaktiviert.
			- Unter Windows XP wird *$mode* immer auf 0 gesetzt, da diese Version nicht mit Direct2D kompatibel ist.

Diagnostic log recording	Lange Ganzzahl	79	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 oder 1 (0 = nicht speichern, 1 = speichern) **Beschreibung:** Startet oder stoppt das Protokollieren der 4D Diagnosedatei. Standardmäßig ist der Wert 0 (nicht protokollieren). 4D kann kontinuierlich eine Reihe Ereignisse, die bei der internen Operationsweise der Anwendung auftreten, in einer Diagnosedatei mitschreiben. Information in dieser Datei ist hilfreich beim Entwickeln von 4D Anwendungen und kann mit Hilfe des 4D Tech Support analysiert werden. Übergeben Sie 1 in diesem Selector, wird im Ordner **Logs** der Anwendung automatisch eine Diagnosedatei mit Namen *DatenbankName.txt* angelegt bzw. geöffnet. Bei einer Größe von 10 MB wird sie geschlossen und es wird eine neue Datei mit Namen *DatenbankName_N.txt* generiert mit der ansteigenden Nummer N. Über den Befehl **LOG EVENT** können Sie in dieser Datei eigene Informationen hinzufügen.
Log command list	Zeichenkette	80	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** String mit der Liste der 4D Befehlsnummern zum Protokollieren, getrennt durch Strichpunkte. Sie können "all" übergeben, um alle Befehle zu protokollieren oder "" (leerer String), um keine Befehle zu protokollieren. **Beschreibung***:* Liste der 4D Befehle, die in der Protokolldatei mitgeschrieben werden (siehe Selector 34, Debug Log Recording). Standardmäßig werden alle 4D Befehle protokolliert. Dieser Selector reduziert die Informationen, die in der Diagnosedatei gesichert wird, durch Einschränken der 4D Befehle, deren Ausführung Sie nicht protokollieren wollen. Sie können z.B. schreiben: `SET DATABASE PARAMETER(Log command list;"277;334") //Nur die Befehle QUERY und QUERY SELECTION protokollieren`
Spellchecker	Lange Ganzzahl	81	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: 0 (Standard) = native OS X Rechtschreibprüfung (Hunspell deaktiviert), 1 = Hunspell Rechtschreibprüfung aktiviert. **Beschreibung**: Aktiviert die Hunspell Rechtschreibprüfung auf OS X. Auf dieser Plattform wird standardmäßig die native Rechtschreibprüfung aktiviert. Sie können jedoch die Hunspell Rechtschreibprüfung aktivieren, z.B. um die Oberfläche Ihrer plattformunabhängigen Anwendungen einheitlich zu gestalten (unter Windows ist nur die Hunspell Rechtschreibprüfung verfügbar). Weitere Informationen dazu finden Sie im Abschnitt **Rechtschreibprüfung**.

Konstante	Typ	Wert	Kommentar
QuickTime support	Lange Ganzzahl	82	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Ja **Werte**: 0 (Standard) = QuickTime deaktiviert, 1 = QuickTime aktiviert **Beschreibung**: 4D Version 14 unterstützt QuickTime Codecs nicht mehr standardmäßig. Zur Wahrung der Kompatibilität können Sie die Unterstützung in Ihrer Anwendung über diesen Selector reaktivieren. Beachten Sie jedoch, dass die QuickTime Unterstützung in zukünftigen 4D Versionen komplett eingestellt wird.
Dates inside objects	Lange Ganzzahl	85	**Reichweite**: Aktueller Prozess **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: String type without time zone (0), String type with time zone (1), Date type (2) (Standard) **Beschreibung**: Definiert wie Datumsangaben innerhalb von Objekten gespeichert werden und sie in JSON importiert/exportiert werden. Mit Date type (Standardwert für Anwendungen, die mit 4Dv17 oder höher erstellt wurden) werden 4D Datumsangaben mit dem Datumstyp im Objekt gespeichert unter Berücksichtigung der lokalen Datumseinstellung. Beim Konvertieren in das JSON Format werden Datumsattribute in Strings ohne Zeitangabe konvertiert. (**Hinweis:** Dies kann über die Option "Verwende Datumstyp statt ISO Datumsformat in Objekten" in den Datenbank-Einstellungen auf der Seite **Seite Kompatibilität** gesetzt werden) Mit String type with time zone werden 4D Datumsangaben in ISO Strings konvertiert und berücksichtigen die lokale Zeitzone. Zum Beispiel erhalten Sie beim Konvertieren des Datums !23.08.2013! in JSON Format 2013-08-22T22:00:00Z, wenn die Operation in Deutschland zu einer Sicherungszeit bei Tageslicht (GMT+2) ausgeführt wird. Dieses Prinzip entspricht der Standardoperation von JavaScript. Das kann zu Fehlern führen, wenn Sie Datumswerte in JSON an jemanden in einer anderen Zeitzone senden wollen. Das ist z.B. der Fall, wenn Sie eine Tabelle über **Selection to JSON** in Deutschland exportieren, die dann in USA über **JSON TO SELECTION** importiert wird. Da Datumsangaben in jeder Zeitzone erneut interpretiert werden, sind die in der Anwendung gespeicherten Werte unterschiedlich. In diesem Fall können Sie den Konvertierungsmodus für Datum ändern. Damit die Zeitzone nicht berücksichtigt wird, übergeben Sie in diesem Selektor String type without time zone. Dann ergibt die Konvertierung des Datums !23.08.2013! in allen Fällen 2013-08-23T00:00:00Z.
Use legacy network layer	Lange Ganzzahl	87	**Reichweite**: 4D im lokalen Modus, 4D Server **Wird zwischen 2 Sitzungen beibehalten**: Ja **Beschreibung**: Setzt oder erhält den aktuellen Status der legacy Netzwerk-Schicht für Client/Server Verbindungen. Die legacy Netzwerk-Schicht ist ab 4D v14 R5 überholt und sollte in Ihren Anwendungen nach und nach durch die Netzwerk-Schicht *ServerNet* ersetzt werden. *ServerNet* ist in den nächsten 4D Releases erforderlich, um auch in Zukunft von Netzwerk Evolutionen zu profitieren. Zur Wahrung der Kompatibilität wird die bisherige Netzwerk-Schicht noch unterstützt, um den allmählichen Übergang für vorhandene Anwendungen zu ermöglichen. Sie wird in konvertierten Anwendungen aus einem Release vor v14 R5 standardmäßig genutzt. Übergeben Sie 1, um für Ihre Client/Server Verbindungen die bisherige Netzwerk-Schicht zu verwenden (und *ServerNet* zu deaktivieren); 0, um sie zu deaktivieren (und *ServerNet* zu verwenden). Diese Eigenschaft lässt sich auch über die Option "Verwende legacy Netzwerk Schicht" auf der **Seite Kompatibilität** der Datenbank-Eigenschaften setzen. Hier wird auch die Vorgehensweise zum Migrieren erläutert. Wir empfehlen, so bald wie möglich in all Ihren Anwendungen auf *ServerNet* umzustellen (siehe auch **Netzwerk und Client-Server Optionen**). Sie müssen die Anwendung neu starten, damit dieser Selektor berücksichtigt wird. **Mögliche Werte**: 0 oder 1 (0 = legacy Schicht nicht verwenden, 1 = legacy Schicht verwenden) **Standardwert:** 0 in Datenbanken, die mit 4D v14 R5 oder neuer erstellt wurden, 1 in Datenbanken, die aus 4D v14 R4 oder früher konvertiert wurden.
SQL Server Port ID	Lange Ganzzahl	88	**Reichweite**: 4D im lokalen Modus und 4D Server. **Wird zwischen 2 Sitzungen beibehalten**: Ja **Beschreibung**:Setzt oder erhält die Nummer des TCP Port, den der in 4D integrierte SQL Server mit 4D im lokalen Modus oder 4D Server verwendet. Der Wert ist standardmäßig 19812. Die TCP Port Nummer lässt sich auch auf der Seite "SQL" in den Datenbank-Eigenschaften setzen. Mit diesem Selector wird die Datenbank-Eigenschaft entsprechend aktualisiert und bleibt auch nach Beenden und Neustart erhalten. **Mögliche Werte**: 0 bis 65535 **Standardwert**: 19812
Circular log limitation	Lange Ganzzahl	90	**Reichweite**: 4D lokal, 4D Server. **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: Jeder Wert vom Typ Zahl, 0 = alle Journale behalten **Beschreibung**: Maximale Anzahl der beizubehaltenden Journale pro Typ. Standardmäßig werden alle Dateien beibehalten. Übergeben Sie einen Wert *X*, werden nur die *letzten X* Dateien behalten, wobei bei Erstellen eines neuen Journals das älteste automatisch entfernt wird. Diese Einstellung gilt jeweils pro Journal für folgende Typen: request logs (Selector 28 und 45), debug log (Selector 34), events log (Selector 79), sowie Web request logs und Web debug logs (Selector 29 und 84 des Befehls **WEB SET OPTION**).

Konstante	Typ	Wert	Kommentar
Number of formulas in cache	Lange Ganzzahl	92	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Mögliche Werte**: Positive Lange Ganzzahl **Standardwert**: 0 (kein Cache) **Beschreibung**: Setzt oder erhält die maximale Anzahl Formeln, die im vom Befehl [#cmd id="63"/] verwendeten Formel-Cache beibehalten werden sollen. Diese Begrenzung gilt für alle Prozesse, jedoch hat jeder Prozess seinen eigenen Formel-Cache. Formeln im Cache zu halten, beschleunigt die Ausführung von **EXECUTE FORMULA** im kompilierten Modus, da jede dieser Formeln nur einmal tokenisiert wird (weitere Info siehe [#title id="8567"/]). Ändern Sie den Cache Wert, wird der vorhandene Inhalt zurückgesetzt, selbst wenn die neue Größe die bisherige übersteigt. Ist die Anzahl Formeln im Cache erreicht, ersetzt die neu ausgeführte Formel die älteste im Cache (FIFO Modus). Dieser Parameter wird nur in kompilierten Anwendungen oder Komponenten berücksichtigt.
Cache flush periodicity	Lange Ganzzahl	95	**Reichweite**: 4D local, 4D Server **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: Lange Ganzzahl > 1 (Sekunden) **Beschreibung**: Erhält oder setzt das Zeitintervall zum Sichern des aktuellen Cache in Sekunden. Eine Änderung dieses Werts überschreibt die Option **Daten-Cache sichern alle X Sekunden** auf der Seite **Seite Datenbank/ Speicher** der Datenbank Eigenschaften für die Sitzung (wird nicht in den Datenbank Eigenschaften gespeichert).
Tips enabled	Lange Ganzzahl	101	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 = Tipps deaktiviert, 1 = Tipps aktiviert (Standard) **Beschreibung:** Setzt oder erhält den aktuellen Status der Anzeige von Tipps für die 4D Anwendung. Tipps sind standardmäßig aktiviert. Beachten Sie, dass dieser Parameter alle 4D Tipps setzt, z.B. Hilfemeldungen für Formulare und Tipps im Editor des Designmodus.
Tips delay	Lange Ganzzahl	102	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** Lange Ganzzahl >= 0 (Ticks) **Beschreibung:** Verzögerte Anzeige von Tipps, nachdem der Mauszeiger in Objekten mit zugewiesenen Hilfemeldungen gesetzt wurde. Wert wird in Ticks gerechnet (1/60stel Sekunde). Standardwert ist 45 Ticks (0.75 Sekunde).
Tips duration	Lange Ganzzahl	103	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** Lange Ganzzahl >= 60 (Ticks) **Beschreibung:** Maximale Anzeigedauer für einen Tipp. Wert wird in Ticks gerechnet (1/60stel Sekunde). Standardwert ist 720 Ticks (12 Sekunden).
Min TLS version	Lange Ganzzahl	105	**Reichweite**: 4D Server, 4D Web Server und 4D SQL Server **Wird zwischen 2 Sitzungen beibehalten**: Nein **Beschreibung**: Gibt die Ebene von Transport Layer Security (TLS) für die Datenverschlüsselung und Authentifizierung zwischen Anwendungen und Servern an. Die definierten Werte gelten global für die Netzwerkebene. Ein geänderter Wert wird erst nach Neustart des Servers verwendet. Das standardmäßige Mindestprotokoll ist TLSv1_2. **Mögliche Werte**: • TLSv1_0 - TLS 1.0, eingeführt in 1999 als Upgrade von SSL v3.0. TLS 1.0 und SSL 3.0 funktioniere n nicht zusammen. • TLSv1_1 - TLS 1.1, eingeführt in 2006 als Update von TLS 1.0. Verbesserungen sind u.a. bessere Sicherheit und Fehlerverwaltung. • TLSv1_2 - TLS 1.2, eingeführt in 2008 als Update von TLS 1.1. Verbesserungen sind u.a. erhöhte Flexibilität, zusätzliche Unterstützung für authentifizierte Verschlüsselungen. **Hinweis**: Das Plug-In 4D Internet Commands nutzt eine andere Netzwerkschicht. Von daher hat dieser Selector keine Auswirkung auf deren TLS Version.
Times inside objects	Lange Ganzzahl	109	**Reichweite**: 4D lokal, 4D Server (alle Prozesse) **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: Times in seconds (0) (Standard), Times in milliseconds (1) **Beschreibung**: Definert, wie Zeitangaben konvertiert und in Objekteigenschaften und Elementen in Collections gespeichert werden, und wie sie in JSON und in Web Areas importiert bzw. daraus exportiert werden. Ab 4D v17 werden Zeitangaben in Objekten standardmäßig in Anzahl von Sekunden konvertiert und gespeichert. In bisherigen Releases wurden Zeitwerte in diesem Kontext als Anzahl von Millisekunden konvertiert und gespeichert. Dieser Selector unterstützt Sie beim Migrieren Ihrer Anwendungen und geht bei Bedarf zurück auf die bisherige Einstellung. **Hinweis**: ORDA Methoden und SQL Engine ignorieren diese Einstellung, sie gehen immer davon aus, dass Zeitwerte die Anzahl von Sekunden sind.

Hinweise:

- Der Parameter *Tabellenname* wird nur von den Selektoren 31, 32, 46 und 47 verwendet. In allen anderen Fällen wird er ignoriert, falls er übergeben ist.
- Einstellungen, die nicht zwischen Sitzungen beibehalten werden, können in der **Datenbankmethode On Startup** oder **Datenbankmethode On Server Startup** gesetzt werden.

Beispiel 1

Nachfolgender Code verhindert unerwartete Timeouts:

```
  `Timeout für aktuellen Prozess auf 3 Stunden erhöhen
SET DATABASE PARAMETER(4D Server Timeout;-60*3)
  `Zeitintensive Operation ohne Steuern durch 4D ausführen
...
WR PRINT MERGE(Bereich;3;0)
...
```

Beispiel 2

Dieses Beispiel erzwingt vorübergehend die Ausführung des Befehls **QUERY BY FORMULA** auf dem Client Rechner:

```
curVal:=Get database parameter([table1];Query By Formula On Server)
  ` Die aktuelle Einstellung speichern
SET DATABASE PARAMETER([table1];Query By Formula On Server;1)
  ` Die Ausführung auf dem Client Rechner erzwingen
QUERY BY FORMULA([table1];myformula)
SET DATABASE PARAMETER([table1];Query By Formula On Server;curVal)
  ` Die aktuelle Einstellung wiederherstellen
```

Beispiel 3

Sie wollen Daten in JSON exportieren, die ein konvertiertes 4D Datum enthalten. Beachten Sie, dass die Konvertierung passiert, wenn das Datum im Objekt gesichert wird. Deshalb müssen Sie den Befehl **SET DATABASE PARAMETER** vor **OB SET** aufrufen:

```
C_OBJECT($o)
SET DATABASE PARAMETER(JSON use local time;0)
OB SET($o ;"myDate";Current date)  // JSON Konvertierung
$json:=JSON Stringify($o)
SET DATABASE PARAMETER(JSON use local time;1)
```

SET UPDATE FOLDER

SET UPDATE FOLDER (OrdnerPfad {; stilleFehler})

Parameter	Typ		Beschreibung
OrdnerPfad	String	➡	Pfadname des Ordners (Paket auf OS X) mit der upgedateten Anwendung
stilleFehler	Boolean	➡	Falsch (Standard) = Fehler protokollieren und anzeigen Wahr = Fehler nur protokollieren

Beschreibung

Der Befehl **SET UPDATE FOLDER** gibt den Ordner mit dem Update der aktuellen doppelklickbaren 4D Anwendung an. Diese Information wird in der 4D Sitzung bis zum Aufrufen von **RESTART 4D** gespeichert. Sie wird bei manuellem Beenden der Anwendung nicht beibehalten.

Dieser Befehl dient zum automatischen Update Prozess für eine doppelklickbare Anwendung (Client/Server oder Einzelplatz). Weitere Informationen dazu finden Sie im Abschnitt **Eigenständige Anwendung erstellen und weitergeben** des Handbuchs *Designmodus*.

Hinweis: Dieser Befehl funktioniert nur mit 4D Server oder einer Einzelplatz-Anwendung mit einkompilierter 4D Volume Desktop.

Im Parameter *OrdnerPfad* übergeben Sie den vollständigen Pfadnamen des Ordners mit der neuen Version der doppelklickbaren Anwendung. Das ist unter Windows der Ordner mit der Anwendung my4DApp.exe, auf OS X das Paket my4DApp.app, die über den Application Builder erstellt wurde.

Hinweis: Wir empfehlen, für die Dateien der neuen Version der Anwendung dieselben Namen wie für die Originale zu verwenden, da der Anwendungsordner während dem Update ersetzt wird. Wenn Sie für diese Dateien unterschiedliche Namen verwenden, funktionieren gespeicherte Tastenkürzel bzw. Pfade nicht.

Sind die Parameter gültig, wird das Update in der Sitzung bis zum Aufrufen von **RESTART 4D** auf "Halten" gesetzt. Haben Sie vor Aufrufen von **RESTART 4D** mehrmals **SET UPDATE FOLDER** ausgeführt, wird der letzte gültige Aufruf berücksichtigt. Treten Unregelmäßigkeiten auf, wird ein Fehler generiert; der Parameter *stilleFehler* definiert, ob diese Fehler angezeigt werden oder nicht (siehe unten).
Sie können im Parameter *OrdnerPfad* einen leeren String ("") übergeben, um die Update Informationen für die aktuelle Sitzung erneut zu setzen.

Der optionale Parameter *stilleFehler* gibt an, wie Fehler während dem Update protokolliert werden:

- Übergeben Sie **Falsch** oder lassen diesen Parameter weg, werden Fehler im Update Journal gespeichert und in einem Dialogfenster Warnung angezeigt.
- Übergeben Sie **Wahr**, werden die Fehler nur im Update Journal gespeichert.

Ausnahme: Konnte das Journal nicht angelegt werden, erscheint eine Meldung, unabhängig, welchen Wert der Parameter *stilleFehler* hat. Weitere Informationen dazu finden Sie unter dem Befehl **Get last update log path**.

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt; andernfalls auf 0. Durch diesen Befehl generierte Fehler können Sie über eine Methode abfangen, die über den Befehl **ON ERR CALL** aufgerufen wird.

Beispiel

Sie haben auf Ihrer Festplatte einen Ordner "MeineUpdates"erstellt, der die neue Version der Anwendung "MeineApp" enthält. Sie wollen Fehler nicht anzeigen. Dazu schreiben Sie folgenden Code:

```
// Syntax unter Windows
SET UPDATE FOLDER("C:\\MeineUpdates"+Folder separator+"MeineApp"+Folder separator;True)

// Syntax auf OS X
SET UPDATE FOLDER("MacHD:MeineUpdates"+Folder separator+"MeineApp.app"+Folder separator;True)
```

Structure file

Structure file {(*)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	⇒	Gibt die Strukturdatei der Host Datenbank zurück
Funktionsergebnis	String	⇐	Kompletter Name der Strukturdatei der Datenbank

Beschreibung

Die Funktion **Structure file** gibt den kompletten Namen der Strukturdatei der Datenbank zurück, mit der Sie gerade arbeiten.

Windows
Arbeiten Sie zum Beispiel mit der Datenbank MeineCDs aus dem Volume G in \DOCS\MeineCDs, gibt der Befehl zurück: G:\DOCS\MeineCDs\MeineCDs.4DB.

Macintosh
Arbeiten Sie zum Beispiel mit der Datenbank MeineCDs aus dem Ordner Dokumente:MeineCDs*ƒ*: auf der Festplatte Macintosh HD, gibt der Befehl zurück: Macintosh HD:Documente:MeineCDs*ƒ*:MeineCDs.

Hinweis: Im Fall einer Anwendung mit einkompilierter 4D Volume Desktop gibt die Funktion den Pfadnamen der Anwendungsdatei (ausführbare Anwendung) unter Windows und OS X zurück. Auf OS X liegt sie innerhalb des Software-Pakets im Ordner [Contents:Mac OS]. Über die Funktion **Application file** erhalten Sie den vollständigen Namen des Software Pakets. Zuerst prüfen Sie die Anwendung mit der Funktion **Application type**, dann führen Sie je nach Kontext die Funktion **Structure file** oder **Application file** aus.

WARNUNG: Rufen Sie diese Funktion mit 4D remote Modus auf, gibt sie nur den Namen der Strukturdatei und nicht den kompletten Namen zurück.

- Wird die Funktion von einer Komponente aus aufgerufen, gilt folgendes:
  - Ist der Parameter * übergeben, gibt die Funktion den langen Namen der Strukturdatei der Host Datenbank zurück.
  - Ist der Parameter * nicht übergeben, gibt die Funktion den langen Namen der Strukturdatei der Komponente zurück. Die Strukturdatei einer Komponente entspricht der Datei .4db oder .4dc der Komponente, die im Ordner Components der Datenbank liegt. Eine Komponente lässt sich auch in Form eines Alias/Tastenkürzel oder eines .4dDatenbankordners/Package installieren:
    - Bei einem Alias/Tastenkürzel gibt die Funktion den Pfadnamen der Originaldatei .4db oder .4dc zurück (Das Alias bzw. Tastenkürzel wird aufgelöst).
    - Bei einem .4dDatenbankordner/Package gibt die Funktion den Pfadnamen der Datei .4db oder .4dc innerhalb dieses Ordners/Package zurück.
- Wird die Funktion über eine Methode der Host Datenbank aufgerufen, gibt die Funktion immer den langen Namen der Strukturdatei der Host Datenbank zurück, unabhängig ob der Parameter * übergeben ist oder nicht.

Beispiel 1

Dieses Beispiel zeigt Name und Position der gerade verwendeten Strukturdatei an:

```
If(Application type#4D Remote mode)
   $vsStructureFilename:=Long name to file name(Structure file)
   $vsStructurePathname:=Long name to path name(Structure file)
   ALERT("Sie arbeiten gerade mit der Datenbank "
   +Char(34)+$vsStructureFilename+Char(34)+" sie liegt in "+Char(34)+$vsStructurePathname+Char(34)+".")
Else
   ALERT("Sie sind angemeldet an die Datenbank "+Char(34)+Structure file+Char(34))
End if
```

Hinweis: Weitere Informationen zu den Projektmethoden ***Long name to file name*** und ***Long name to path name*** finden Sie im Abschnitt **Nützliche Projektmethoden für Dokumente auf der Festplatte**.

Beispiel 2

Folgendes Beispiel fragt ab, ob die Methode über eine Komponente aufgerufen wird:

```
C_BOOLEAN($0)
$0:=(Structure file#Structure file(*))
  ` $0=True wenn die Methode über eine Komponente aufgerufen wird
```

VERIFY CURRENT DATA FILE

VERIFY CURRENT DATA FILE {(Objekte ; Optionen ; Methode {; TabellenArray {; FelderArray}})}

Parameter	Typ		Beschreibung
Objekte	Lange Ganzzahl	➡	Zu prüfende Objekte
Optionen	Lange Ganzzahl	➡	Zu prüfende Optionen
Methode	Text	➡	Name der 4D Callback Methode
TabellenArray	Array Lange Ganzzahl	➡	Nummmern der zu prüfenden Tabellen
FelderArray	Array 2D Ganzzahl, Array 2D Lange Ganzzahl, Array 2D Zahl	➡	Nummern der zu prüfenden Indizes

Beschreibung

Der Befehl **VERIFY CURRENT DATA FILE** prüft die Struktur der Objekte in der 4D Datendatei, die 4D derzeit geöffnet hat. Dieser Befehl arbeitet genauso wie der Befehl **VERIFY DATA FILE** mit dem Unterschied, dass er nur für die aktuelle Datendatei der offenen Anwendung gilt. Er benötigt deshalb keine Parameter zum Definieren der Struktur- und der Datendatei.
Die Beschreibung der Parameter finden Sie unter dem Befehl **VERIFY DATA FILE**.

Übergeben Sie den Befehl **VERIFY CURRENT DATA FILE** ohne Parameter, wird die Überprüfung mit den Standardwerten der Parameter ausgeführt:

- *Objekte* = Alles prüfen (= Wert 16)
- *Optionen* = 0 (Prüfbericht wird angelegt)
- *Methode* = ""
- *TabellenArray* und *FelderArray* werden weggelassen

Nach Ausführen dieses Befehls wird der Daten-Cache geleert und alle Operationen, die auf Daten zugreifen, sind während der Überprüfung gesperrt.

Systemvariablen und Mengen

Existiert die Callback Methode nicht, wird keine Überprüfung durchgeführt, ein Fehler erzeugt und die Systemvariable OK auf 0 (Null) gesetzt. Wurde ein Prüfbericht generiert, wird sein kompletter Pfadname in der Systemvariable *Document* zurückgegeben.

VERIFY DATA FILE

VERIFY DATA FILE (StrukturPfad ; DatenPfad ; Objekte ; Optionen ; Methode {; TabellenArray {; FelderArray}})

Parameter	Typ		Beschreibung
StrukturPfad	Text	➡	Pfadname der zu prüfenden 4D Strukturdatei
DatenPfad	Text	➡	Pfadname der zu prüfenden Datendatei
Objekte	Lange Ganzzahl	➡	Zu prüfende Objekte
Optionen	Lange Ganzzahl	➡	Zu prüfende Optionen
Methode	Text	➡	Name der 4D Callback Methode
TabellenArray	Array Lange Ganzzahl	➡	Nummern der zu prüfenden Tabellen
FelderArray	Array 2D Ganzzahl, Array 2D Lange Ganzzahl, Array 2D Zahl	➡	Nummern der zu prüfenden Indizes

Beschreibung

Der Befehl **VERIFY DATA FILE** prüft die Struktur der Objekte in der 4D Datendatei, definiert durch die Parameter *StrukturPfad* und *DatenPfad*.

Hinweis: Weitere Informationen dazu finden Sie im Abschnitt **Beschreibung der 4D Dateien** des Handbuchs *4D Designmodus*.

- *StrukturPfad* bestimmt die Strukturdatei (kompiliert oder nicht), die der zu prüfenden Datendatei zugeordnet ist. Das kann die offene Strukturdatei oder jede andere Strukturdatei sein. Sie müssen den vollständigen Pfadnamen in der Schreibweise des Betriebssystems angeben. Sie können auch einen leeren String übergeben. Dann erscheint der Standard Öffnen-Dialog, in dem der Benutzer die entsprechende Strukturdatei auswählen kann.
- *DatenPfad* bestimmt die 4D Datendatei (.4DD). Sie muss zur Strukturdatei passen, die im Parameter *Strukturdatei* definiert wurde.

Achtung: Sie können die aktuelle Strukturdatei bestimmen, die Datendatei muss jedoch nicht die aktuelle (offene) Datei sein. Mit dem Befehl **VERIFY CURRENT DATA FILE** können Sie prüfen, ob die Datendatei offen ist. Versuchen Sie, die aktuelle Datendatei mit **VERIFY DATA FILE** zu prüfen, wird ein Fehler erzeugt.

Die festgelegte Datendatei wird im Nur-Lesen Modus geöffnet. Stellen Sie sicher, dass keine Anwendung im Schreibmodus auf diese Datei zugreift, denn das kann die Ergebnisse der Prüfung beeinträchtigen.
Im Parameter *DatenPfad* können Sie einen leeren String, einen Dateinamen oder einen kompletten Pfadnamen in der Schreibweise des Betriebssystems übergeben. Bei einem leeren String erscheint der Standard Öffnen-Dialog, so dass der Benutzer die entsprechende Datei auswählen kann. Beachten Sie, dass Sie dann nicht die aktuelle Datendatei auswählen können. Übergeben Sie nur den Namen einer Datendatei, sucht 4D danach auf derselben Ebene wie die festgelegte Strukturdatei.

- Im Parameter *Objekte* definieren Sie, welche Objekttypen geprüft werden sollen. Es gibt zwei Typen: Datensätze und Indizes. Sie können folgende Konstanten unter dem Thema **Datendatei Wartung** verwenden:

Konstante	Typ	Wert	Kommentar
Verify all	Lange Ganzzahl	16	
Verify indexes	Lange Ganzzahl	8	Diese Option prüft die physikalische Konsistenz der Indizes, ohne Verknüpfung zu den Daten. Sie zeigt ungültige Verknüpfungen an, kann jedoch keine doppelten Verknüpfungen ausfindig machen (zwei Indizes zeigen auf den gleichen Datensatz). Diese Art Fehler lässt sich nur mit der Option Verify all ausfindig machen.
Verify records	Lange Ganzzahl	4	

  Um Datensätze und Indizes zu prüfen, übergeben Sie die Summe aus Verify Records+Verify Indexes. Mit dem Wert 0 (Null) erhalten Sie das gleiche Ergebnis. Die Option Verify All führt eine vollständige interne Überprüfung aus (Datensätze + Indizes). Diese Überprüfung ist kompatibel mit dem Erstellen eines Prüfberichts.
- Im Parameter *Optionen* setzen Sie Optionen zum Überprüfen. Unter dem Thema **Datendatei Wartung** gibt es folgende Optionen.

Konstante	Typ	Wert	Kommentar
Do not create log file	Lange Ganzzahl	16384	Dieser Befehl erstellt generell ein Logbuch im XML Format. Mit dieser Option wird kein Logbuch angelegt.
Timestamp log file name	Lange Ganzzahl	262144	Ist diese Option übergeben, zeigt der Name des generierten Logbuchs Datum und Uhrzeit seiner Erstellung und ersetzt dann folglich nicht das vorige Logbuch. Standardmäßig hat der Name eines Logbuchs keinen Zeitstempel und das vorige Logbuch wird überschrieben.

  Der Befehl **VERIFY DATA FILE** legt generell einen Prüfbericht im XML-Format an (siehe am Ende der Beschreibung). Übergeben Sie diese Option, wird kein Prüfbericht angelegt. Um den Prüfbericht anzulegen, übergeben Sie 0 (Null) in *Optionen*.
- Mit dem Parameter *Methode* setzen Sie eine Callback Methode, die während der Überprüfung regelmäßig aufgerufen wird. Übergeben Sie einen leeren String, wird keine Methode aufgerufen. Existiert die übergebene Methode nicht, wird die Überprüfung nicht ausgeführt. Es wird ein Fehler erzeugt und die Variable OK wird auf 0 (Null) gesetzt.
  Wird die Methode aufgerufen, kann sie je nach den zu prüfenden Objekten und dem auslösendem Ereignis bis zu 5 Parameter empfangen (siehe Tabelle Aufrufe). Sie müssen diese Parameter in der Methode deklarieren:

- $1 Lange Ganzzahl Meldungstyp (siehe Tabelle)
- $2 Lange Ganzzahl Objekttyp
- $3 Text Meldung
- $4 Lange Ganzzahl Tabellennummer
- $5 Lange Ganzzahl Reserviert

Nachfolgende Tabelle beschreibt den Inhalt der Parameter je nach Ereignistyp:

Ereignis	$1 (Lange Ganzzahl)	$2 (Lange Ganzzahl)	$3 (Text)	$4 (Lange Ganzzahl)	$5 (Lange Ganzzahl)
Meldung	1	0	Meldung über Verlauf	Erledigter Prozentsatz (0-100)	Reserviert
Überprüfungsende (*)	2	Objekttyp (**)	Meldungstext OK	Tabellen- oder Indexnummer	Reserviert
Fehler	3	Objekttyp (**)	Text der Fehlermeldung	Tabellen- oder Indexnummer	Reserviert
Ausführungsende	4	0	Fertig	0	Reserviert
Warnung	5	Objekttyp (**)	Text der Fehlermeldung	Tabellen- oder Indexnummer	Reserviert

(*) Ist der Modus Verify All, wird nie das Ereignis *Überprüfungsende* ($1=2) zurückgegeben. Es wird nur für die Modi Verify Records oder Verify Indexes verwendet.
(**) Objekttyp: Wurde ein Objekt bestätigt, kann eine Meldung "beendet" ($1=2), Fehler ($1=3) oder Warnung ($1=5) gesendet werden. Für $2 sind folgende Objekttypen möglich:

- 0 = unbestimmt
- 4 = Datensatz
- 8 = Index
- 16 = Strukturobjekt (Vorabprüfung der Datendatei).

Sonderfall: Ist $4 = 0 für $1=2, 3 oder 5 betrifft die Meldung keine Tabelle oder Index, sondern die gesamte Datendatei.

Die Callback Methode muss auch einen Wert in $0 zurückgeben (Lange Ganzzahl), der die Ausführung der Operation prüft:

- Ist $0 = 0, läuft die Operation normal weiter
- Ist $0 = -128, stoppt die Operation ohne Erzeugen eines Fehlers
- Ist $0 = ein anderer Wert, stoppt die Operation und der in $0 übergebene Wert wird als Fehlernummer zurückgegeben. Dieser Fehler lässt sich über eine Fehlerverwaltungsmethode abfangen.

Hinweis: Wurde das Ereignis ($4=1) generiert, können Sie die Ausführung nicht via $0 unterbrechen.

Dieser Befehl kann auch 2 optionale Arrays verwenden:

- Das Array *TabellenArray* enthält die Nummern der Tabellen, deren Datensätze geprüft werden sollen. Damit können Sie die Überprüfung auf bestimmte Tabellen begrenzen. Ist dieser Parameter nicht übergeben, oder das Array leer und der Parameter *Objekte* enthält die Konstante Verify Records, werden alle Tabellen geprüft.
- Das Array *FelderArray* enthält die Nummern der indizierten Felder, deren Indizes überprüft werden sollen.
  Ist dieser Parameter nicht übergeben, oder das Array leer und der Parameter *Objekte* enthält die Konstante Verify Indexes, werden alle Indizes geprüft.
  Der Befehl ignoriert nicht-indizierte Datenfelder.
  Enthält ein Datenfeld mehrere Indizes, werden alle geprüft. Ist das Datenfeld Teil eines zusammengesetzten Index, wird der gesamte Index geprüft.
  Sie müssen in *FelderArray* ein 2D Array übergeben. Für jede Zeile des Array gilt:
  - Das Element {0} enthält die Tabellennummer
  - Die anderen Elemente {1...x} enthalten die Nummern der Datenfelder.

Der Befehl **VERIFY DATA FILE** erstellt standardmäßig einen Prüfbericht im XML-Format (wenn Sie nicht die Konstante Do not create log file übergeben haben, siehe Parameter *Optionen*). Er liegt im Ordner **Logs** der aktuellen Datenbank und hat den Namen ihrer Strukturdatei.
Beispiel: Bei einer Strukturdatei mit Namen “myDB” lautet der Prüfbericht “myDB_Verify_Log.xml.”
Haben Sie die Option Timestamp log file name übergeben, zeigt der Name des Prüfberichts auch Datum und Uhrzeit der Erstellung im Format "YYYY-MM-DD HH-MM-SS", der Name im Beispiel lautet dann: “myDB_Verify_Log_2015-09-27 15-20-35.xml”. Das bedeutet, dass ein neuer Prüfbericht nicht den vorigen ersetzt und dann nicht benötigte Dateien per Hand entfernt werden müssen.
Unabhängig von der gewählten Option wird der Pfad des generierten Prüfberichts nach Ausführung des Befehls in der Systemvariablen *Document* zurückgegeben.

Beispiel 1

Einfache Überprüfung von Daten und Indizes:

```
VERIFY DATA FILE($StructName;$DataName;Verify indexes+Verify records;Do not create log file;"")
```

Beispiel 2

Vollständige Überprüfung mit Prüfbericht

```
VERIFY DATA FILE($StructName;$DataName;Verify all;0;"")
```

Beispiel 3

Nur Datensätze überprüfen

```
VERIFY DATA FILE($StructName;$DataName;Verify records;0;"")
```

Beispiel 4

Nur Datensätze der Tabellen 3 und 7 überprüfen:

```
ARRAY LONGINT($arrTableNums;2)
$arrTableNums{1}:=3
$arrTableNums{2}:=7
VERIFY DATA FILE($StructName;$DataName;Verify records;0;"FollowScan";$arrTableNums)
```

Beispiel 5

Bestimmte Indizes überprüfen (Index von Feld 1 der Tabelle 4 und Index der Felder 2 und 3 der Tabelle 5)

```
ARRAY LONGINT($arrTableNums;0)  `wird nicht verwendet, ist aber zwingend
ARRAY LONGINT($arrIndex;2;0)  `2 Zeilen (Spalten werden später hinzugefügt)
$arrIndex{1}{0}:=4  `Tabellennummer in Element 0
APPEND TO ARRAY($arrIndex{1};1)  `Nummer des 1. zu prüfenden Feldes
$arrIndex{2}{0}:=5  `Tabellennummer in Element 0
APPEND TO ARRAY($arrIndex{2};2)  `Nummer des 1. zu prüfenden Feldes
APPEND TO ARRAY($arrIndex{2};3)  `Nummer des 2. zu prüfenden Feldes
VERIFY DATA FILE($StructName;$DataName;Verify indexes;0;"FollowScan";$arrTableNums;$arrIndex)
```

Systemvariablen und Mengen

Existiert die Callback Methode nicht, wird keine Überprüfung durchgeführt, ein Fehler erzeugt und die Systemvariable OK auf 0 (Null) gesetzt. Wurde ein Prüfbericht generiert, wird sein kompletter Pfadname in der Systemvariable *Document* zurückgegeben.

Version type

Version type -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Demo- oder Vollversion, 64-bit oder 32-bit Version

Beschreibung

Die Funktion **Version type** gibt einen numerischen Wert zurück, der die Art der laufenden Version von 4D oder 4D Server anzeigt. 4D bietet folgende vordefinierten Konstanten unter dem Thema **4D Umgebung**:

Konstante	Typ	Wert	Kommentar
64 bit version	Lange Ganzzahl	1	
Demo version	Lange Ganzzahl	0	
Merged application	Lange Ganzzahl	2	Version ist eine Anwendung mit einkompilierter 4D Volume Desktop

Hinweis: In den aktuellen 4D Versionen ist der Demo-Modus nicht verfügbar.

Hinweis: Version type gibt einen Wert in Form eines bit Feldes zurück; für seine Interpretation müssen bitwise Operatoren verwendet werden (siehe Beispiele).

Hinweis zur Kompatibilität: In 4D Versionen vor 13.2 waren für diese Funktion andere Konstanten verfügbar. Sie wurden geändert, da sie einige Fälle nicht richtig bearbeitet haben. Als Konsequenz müssen Sie auch Ihren Code anpassen (siehe Beispiel). Wenn Sie lieber die bisherige Funktionsweise beibehalten wollen, können Sie die Konstanten in Ihrem Code mit den vorherigen Werten ersetzen: 2 für 64 bit Version, 1 für Demo Version, 0 für Standardversion.

Beispiel 1

Ihre 4D Anwendung enthält spezifischen Code, der auf der laufenden Version basiert. Sie können die Ausführungsumgebung mit folgendem Code testen:

```
If(Version type?? 64 bit Version)
  // Wir sind in einer 64-bit Version
Else
  // Wir sind in einer 32-bit Version
End if
```

Beispiel 2

Dieser Test ermöglicht unterschiedlichen Code, je nachdem, ob die Version eine Anwendung mit einkompilierter 4D Volume License ist oder eine Datenbank, die mit 4D oder 4D Server geöffnet wurde:

```
If(Version type?? Merged application)
  // Wir sind in einer Anwendung mit einkompilierter 4D Volume License
Else
  // Wir sind in einer Datenbank, die von 4D ausgeführt wird
End if
```

_o_ADD DATA SEGMENT

_o_ADD DATA SEGMENT
Dieser Befehl benötigt keine Parameter

Beschreibung

Hinweis zur Kompatibilität: Datensegmente werden ab 4D Version 11 nicht mehr unterstützt, da die Größe von Dateien jetzt virtuell unbegrenzt ist. Wird dieser Befehl aufgerufen, führt er nichts aus.

_o_DATA SEGMENT LIST

_o_DATA SEGMENT LIST (Segmente)

Parameter	Typ		Beschreibung
Segmente	Array String	⇦	Kompletter Name der Datensegmente für die Datenbank

Hinweis zur Kompatibilität

Seit 4D Version 11 werden Datensegmente nicht mehr unterstützt (die Größe der Datendatei ist unbegrenzt). Dieser Befehl gibt jetzt systematisch ein Array mit einem Element zurück, das den Pfadnamen der Datendatei der Anwendung enthält.

Arrays

- Einführung in Arrays
- Arrays erstellen
- Arrays und Formularobjekte
- Arrays und die 4D Programmiersprache
- Arrays und Zeiger
- Element Null eines Array verwenden
- Zweidimensionale Arrays
- Arrays und Speicher
- 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*

Einführung in Arrays

In **Arrays** werden Variablen des gleichen Typs abgelegt oder gruppiert. Jeder Wert im Array wird als **Element** bezeichnet. Die **Größe** des Array ist die Anzahl der darin enthaltenen Elemente. Beim Erstellen wird eine Größe festgelegt, die beliebig veränderbar ist. Sie können Elemente hinzufügen, einfügen oder löschen bzw. das Array mit demselben Befehl anpassen, der es erstellt hat.

Sie erstellen ein Array mit einem der Array-Befehle. Weitere Informationen dazu finden Sie im Abschnitt **Arrays erstellen**.

Elemente werden von **1 bis N** durchnumeriert. N ist die Größe des Array. Ein Array hat immer ein **Element Null**. Sie können darauf wie auf jedes andere Element im Array zugreifen, es erscheint jedoch nicht beim Anzeigen in einem Formular. Es kann trotzdem uneingeschränkt in der Programmiersprache verwendet werden. Weitere Informationen dazu finden Sie im Abschnitt **Element Null eines Array verwenden**.

Arrays sind 4D Variablen. Von daher hat ein Array einen Bereich und wendet bis auf wenige Ausnahmen die Regeln der 4D Programmiersprache an. Weitere Informationen dazu finden Sie in den Abschnitten **Arrays und die 4D Programmiersprache** sowie **Arrays und Zeiger**.

Arrays sind Objekte der Programmiersprache; Sie können Arrays erstellen und einsetzen, die für den Benutzer unsichtbar bleiben. Arrays sind ebenso Objekte der Benutzeroberfläche. Weitere Informationen dazu finden sie in den Abschnitten **Arrays und Formularobjekte** und **Gruppierte rollbare Bereiche**.

In Arrays werden überschaubare Datenmengen für eine kurze Zeitspanne im Hauptspeicher gehalten. Array-Operationen sind daher sehr schnell. Sie können die Array-Elemente kopieren, sortieren, suchen ….. Benötigen Sie Arrays nicht mehr, löschen Sie sie, um Speicherplatz zu sparen. Weitere Informationen dazu finden Sie im Abschnitt **Arrays und Speicher**.

Arrays erstellen

Sie erstellen ein Array mit einem Befehl, der ein Array deklariert. Es gibt folgende Arten von Array-Befehlen:

Befehl	Erstellt ein Array oder passt seine Größe an von
ARRAY INTEGER	2 Byte Werten vom Typ Ganzzahl
ARRAY LONGINT	4 Byte Werten vom Typ Lange Ganzzahl
ARRAY REAL	Werten vom Typ Zahl
ARRAY TEXT	Werten vom Typ Text (bis zu 2 GB Text pro Element) (*)
_o_ARRAY STRING	Werten vom Typ Text (obsolet)
ARRAY DATE	Werten vom Typ Datum
ARRAY BOOLEAN	Werten vom Typ Boolean
ARRAY PICTURE	Werten vom Typ Bild
ARRAY POINTER	Werten vom Typ Zeiger
ARRAY OBJECT	Programmiersprache Objekten
ARRAY BLOB	BLOBs
ARRAY TIME	Zeiten

Jeder dieser Befehle kann ein- oder zweidimensionale Arrays erstellen oder in der Größe anpassen. Weitere Informationen zu zweidimensionalen Arrays finden Sie im Abschnitt **Zweidimensionale Arrays**.

(*) Es gibt keinen Unterschied zwischen Arrays vom Typ Text und String. Der Parameter *strLen* im Befehl **_o_ARRAY STRING** wird ignoriert. Wir empfehlen, Arrays vom Typ Text zu verwenden. Der Befehl **_o_ARRAY STRING** wird nur zur Wahrung der Kompatibilität beibehalten.

Anlegen eines Array vom Typ Ganzzahl mit 10 Elementen:

```
ARRAY INTEGER(aiAnArray;10)
```

Anpassen dieses Array auf 20 Elemente:

```
ARRAY INTEGER(aiAnArray;20)
```

Anpassen dieses Array auf keine Elemente:

```
ARRAY INTEGER(aiAnArray;0)
```

Um auf die Elemente zuzugreifen, schreiben Sie den Tabellennamen, gefolgt vom Indexwert in geschweiften Klammern ({…}). Dieser Wert heißt **Elementnummer**. Folgender Code fügt fünf Namen in das Array mit Namen *atNames* ein und zeigt sie dann in Fenstern mit Meldungen an:

```
ARRAY TEXT(atNames;5)
atNames{1}:="Richard"
atNames{2}:="Sarah"
atNames{3}:="Sam"
atNames{4}:="Jane"
atNames{5}:="John"
For($vlElem;1;5)
   ALERT("Element #"+String($vlElem)+" ist gleich: "+atNames{$vlElem})
End for
```

Sie können das Element auch über eine Variable vom Typ numerisch ansprechen, z.B. *atNames{$vlElem}*.
Über eine Schleife (**For...End for**, **Repeat...Until** oder **While...End while**) können kompakte Teile des Code alle oder bestimmte Elemente in einem Array ansprechen.

Arrays und andere Bereiche der 4D Programmiersprache

Auch andere 4D Befehle können Arrays erstellen und damit arbeiten:

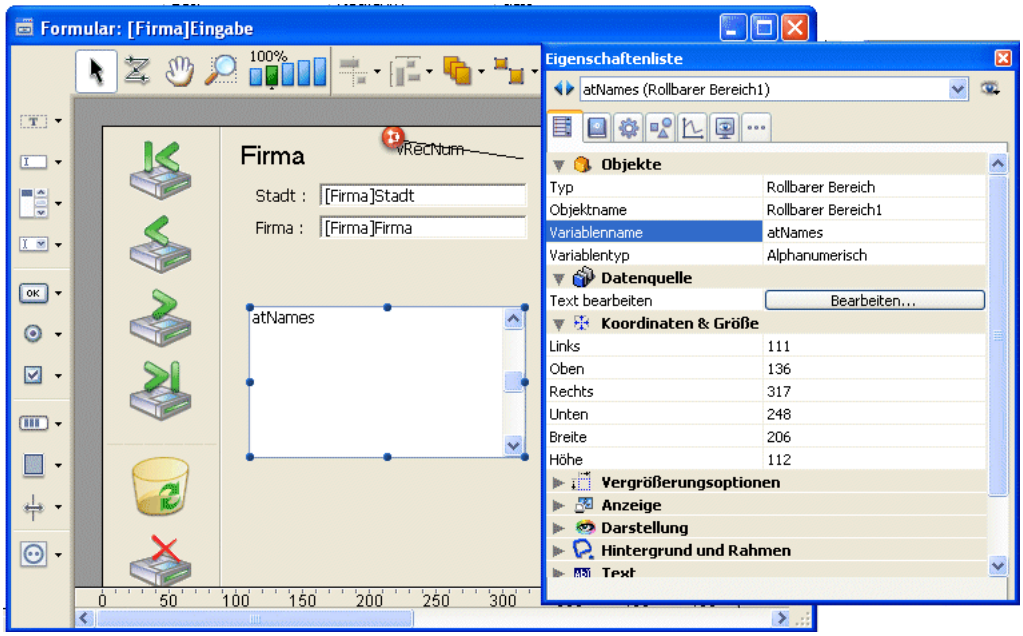
- Mit den Befehlen **SELECTION RANGE TO ARRAY**, **SELECTION TO ARRAY**, **ARRAY TO SELECTION** und **DISTINCT VALUES** arbeiten Sie mit Arrays und Datensatzauswahlen.
- Objekte vom Typ Listbox basieren auf Arrays; einige Befehle unter dem Thema "Listbox" arbeiten mit Arrays, z.B. **LISTBOX INSERT ROWS**.
- Mit dem Befehl **GRAPH** erstellen Sie Grafiken und Diagramme über in Tabellen und Arrays gespeicherte Werte.
- Die Befehle **LIST TO ARRAY** und **ARRAY TO LIST**.
- Viele Befehle erstellen Arrays in einem Aufruf, z.B. **FONT LIST**, **WINDOW LIST**, **VOLUME LIST**, **FOLDER LIST**, **DOCUMENT LIST**, **GET SERIAL PORT MAPPING**, **SAX GET XML ELEMENT**, etc.

Arrays und Formularobjekte

Arrays sind Objekte der Programmiersprache —Sie können Arrays erstellen und benutzen, die für den Benutzer unsichtbar sind. Arrays sind ebenso Objekte der Benutzeroberfläche. Arrays unterstützen folgende Arten von **Formularobjekten**:

- PopUp-Menü
- Dropdown-Listen
- Combo Boxen
- Rollbare Bereiche (ab 4D v13 überholt)
- Registerkarten
- Listboxen

Sie können diese Objekte entweder im Designmodus im Formulareditor über die Eigenschaftenliste mit der Schaltfläche **Standardwerte einrichten** oder über die Array-Befehle programmieren (außer für Listboxen).
Bei diesen Objekten können Sie das Array nach einem **ausgewählten Element** abfragen und so herausfinden, welches Element innerhalb des Objekts ausgewählt bzw. angeklickt wurde.
Setzen Sie ein Array zur Unterstützung eines Formularobjekts ein, ist es sowohl ein Objekt der Programmiersprache als auch der Benutzeroberfläche. Sie erstellen z.B. einen rollbaren Bereich in einem Formular:



Der Name der zugewiesenen Variablen, hier *atNames* ist der Name des Array, das Sie zum Erstellen und Verwalten des rollbaren Bereichs verwenden.

Hinweise:

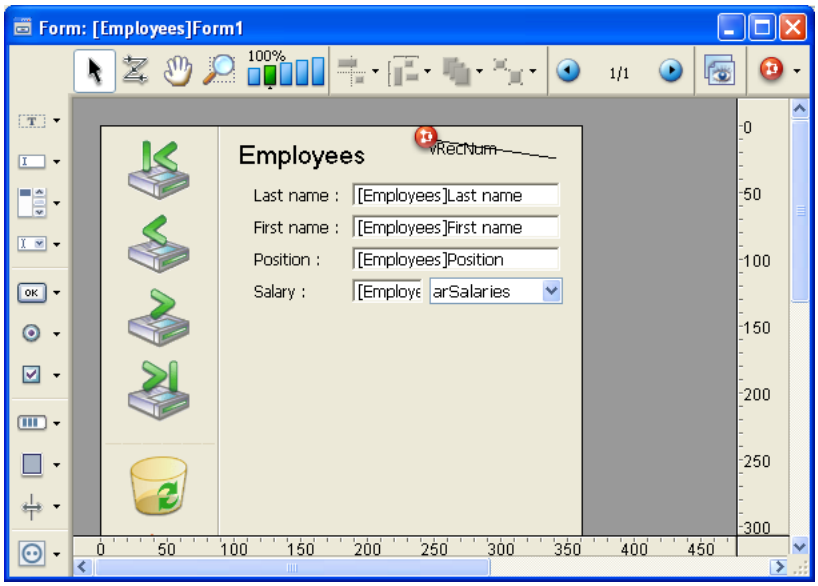
- Das **ausgewählte Element** des Array wird intern in einer Variablen vom Typ Ganzzahl gespeichert. Deshalb lässt es sich nicht mit Arrays verwenden, die mehr als 32.767 Elemente enthalten. Darüberhinaus passt solch eine hohe Anzahl an Elementen auch nicht zur Anzeige als Formularobjekt.
- Sie können keine zweidimensionalen Arrays oder Zeiger-Arrays anzeigen.
- Die Verwaltung von Objekten des Typs **Listbox**, welches mehrere Arrays enthalten kann, ist sehr vielseitig. Ausführliche Informationen dazu finden Sie im Abschnitt **Einführung in Listboxen**.

Eine Dropdown-Liste erstellen

Folgendes Beispiel zeigt, wie Sie ein Array füllen und in einer Dropdown-Liste anzeigen. Mit dem Befehl **ARRAY REAL** erstellen Sie ein Array mit Namen *arSalaries*. Es enthält alle Standardlöhne, die in einer Firma ausgezahlt werden. Wählt der Benutzer ein Element aus der Dropdown-Liste, wird dem Datenfeld *[Employees]Salary* der in der Design- oder Anwendungsumgebung gewählte Wert zugewiesen.

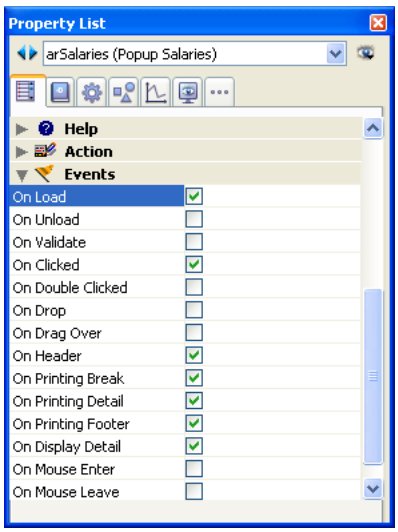
Dropdown-Liste arSalaries in Formular anlegen

Erstellen Sie eine Dropdown-Liste mit Namen *arSalaries*. Er sollte mit dem Namen des Array identisch sein.

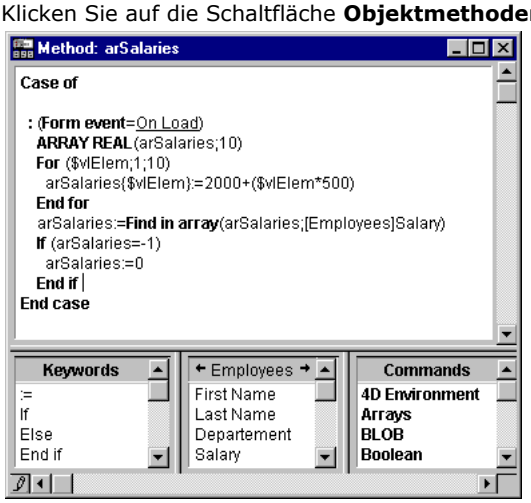


Array initialisieren

Initialisieren Sie das Array *arSalaries* mit dem Ereignis *On Load* für das Objekt. Dazu müssen Sie dieses Ereignis in der **Eigenschaftenliste** aktivieren:



Klicken Sie auf die Schaltfläche **Objektmethoden** und erstellen Sie folgende Methode:



Die Programmierzeilen:

```
ARRAY REAL(arSalaries;10)
For($vlElem;1;10)
    arSalaries{$vlElem}:=2000+($vlElem*500)
End for
```

erstellen ein Array mit den Zahlen 2500, 3000... 7000.

Die Programmierzeilen:

```
arSalaries:=Find in array(arSalaries;[Employees]Salary)
If(arSalaries=-1)
```

```
    arSalaries:=0
End if
```

erzeugen einen neuen Datensatz oder ändern einen bestehenden Datensatz.

- Erstellen Sie einen neuen Datensatz, ist das Datenfeld *[Employees]Salary* zu Beginn gleich Null. In diesem Fall findet **Find in array** keinen Wert im Array und gibt -1 zurück. Die Abfrage *If (arSalaries=-1)* setzt *arSalaries* auf Null und zeigt so an, dass in der Dropdown-Liste kein Element ausgewählt ist.
- Ändern Sie einem bestehenden Datensatz, findet **Find in array** den Wert im Array und setzt das ausgewählte Element der Dropdown-Liste auf den aktuellen Wert des Datenfeldes. Befindet sich der Wert für einen bestimmten "Employe" nicht in der Liste, hebt die Abfrage *If (arSalaries=-1)* die Auswahl in der Liste auf.

Hinweis: Weitere Informationen dazu finden Sie im nächsten Abschnitt.

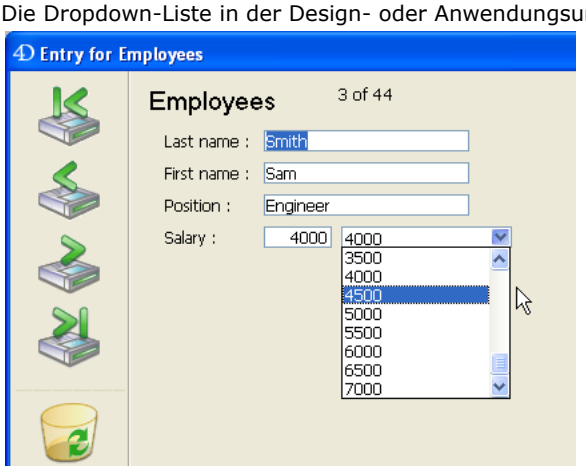
Ausgewählten Wert in Datenfeld [Employees]Salary übertragen

Um den ausgewählten Wert aus der Dropdown-Liste *arSalaries* zu übertragen, müssen Sie nur das Ereignis *On Clicked* auf das Objekt anwenden. Die Nummer des ausgewählten Elements ist der Wert des Array *arSalaries* selbst. Der Ausdruck *arSalaries{arSalaries}* gibt so den in der Drop-Down-Liste gewählten Wert zurück.

Vervollständigen Sie die Methode für das Objekt *arSalaries* wie folgt:

```
Case of
   :(Form event=On Load)
      ARRAY REAL(arSalaries;10)
      For($vlElem;1;10)
         arSalaries{$vlElem}:=2000+($vlElem*500)
      End for
      arSalaries:=Find in array(arSalaries;[Employees]Salary)
      If(arSalaries=-1)
         arSalaries:=0
      End if
   :(Form event=On Clicked)
      [Employees]Salary:=arSalaries{arSalaries}
End case
```

Die Dropdown-Liste in der Design- oder Anwendungsumgebung sieht folgendermaßen aus:



Der nächste Abschnitt beschreibt die allgemeingültigen Operationen für Arrays, die Sie als Formularobjekte verwenden.

Tabellengröße feststellen

Mit der Funktion **Size of array** erhalten Sie die aktuelle Größe des Array. Gehen wir vom vorigen Beispiel aus. Folgende Programmierzeile würde 5 anzeigen:

```
ALERT("The size of the array atNames is: "+String(Size of array(atNames)))
```

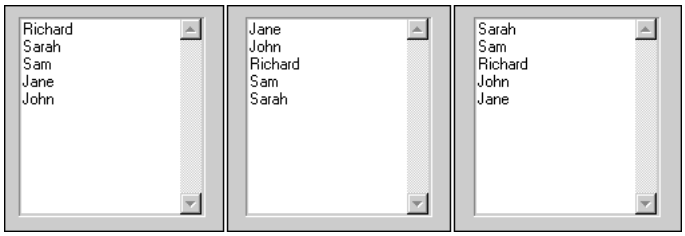
Elemente des Array neu ordnen

Mit dem Befehl **SORT ARRAY** ordnen Sie die Elemente eines Array neu, mit dem Befehl **MULTI SORT ARRAY** mehrere Arrays. Wir wählen wieder das vorige Beispiel und gehen davon aus, dass das Array als rollbarer Bereich angezeigt wird:

a. Zu Beginn sieht der Bereich wie die Liste auf der linken Seite aus.

b. Nach Ausführen der Programmierzeile: *SORT ARRAY(atNames;>)* sieht der Bereich wie die Liste in der Mitte aus.

c. Nach Ausführen der Programmierzeile: *SORT ARRAY(atNames;<)* sieht der Bereich wie die Liste auf der rechten Seite aus.



Elemente hinzufügen oder löschen

Mit den Befehlen **APPEND TO ARRAY**, **INSERT IN ARRAY** und **DELETE FROM ARRAY** löschen oder fügen Sie Elemente hinzu.

Klicks in Array: Das ausgewählte Element testen

Wir nehmen das vorige Beispiel und gehen davon aus, dass das Array als rollbarer Bereich angezeigt wird. Klicks in diesem Bereich fragen Sie wie folgt ab:

```
  ` Objektmethode rollbarer Bereich atNames
Case of
   :(Form event=On Load)
  ` Initialisiere Array (wie oben angezeigt)
      ARRAY TEXT(atNames;5)
  ` ...
   :(Form event=On Unload)
  ` Wir benötigen das Array nicht mehr
      CLEAR VARIABLE(atNames)

   :(Form event=On Clicked)
      If(atNames#0)
         vtInfo:="Sie haben geklickt auf: "+atNames{atNames}
      End if
   :(Form event=On Double Clicked)
      If(atNames#0)
         ALERT("Sie haben doppelgeklickt auf: "+atNames{atNames}
      End if
End case
```

Hinweis: Diese Ereignisse müssen in der Eigenschaftenliste für das Objekt aktiviert sein.

Mit der Syntax *atNames{$vlElem}* können Sie mit einem bestimmten Element des Array arbeiten, während die Syntax *atNames* die Nummer des ausgewählten Elements im Array zurückgibt. Die Syntax *atNames{atNames}* bedeutet also "Wert des ausgewählten Elements im Array *atNames*". Ist kein Element ausgewählt, hat *atNames* den Wert 0 (Null), die Abfrage *If (atNames#0)* findet heraus, ob derzeit ein Element ausgewählt ist oder nicht.

Ausgewähltes Element setzen

In ähnlicher Weise können Sie per Programmierung das ausgewählte Element ändern, indem Sie dem Array einen Wert zuweisen.

Beispiele

```
  ` Wählt das erste Element (wenn das Array nicht leer ist)
atNames:=1

  ` Wählt das letzte Element (wenn das Array nicht leer ist)
atNames:=Size of array(atNames)

  ` Deaktiviert das ausgewählte Element (sofern vorhanden), es ist dann kein Element ausgewählt
atNames:=0

If((0<atNames)&(atNames<Size of array(atNames))
  ` Wählt, falls möglich, das nächste Element nach dem ausgewählten Element
   atNames:=atNames+1
End if

If(1<atNames)
  ` Wählt, falls möglich, das nächste Element vor dem ausgewählten Element
   atNames:=atNames-1
End if
```

Wert in Array suchen

Die Funktion **Find in array** sucht nach einem bestimmten Wert im Array. Im oben erwähnten Beispiel wählt folgender Code das Element mit dem Wert "Richard," wenn dies im Dialogfenster Request eingetragen ist:

```
$vsName:=Request("Gib den Vornamen ein:")
If(OK=1)
    $vlElem:=Find in array(atNames;$vsName)
    If($vlElem>0)
        atNames:=$vlElem
    Else
        ALERT("Es gibt keinen "+$vsName+" in dieser Liste mit Vornamen.")
    End if
End if
```

PopUp-Menüs, Dropdown-Listen, rollbare Bereiche und Registerkarten lassen sich normalerweise auf dieselbe Art abfragen. Natürlich ist kein weiterer Code erforderlich, um Objekte jedes Mal auf dem Bildschirm neu zu zeichnen, wenn Sie den Wert eines Elements ändern, hinzufügen oder löschen.

Hinweis: Für Registerkarten mit Icons und aktiven bzw. inaktiven Registern müssen Sie als unterstützendes Objekt eine hierarchische Liste verwenden. Weitere Informationen dazu finden Sie im Beispiel zur Funktion **Count tasks**.

Combo Boxen

PopUp-Menüs, Dropdown-Listen, rollbare Bereiche und Registerkarten verwalten Sie mit den oben beschriebenen Algorithmen. Combo Boxen dagegen müssen Sie anders verwalten.

Eine Combo Box ist ein eingebbarer Text, dem eine Liste mit Werten (die Elemente aus dem Array) angehängt ist. Der Benutzer kann einen Wert aus der Liste auswählen und dann den Text editieren. Von daher lässt sich der Begriff ausgewähltes Element nicht auf Combo Boxen anwenden. In Combo Boxen gibt es nie ein ausgewähltes Element. Immer wenn der Benutzer einen Wert aus der angehängten Liste auswählt, wird dieser in das Element Null des Array gelegt. Wenn der Benutzer dann den Text editiert, wird der vom Benutzer geänderte Wert ebenfalls in das Element Null gelegt.

Beispiel

```
  ` Objektmethode Combo Box asColors
Case of
    :(Form event=On Load)
        ARRAY STRING(31;asColors;3)
        asColors{1}:="Blue"
        asColors{2}:="White"
        asColors{3}:="Red"
    :(Form event=On Clicked)
        If(asColors{0}#"")
  ` Das Objekt ändert automatisch seinen Wert
  ` Das Ereignis On Clicked ist in einer Combo Box
  ` nur bei zusätzlichen Aktionen erforderlich
        End if
    :(Form event=On Data Change)
  ` Find in array ignoriert Element 0, gibt also -1 oder >0 zurück
        If(Find in array(asColors;asColors{0})<0)
  ` Eingegebener Wert gehört nicht zu den diesem Objekt angehängten Werten
  ` Füge Wert für das nächste Mal in der Liste hinzu.
            APPEND TO ARRAY(asColors;asColors{0})
        Else
  ` Eingegebener Wert gehört zu den diesem Objekt angehängten Werten
        End if
End case
```

Arrays und die 4D Programmiersprache

Arrays sind 4D Variablen. Von daher haben Sie einen Geltungsbereich und folgen – bis auf wenige Ausnahmen – den Regeln der 4D Programmiersprache.

Lokale, Prozess- und Interprozess-Arrays

Sie können lokale, Prozess- oder Interprozess-Arrays erstellen und einsetzen. Beispiele:

```
ARRAY INTEGER($aiCodes;100)
  ` Erstellt ein lokales Array mit 100 2-byte Werten vom Typ Zahl
ARRAY INTEGER(aiCodes;100)
  ` Erstellt ein Prozess-Array mit 100 2-byte Werten vom Typ Zahl
ARRAY INTEGER(◊aiCodes;100)
  ` Erstellt ein Interprozess-Array mit 100 2-byte Werten vom Typ Zahl
```

Der Geltungsbereich dieser Arrays ist derselbe wie für lokale, Prozess- und Interprozessvariablen:

Lokale Arrays

Lokale Arrays beginnen mit dem Dollarzeichen (*$*).

Sie gelten für die Methode, in der sie erstellt wurden und werden gelöscht, sobald die Methode endet. Sie können in zwei verschiedenen Methoden denselben Namen haben, denn es sind in der Tat zwei verschiedene Variablen mit unterschiedlichem Geltungsbereich.

Erstellen Sie ein lokales Array in einer Formularmethode, Objektmethode oder Projektmethode, die als Unterroutine von den beiden vorigen Methodenarten aufgerufen wird, wird das Array bei jedem Aufrufen der Formular- bzw. der Objektmethode erstellt und gelöscht. Mit anderen Worten, das Array wird für jedes Formularereignis neu erstellt und gelöscht. Folglich können Sie lokale Variablen weder für Formulare, noch zum Anzeigen oder Drucken verwenden.

Verwenden Sie lokale Arrays sowie lokale Variablen so oft wie möglich. Das verringert den Speicherplatz, der zum Laufen Ihrer Anwendung erforderlich ist.

Prozess-Arrays

Prozess-Arrays beginnen mit einem Buchstaben.

Sie gelten für den Prozess, in dem sie erstellt wurden und werden gelöscht, sobald der Prozess endet oder abbricht. Ein Prozess-Array hat automatisch eine Instanz pro Prozess. Deshalb ist das Array in den Prozessen vom selben Typ. Ihr Inhalt ist dagegen für jeden Prozess anders.

Interprozess-Arrays

Interprozess-Arrays beginnen unter Windows und auf Macintosh mit dem Symbol <>. Auf Macintosh kann auch das Zeichen ◊ (Wahl-/Großschreibtaste + Buchstabe v) verwendet werden. Sie gelten für alle Prozesse einer Arbeitssitzung. Sie sollten sie nur für den gemeinsamen Zugriff auf Daten und Übertragungsinformationen zwischen Prozessen einsetzen.

Tipp: Wissen Sie bereits im voraus, dass auf ein Interprozess-Array mehrere Prozesse zugreifen, was u.U. zu Konflikten führt, schützen Sie den Zugriff auf dieses Array mit einer Semaphore. Weitere Informationen dazu finden Sie im Beispiel zur Funktion **Semaphore**.

Hinweis: Sie können in Formularen auch Formularobjekte wie rollbare Bereiche, Dropdown-Listen, etc. über Prozess- und Interprozess-Arrays erstellen.

Ein Array als Parameter übergeben

Ein Array lässt sich einem 4D Befehl oder der Routine eines 4D Plug-In als Parameter übergeben, jedoch nicht in einer Benutzermethode. Dafür müssen Sie einen Zeiger auf das Array übergeben, der dann der Parameter für die Methode ist. Weitere Informationen dazu finden Sie im Abschnitt **Arrays und Zeiger**.

Array einem anderen Array zuweisen

Im Gegensatz zu Variablen vom Typ Text oder Alphanumerisch können Sie ein Array nicht einem anderen zuweisen. Dafür müssen Sie den Befehl **COPY ARRAY** verwenden.

Arrays und Zeiger

Sie können ein Array als Parameter für einen 4D Befehl oder eine Routine eines 4D Plug-In übergeben, jedoch nicht für eine Benutzermethode. Dafür müssen Sie dem Array einen Zeiger übergeben, der dann für die Benutzermethode als Parameter dient.
Sie können als Parameter Prozess- und Interprozess-Arrays, sowie Prozess- und lokale Arrays übergeben.

- Wir gehen von folgendem Beispiel aus:

```
If((0<atNames)&(atNames<Size of array(atNames))
   ` Wählt, falls möglich, das nächste Element nach dem ausgewählten Element
   atNames:=atNames+1
End if
```

Müssen Sie denselben Vorgang für 50 verschiedene Arrays in mehreren Formularen ausführen, vermeiden Sie mit nachfolgender Methode, 50 Mal dasselbe zu schreiben:

```
   ` Projektmethode SELECT NEXT ELEMENT
   ` SELECT NEXT ELEMENT ( Zeiger )
   ` SELECT NEXT ELEMENT ( -> Array )

   C_POINTER($1)

   If((0<$1->)&($1-><Size of array($1->))
      $1->:=$1->+1  ` Wählt, falls möglich, das nächste Element nach dem ausgewählten Element
   End if
```

Schreiben Sie nun:

```
   SELECT NEXT ELEMENT(->atNames)
   ` ...
   SELECT NEXT ELEMENT(->asZipCodes)
   ` ...
   SELECT NEXT ELEMENT(->alRecordIDs)
   ` ... usw.
```

- Folgende Projektmethode gibt die Summe aller Elemente eines numerischen Array zurück (Ganzzahl, Lange Ganzzahl, Zahl):

```
   ` Array sum
   ` Array sum ( Pointer )
   ` Array sum ( -> Array )

   C_REAL($0)

   $0:=0
   For($vlElem;1;Size of array($1->))
      $0:=$0+$1->{$vlElem}
   End for
```

Hinweis: Seit 4D v13 können Sie einfach die Funktion **Sum** verwenden, um die Summe von Elementen in einem numerischen Array zu berechnen.

```
   $vlSum:=Array sum(->arSalaries)
   ` ...
   $vlSum:=Array sum(->aiDefectCounts)
   ` ..
   $vlSum:=Array sum(->alPopulations)
```

- Folgende Projektmethode setzt die Elemente eines Array vom Typ Alphanumerisch oder Text in Kapitälchen (Anfangsbuchstabe groß):

```
   ` CAPITALIZE ARRAY
   ` CAPITALIZE ARRAY ( Pointer )
   ` CAPITALIZE 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
```

Schreiben Sie nun:

```
CAPITALIZE ARRAY(->atSubjects)
  ` ...
CAPITALIZE ARRAY(->asLastNames)
```

Die Kombination aus Arrays, Zeigern und Schleifen, wie z.B. **For...End for**, ergibt viele kleine nützliche Projektmethoden zum Verwalten von Arrays.

Element Null eines Array verwenden

Ein Array hat immer ein Element Null. Es erscheint zwar nicht in einem Array für ein Formularobjekt, es kann jedoch fast uneingeschränkt in der Programmiersprache eingesetzt werden.
Ausnahme: In einem Array vom Typ **Listbox** wird das Element Null intern zum Speichern des vorigen Wertes eines Elements in Bearbeitung verwendet. Sie können es also in diesem spezifischen Kontext nicht verwenden.

Ein Beispiel dafür haben Sie bereits im Abschnitt **Arrays und Formularobjekte** unter Combo Boxen gesehen. Hier folgt ein weiteres Beispiel:

Eine bestimmte Aktion soll nur ausgeführt werden, wenn ein anderes Element als das zuvor gewählte angeklickt wird. Dazu müssen Sie jedes ausgewählte Element verfolgen. Dazu könnten Sie eine Prozessvariable mit der Elementnummer des ausgewählten Elements einsetzen oder wie nachfolgend beschrieben, das Element Null:

```
  ` Objektmethode rollbarer Bereich atNames
Case of
   :(Form event=On Load)
  ` Initialisiere das Array (wie bereits oben gezeigt)
      ARRAY TEXT(atNames;5)
  ` ...
  ` Initialisiere das Element Null mit der Nummer des gerade ausgewählten
  ` Elements im Texformular
  ` Hier starten Sie mit keinem ausgewählten Element
      atNames{0}:="0"

   :(Form event=On Unload)
  ` Wir benötigen das Array nicht mehr
      CLEAR VARIABLE(atNames)

   :(Form event=On Clicked)
      If(atNames#0)
         If(atNames#Num(atNames{0}))
            vtInfo:="Sie haben geklickt auf: "+atNames{atNames}+" und es wurde vorher noch nicht ausgewählt."
            atNames{0}:=String(atNames)
         End if
      End if
   :(Form event=On Double Clicked)
      If(atNames#0)
         ALERT("Sie haben doppelgeklickt auf: "+atNames{atNames}
      End if
End case
```

Zweidimensionale Arrays

Jeder Befehl zum Erstellen von Arrays kann ein- oder zweidimensionale Arrays erstellen bzw. in der Größe anpassen. Beispiel:

```
ARRAY TEXT(atTopics;100;50)  ` Erstellt ein Array vom Typ Text mit 100 Reihen in 50 Spalten
```

Zweidimensionale Arrays sind Objekte der Programmiersprache; von daher lassen sie sich weder anzeigen noch ausdrucken.
Im oben angezeigten Beispiel gilt:

- *atTopics* ist ein zweidimensionales Array
- *atTopics{8}{5}* ist das 5. Element (5. Spalte...) der 8. Reihe
- *atTopics{20}* ist die 20. Reihe und selbst ein eindimenisonales Array
- *Size of array(atTopics)* gibt 100 zurück, das ist die Anzahl der Reihen
- *Size of array(atTopics{17})* gibt 50 zurück, das ist die Anzahl der Spalten für die 17. Reihe

Folgendes Beispiel speichert für jedes Datenfeld jeder Tabelle einen Zeiger in einem zweidimensionalen Array:

```
C_LONGINT($vlLastTable;$vlLastField)
C_LONGINT($vlFieldNumber)
  `Erstelle zu Beginn soviele leere Reihen wie Tabellen vorhanden sind
$vlLastTable:=Get last table number
ARRAY POINTER(<>apFields;$vlLastTable;0)  `2D Array mit X Zeilen und Null Spalten
  `Für jede Tabelle
For($vlTable;1;$vlLastTable)
   If(Is table number valid($vlTable))
      $vlLastField:=Get last field number($vlTable)
  ` Setze den Wert der Elemente
      $vlColumnNumber:=0
      For($vlField;1;$vlLastField)
         If(Is field number valid($vlTable;$vlField))
            $vlColumnNumber:=$vlColumnNumber+1
  `Füge eine Spalte in einer Zeile der Tabelle in Bearbeitung ein.
            INSERT IN ARRAY(<>apFields{$vlTable};$vlColumnNumber;1)
  `Weise die "Zelle" mit dem Zeiger zu
            <>apFields{$vlTable}{$vlColumnNumber}:=Field($vlTable;$vlField)
         End if
      End for
   End if
End for
```

Unter der Voraussetzung, dass dieses zweidimensionale Array initialisiert wurde, erhalten Sie nun die Zeiger auf die Datenfelder für eine bestimmte Tabelle:

```
  ` Erhalte Zeiger auf die Datenfelder für die gerade angezeigte Tabelle:
COPY ARRAY(◇apFields{Table(Current form table)};$apTheFieldsIamWorkingOn)
  ` Initialisiere Datenfelder vom Typ Boolean und Datum
For($vlElem;1;Size of array($apTheFieldsIamWorkingOn))
   Case of
      :(Type($apTheFieldsIamWorkingOn{$vlElem}->)=Is date)
         $apTheFieldsIamWorkingOn{$vlElem}->:=Current date
      :(Type($apTheFieldsIamWorkingOn{$vlElem}->)=Is Boolean)
         $apTheFieldsIamWorkingOn{$vlElem}->:=True
   End case
End for
```

Hinweis: Wie Sie in diesem Beispiel sehen, können Reihen in zweidimensionalen Arrays dieselbe oder verschiedene Größen haben.

Arrays und Speicher

Ein Array wird im Gegensatz zu Daten, die Sie in Tabellen und Datensätzen auf der Festplatte speichern, immer vollständig im Speicher gehalten.

Geben Sie zum Beispiel alle Postleitzahlen in einer Tabelle *[PLZ]* ein, enthält sie ca. 10.000 Datensätze. Die Tabelle enthält natürlich weitere Datenfelder, wie Landeskennzahl und Stadt. Wählen Sie nun das Postleitzahlengebiet 8, erstellt die 4D Datenbank-Engine die entsprechende Datensatzauswahl in der Tabelle *[PLZ]*, und lädt die Datensätze nur bei Bedarf, also z.B. zum Anzeigen auf dem Bildschirm oder zum Drucken. Mit anderen Worten, Sie arbeiten mit einer geordneten Reihe von Werten vom selben Typ, die die Engine teilweise von der Festplatte in den Speicher lädt.

Dieses Vorgehen ist für Arrays undenkbar. Das hat folgende Gründe:

- Zum Verwalten der drei Informationen Landeskennzahl, Postleitzahl und Stadt müssten Sie drei umfangreiche Arrays im Speicher halten.
- Da ein Array immer vollständig im Speicher gehalten wird, müssten Sie alle Informationen dieser Arrays während der ganzen Arbeitssitzung im Speicher halten, auch wenn Sie die Daten nicht ständig benötigen.
- Diese Arrays müssten bei jedem Starten bzw. Beenden der Datenbank komplett geladen und dann auf der Festplatte gesichert werden, selbst wenn die Daten während der ganzen Arbeitssitzung weder benutzt noch verändert wurden.

Fazit: In Arrays sollten überschaubare Datenmengen für eine kurze Zeitspanne gehalten werden. Da Arrays im Hauptspeicher gehalten werden, sind Array-Operationen sehr schnell. Sie können im Handumdrehen Array-Elemente kopieren, sortieren, suchen... .

Unter bestimmten Umständen müssen Sie jedoch Arrays mit hunderten oder tausenden von Elementen einsetzen. Nachfolgende Tabelle zeigt die Formel zum Berechnen der Speicherbelegung für jeden Array-Typ:

Array-Typ	**Formel für Speicherbelegung in Bytes**
Boolean	(31+Anzahl der Elemente)/8
Datum	(1+Anzahl der Elemente) * 6
String	(1+Anzahl der Elemente) * (Summe der Größe jedes Texts)
Ganzzahl	(1+Anzahl der Elemente) * 2
Lange Ganzzahl	(1+Anzahl der Elemente) * 4
Bild	(1+Anzahl der Elemente) * 4 + Summe der Größen jedes Bilds im Array
Zeiger	(1+Anzahl der Elemente) * 16
Zahl	(1+Anzahl der Elemente) * 8
Text	(1+Anzahl der Elemente) * (Summe der Größe jedes Texts)
Zweidimemsional	(1+Anzahl der Elemente) * 12 + Summe der Größe jedes Array

Hinweise:

- Die Größe eines Textes im Speicher wird mit der Formel ((Länge + 1)* 2) berechnet.
- Das ausgewählte Element, die Anzahl der Elemente und das Array selbst benötigen ein paar zusätzliche Bytes.

Beim Arbeiten mit umfangreichen Arrays sollten Sie prüfen, ob der Speicher ausreicht. Setzen Sie dazu die Array-Erstellung an den Anfang und fragen Sie Fehler mit einer Projektmethode **ON ERR CALL** ab. Beispiel:

```
  `Über Nacht soll eine Operation laufen, bei der umfangreiche Arrays erstellt werden
  `müssen. Anstatt das Auftreten von Fehlern mitten in der Nacht zu riskieren, lege das
  `Erstellen von Arrays an den Beginn der Operation und prüfe die Fehler zu diesem Zeitpunkt:
gError:=0  ` Nimm an, es gibt keinen Fehler
ON ERR CALL("ERROR HANDLING")  ` Installiere eine Methode zur Fehlersuche
ARRAY STRING(63;asThisArray;50000)  ` Ungefähr 3125K
ARRAY REAL(arThisAnotherArray;50000)  ` 488K
ON ERR CALL("")  ` Fehlersuche ist nicht mehr nötig
If(gError=0)
  ` Arrays können erstellt werden
  ` die Operation kann fortfahren
Else
   ALERT("Diese Operation benötigt mehr Speicher!")
End if
  ` Arrays werden nicht mehr benötigt
CLEAR VARIABLE(asThisArray)
CLEAR VARIABLE(arThisAnotherArray)
```

Die Projektmethode ***ERROR HANDLING*** ist folgende:

```
  ` Projektmethode ERROR HANDLING
gError:=Error  ` Gib den Fehlercode zurück
```

APPEND TO ARRAY

APPEND TO ARRAY (ArrayName ; Wert)

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Array zum Anfügen eines Elements
Wert	Ausdruck	➡	Anzufügender Wert

Beschreibung

Der Befehl **APPEND TO ARRAY** fügt am Ende von Array ein neues Element hinzu und weist ihm *Wert* zu. Existiert *Array* nicht im interpretierten Modus, erstellt der Befehl dieses nach dem in *Wert* angegebenen Typ.

Dieser Befehl arbeitet mit allen Arten von Arrays: String, Zahl, Boolean, Datum, Zeiger und Bild.
Der Typ von *Wert* muss zum Array-Typ passen, sonst erscheint der Syntaxfehler Nr. 54 (Die Argumente sind nicht kompatibel). Nachfolgende Werte werden jedoch akzeptiert:

- *Array* String (Text oder String) akzeptiert jeden Wert vom Typ Text oder String.
- *Array* Zahl (Ganzzahl, Lange Ganzzahl oder Zahl) akzeptiert jeden Wert vom Typ Ganzzahl, Lange Ganzzahl, Zahl oder Zeit.

Beispiel

Der folgende Code:

```
INSERT IN ARRAY($myarray;Size of array($myarray)+1)
$myarray{Size of array($myarray)}:=$myvalue
```

...lässt sich ersetzen durch:

```
APPEND TO ARRAY($myarray;$myvalue)
```

ARRAY BLOB

ARRAY BLOB (ArrayName ; Größe {; Größe2})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Name des Array
Größe	Lange Ganzzahl	➡	Anzahl der Array Elemente oder Größe Arrays, wenn Größe2 definiert ist
Größe2	Lange Ganzzahl	➡	Anzahl der 2D Array Elemente

Beschreibung

Der Befehl **ARRAY BLOB** erstellt bzw. passt ein Array mit Elementen vom Typ Blob im Speicher an.

Der Parameter *ArrayName* ist der Name des Array.

Der Parameter *Größe* ist die Anzahl der Array Elemente

Der Parameter *Größe2* ist optional. Ist er übergeben, erstellt dieser Befehl ein zwei-dimensionales Array. Dann gibt Größe die Anzahl Zeilen und *Größe2* die Anzahl Spalten in jedem Array an. Jede Zeile in einem zweidimensionalen Array lässt sich als Element und als Array bearbeiten. Das heißt, Sie können über andere Befehle aus diesem Kapitel komplette Arrays in ein zweidimensionales Array einfügen und entfernen, wenn Sie mit der ersten Dimension des Array arbeiten.

Wenden Sie den Befehl **ARRAY BLOB** auf ein vorhandenes Array an, passiert folgendes:

- Wenn Sie vergrößern, ändern sich vorhandene Elemente nicht und neue Elemente werden in ein leeres BLOB initialisiert (**BLOB size**= 0).
- Wenn Sie verkleinern, werden Elemente am unteren Ende des Array gelöscht und gehen verloren.

Beispiel 1

Dieses Beispiel erstellt ein Prozess Array mit 100 Elementen vom Typ BLOB:

```
ARRAY BLOB(arrBlob;100)
```

Beispiel 2

Dieses Beispiel erstellt ein lokales Array mit 100 Zeilen, die jeweils 50 Elemente vom Typ BLOB enthalten:

```
ARRAY BLOB($arrBlob;100;50)
```

Beispiel 3

Dieses Beispiel erstellt ein lokales Array mit 100 Zeilen, die jeweils 50 Elemente vom Typ BLOB enthalten. Die Variable *$vByteValue* empfängt das 10. Byte des BLOB, das in der 7. Spalte und der 5. Zeile des Array vom Typ BLOB liegt:

```
C_INTEGER($vByteValue)
ARRAY BLOB($arrValues;100;50)
...
$vByteValue:=$arrValues{5}{7}{9}
```

ARRAY BOOLEAN

ARRAY BOOLEAN (ArrayName ; Größe {; Größe2})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Name des Array
Größe	Lange Ganzzahl	➡	Anzahl der Elemente im Array oder Anzahl der Reihen, wenn Größe2 angegeben ist
Größe2	Lange Ganzzahl	➡	Anzahl der Spalten in zweidimensionalem Array

Beschreibung

Der Befehl **ARRAY BOOLEAN** erstellt und/oder passt ein Array mit Elementen vom Typ *Boolean* im Speicher an.

- Der Parameter *ArrayName* ist der Name des Array.
- Der Parameter *Größe* ist die Anzahl der Elemente im Array.
- Der Parameter *Größe2* ist optional; ist *Größe2* angegeben, wird ein zweidimensionales Array erstellt. In diesem Fall gibt *Größe* die Anzahl der Reihen und *Größe2* die Anzahl der Spalten in jedem Array an. In einem zweidimensionalen Array kann jede Reihe sowohl als Element als auch als Array behandelt werden. So können Sie, während Sie mit der ersten Dimension des Array arbeiten, mit anderen Befehlen ganze Arrays in einem zweidimensionalen Array einfügen oder löschen.

Wenden Sie **ARRAY BOOLEAN** auf ein bestehendes Array an, gilt folgendes:

- Erweitern Sie die Größe des Array, bleiben die vorhandenen Elemente unverändert, die neuen Elemente werden auf Falsch initialisiert.
- Verringern Sie die Größe des Array, gehen die letzten aus dem Array gelöschten Elemente verloren.

Beispiel 1

In einigen Fällen bietet es sich an, nicht Arrays vom Typ Boolean sondern vom Typ Ganzzahl zu verwenden, wobei jedes Element ungleich Null "wahr" und gleich Null "falsch" ist. Allerdings benötigt ein Array vom Typ Ganzzahl 16mal mehr Speicher als ein Array vom Typ *Boolean*.

Beispiel 2

Dieses Beispiel erstellt ein Prozess-Array mit 100 Elementen vom Typ *Boolean*:

```
ARRAY BOOLEAN(abValues;100)
```

Beispiel 3

Dieses Beispiel erstellt ein lokales Array mit 100 Reihen mit 50 Elementen vom Typ *Boolean*:

```
ARRAY BOOLEAN($abValues;100;50)
```

ARRAY DATE

ARRAY DATE (ArrayName ; Größe {; Größe2})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Name des Array
Größe	Lange Ganzzahl	➡	Anzahl der Elemente im Array oder Anzahl der Reihen, wenn Größe2 angegeben ist
Größe2	Lange Ganzzahl	➡	Anzahl der Spalten im zweidimensionalen Array

Beschreibung

Der Befehl **ARRAY DATE** erstellt und/oder passt ein Array mit Elementen vom Typ *Datum* im Speicher an.

- Der Parameter *ArrayName* ist der Name des Array.
- Der Parameter *Größe* ist die Anzahl der Elemente im Array.
- Der Parameter *Größe2* ist optional; ist *Größe2* angegeben, wird ein zweidimensionales Array erstellt. In diesem Fall gibt *Größe* die Anzahl der Reihen und *Größe2* die Anzahl der Spalten in jedem Array an. In einem zweidimensionalen Array kann jede Reihe sowohl als Element als auch als Array behandelt werden. So können Sie, während Sie mit der ersten Dimension des Array arbeiten, mit anderen Befehlen ganze Arrays in einem zweidimensionalen Array einfügen oder löschen.

Wenden Sie **ARRAY DATE** auf ein bestehendes Array an, gilt folgendes:

- Erweitern Sie die Größe des Array, bleiben die vorhandenen Elemente unverändert, die neuen Elemente werden auf das Datum Null (!00.00.00!) initialisiert.
- Verringern Sie die Größe des Array, gehen die letzten aus dem Array gelöschten Elemente verloren.

Beispiel 1

Dieses Beispiel erstellt ein Prozess-Array mit 100 Elementen vom Typ *Datum*:

```
ARRAY DATE(adValues;100)
```

Beispiel 2

Dieses Beispiel erstellt ein lokales Array mit 100 Reihen mit 50 Elementen vom Typ *Datum*:

```
ARRAY DATE($adValues;100;50)
```

Beispiel 3

Dieses Beispiel erstellt ein Interprozess-Array mit 50 Elementen vom Typ *Datum* und setzt jedes Element auf das aktuelle Datum plus der Anzahl Tage, die gleich der Elementnummer sind:

```
ARRAY DATE(◇adValues;50)
For($vlElem;1;50)
   ◇adValues{$vlElem}:=Current date+$vlElem
End for
```

ARRAY INTEGER

ARRAY INTEGER (ArrayName ; Größe {; Größe2})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Name des Array
Größe	Lange Ganzzahl	➡	Anzahl der Elemente im Array oder Anzahl der Reihen, wenn Größe2 angegeben ist
Größe2	Lange Ganzzahl	➡	Anzahl der Spalten in zweidimensionalem Array

Beschreibung

Der Befehl **ARRAY INTEGER** erstellt und/oder passt im Speicher ein Array mit 2-byte Elementen vom Typ *Ganzzahl* an.

- Der Parameter *ArrayName* ist der Name des Array.
- Der Parameter *Größe* ist die Anzahl der Elemente im Array.
- Der Parameter *Größe2* ist optional; ist *Größe2* angegeben, wird ein zweidimensionales Array erstellt. In diesem Fall gibt *Größe* die Anzahl der Reihen und *Größe2* die Anzahl der Spalten in jedem Array an. In einem zweidimensionalen Array kann jede Reihe sowohl als Element als auch als Array behandelt werden. Sie können also, während Sie mit der ersten Dimension des Array arbeiten, mit anderen Befehlen ganze Arrays in einem zweidimensionalen Array einfügen oder löschen.

Wenden Sie **ARRAY INTEGER** auf ein bestehendes Array an, gilt folgendes:

- Erweitern Sie die Größe des Array, bleiben die vorhandenen Elemente unverändert, die neuen Elemente werden auf Null (0) initialisiert.
- Verringern Sie die Größe des Array, gehen die letzten aus dem Array gelöschten Elemente verloren.

Beispiel 1

Dieses Beispiel erstellt ein Prozess-Array mit 100 2-byte Elementen vom Typ *Ganzzahl*:

```
ARRAY INTEGER(aiValues;100)
```

Beispiel 2

Dieses Beispiel erstellt ein lokales Array mit 100 Reihen mit 50 2-byte Elementen vom Typ *Ganzzahl*:

```
ARRAY INTEGER($aiValues;100;50)
```

Beispiel 3

Dieses Beispiel erstellt ein Interprozess-Array mit 50 2-byte Elementen vom Typ *Ganzzahl* und setzt jedes Element auf seine Elementnummer:

```
ARRAY INTEGER(◇aiValues;50)
For($vlElem;1;50)
   ◇aiValues{$vlElem}:=$vlElem
End for
```

ARRAY LONGINT

ARRAY LONGINT (ArrayName ; Größe {; Größe2})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Name des Array
Größe	Lange Ganzzahl	➡	Anzahl der Elemente im Array oder Anzahl der Reihen, wenn Größe2 angegeben ist
Größe2	Lange Ganzzahl	➡	Anzahl der Spalten in zweidimensionalem Array

Beschreibung

Der Befehl **ARRAY LONGINT** erstellt und/oder passt im Speicher ein Array mit 4-byte Elementen vom Typ *Lange Ganzzahl* an.

- Der Parameter *ArrayName* ist der Name des Array.
- Der Parameter *Größe* ist die Anzahl der Elemente im Array.
- Der Parameter *Größe2* ist optional; ist *Größe2* angegeben, wird ein zweidimensionales Array erstellt. In diesem Fall gibt *Größe* die Anzahl der Reihen und *Größe2* die Anzahl der Spalten in jedem Array an. In einem zweidimensionalen Array kann jede Reihe sowohl als Element als auch als Array behandelt werden. Sie können also, während Sie mit der ersten Dimension des Array arbeiten, mit anderen Befehlen ganze Arrays in einem zweidimensionalen Array einfügen oder löschen.

Wenden Sie **ARRAY LONGINT** auf ein bestehendes Array an, gilt folgendes:

- Erweitern Sie die Größe des Array, bleiben die vorhandenen Elemente unverändert, die neuen Elemente werden auf Null (0) initialisiert.
- Verringern Sie die Größe des Array, gehen die letzten aus dem Array gelöschten Elemente verloren.

Beispiel 1

Dieses Beispiel erstellt ein Prozess-Array mit 100 4-byte Elementen vom Typ *Lange Ganzzahl*:

```
ARRAY LONGINT(alValues;100)
```

Beispiel 2

Dieses Beispiel erstellt ein lokales Array mit 100 Reihen mit 50 4-byte Elementen vom Typ *Lange Ganzzahl*:

```
ARRAY LONGINT($alValues;100;50)
```

Beispiel 3

Dieses Beispiel erstellt ein Interprozess-Array mit 50 4-byte Elementen vom Typ *Lange Ganzzahl* und setzt jedes Element auf seine Elementnummer:

```
ARRAY LONGINT(◇alValues;50)
For($vlElem;1;50)
   ◇alValues{$vlElem}:=$vlElem
End for
```

ARRAY OBJECT

ARRAY OBJECT (ArrayName ; Größe {; Größe2})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Name des Array
Größe	Lange Ganzzahl	➡	Anzahl der Array Elemente oder Anzahl Arrays, wenn Größe2 definiert ist
Größe2	Lange Ganzzahl	➡	Anzahl der 2D Array Elemente

Beschreibung

Der Befehl **ARRAY OBJECT** erstellt bzw. passt ein Array mit Elementen vom Typ Programmiersprache Objekt im Speicher an.

Der Parameter *ArrayName* ist der Name des Array. Sie können jeden Namen verwenden, der die 4D Konventionen berücksichtigt.

Der Parameter *Größe* ist die Anzahl der Array Elemente.

Der Parameter *Größe2* ist optional. Ist er übergeben, erstellt dieser Befehl ein zwei-dimensionales Array. Dann gibt *Größe* die Anzahl Zeilen und *Größe2* die Anzahl Spalten in jedem Array an. Jede Zeile in einem zwei-dimensionalen Array lässt sich als Element und als Array bearbeiten. Das heißt, Sie können über andere Befehle aus diesem Kapitel komplette Arrays in ein zwei-dimensionales Array einfügen und entfernen, wenn Sie mit der ersten Dimension des Array arbeiten.

Wenden Sie den Befehl **ARRAY OBJECT** auf ein vorhandenes Array an, passiert folgendes:

- Wenn Sie es vergrößern, ändern sich vorhandene Elemente nicht und neue Elemente sind undefiniert. Über die Funktion **OB Is defined** können Sie testen, ob ein Element definiert ist.
- Wenn Sie es verkleinern, werden Elemente am unteren Ende des Array gelöscht und gehen verloren.

Beispiel 1

Ein Prozess Array mit 100 Elementen vom Typ Objekt erstellen:

```
ARRAY OBJECT(arrObjects;100)
```

Beispiel 2

Ein lokales Array mit 100 Zeilen mit jeweils 50 Elementen vom Typ Objekt erstellen:

```
ARRAY OBJECT($arrObjects;100;50)
```

Beispiel 3

Ein lokales Objekt Array erstellen und füllen:

```
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 (ArrayName ; Größe {; Größe2})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Name des Array
Größe	Lange Ganzzahl	➡	Anzahl der Elemente im Array, oder Anzahl der Reihen, wenn Größe2 angegeben ist
Größe2	Lange Ganzzahl	➡	Anzahl der Spalten in zweidimensionalem Array

Beschreibung

Der Befehl **ARRAY PICTURE** erstellt und/oder passt ein Array mit Elementen vom Typ *Bild* im Speicher an.

- Der Parameter *ArrayName* ist der Name des Array.
- Der Parameter *Größe* ist die Anzahl der Elemente im Array.
- Der Parameter *Größe2* ist optional; ist *Größe2* angegeben, wird ein zweidimensionales Array erstellt. In diesem Fall gibt *Größe* die Anzahl der Reihen und *Größe2* die Anzahl der Spalten in jedem Array an. In einem zweidimensionalen Array kann jede Reihe sowohl als Element als auch als Array behandelt werden. So können Sie, während Sie mit der ersten Dimension des Array arbeiten, mit anderen Befehlen ganze Arrays in einem zweidimensionalen Array einfügen oder löschen.

Wenden Sie **ARRAY PICTURE** auf ein bestehendes Array an, gilt folgendes:

- Erweitern Sie die Größe des Array, bleiben die vorhandenen Elemente unverändert, die neuen Elemente werden auf leere Bilder initialisiert. Das bedeutet, wird die Funktion **Picture size** auf eines dieser Elemente angewendet, gibt sie 0 zurück.
- Verringern Sie die Größe des Array, gehen die letzten aus dem Array gelöschten Elemente verloren.

Beispiel 1

Dieses Beispiel erstellt ein Prozess-Array mit 100 Elementen vom Typ *Bild*:

```
ARRAY PICTURE(agValues;100)
```

Beispiel 2

Dieses Beispiel erstellt ein lokales Array mit 100 Reihen mit 50 Elementen vom Typ *Bild*:

```
ARRAY PICTURE($agValues;100;50)
```

Beispiel 3

Dieses Beispiel erstellt ein Interprozess-Array mit Elementen vom Typ *Bild* und lädt jedes Bild in ein Element des Array. Die Größe ist gleich der Anzahl der für die Datenbank verfügbaren *'PICT'* Ressources. Der Ressourcenname des Array beginnt mit *"User Intf/"*:

```
RESOURCE LIST("PICT";$aiResIDs;$asResNames)
ARRAY PICTURE(◇agValues;Size of array($aiResIDs))
$vlPictElem:=0
For($vlElem;1;Size of array(◇agValues))
   If($asResNames{$vlElem}="User Intf/@")
      $vlPictElem:=$vlPictElem+1
      GET PICTURE RESOURCE("PICT";$aiResIDs{$vlElem};$vgPicture)
      ◇agValues{$vlPictElem}:=$vgPicture
   End if
End for
ARRAY PICTURE(◇agValues;$vlPictElem)
```

ARRAY POINTER

ARRAY POINTER (ArrayName ; Größe {; Größe2})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Name des Array
Größe	Lange Ganzzahl	➡	Anzahl der Elemente im Array, oder Anzahl der Reihen, wenn Größe2 angegeben ist
Größe2	Lange Ganzzahl	➡	Anzahl der Spalten in zweidimensionalem Array

Beschreibung

Der Befehl **ARRAY POINTER** erstellt und/oder passt im Speicher ein Array mit Elementen vom Typ *Zeiger* an.

- Der Parameter *ArrayName* ist der Name des Array.
- Der Parameter *Größe* ist die Anzahl der Elemente im Array.
- Der Parameter *Größe2* ist optional; ist *Größe2* angegeben, wird ein zweidimensionales Array erstellt. In diesem Fall gibt *Größe* die Anzahl der Reihen und *Größe2* die Anzahl der Spalten in jedem Array an. In einem zweidimensionalen Array kann jede Reihe sowohl als Element als auch als Array behandelt werden. So können Sie, während Sie mit der ersten Dimension des Array arbeiten, mit anderen Befehlen ganze Arrays in einem zweidimensionalen Array einfügen oder löschen.

Wenden Sie **ARRAY POINTER** auf ein bestehendes Array an, gilt folgendes:

- Erweitern Sie die Größe des Array, bleiben die vorhandenen Elemente unverändert, die neuen Elemente werden auf den Zeiger Null initialisiert, d.h. **Is nil pointer** gibt bei Anwendung auf eines dieser Elemente wahr zurück.
- Verringern Sie die Größe des Array, gehen die letzten aus dem Array gelöschten Elemente verloren.

Beispiel 1

Dieses Beispiel erstellt ein Prozess-Array mit 100 Elementen vom Typ *Zeiger*:

```
ARRAY POINTER(apValues;100)
```

Beispiel 2

Dieses Beispiel erstellt ein lokales Array mit 100 Reihen mit 50 Elementen vom Typ *Zeiger*:

```
ARRAY POINTER($apValues;100;50)
```

Beispiel 3

Dieses Beispiel erstellt ein Interprozess-Array mit Elementen vom Typ *Zeiger*. Jedes Element zeigt auf die Tabelle mit derselben Nummer wie das Element. Die Größe des Array entspricht der Anzahl Tabellen in der Datenbank. Bei einer gelöschten Tabelle gibt die Zeile **Is nil pointer** zurück:

```
ARRAY POINTER(◇apValues;Get last table number)
For($vlElem;1;Size of array(◇apValues);1;-1)
   If(Is table number valid($vlElem))
      ◇apValues{$vlElem}:=Table($vlElem)
   End if
End for
```

ARRAY REAL

ARRAY REAL (ArrayName ; Größe {; Größe2})

Parameter	Typ		Beschreibung
ArrayName	Array	⇒	Name des Array
Größe	Lange Ganzzahl	⇒	Anzahl der Elemente im Array oder Anzahl der Reihen, wenn Größe2 angegeben ist
Größe2	Lange Ganzzahl	⇒	Anzahl der Spalten in zweidimensionalem Array

Beschreibung

Der Befehl **ARRAY REAL** erstellt und/oder passt ein Array mit Elementen vom Typ *Zahl* im Speicher an.

- Der Parameter *ArrayName* ist der Name des Array.
- Der Parameter *Größe* ist die Anzahl der Elemente im Array.
- Der Parameter *Größe2* ist optional; ist *Größe2* angegeben, wird ein zweidimensionales Array erstellt. In diesem Fall gibt *Größe* die Anzahl der Reihen und *Größe2* die Anzahl der Spalten in jedem Array an. In einem zweidimensionalen Array kann jede Reihe sowohl als Element als auch als Array behandelt werden. Sie können also, während Sie mit der ersten Dimension des Array arbeiten, mit anderen Befehlen ganze Arrays in einem zweidimensionalen Array einfügen oder löschen.

Wenden Sie **ARRAY REAL** auf ein bestehendes Array an, gilt folgendes:

- Erweitern Sie die Größe des Array, bleiben die vorhandenen Elemente unverändert, die neuen Elemente werden auf Null (0) initialisiert.
- Verringern Sie die Größe des Array, gehen die letzten aus dem Array gelöschten Elemente verloren.

Beispiel 1

1. Dieses Beispiel erstellt ein Prozess-Array mit 100 Elementen vom Typ *Zahl*:

```
ARRAY REAL(arValues;100)
```

Beispiel 2

Dieses Beispiel erstellt ein lokales Array mit 100 Reihen mit 50 Elementen vom Typ *Zahl*:

```
ARRAY REAL($arValues;100;50)
```

Beispiel 3

Dieses Beispiel erstellt ein Interprozess-Array mit 50 Elementen vom Typ *Zahl* und setzt jedes Element auf seine Elementnummer:

```
ARRAY REAL(<>arValues;50)
For($vlElem;1;50)
   <>arValues{$vlElem}:=$vlElem
End for
```

ARRAY TEXT

ARRAY TEXT (ArrayName ; Größe {; Größe2})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Name des Array
Größe	Lange Ganzzahl	➡	Anzahl der Elemente im Array oder Anzahl der Spalten, wenn Größe2 angegeben
Größe2	Lange Ganzzahl	➡	Anzahl der Spalten in zweidimensionalem Array

Beschreibung

The Befehl **ARRAY TEXT** erstellt und/oder passt die Größe des Array mit Elementen vom Typ *Text* im Speicher an.

- Der Parameter *ArrayName* ist der Name des Array.
- Der Parameter *Größe* ist die Anzahl der Elemente im Array.
- Der Parameter *Größe2* ist optional; ist *Größe2* angegeben, wird ein zweidimensionales Array erstellt. In diesem Fall gibt *Größe* die Anzahl der Reihen und *Größe2* die Anzahl der Spalten in jedem Array an. In einem zweidimensionalen Array kann jede Reihe sowohl als Element als auch als Array behandelt werden. So können Sie, während Sie mit der ersten Dimension des Array arbeiten, mit anderen Befehlen ganze Arrays in einem zweidimensionalen Array einfügen oder löschen.

Wenden Sie **ARRAY TEXT** auf ein bestehendes Array an, gilt folgendes:

- Erweitern Sie die Größe des Array, bleiben die vorhandenen Elemente unverändert, die neuen Elemente werden auf "" (leere Zeichenkette) initialisiert.
- Verringern Sie die Größe des Array, gehen die letzten aus dem Array gelöschten Elemente verloren.

Beispiel 1

Dieses Beispiel erstellt ein Prozess-Array mit 100 Elementen vom Typ *Text*:

```
ARRAY TEXT(atValues;100)
```

Beispiel 2

Dieses Beispiel erstellt ein lokales Array mit 100 Reihen mit 50 Elementen vom Typ *Text*:

```
ARRAY TEXT($atValues;100;50)
```

Beispiel 3

Dieses Beispiel erstellt ein Interprozess-Array mit 50 Elementen vom Typ *Text* und setzt jedes Element auf den Wert "Element #" gefolgt von seiner Elementnummer:

```
ARRAY TEXT(◊atValues;50)
For($vlElem;1;50)
   ◊atValues{$vlElem}:="Element #"+String($vlElem)
End for
```

ARRAY TIME

ARRAY TIME (ArrayName ; Größe {; Größe2})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Name des Array
Größe	Lange Ganzzahl	➡	Anzahl der Array Elemente oder Anzahl Arrays, wenn Größe2 definiert ist
Größe2	Lange Ganzzahl	➡	Anzahl der 2D Array Elemente

Beschreibung

Der Befehl **ARRAY TIME** erstellt bzw. passt ein Array mit Elementen vom Typ Zeit im Speicher an.

Zur Erinnerung: In 4D v14 lassen sich Zeitangaben als numerische Werte bearbeiten. In früheren 4D Versionen mussten Sie, um ein Array mit Zeiten zu verwalten, ein Array Lange Ganzzahl mit einem Anzeigeformat kombinieren..

Der Parameter *ArrayName* ist der Name des Array.

Der Parameter *Größe* ist die Anzahl der Array Elemente

Der Parameter *Größe2* ist optional. Ist er übergeben, erstellt dieser Befehl ein zweidimensionales Array. Dann gibt Größe die Anzahl Zeilen und *Größe2* die Anzahl Spalten in jedem Array an. Jede Zeile in einem zweidimensionalen Array lässt sich als Element und als Array bearbeiten. Das heißt, Sie können über andere Befehle aus diesem Kapitel komplette Arrays in ein zweidimensionales Array einfügen und entfernen, wenn Sie mit der ersten Dimension des Array arbeiten.

Wenden Sie den Befehl **ARRAY TIME** auf ein vorhandenes Array an, passiert folgendes:

- Wenn Sie es vergrößern, ändern sich vorhandene Elemente nicht und neue Elemente werden auf Nullwerte vom Typ Zeit initialisiert (00:00:00).
- Wenn Sie es verkleinern, werden Elemente am unteren Ende des Array gelöscht und gehen verloren.

Wenden Sie **SELECTION TO ARRAY** oder **SELECTION RANGE TO ARRAY** auf ein Feld vom Typ Zeit an, müssen Sie beachten, dass diese Befehle nur ein Array vom Typ Zeit anlegen, wenn für das Array noch kein anderer Typ definiert wurde, wie z.B. Lange Ganzzahl.

Beispiel 1

Dieses Beispiel erstellt ein Prozess Array mit 100 Elementen vom Typ Zeit:

```
ARRAY TIME(arrTimes;100)
```

Beispiel 2

Dieses Beispiel erstellt ein lokales Array mit 100 Zeilen mit jeweils 50 Elementen vom Typ Zeit:

```
ARRAY TIME($arrTimes;100;50)
```

Beispiel 3

Da Zeit Arrays numerische Werte akzeptieren, ist folgender Code gültig:

```
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 (ArrayName ; Liste {; EintragRefs})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Array, aus dem Array-Elemente zu kopieren sind
Liste	String, ListRef	➡	Name oder Referenz der Liste, in die Array-Elemente zu kopieren sind
EintragRefs	Array	➡	Numerisches Array mit den Referenznummern der Items

Beschreibung

Der Befehl **ARRAY TO LIST** erstellt oder ersetzt *Liste* (wie im Listeneditor der Designumgebung definiert) mit den Elementen von *Array*.

Im Parameter *Liste* können Sie eine Auswahlliste (String) oder Referenz auf eine hierarchische Liste (*ListRef*) übergeben. Bei *ListRef* funktioniert der Befehl nur, wenn die Liste bereits erstellt ist, z.B. über die Funktion **New list**.

Verwenden Sie den optionalen Parameter *itemRefs*, muss er ein numerisches Array sein, das mit *Array* synchronisiert ist. Jedes Element gibt dann die Referenznummer der Einträge aus der Liste für das entsprechende Element in *Array* an. Verwenden Sie ihn nicht, setzt 4D automatisch die Referenznummer der Einträge aus der Liste auf 1, 2... N.

Hinweis zur Kompatibilität: Verwenden Sie **ARRAY TO LIST** mit Bedacht, da es folgende Beschränkungen gibt:

- Da dieser Befehl die Anwendungsstruktur verändert (Listen werden in der Strukturdatei gespeichert), gehen alle lokal ausgeführten Änderungen verloren, wenn die Strukturdatei während der Produktion aktualisiert wird.
- Dieser Befehl lässt sich in einer Komponente nicht verwenden, da sie mit ihrer Struktur im Nur-Lesen Modus geladen wird.
- Der Befehl kann nur die Einträge der 1. Ebene in der Liste definieren.

Mit **ARRAY TO LIST** können Sie weiterhin eine Liste mit den Elementen eines Array erstellen. Für den uneingeschränkten Einsatz von Listen mit Werten empfehlen wir, die Befehle im Kapitel **Hierarchische Listen** zu verwenden.

Beispiel

Folgendes Beispiel kopiert das Array *atRegions* in die Liste mit Namen "Regions:"

```
ARRAY TO LIST(atRegions;"Regions")
```

Beispiel

Nicht-wiederholte Werte eines Feldes in eine Liste setzen, um z.B. ein hierarchisches PopUp Menü zu erstellen. Sie schreiben wie folgt:

```
ALL RECORDS([Company])
DISTINCT VALUES([Company]country;$arrCountries)
CountryList:=New list
ARRAY TO LIST($arrCountries;CountryList)
```

Fehlerverwaltung

Der Fehler *-9957* tritt auf, wenn **ARRAY TO LIST** auf eine Liste angewendet wird, die gerade in der Designumgebung im Listeneditor angelegt wird. Sie können diesen Fehler mit einer Projektmethode **ON ERR CALL** abfangen.

ARRAY TO SELECTION

ARRAY TO SELECTION {(ArrayName ; Feldname {; ArrayName2 ; Feldname2 ; ... ; ArrayNameN ; FeldnameN}{; *})}

Parameter	Typ		Beschreibung
ArrayName	Array	⇒	Array, das in die Auswahl kopiert werden soll
Feldname	Feld	⇐	Feld, das die Array-Werte erhält
*	Operator	⇒	Auf Ausführung warten

Beschreibung

Der Befehl **ARRAY TO SELECTION** kopiert eine oder mehrere Arrays in eine Auswahl von Datensätzen. Alle angezeigten Datenfelder müssen zur gleichen Tabelle gehören.

Besteht zur Zeit des Aufrufs eine Auswahl, werden die Array-Elemente in der Reihenfolge des Array und der Datensätze übertragen. Gibt es mehr Elemente als Datensätze, werden neue Datensätze angelegt. Sowohl die neuen als auch die bestehenden Datensätze werden automatisch gesichert. Alle Arrays müssen dieselbe Anzahl Elemente enthalten. Sind die Arrays von unterschiedlicher Größe, wird ein Syntax Fehler generiert.

Hinweis: Da dieser Befehl neue Datensätze erstellen kann, berücksichtigt er nicht den evtl. vorhandenen Status Nur-Lesen der Tabelle. Weitere Informationen dazu finden Sie im Abschnitt **Überblick zu Datensatz sperren**.

Dieser Befehl führt das Umgekehrte von **SELECTION TO ARRAY** aus. **ARRAY TO SELECTION** akzeptiert jedoch nicht Datenfelder aus verschiedenen Tabellen. Das gilt auch für verknüpfte Tabellen, selbst wenn sie automatisch sind.

Übergeben Sie den Parameter *, führt 4D die entsprechende Zeile der Anweisung nicht sofort aus, sondern legt sie stattdessen in den Speicher; auf diese Weise können Sie Zeilen, die mit * enden, stapeln. All diese auf Ausführung wartenden Zeilen werden durch eine abschließende Anweisung **ARRAY TO SELECTION** ohne den Parameter * ausgeführt. Aus diesem Grund lässt sich der Befehl jetzt ohne weitere Parameter aufrufen.
Analog zum Befehl **QUERY** können Sie so eine komplexe Anweisung in eine Reihe von Zeilen aufteilen, die leichter zu lesen und zu pflegen ist. Sie können auch Anweisungen dazwischen einfügen.

WARNUNG: Verwenden Sie **ARRAY TO SELECTION** mit Vorsicht, da er die Information in vorhandenen Datensätzen überschreibt. Ist beim Aufruf von **ARRAY TO SELECTION** ein Datensatz durch einen anderen Prozess gesperrt, wird er nicht geändert. Gesperrte Datensätze werden in die Prozessmenge mit Namen *LockedSet* gelegt. Diese Menge kann nach dem Ausführen des Befehls überprüft werden.

Hinweis: Dieser Befehl berücksichtigt nicht den Status Nur-Lesen bzw. Lesen-Schreiben der Tabelle zum jeweiligen Feld.

4D Server: Der Befehl wurde für 4D Server optimiert. Arrays werden vom Client-Rechner zum Server gesendet. Die Datensätze werden auf dem Server-Rechner geändert oder erstellt. Da diese Anfrage synchron bearbeitet wird, muss der Client-Rechner warten, bis die Operation erfolgreich abgeschlossen ist. Gesperrte Datensätze werden in der Multi-User bzw. Multi-Prozessumgebung nicht überschrieben.

Beispiel 1

Im folgenden Beispiel legen die beiden Arrays *asLastNames* und *asCompanies* Daten in die Tabelle *[People]*. Werte aus dem Array *asLastNames* werden in das Datenfeld *[People]Last Name* gelegt, Werte aus dem Array *asCompanies* in das Datenfeld *[People]Company*:

```
ARRAY TO SELECTION(asLastNames;[People]Last Name;asCompanies;[People]Company)
```

Beispiel 2

Eine Datensatzauswahl mit den durch den Wert Option festgelegten Feldern kopieren:

```
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)  //Das CV Feld laden
   SELECTION TO ARRAY([Candidate]cv;$_cv;*)
End if
If(withPhoto)  //Das Feld Photo laden
   SELECTION TO ARRAY([Candidate]photo;$_photo;*)
End if
SELECTION TO ARRAY  //Kopie ausführen

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 (BooleanArr {; Mengenname})

Parameter	Typ		Beschreibung
BooleanArr	Array Boolean	⇐	Array, das angibt, ob ein Datensatz in Menge ist oder nicht
Mengenname	String	⇒	Name der Menge, ohne Angabe UserSet

Beschreibung

Der Befehl **BOOLEAN ARRAY FROM SET** füllt ein Array vom Typ Boolean und gibt für jeden Datensatz der Tabelle an, ob er in *Mengenname* ist oder nicht. Die Elemente im Array sind so sortiert wie die Datensätze in der Tabelle erstellt wurden (absolute Anzahl Datensätze).
Ist N die Anzahl der Datensätze in der Tabelle, entspricht Element 0 des Array dem Datensatz mit der Nummer 0, Element 1 des Array dem Datensatz mit der Nummer 1, etc.

Jedes Element des Array ist:

- *True* wenn der dazugehörige Datensatz zur Menge gehört.
- *False* wenn der dazugehörige Datensatz nicht zur Menge gehört.

Warnung: Die Gesamtanzahl der Elemente im Array *BooleanArr* ist nicht relevant. Sie kann sich strukturbedingt von der aktuell in der Tabelle vorhandenen Anzahl der Datensätze unterscheiden. Evtl. zusätzliche Elemente werden auf **False** gesetzt.

Übergeben Sie den Parameter *Mengenname* nicht, verwendet der Befehl *UserSet* im aktuellen Prozess.

COPY ARRAY

COPY ARRAY (Quelle ; Ziel)

Parameter	Typ		Beschreibung
Quelle	Array	⇒	Zu kopierendes Array
Ziel	Array	⇐	Array, in das kopiert werden soll

Beschreibung

Der Befehl **COPY ARRAY** erstellt oder überschreibt das Array *Ziel* mit dem exakten Inhalt, Typ und der Größe des Array *Quelle*. Mit diesem Befehl können Sie also nicht einzelne Array-Elemente eines anderen Array anfügen.

Quell- und Ziel-Array können ein lokales, Prozess- oder Interprozess-Array sein. Der Geltungsbereich spielt beim Kopieren keine Rolle.

Hinweis:

- Im kompilierten Modus muss das Array *Ziel* vom gleichen Typ wie das Array *Quelle* sein.
- Beim Kopieren von Objekt Arrays werden nur die Referenzen auf die darin enthaltenen Objekte dupliziert und nicht die Objekte selbst, d.h. Änderungen in einem Objekt in einem Array werden auf alle vorhandenen Instanzen des Objekts in kopierten Arrays angewandt.
  Zum Duplizieren von Objekten müssen Sie die Funktion **OB Copy** verwenden.

Beispiel

Folgendes Beispiel füllt das Array mit Namen C und erstellt dann ein neues Array mit Namen D. Es hat dieselbe Größe und denselben Inhalt wie C:

```
ALL RECORDS([People]) ` Wähle alle Datensätze in People
SELECTION TO ARRAY([People]Company;C)
  ` Lege Daten von Datenfeld Company in Array C
COPY ARRAY(C;D) ` Kopiere Array C in Array D
```

Count in array

Count in array (ArrayName ; Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ArrayName	Array	→	Array für das Zählen
Wert	Ausdruck	→	Zu zählender Wert
Funktionsergebnis	Lange Ganzzahl	↩	Anzahl der gefundenen Einträge

Beschreibung

Die Funktion **Count in array** gibt an, wie oft *Wert* in *Array* gefunden wird.

Sie lässt sich mit folgenden Array-Typen verwenden: Text, Alpha, Numerisch, Datum, Zeiger und Boolean. Die Parameter *Array* und *Wert* müssen vom gleichen Typ oder mindestens miteinander kompatibel sein.
Passt kein Element in *Array* zu *Wert*, gibt die Funktion 0 (Null) zurück.

Beispiel

Mit nachfolgendem Code können Sie die Anzahl der gewählten Zeilen in einer Listbox anzeigen:

```
`tBList ist der Name mit den Spalten der Listbox
ALERT(String(Count in array(tBList;True))+" In Listbox gewählte Zeile(n)")
```

DELETE FROM ARRAY

DELETE FROM ARRAY (ArrayName ; Beginn {; AnzElemente})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Array, in dem Elemente zu löschen sind
Beginn	Lange Ganzzahl	➡	Position, ab der Elemente gelöscht werden sollen
AnzElemente	Lange Ganzzahl	➡	Anzahl der zu löschenden Elemente, bzw. 1 Element, falls nichts angegeben ist

Beschreibung

Der Befehl **DELETE FROM ARRAY** löscht die Elemente von *Array* ab *Beginn*.

AnzElemente ist optional. Wird der Parameter nicht angegeben, wird nur ein einziges Element gelöscht, sonst die angegebene Anzahl der Elemente. Die Größe der Tabelle wird automatisch aktualisiert.

Beispiel 1

Folgendes Beispiel löscht ab Element 5 drei Elemente:

```
DELETE FROM ARRAY(anArray;5;3)
```

Beispiel 2

Folgendes Beispiel löscht das letzte Element eines Array, sofern es vorhanden ist:

```
$vlElem:=Size of array(anArray)
If($vlElem>0)
   DELETE FROM ARRAY(anArray;$vlElem)
End if
```

DISTINCT ATTRIBUTE PATHS

DISTINCT ATTRIBUTE PATHS (ObjektFeld ; PfadArray)

Parameter	Typ		Beschreibung
ObjektFeld	Feld	➡	Indiziertes Objektfeld
PfadArray	Array Text	⬅	Array für die Liste der verschiedenen Pfade

Beschreibung

Der Befehl **DISTINCT ATTRIBUTE PATHS** gibt die Liste der verschiedenen Pfade zurück, die im indizierten Objektfeld, angegeben im Parameter *ObjektFeld* für die aktuelle Auswahl der dazugehörigen Tabelle gefunden werden.

Bitte beachten Sie, dass *ObjektFeld* indiziert und vom Typ Objekt sein muss, andernfalls wird ein Fehler zurückgegeben.

Nach dem Aufruf ist *PfadArray* genauso groß wie die Anzahl der verschiedenen Pfade in der Auswahl. Pfade zu eingebundenen Objektattributen werden mit der Standard Notation mit Punkt zurückgegeben, z.B. "company.address.number". Beachten Sie, dass Namen von Objektattributen zwischen Groß- und Kleinschreibung unterscheiden. Der Befehl verändert nicht die aktuelle Auswahl bzw. den aktuellen Datensatz.

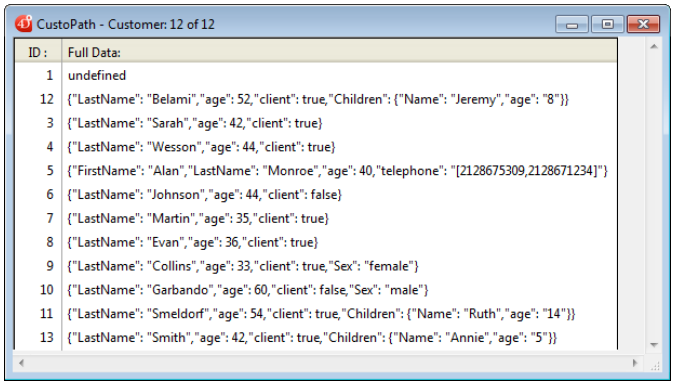
In *PfadArray* wird die Liste der verschiedenen Pfade in alphabetischer (diakritischer) Reihenfolge zurückgegeben.

Hinweise:

- Datensätze mit einem undefinierten Wert in *ObjektFeld* werden nicht berücksichtigt.
- Während einer Transaktion erstellte Attributpfade werden vom Befehl berücksichtigt. Beachten Sie, dass diese Pfade im Index des Objektfeldes beibehalten werden, selbst wenn die Transaktion abgebrochen wurde.

Beispiel

Ihre Datenbank enthält ein Objektfeld [Customer]full_Data (indexed) mit 15 Datensätzen:



Führen Sie diesen Code aus:

```
ARRAY TEXT(aTPaths;0)
ALL RECORDS([Customer])
DISTINCT ATTRIBUTE PATHS([Customer]full_Data;aTPaths)
```

Das Array *aTPaths* erhält folgende Elemente:

Element	Wert
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"

DISTINCT ATTRIBUTE VALUES

DISTINCT ATTRIBUTE VALUES (ObjektFeld ; Pfad ; WerteArray)

Parameter	Typ		Beschreibung
ObjektFeld	Feld	➡	Objektfeld, aus dem Sie die Liste der verschiedenen Attributwerte erhalten wollen
Pfad	Text	➡	Pfad des Attributs, dessen verschiedene Werte Sie erhalten wollen
WerteArray	Array Text, Array Lange Ganzzahl, Array Boolean, Array Datum, Time array	⬅	Verschiedene Werte im Attributpfad

Beschreibung

Der Befehl **DISTINCT ATTRIBUTE VALUES** erstellt und füllt den Parameter *WerteArray* mit nicht-wiederholten (einmaligen) Werten aus dem Attribut *Pfad* im Feld *ObjektFeld* für die aktuelle Auswahl der dazugehörigen Tabelle. Bitte beachten Sie, dass *ObjektFeld* indiziert und vom Typ Objekt sein muss, andernfalls wird ein Fehler zurückgegeben. Der Befehl ist mit indizierten und nicht-indizierten Feldern verwendbar.

In *Pfad* übergeben Sie einen gültigen Attributpfad. Definieren Sie Pfade zu eingebundenen Objektattributen mit der Standard Notation mit Punkt, z.B. "company.address.number". Beachten Sie, dass Namen von Objektattributen zwischen Groß- und Kleinschreibung unterscheiden.

Das in *WerteArray* übergebene Array muss vom gleichen Typ sein wie das als Parameter übergebene Attribut *Pfad*. Die Werte müssen skalar sein und können vom Typ Zahl, Text, Datum und Zeit sein (Zeiger, Objekte, Blobs oder Bilder werden nicht unterstützt). Stellen Sie sicher, dass alle Werte für Feldattribute vom gleichen Typ sind; sonst wird ein Fehler zurückgegeben. Beispiel: Enthält das Attribut *Pfad* in einem Datensatz den Text Montag und in einem anderen die Zahl 10125, wird ein Fehler zurückgegeben.

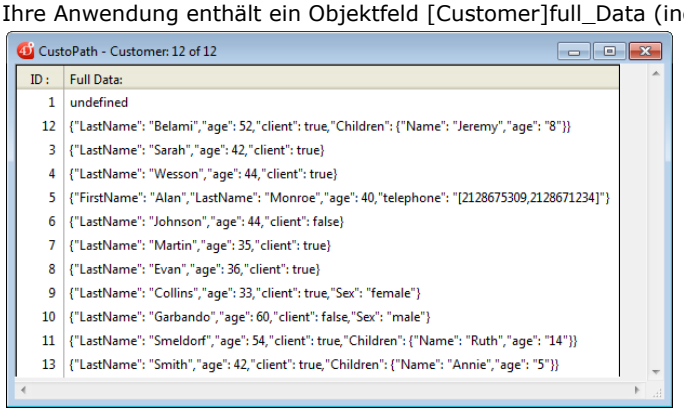
Nach dem Aufruf ist das Array genauso groß wie die Anzahl der verschiedenen Werte in der Auswahl. Der Befehl verändert nicht die aktuelle Auswahl bzw. den aktuellen Datensatz.

Die virtuelle Eigenschaft .length verwenden

Mit diesem Befehl können Sie die virtuelle Eigenschaft "length" verwenden. Sie ist für alle Attribute vom Typ Array automatisch verfügbar und liefert die Größe des Array, z.B. die Anzahl der darin enthaltenen Elemente. Diese Eigenschaft wurde für Suchläufe eingerichtet (siehe **QUERY BY ATTRIBUTE**). Sie können sie auch mit dem Befehl **DISTINCT ATTRIBUTE VALUES** verwenden, um die verschiedenen Array Größen für ein Attribut zu erhalten

Beispiel

Ihre Anwendung enthält ein Objektfeld [Customer]full_Data (indexed) mit 12 Datensätzen:



Führen Sie folgende Anweisung aus:

```
ARRAY LONGINT(aLAges;0)
ALL RECORDS([Customer])
 //erhält die verschiedenen Werte für das Attribut "age"
DISTINCT ATTRIBUTE VALUES([Customer]full_Data;"age";aLAges)
```

erhält das Array *aLAges* folgende 9 Elemente:

```
//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 (Feldname ; ArrayName {; AnzahlArray})

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Feld oder Unterfeld für Daten
ArrayName	Array	⬅	Array, das indizierte Daten aus Feldern erhält
AnzahlArray	Array Lange Ganzzahl, Array Zahl	⬅	Array zum Empfangen der Anzahl jedes Werts

Beschreibung

Der Befehl **DISTINCT VALUES** erstellt und füllt *Array* mit nicht-wiederholten (einmaligen) Werten aus dem Datenfeld *Feldname* für die aktuelle Auswahl der Tabelle, zu der das Datenfeld gehört.

Sie können in diesem Befehl jedes **indizierbare** Feld übergeben, d.h. dessen Typ die Indizierung unterstützt, auch wenn es derzeit nicht indiziert ist.
Bei Ausführung auf nicht-indizierte Felder ist der Befehl jedoch langsamer. Beachten Sie, dass er in diesem Fall auch den aktuellen Datensatz verliert.

DISTINCT VALUES durchläuft und entnimmt nicht-wiederholte Werte nur aus der aktuellen Auswahl der Datensätze.

Hinweis: DISTINCT VALUES berücksichtigt beim Aufrufen während einer noch nicht abgeschlossenen Transaktion die Datensätze, die in dieser Transaktion erstellt wurden.

Erstellen Sie ein Array vor dem Aufrufen, erwartet **DISTINCT VALUES** einen Array-Typ, der mit dem übergebenen Datenfeld kompatibel ist. Ansonsten erstellt **DISTINCT VALUES** im interpretierten Modus ein Array von eigenem Typ. Ist das Datenfeld vom Typ Zeit, erwartet oder erstellt der Befehl ein Array vom Typ Lange Ganzzahl.

Das von **DISTINCT VALUES** erwartete Array muss vom gleichen Typ sein wie das Feld, das als erster Parameter übergeben ist. Andernfalls wird das Array neu typisiert. Es gibt eine Ausnahme-I für diese Regelung: Ist das Datenfeld vom Typ Bild (und mit einem Volltext-Index verknüpft), muss das dazugehörige Array vom Typ Text sein.

Nach dem Aufruf ist die Größe des Array gleich der Anzahl der in der Auswahl gefundenen eindeutigen Werte. Der Befehl ändert weder die aktuelle Auswahl noch den aktuellen Datensatz. Da **DISTINCT VALUES** mit dem Index des Datenfelds arbeitet, werden die Elemente in *Array* in aufsteigender Reihenfolge sortiert übertragen. Bei anderer Reihenfolge müssen Sie nach dem Befehl **SORT ARRAY** aufrufen.

Hinweis: Wird **DISTINCT VALUES** mit einem Text- oder Bildfeld mit zugewiesenem Volltext-Index ausgeführt, füllt der Befehl das Array mit den Schlüsselwörtern des Index. Deshalb variieren die zurückgegebenen Werte im Gegensatz zu anderen Datentypen je nach Vorhandensein des Index. Der Volltext-Index wird immer berücksichtigt, selbst wenn dem Feld auch ein Standard-Index zugewiesen ist. Ist dem Datenfeld vom Typ Text oder Bild kein Volltext-Index zugeordnet, wird das Array leer zurückgegeben.

Der Befehl akzeptiert ein Array *AnzahlArray* als optionalen Parameter. Sie können entweder ein Array vom Typ Lange Ganzzahl oder Zahl übergeben. *AnzahlArray* gibt für jeden nicht-wiederholten Wert in *Feld* die Anzahl der gefundenen Vorkommen in der aktuellen Auswahl zurück. Sie wird automatisch an die Anzahl der Elemente in *Array* angepasst. Beispiel: Für eine Auswahl, die drei Datensätze enthält mit den Feldwerten "A", "B" und "A", enthält *Array* {A;B} und *AnzahlArray* {2;1}.

Hinweis: Der Parameter *AnzahlArray* wird nicht von Feldern vom Typ Text oder Bild mit zugewiesenen Volltext-Indizes unterstützt. In diesem Kontext wird er leer zurückgegeben.

WARNUNG: DISTINCT VALUES erstellt u.U. umfangreiche Arrays, je nach Größe der Auswahl und Anzahl der verschiedenen Werte in den Datensätzen. Da Arrays im Speicher bleiben, empfehlen wir, das Ergebnis nach Ende des Befehls zu testen. Prüfen Sie die Größe jedes resultierenden Array oder sichern Sie den Aufruf dieses Befehls mit einer Fehlerverwaltungsmethode **ON ERR CALL**.

4D Server: DISTINCT VALUES wurde für 4D Server optimiert. Das Array wird auf dem Server erstellt und berechnet; es wird dann komplett auf den Client-Rechner übertragen.

Hinweis: Dieser Befehl unterstützt keine Felder vom Typ Objekt.

Beispiel 1

Folgendes Beispiel erstellt eine Städteliste aus der aktuellen Auswahl und teilt dem Benutzer die Anzahl der Städte mit, in denen die Firma Läden unterhält:

```
ALL RECORDS([Retail Outlets])  ` Erstelle eine Datensatzauswahl
DISTINCT VALUES([Retail Outlets]City;asCities)
ALERT("Die Firma hat Läden in "+String(Size of array(asCities))+" cities.")
```

Beispiel 2

Sie wollen für das Feld "Pictures" eine komplette Liste der Schlüsselwörter im Volltext-Index erhalten:

```
ALL RECORDS([PICTURES])
ARRAY TEXT(<>_MyKeywords;10)
DISTINCT VALUES([PICTURES]Photos;<>_MyKeywords)
```

Beispiel 3

Sie wollen für statistische Berechnungen die Anzahl der unterschiedlichen Werte in einem Feld in absteigender Reihenfolge sortieren:

```
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 (ArrayName ; Wert {; Start}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Array, in dem gesucht werden soll
Wert	Ausdruck	➡	Wert, der gesucht werden soll
Start	Lange Ganzzahl	➡	Element, bei dem die Suche beginnt
Funktionsergebnis	Lange Ganzzahl	⮌	Nummer des ersten gefundenen Elements im Array

Beschreibung

Die Funktion **Find in array** gibt die Nummer des ersten gefundenen Elements in *ArrayName* zurück, das zu *Wert* passt.

Find in array funktioniert mit Arrays vom Typ Text, Alphanumerisch, Numerisch, Datum, Zeiger und Boolean. Die Parameter *ArrayName* und *Wert* müssen vom selben Typ sein.

Wert muss genau mit dem gesuchten Element übereinstimmen. Es gelten dieselben Regeln wie für den Operator *Ist gleich* (siehe **Vergleichsoperatoren**). Wurde kein passendes Element gefunden, gibt **Find in array** den Wert - 1 zurück.

Ist *Start* angegeben, startet die Funktion die Suche mit der hier angegebenen Nummer des Elements. Ist *Start* nicht angegeben, wird ab dem ersten Element gesucht.

Beispiel 1

Folgende Projektmethode löscht alle leeren Elemente aus einem Array vom Typ Alphanumerisch oder Text, dessen Zeiger als Parameter übergeben wurde:

```
  ` Projektmethode CLEAN UP ARRAY
  ` CLEAN UP ARRAY ( Pointer )
  ` CLEAN UP ARRAY ( -> Text oder String Array )

C_POINTER($1)
Repeat
   $vlElem:=Find in array($1->;"")
   If($vlElem>0)
      DELETE FROM ARRAY($1->;$vlElem)
   End if
Until($vlElem<0)
```

Anschließend können Sie schreiben:

```
ARRAY TEXT(atSomeValues;...)
  ` ...
  ` Führe dies und das mit dem Array aus
  ` ...
  ` Lösche leere alphanumerische Elemente
CLEAN UP ARRAY(->atSomeValues)
```

Beispiel 2

Folgende Projektmethode wählt das erste Element eines Array aus, dessen Zeiger als erster Parameter übergeben wurde. Dies ist der Wert der Variablen bzw. des Datenfelds, dessen Zeiger als Parameter übergeben wurde:

```
  ` Projektmethode SELECT ELEMENT
  ` SELECT ELEMENT ( Pointer ; Pointer)
  ` SELECT ELEMENT ( -> Text oder String Array ; -> Text oder String Variable oder Feld )

$1->:=Find in array($1->;$2->)
If($1->=-1)
   $1->:=0  ` Wurde kein Element gefunden, setze Array auf kein ausgewähltes Element
End if
```

Anschließend können Sie schreiben:

```
  ` Objektmethode PopUp-Menü asGender
Case of
   :(Form event=On Load)
      SELECT ELEMENT(->asGender;->[People]Gender)

End case
```

Hinweis: Dieses Beispiel verwendet das **ausgewählte Element** des Array. Beachten Sie, dass es ohne Bedeutung ist, wenn das Array mehr als 32.767 Elemente enthält (siehe **Arrays und Formularobjekte**). In diesem Fall müssen Sie eine Variable Lange Ganzzahl verwenden, um das Ergebnis von **Find in array** zu speichern.

Find in sorted array

Find in sorted array (Array ; Wert ; > oder < {; ErstePos {; LetztePos}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Array	Array	➡	Array zum Suchen
Wert	Ausdruck	➡	Wert (gleicher Typ wie Array) zum Suchen im Array
> oder <	Operator	➡	> wenn Array in aufsteigender Reihenfolge sortiert ist, < wenn Array in absteigender Reihefolge sortiert ist
ErstePos	Lange Ganzzahl	⬅	Position des ersten Vorkommens, wenn der Wert gefunden wird; sonst Position, an der der Wert eingefügt werden soll.
LetztePos	Lange Ganzzahl	⬅	Postion des letzten Vorkommens, wenn der Wert gefunden wird; sonst gleicher Wert wie ErstePos
Funktionsergebnis	Boolean	➲	Wahr, wenn mindestens ein Element im Array zum Wert passt, sonst Falsch

Beschreibung

Die Funktion **Find in sorted array** gibt **wahr** zurück, wenn mindestens ein Element im sortierten *Array* zum Parameter *Wert* passt. Sie gibt optional auch die Position des Elements zurück. Im Gegensatz zu **Find in array** funktioniert **Find in sorted array** nur mit einem sortierten *Array* und gibt außerdem die Position der Vorkommen an. So lassen sich auch Elemente einfügen, wenn keine Vorkommen gefunden werden.

Das *Array* muss bereits sortiert und in derselben Reihenfolge sein, wie im Parameter *> oder <* angegeben wurde, d.h. "größer als" für aufsteigende und "kleiner als" für absteigende Reihenfolge. Andernfalls ist das Suchergebnis u.U. nicht korrekt. **Find in sorted array** nutzt die Sortierung und einen binären Suchalgorithmus, was bei umfangreichen vorsortierten Arrays in der Regel effektiver ist. Weitere Informationen dazu finden Sie unter Wikipedia, binäre Suche.

In folgenden Fällen ignoriert die Funktion die Sortierrichtung und arbeitet wie die Funktion **Find in array** (sequentielle Suche, gibt für *ErstePos* und *LetztePos* -1 zurück, wenn *Wert* nicht gefunden wird):

- bei Arraytypen, die sich nicht sortieren lassen, wie z.B. Zeiger Arrays,
- bei Anwendungen im Nicht-Unicode Modus (Kompatibilitätsmodus) und wenn das Array vom Typ String oder Text ist,
- wenn in einem Array vom Typ Text nach einem String mit Joker ('@') am Anfang oder in der Mitte gesucht wird. Hier ist kein binärer Suchalgorithmus möglich, da die passenden Elemente im Array evtl. nicht in fortlaufender Reihenfolge sind.

Gibt die Funktion **False** zurück, kann der in *ErstePos* zurückgegebene Wert an **INSERT IN ARRAY** übergeben werden, um den *Wert* in das bereits vorsortierte Array einzufügen. Das läuft schneller, als erst einen neuen Eintrag ans Ende des Array zu setzen und dann **SORT ARRAY** aufzurufen, um es an die richtige Stelle zu bewegen.

Die in *LetztePos* und *ErstePos* zurückgegebenen Werte lassen sich miteinander kombinieren, um jedes Element des Arrays zu durchlaufen, das zu *Wert* passt (mit einer **For...End for** Schleife) oder um die Anzahl der Vorkommen zu finden (wie auch mit dem Befehl **Count in array**, nur schneller).

Beispiel 1

Bei Bedarf einen Wert einfügen und dabei das sortierte Array beibehalten:

```
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
```

Beispiel 2

Die Anzahl der Vorkommen von Strings finden, die mit "test" beginnen und einen String erstellen, der diese Elemente zusammenfasst:

```
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 (ArrayName ; Start {; AnzElemente})

Parameter	Typ		Beschreibung
ArrayName	Array	⇒	Name des Array
Start	Lange Ganzzahl	⇒	Position, ab der die Elemente hinzugefügt werden sollen
AnzElemente	Lange Ganzzahl	⇒	Anzahl der Elemente, die hinzugefügt werden sollen, 1 Element, falls nichts angegeben ist

Beschreibung

Der Befehl **INSERT IN ARRAY** fügt eine beliebige Anzahl von Elementen in das Array *ArrayName* ein. Die neuen Elemente werden vor *Start* eingefügt und nach dem leeren Wert des Array-Typs initialisiert. Alle Elemente nach *Start* werden um 1 oder gemäß *AnzElemente* verschoben.

Ist *Start* größer als das Array, werden die Elemente am Ende des Array hinzugefügt.

AnzElemente gibt die Anzahl der einzufügenden Elemente an. Ist dieser Parameter nicht angegeben, wird nur ein Element eingefügt. Das Array wird automatisch in der Größe angepasst.

Beispiel 1

Dieses Beispiel fügt ab Element 10 fünf neue Elemente ein:

```
INSERT IN ARRAY(anArray;10;5)
```

Beispiel 2

Dieses Beispiel fügt ein Element am Tabellenende hinzu:

```
$vlElem:=Size of array(anArray)+1
INSERT IN ARRAY(anArray;$vlElem)
anArray{$vlElem}:=...
```

LIST TO ARRAY

LIST TO ARRAY (Liste ; ArrayName {; EintragRefs})

Parameter	Typ		Beschreibung
Liste	String, ListRef	➡	Name oder Referenz der Liste, aus der Einträge der 1. Ebene zu kopieren sind
ArrayName	Array	⬅	Array, in das Einträge der Liste zu kopieren sind
EintragRefs	Array	⬅	Referenznummern für Einträge der Liste

Beschreibung

Der Befehl **LIST TO ARRAY** erstellt oder überschreibt *ArrayName* mit den Einträgen der 1. Ebene aus *Liste*.

Im Parameter *Liste* können Sie entweder den Namen eine Auswahlliste (String) oder eine Referenz auf eine hierarchische Liste übergeben.

Haben Sie das Array zuvor nicht vom Typ Alphanumerisch oder Text definiert, erstellt **LIST TO ARRAY** standardmäßig ein Array vom Typ Text.

Hinweis: Im kompilierten Modus muss das Array *Ziel* vom gleichen Typ wie das Array *Quelle* sein.

Der Parameter *EintragRefs* ist optional (ein numerisches Array). Er gibt die Referenznummern für Einträge in der Auswahlliste zurück.

Mit **LIST TO ARRAY** können Sie ein Array mit Einträgen auf der 1. Ebene einer hierarchischen Liste erstellen. Sie können jedoch nicht mit untergeordneten Einträgen arbeiten. Wir empfehlen dazu, die Befehle für hierarchische Listen, insbesondere **Load list** zu verwenden.

Beispiel 1

Folgendes Beispiel kopiert die Einträge der Liste Regions in ein Array mit Namen *atRegions*:

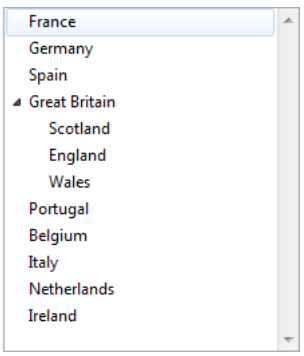
```
LIST TO ARRAY("Regions";atRegions)
```

Beispiel 2

Nehmen wir folgende hierarchische Liste:

```
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)
```

Diese Liste könnte aussehen wie folgt:



Führen Sie folgende Anweisung aus:

```
LIST TO ARRAY(myList1;$MyArray)
```

...erhalten Sie

```
$MyArray{1}="France"
$MyArray{2}="Germany"
$MyArray{3}="Spain"
$MyArray{4}="Great Britain"
$MyArray{5}="Portugal"
...
```

LONGINT ARRAY FROM SELECTION

LONGINT ARRAY FROM SELECTION (Tabellenname ; DatensatzArray {; Auswahl})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle der aktuellen Auswahl ohne Angabe Standardtabelle
DatensatzArray	Array Lange Ganzzahl	⬅	Array der Datensatznummern
Auswahl	String	➡	Name der temporären Auswahl ohne Angabe aktuelle Auswahl

Beschreibung

Der Befehl **LONGINT ARRAY FROM SELECTION** füllt das Array *DatensatzArray* mit den (absoluten) Datensatznummern aus *Auswahl*.

Übergeben Sie den Parameter *Auswahl* nicht, verwendet der Befehl die aktuelle Auswahl von *Tabellenname*.

Hinweis: Das Element des Arrays mit der Nummer 0 wird auf -1 initialisiert.

Beispiel

Die Anzahl der Datensätze in der aktuellen Auswahl wiederfinden:

```
ARRAY LONGINT($_arrRecNum;0)  //zwingend für kompilierten Modus
LONGINT ARRAY FROM SELECTION([Clients];$_arrRecNum)
```

MULTI SORT ARRAY

MULTI SORT ARRAY (ArrayName {; Sortieren}{; ArrayName2 ; Sortieren2 ; ... ; ArrayNameN ; SortierenN})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Zu sortierende Array(s)
Sortieren	Operator	➡	">" aufsteigend sortieren oder "<" absteigend sortieren Ohne Angabe "aa" keine Sortierung

MULTI SORT ARRAY (ZeigerArrayName ; SortArrayName)

Parameter	Typ		Beschreibung
ZeigerArrayName	Array Zeiger	➡	Array der Array Zeiger
SortArrayName	Array Lange Ganzzahl	➡	Sortierfolge Array (1 = aufsteigend, -1 = absteigend, 0 = Anpassung an vorige Sortierung)

Beschreibung

Der Befehl **MULTI SORT ARRAY** sortiert einen Satz Arrays auf mehreren Ebenen.

Der Befehl akzeptiert zwei Syntaxarten.

• **Erste Syntax: MULTI SORT ARRAY (Array{; Sortieren}{; Array2; Sortieren2; ...; arrayN; SortierenN})**
Dies ist die einfachste Syntax; hier übergeben Sie direkt die Namen der synchronisierten Arrays, auf die Sie eine Sortierung nach mehreren Kriterien anwenden wollen.
Sie können eine unbegrenzte Anzahl an Paaren übergeben (*Array;> oder <*) und/oder nur Arrays. Alle als Parameter übergebenen Arrays werden aufeinander abgestimmt sortiert.
Sie können Arrays jeder Art übergeben mit Ausnahme von Zeigern oder Bildern. Sie können nach einem Element in einem zweidimensionalen Array (z.B. *ein2DArray{$vlDiesesElement}*) sortieren, jedoch nicht nach dem 2D Array selbst, z.B. *ein2DArray*).

Wollen Sie den Inhalt eines Array als Sortierkriterium zu verwenden, übergeben Sie den Parameter *Sortieren*. Der Wert des Parameters (> oder <) bestimmt, ob das Array in aufsteigender oder in absteigender Reihenfolge sortiert wird. Ist der Parameter *Sortieren* nicht angegeben, wird der Inhalt des Array nicht als Sortierkriterium verwendet.

Hinweis: Sie müssen mindestens ein Sortierkriterium übergeben, damit der Befehl funktioniert. Gibt es kein Sortierkriterium, wird ein Fehler generiert.

Die Sortierebenen richten sich nach der Reihenfolge, in welcher die Arrays im Befehl übergeben wurden. Die Position eines Array mit einem Sortierkriterium in der Syntax bestimmt seine Sortierebene.

• **Zweite Syntax: MULTI SORT ARRAY (ZeigerArrayName; SortArrayName)**
Diese Syntax ist komplexer und hilfreich bei generischer Entwicklung. Sie können z.B. eine generische Methode erstellen, um Arrays jeglichen Typs zu sortieren, ja sogar die Entsprechung eines generischen Befehls **SORT ARRAY** erstellen.
Der Parameter *ZeigerArrayName* enthält den Namen eines Arrays mit Array Zeigern; jedes Element dieses Arrays ist ein Zeiger auf ein zu sortierendes Array. Die Sortierung erfolgt in der Reihenfolge, wie die Array Zeiger in *ZeigerArrayName* definiert sind.

Hinweis: *ZeigerArrayName* kann das Array eines lokalen ($ptrArrayName), Prozess (ZeigerArrayName) oder Interprozess (<>ZeigerArrayName) Zeigers sein.
Umgekehrt dürfen die Elemente dieses Array nur auf Arrays vom Typ Prozess oder Interprozess zeigen.

Der Parameter *SortArrayName* Parameter kann das Array eines lokalen ($ZeigerArrayName), Prozess (ZeigerArrayName) oder Interprozess (<>ZeigerArrayName) Zeigers sein.
Umgekehrt dürfen die Elemente dieses Array nur auf Arrays vom Typ Prozess oder Interprozess zeigen.
0 = Das Array wird nicht als Sortierkriterium verwendet, sondern muss passend zu den anderen Sortierungen sortiert werden.
1 = Aufsteigend sortieren.
-1 = Absteigend sortieren

Hinweise: Arrays vom Typ Zeiger oder Bild lassen sich nicht sortieren. Sie können ein Element eines zweidimensionalen Array (z.B. a2DArray{$vlThisElement}) sortieren, jedoch nicht das 2D Array selbst (z.B. a2DArray).
Jedes Element des Array *ZeigerArrayName* muss eine Entsprechung im Array *SortArrayName* haben. Deshalb müssen beide Arrays exakt dieselbe Anzahl Elemente haben.

Beispiel 1

Folgendes Beispiel verwendet die 1. Syntax: Es erstellt vier Arrays und sortiert sie nach Stadt in aufsteigender Reihenfolge, dann nach Lohn in absteigender Reihenfolge mit den beiden letzten Arrays *names_array* und *telNum_array*, die mit dem vorigen Sortierkriterium synchronisiert wird:

```
ALL RECORDS([Employees])
SELECTION TO ARRAY([Employees]City;cities;[Employees]Salary;
salaries;[Employees]Name;names;[Employees]TelNum;telNums)
MULTI SORT ARRAY(cities;>;salaries;<;names;telNums)
```

Wollen Sie das Array *Namen* als 3. Sortierkriterium verwenden, fügen Sie nach dem Parameter *names_array* das Zeichen > oder < zu.

Beachten Sie, dass die Syntax:

```
MULTI SORT ARRAY(cities;>;salaries;names;telNums)
```

gleichbedeutend ist mit:

```
SORT ARRAY(cities;salaries;names;telNums;>)
```

Beispiel 2

Folgendes Beispiel verwendet die 2. Syntax: Es erstellt vier Arrays und sortiert sie nach Stadt (aufsteigend) und Firma (absteigend); die beiden letzten Arrays names_Array und telNum_Array werden an die vorigen Sortierkriterien angepasst:

```
ALL RECORDS([Employees])
SELECTION TO ARRAY([Employees]City;cities;[Employees]Company;
companies;[Employees]Name;names;[Employees]TelNum;telNums)
ARRAY POINTER(pointers_Array;4)
ARRAY LONGINT(sorts_Array;4)
pointers_Array{1}:=->cities
sorts_Array{1}:=1
pointers_Array{2}:=->companies
sorts_Array{2}:=-1
pointers_Array{3}:=->names
sorts_Array{3}:=0
pointers_Array{4}:=->telNums
sorts_Array{4}:=0
MULTI SORT ARRAY(pointers_Array;sorts_Array)
```

Wollen Sie als drittes Sortierkriterium ein Array mit Namen verwenden, müssen Sie im Element sorts_Array{3} den Wert 1 zuweisen. Sollen die Arrays nur nach der Stadt sortiert werden, übergeben Sie in den Elementen sorts_Array{2}, sorts_Array{3} und sorts_Array{4} den Wert 0 (Null). Auf diese Weise erhalten Sie ein identisches Ergebnis zu *SORT ARRAY(cities;companies;names;telNums;>)*.

SELECTION RANGE TO ARRAY

SELECTION RANGE TO ARRAY (Start ; Ende {; Feld | Tabelle ; ArrayName} {; Feld | Tabelle2 ; ArrayName2 ; ... ; Feld | TabelleN ; ArrayNameN})

Parameter	Typ		Beschreibung
Start	Lange Ganzzahl	➡	Datensatznummer, ab der Daten geholt werden
Ende	Lange Ganzzahl	➡	Datensatznummer, bis zu der Daten geholt werden
Feld | Tabelle	Feld, Tabelle	➡	Feld zum Daten holen oder Tabelle zum Holen von Datensatznummern
ArrayName	Array	⬅	Array, das Datenfelder bzw. Datensatznr. erhält

Beschreibung

Der Befehl **SELECTION RANGE TO ARRAY** erstellt ein oder mehrere Arrays und kopiert Daten aus Feldern oder Datensatznummern aus der aktuellen Auswahl in die Arrays.

SELECTION RANGE TO ARRAY gilt, im Gegensatz zu **SELECTION TO ARRAY**, die für die gesamte aktuelle Auswahl gilt, nur für den durch die Parameter *Start* und *Ende* festgelegten Bereich.

Sie müssen für diesen Befehl in *Start* und *Ende* Datensatznummern übergeben. Verwenden Sie dafür die Formel *1 <= start <= end <= Records in selection ([...])*.

Übergeben Sie *1 <= start = end < Records in selection ([...])*, werden Datenfelder bzw. die Datensatznummer des Datensatzes geladen, für den gilt *Start = End*.

Übergeben Sie falsche Datensatznummern, führt der Befehl folgendes aus:

- Ist *Ende* > *Records in selection ([...])*, gibt er Werte zurück vom in *Start* festgelegten Datensatz bis zum letzten ausgewählen Datensatz.
- Ist *Start* > *Ende*, gibt er nur Werte aus dem in *Start* festgelegten Datensatz zurück.
- Passen beide Parameter nicht zur Größe der Auswahl, gibt er leere Arrays zurück.

SELECTION RANGE TO ARRAY gilt wie **SELECTION TO ARRAY** für die Auswahl der Tabelle, die im ersten Parameter festgelegt wurde.

Mit **SELECTION RANGE TO ARRAY** können Sie wie mit **SELECTION TO ARRAY**:

- Werte aus einem oder mehreren Datenfeldern laden.
- Mit der Syntax *...;[table];Array;...* Datensatznummern laden.
- Werte aus verknüpften Datenfeldern laden, wenn zwischen den Tabellen eine automatische Viele-zu-Eine Verknüpfung besteht oder zuvor **SET AUTOMATIC RELATIONS** aufgerufen wurde, um manuelle Viele-zu-Eine Verknüpfung zu automatisieren. In beiden Fällen werden Werte aus Tabellen mit mehrstufigen Viele-zu-Eine Verknüpfungen geladen.

Jedes Array erhält den Typ des Datenfelds.

Wenden Sie **SELECTION RANGE TO ARRAY** auf ein Feld vom Typ Zeit an, müssen Sie beachten, dass diese Befehle nur ein Array vom Typ Zeit anlegen, wenn für das Array noch kein anderer Typ definiert wurde. So bleibt zum Beispiel im folgenden Kontext *myArray* ein Array vom Typ Lange Ganzzahl:

```
ARRAY LONGINT(myArray;0)
SELECTION TO ARRAY([myTable]myTimeField;myArray)
```

Datensatznummern werden in ein Array vom Typ *Lange Ganzzahl* geladen.

Hinweis: Sie können den Befehl **SELECTION RANGE TO ARRAY** nur mit den Parametern *Start* und *Ende* aufrufen. Mit dieser speziellen Syntax können Sie für eine eingeschränkte Auswahl über den Parameter * eine gesammelte Reihe von **SELECTION TO ARRAY** Befehlen ausführen (siehe Beispiel 4).

4D Server: SELECTION RANGE TO ARRAY wurde für 4D Server optimiert. Jedes Array wird auf dem Server erstellt und dann komplett auf den Client-Rechner übertragen.

WARNUNG: SELECTION RANGE TO ARRAY erstellt u.U. umfangreiche Arrays, je nach dem in *Start* und *Ende* festgelegten Bereich bzw. Typ und Umfang der zu ladenden Daten. Da Arrays im Speicher bleiben, empfehlen wir, das Ergebnis nach Ende des Befehls zu testen. Prüfen Sie die Größe jedes resultierenden Array oder sichern Sie den Aufruf dieses Befehls mit einer Projektmethode **ON ERR CALL**.

War der Befehl erfolgreich, ist die Größe jedes resultierenden Array gleich *(Ende-Start)+1*, außer der Parameter *Ende* ist größer als die Anzahl der Datensätze in der Auswahl. In diesem Fall enthält jedes resultierende Array *(Records in selection([...])-Start)+1* Elemente.

Beispiel 1

Folgender Code berücksichtigt die ersten 50 Datensätze aus der aktuellen Auswahl für die Tabelle *[Invoices]*. Er lädt die Werte aus dem Datenfeld *[Invoices]Invoice ID* und dem verknüpften Datenfeld *[Customers]Customer ID*.

```
SELECTION RANGE TO ARRAY(1;50;[Invoices]Invoice ID;alInvoID;
[Customers]Customer ID;alCustID)
```

Beispiel 2

Folgender Code berücksichtigt die letzten 50 Datensätze aus der aktuellen Auswahl für die Tabelle *[Invoices]*. Er lädt die Datensatznummern aus *[Invoices]* sowie aus der Verknüpfung zu *[Customers]*:

```
lSelSize:=Records in selection([Invoices])
SELECTION RANGE TO ARRAY(lSelSize-49;lSelSize;[Invoices];alInvRecN;
[Customers];alCustRecN)
```

Beispiel 3

Folgender Code erstellt, in Segmenten zu je 1000 Datensätzen, eine umfangreiche Auswahl, die nicht als ein Ganzes in Arrays geladen werden kann:

```
lMaxPage:=1000
lSelSize:=Records in selection([Phone Directory])
For($lPage ;1;1+((lSelSize-1)\lMaxPage))
  ` Lade Werte und/oderDatensatznummern
   SELECTION RANGE TO ARRAY(1+(lMaxPage*($lPage-1));
   lMaxPage*$lPage;...;...;...;...;...;...)
  ` Führe etwas aus mit den Arrays
End for
```

Beispiel 4

Die ersten 50 aktuellen Datensätze der Tabelle [Invoices] zum Laden verschiedener Arrays verwenden:

```
 // Anweisungen sammeln
SELECTION TO ARRAY([Invoices]InvoiceRef;arrLInvRef;*)
SELECTION TO ARRAY([Invoices]Date;arrDInvDate;*)
SELECTION TO ARRAY([Clients]ClientRef;arrLClientRef;*)
 // Gesammelte Anweisungen ausführen
SELECTION RANGE TO ARRAY(1;50)
```

SELECTION TO ARRAY

SELECTION TO ARRAY {(Feld | Tabelle ; ArrayName {; Feldname ; ArrayName {; Feldname2 ; ArrayName2 ; ... ; FeldnameN ; ArrayNameN}}{; *})}

Parameter	Typ		Beschreibung
Feld | Tabelle	Feld, Tabelle	➡	Feld zum Holen von Daten oder Tabelle zum Holen von Datensatznummern
ArrayName	Array	⬅	Array, das Feldwerte oder Datensatznummern erhalten soll
Feldname	Feld	➡	Wiederzufindendes Feld in Array
ArrayName	Array	⬅	Array zum Empfangen von Felddaten
*	Operator	➡	Auf Ausführung warten

Beschreibung

Der Befehl **SELECTION TO ARRAY** erstellt ein oder mehrere Arrays und kopiert Daten in Feldern oder Datensatznummern aus der aktuellen Auswahl in die Arrays.

SELECTION TO ARRAY gilt für die Auswahl der Tabelle, die im ersten Parameter (Tabellenname oder Feldname) angegeben ist. Damit können Sie:

- Werte aus einem oder mehreren Datenfeldern laden.
- Mit der Syntax *...;[table];array;...* Datensatznummern laden.
- Werte aus verknüpften Datenfeldern laden, wenn zwischen den Tabellen eine automatische Viele-zu-Eine Verknüpfung besteht oder zuvor **SET AUTOMATIC RELATIONS** aufgerufen wurde, um manuelle Viele-zu-Eine Verknüpfung zu automatisieren. In beiden Fällen werden Werte aus Tabellen mit mehrstufigen Viele-zu-Eine Verknüpfungen geladen.

Jedes Array erhält den Typ des Datenfeldes.

Wenden Sie **SELECTION TO ARRAY** auf ein Feld vom Typ Zeit an, müssen Sie beachten, dass diese Befehle nur ein Array vom Typ Zeit anlegen, wenn für das Array noch kein anderer Typ definiert wurde. So bleibt zum Beispiel im folgenden Kontext *myArray* ein Array vom Typ Lange Ganzzahl:

```
ARRAY LONGINT(myArray;0)
SELECTION TO ARRAY([myTable]myTimeField;myArray)
```

Datensatznummern werden in ein Array vom Typ *Lange Ganzzahl* geladen.

Übergeben Sie den Parameter *, führt 4D die entsprechende Zeile der Anweisung nicht sofort aus, sondern legt sie stattdessen in den Speicher; auf diese Weise können Sie Zeilen, die mit * enden, stapeln. Die auf Ausführung wartenden Zeilen werden durch eine abschließende Anweisung **SELECTION TO ARRAY** ohne den Parameter * ausgeführt. Aus diesem Grund lässt sich der Befehl jetzt ohne weitere Parameter aufrufen.
Die Arraytypen werden beim Ausführen der letzten Zeile (ohne *) geprüft.
Analog zum Befehl **QUERY** können Sie so eine komplexe Anweisung in eine Reihe von Zeilen aufteilen, die leichter zu lesen und zu pflegen ist. Sie können auch Anweisungen dazwischen einfügen oder ein Array innerhalb einer Schleife anlegen. Weitere Informationen dazu finden Sie im 2. Beispiel des Befehls **ARRAY TO SELECTION**.

4D Server: SELECTION TO ARRAY wurde für 4D Server optimiert. Jedes Array wird auf dem Server erstellt und dann komplett auf den Client-Rechner übertragen.

WARNUNG: SELECTION TO ARRAY erstellt u.U. umfangreiche Arrays, je nach der Größe der aktuellen Auswahl bzw. Typ und Umfang der zu ladenden Daten. Da Arrays im Speicher bleiben, empfehlen wir, das Ergebnis nach Ende des Befehls zu testen. Prüfen Sie die Größe jedes resultierenden Array oder sichern Sie den Aufruf dieses Befehls mit einer Projektmethode **ON ERR CALL**.

Hinweis: Nach Aufrufen von **SELECTION TO ARRAY** bleiben die aktuelle Auswahl und der aktuelle Datensatz gleich, der aktuelle Datensatz wird jedoch nicht länger geladen. Benötigen Sie die Werte der Datenfelder im aktuellen Datensatz, verwenden Sie nach Aufrufen von **SELECTION TO ARRAY** den Befehl **LOAD RECORD**.

Beispiel 1

Im folgenden Beispiel besteht eine automatische Verknüpfung von der Tabelle *[People]* zur Tabelle [Company]. Die beiden Arrays *asLastName* und *asCompanyAddr* werden gemäß der Anzahl der in der Tabelle *[People]* ausgewählten Datensätze dimensioniert und enthalten Informationen aus beiden Tabellen:

```
SELECTION TO ARRAY([People]Last Name;asLastName;[Company]Address;asCompanyAddr)
```

Beispiel 2

Folgendes Beispiel gibt die Datensatznummern von *[Clients]* im Array *alRecordNumbers* zurück und die Datenfeldwerte aus *[Clients]Names* im Array *asNames*:

```
SELECTION TO ARRAY([Clients];alRecordNumbers;[Clients]Names;asNames)
```

Dasselbe Beispiel können Sie auch so schreiben:

```
SELECTION TO ARRAY([Clients];alRecordNumbers;*)
SELECTION TO ARRAY([Clients]Names;asNames;*)
```

SELECTION TO ARRAY

Size of array

Size of array (ArrayName) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Array, dessen Größe gesucht ist
Funktionsergebnis	Lange Ganzzahl	➲	Gibt die Anzahl der Elemente im Array zurück

Beschreibung

Die Funktion **Size of array** gibt die Anzahl der Elemente aus *ArrayName* zurück.

Beispiel 1

Diese Anweisung gibt die Anzahl der Elemente von *anArray* zurück:

```
vlSize:=Size of array(anArray)  ` vlSize erhält die Größe von anArray
```

Beispiel 2

Diese Anweisung gibt die Anzahl der Reihen in einem zweidimensionalen Array zurück:

```
vlRows:=Size of array(a2DArray)  ` vlRows erhält die Größe von a2DArray
```

Beispiel 3

Diese Anweisung gibt die Anzahl der Spalten für eine Reihe in einem zweidimensionalen Array zurück:

```
vlColumns:=Size of array(a2DArray{10})
  ` vlColumns erhält die Größe von a2DArray{10}
```

SORT ARRAY

SORT ARRAY (ArrayName {; ArrayName2 ; ... ; ArrayNameN}{; > oder <})

Parameter	Typ		Beschreibung
ArrayName	Array	➡	Zu sortierendes Array
> oder <	Operator	➡	">" Sortieren in aufsteigender Reihenfolge, oder "<". Sortieren in absteigender Reihenfolge, bzw. aufsteigend, wenn nichts angegeben ist

Beschreibung

Der Befehl **SORT ARRAY** sortiert eine oder mehrere Arrays in der festgelegten auf- oder absteigenden Sortierreihenfolge.

Hinweis: Arrays vom Typ *Zeiger* oder *Bild* lassen sich nicht sortieren. Sie können die Elemente eines zweidimensionalen Array (z.B. *a2DArray{$vlThisElem}*) sortieren, jedoch nicht das zweidimensionale Array selbst (z.B. *a2DArray*).

Der letzte Parameter gibt an, ob *ArrayName* aufsteigend oder absteigend sortiert werden soll. Das Symbol "größer als" (>) sortiert aufsteigend; das Symbol "kleiner als" (<) sortiert absteigend. Ist keine Sortierreihenfolge angegeben, wird das Array aufsteigend sortiert.

Bei mehreren Arrays wird nach der Reihenfolge des ersten Array sortiert, hier wird keine mehrstufige Sortierung ausgeführt. Um synchronisierte Arrays zu sortieren, können Sie den Befehl **MULTI SORT ARRAY** verwenden.

Beispiel 1

Dieses Beispiel lädt zwei Arrays und sortiert sie nach Firma:

```
ALL RECORDS([People])
SELECTION TO ARRAY([People]Name;asNames;[People]Company;asCompanies)
SORT ARRAY(asCompanies;asNames;>)
```

Da **SORT ARRAY** jedoch nicht mehrstufig sortiert, sind die Namen innerhalb der Firma in wilkürlicher Reihenfolge. Um auch nach Namen zu sortieren, schreiben Sie:

```
ALL RECORDS([People])
ORDER BY([People];[People]Company;>;[People]Name;>)
SELECTION TO ARRAY([People]Name;asNames;[People]Company;asCompanies)
```

Beispiel 2

Zeigen Sie die Namen aus der Tabelle *[People]* in einem Palettenfenster an. Durch Anklicken der entsprechenden Schaltflächen können Sie die Namen von A zu Z oder von Z zu A sortieren. Da manche Namen öfter vorkommen können, sollten Sie auch mit einem einmaligen und indizierten Datenfeld *[People]ID number* arbeiten. Klicken Sie in die Namensliste, finden Sie den Datensatz, der zum angeklickten Namen gehört. Durch ein synchrones ausgeblendetes Array mit den Kenn-Nummern stellen Sie sicher, dass der richtige Datensatz aufgerufen wird:

```
  ` Objektmethode Array asNames
Case of
   :(Form event=On Load)
      ALL RECORDS([People])
      SELECTION TO ARRAY([People]Name;asNames;[People]ID number;allIDs)
      SORT ARRAY(asNames;allIDs;>)
   :(Form event=On Unload)
      CLEAR VARIABLE(asNames)
      CLEAR VARIABLE(allIDs)
   :(Form event=On Clicked)
      If(asNames#0)
  ` Verwende das Array allIDs, um den richtigen Datensatz zu finden
         QUERY([People];[People]ID Number=allIDs{asNames})
  ` Führe etwas aus mit diesem Datensatz
      End if
End case

  ` Objektmethode Schaltfläche bA2Z
  ` Sortiere Arrays in aufsteigender Reihenfolge und erhalte sie synchron
SORT ARRAY(asNames;allIDs;>)

  ` Objektmethode Schaltfläche bZ2A
  ` Sortiere Arrays in absteigender Reihenfolge und erhalte sie synchron
SORT ARRAY(asNames;allIDs;<)
```

TEXT TO ARRAY

TEXT TO ARRAY (varText ; arrText ; Breite ; Schriftname ; Schriftgröße {; Schriftstil {; *}})

Parameter	Typ		Beschreibung
varText	Text	➡	Originaltext zum Unterteilen
arrText	Array Text	⬅	Array, das den in Wörter oder Zeilen unterteilten Text enthält
Breite	Lange Ganzzahl	➡	Maximale Breite des String (in Pixel)
Schriftname	Text	➡	Name der Schrift
Schriftgröße	Lange Ganzzahl	➡	Größe der Schrift
Schriftstil	Lange Ganzzahl	➡	Stil der Schrift
*	Operator	➡	Mit Stern = interpretiere Text als multistyle

Beschreibung

Der Befehl **TEXT TO ARRAY** wandelt eine Textvariable in ein Text-Array um. Der Originaltext in *varText* (formatiert oder nicht) wird unterteilt und jeder Teil wird ein Element des Array *arrText*, das der Befehl zurückgibt. Dieser Befehl lässt sich z.B. verwenden, um Seiten oder Spalten mit Text in einer vorgegebenen Größe zu füllen.
Der Originaltext wird gemäß einer festgelegten Zeilenbreite in "Wörter" unterteilt, die auch die Stilelemente berücksichtigen.

Im Parameter *varText* übergeben Sie den Text, der in Array-Elemente unterteilt werden soll. Dieser Text kann multistyle sein oder auch nicht. Bei Multistyle-Text werden einige Parameter ignoriert.

Im Parameter *arrText* übergeben Sie den Namen des Array, das mit dem unterteilten Text gefüllt werden soll.

Im Parameter *Breite* übergeben Sie eine Größe in Pixel, welche die maximale Zeilenlänge zum Einteilen des Textes festlegt. Der Befehl bewertet für den gesamten Text die maximale Anzahl Wörter, die in diese Breite passt, und berücksichtigt seine grafischen Attribute (Schriftart, Schriftstil).

- Bei Multistyle-Text werden die Stile des Originaltextes berücksichtigt und die nachfolgenden Parameter ignoriert - sofern sie übergeben wurden. In diesem Fall behalten die Textzeilen im resultierenden Array ihre Originalstile bei. Sie lassen sich z.B. durch eine Variable vom Typ Text oder String der Reihe nach drucken.
- Bei Rohtext (ohne Stile) müssen Sie alle Parameter übergeben, damit der Befehl die Länge der Zeilen berechnen kann.

Jedes Array Element muss mindestens ein Wort erhalten. Ist die angegebene Breite zu schmal für die strikt einzuhaltende Unterteilungsregel, wird das Array so nah wie möglich an diesem Parameter gefüllt und die Variable OK wird auf 0 gesetzt. Definieren Sie eine Breite von 3 Pixel, sind die meisten Wörter wahrscheinlich länger als dieser Wert. In diesem Fall wird die Variable OK auf 0 gesetzt.
Das bedeutet auch, dass die maximale Größe des zurückgegebenen Array theoretisch gleich der Anzahl Wörter ist, die in *varText* gefunden werden.

In den Parametern *Schriftname* und *Schriftgröße* übergeben Sie Schriftname und -größe, die der Befehl für *varText* beim Unterteilen einbeziehen muss. Diese Parameter sind bei Rohtext zwingend.

Im Parameter *Schriftstil* übergeben Sie eine oder mehrere Konstanten unter dem Thema **Schriftstile**:

Konstante	Typ	Wert
Bold	Lange Ganzzahl	1
Italic	Lange Ganzzahl	2
Plain	Lange Ganzzahl	0
Underline	Lange Ganzzahl	4

Dieser Parameter ist optional; wird er weggelassen, wird der Stil Normal verwendet.

Ist der optionale Parameter * übergeben, müssen die Parameter *Schriftname*, *Schriftgröße* bzw. *Schriftstil* für Multistyle-Texte berücksichtigt werden, sofern diese Parameter nicht im Originaltext definiert sind. Sind sie dagegen bereits im Originaltext definiert, werden diese Parameter in allen Fällen ignoriert.

Beispiel 1

Einen Multistyle-Text in Zeilen mit einer maximalen Breite von 200 Pixel unterteilen:

```
TEXT TO ARRAY(theText;TextArray;200;"Arial";20;Normal;*)
 // die Attribute Arial, 20 und Normal werden nur berücksichtigt, wenn sie nicht im Text definiert sind
```

Beispiel 2

Wir wollen Rohtext in Zeilen mit einer maximalen Breite von 350 Pixel in der Schrift Bodoni Bold, Größe 14 unterteilen. Da dieser Befehl nicht korrekt arbeitet, wenn die Schrift fehlt, ist es wichtig, das zu prüfen:

```
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
 // andere Schrift verwenden
End if
```

Beispiel 3

Multistyle-Text soll ohne Stile in der Schrift Arial Normal, Größe 12 mit einer maximalen Breite von 600 Pixel gedruckt werden:

```
 // Multistyle Text in Rohtext umwandeln
$RawText:=OBJECT Get plain text(vText)
 // das Array füllen
TEXT TO ARRAY($RawText;TextArray;600;"Arial";12)
```

Beispiel 4

Sie müssen in einem 400-Pixel breiten Bereich einen Text mit maximal 80 Zeilen drucken und dafür die größtmögliche Schrift verwenden (begrenzt auf 24 Punkt). Sie schreiben wie folgt:

```
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 (Länge ; ArrayName ; Größe {; Größe2})

Parameter	Typ		Beschreibung
Länge	Lange Ganzzahl	➡	Länge der Zeichenkette (1... 255)
ArrayName	Array	➡	Name des Array
Größe	Lange Ganzzahl	➡	Anzahl der Elemente im Array der Anzahl der Reihen, wenn Größe2 angegeben ist
Größe2	Lange Ganzzahl	➡	Anzahl der Spalten in zweidimensionalem Array

Hinweis zur Kompatibilität

Der Befehl **_o_ARRAY STRING** arbeitet exakt wie der Befehl **ARRAY TEXT** (Der Parameter *Länge* wird ignoriert). Wir empfehlen jetzt, in Ihren 4D Entwicklungen nur den Befehl **ARRAY TEXT** zu verwenden.

Auswahl

- 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 {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe

Beschreibung

Der Befehl **ALL RECORDS** wählt alle Datensätze der Tabelle *Tabellenname* aus, d. h., alle Datensätze befinden sich in der aktuellen Auswahl. Der erste Datensatz wird als aktueller Datensatz in den Hauptspeicher geladen. Die Datensätze erscheinen in der Standardreihenfolge, also so, wie sie auf der Festplatte gespeichert sind.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich **ALL RECORDS** auf die Haupttabelle.

Arbeiten Sie mit Suchbefehlen wie **QUERY**, brauchen Sie **ALL RECORDS** nicht zuvor aufzurufen. Die Suchbefehle beziehen sich immer auf die ganze Tabelle.

Beispiel

Folgendes Beispiel zeigt alle Datensätze aus der Tabelle [People]:

```
ALL RECORDS([People]) ` Wähle alle Datensätze in der Tabelle
DISPLAY SELECTION([People]) ` Zeige Datensätze in Ausgabeformular an
```

APPLY TO SELECTION

APPLY TO SELECTION (Tabellenname ; Anweisung)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle für die Anweisung
Anweisung	Anweisung	➡	Eine Code-Zeile oder eine Methode

Beschreibung

Der Befehl **APPLY TO SELECTION** wendet für den laufenden Prozess eine Formel auf alle Datensätze der aktuellen Auswahl der Tabelle *Tabellenname* an. *Anweisung* kann eine Zeile Code oder eine Methode sein. Ändert *Anweisung* nur einen Wert in einem Datenfeld der Tabelle *Tabellenname*, werden die Datensätze automatisch gesichert. Werden keine Datenfelder geändert, werden die Datensätze nur sequentiell gelesen. Sie sind dann für andere Prozesse nicht gesperrt.
Ist die aktuelle Auswahl leer, hat **APPLY TO SELECTION** keine Auswirkung. Bei einer automatischen Verknüpfung kann *Anweisung* auch ein Datenfeld aus einer verknüpften Tabelle enthalten.

Die Ausführung von **APPLY TO SELECTION** wird in einem gesonderten Fenster durch einen Ablaufbalken angezeigt. Der Benutzer kann die Operation in diesem Fenster abbrechen. Rufen Sie vor diesem Befehl den Befehl **MESSAGES OFF** auf, wird der Ablaufbalken nicht angezeigt.

Mit **APPLY TO SELECTION** können Sie Information von einer Auswahl an Datensätzen sammeln, z.B. eine Summe oder eine Auswahl ändern. Sie können beispielsweise den ersten Buchstaben eines Datenfeldes in Großschreibung umwandeln. Verwenden Sie diesen Befehl innerhalb einer Transaktion, können Sie alle Änderungen durch Abbrechen der Transaktion rückgängig machen.

4D Server: Der Server führt in *Anweisung* übergebene Befehle nicht aus. Jeder Datensatz der Auswahl wird zum Ändern zur Arbeitsstation zurückgesendet.

Beispiel 1

Folgendes Beispiel wandelt alle Namen der Tabelle [Employees] in Großbuchstaben um:

```
APPLY TO SELECTION([Employees];[Employees]Last Name
:=Uppercase([Employees]Last Name))
```

Beispiel 2

Ist ein Datensatz während der Ausführung von **APPLY TO SELECTION** gesperrt und wird geändert, wird er nicht gesichert. Alle gesperrten Datensätze werden in der Menge *LockedSet* abgelegt. Prüfen Sie diese Menge nach der Ausführung von **APPLY TO SELECTION** auf gesperrte Datensätze. Folgende Schleife wird ausgeführt, bis alle Datensätze geändert sind:

```
Repeat
   APPLY TO SELECTION([Employees];[Employees]Last Name:=Uppercase([Employees]Last Name))
   USE SET("LockedSet")  ` Wähle nur die gesperrten Datensätze
Until(Records in set("LockedSet")=0)
  ` Beende, wenn es keine gesperrten Datensätze mehr gibt
```

Beispiel 3

Dieses Beispiel verwendet eine Methode:

```
ALL RECORDS([Employees])
APPLY TO SELECTION([Employees];M_Cap)
```

Systemvariablen und Mengen

Klickt der Benutzer im Ablaufbalken auf die Schaltfläche Stop, hat die Systemvariable OK den Wert 0 (Null), sonst den Wert 1.

Before selection

Before selection {(Tabellenname)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe
Funktionsergebnis	Boolean	⇐	Ja (TRUE) oder Nein (FALSE)

Beschreibung

Die Funktion **Before selection** gibt *TRUE* zurück, wenn **PREVIOUS RECORD** über den ersten Datensatz der Auswahl der Tabelle *Tabellenname* hinaus gelesen hat. Ist die aktuelle Auswahl leer, hat sie keine Auswirkung.

Mit den Befehlen **FIRST RECORD**, **LAST RECORD** oder **GOTO SELECTED RECORD** können Sie dann wieder einen aktuellen Datensatz bestimmen, ohne die Auswahl verändern zu müssen.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich die Funktion auf die Haupttabelle.

Wird der erste Kopfteil eines Berichts gedruckt, gibt **Before selection** ebenfalls *TRUE* zurück. Mit folgendem Code können Sie den ersten Kopfteil prüfen und einen speziellen Kopfteil für die erste Seite drucken:

```
  ` Formularmethode als Ausgabeformular für einen Summenbericht
$vpFormTable:=Current form table
Case of
  ` ...
   :(Form event=On Header)
  ` Es wird gleich ein Kopfteil gedruckt
      Case of
         :(Before selection($vpFormTable->))
  ` Code für den ersten Kopfteil kommt hier
  ` ...
      End case
End case
```

Beispiel

Diese Formularmethode wird während dem Drucken eines Berichts eingesetzt. Sie setzt eine Variable *vTitle*, um den Kopfteil für die erste Seite zu drucken:

```
  ` Formularmethode [Finances];"Übersicht"
Case of
  ` ...
   :(Form event=On Header)
      Case of
         :(Before selection([Finances))
            vTitle:="Firmenbericht 1997"  ` Setze Titel für die erste Seite
         Else
            vTitle:=""  ` Lösche Titel für alle anderen Seiten
      End case
End case
```

CREATE SELECTION FROM ARRAY

CREATE SELECTION FROM ARRAY (Tabellenname ; DatensatzArray {; Auswahlname})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Erstellen der Auswahl
DatensatzArray	Lange Ganzzahl, Array Boolean	➡	Array der Datensatznummern oder Array der Booleans: Wahr = Datensatz in Auswahl Falsch = Datensatz nicht in Auswahl
Auswahlname	String	➡	Name der zu erstellenden temporären Auswahl Ohne Angabe Befehl auf die aktuelle Auswahl anwenden

Beschreibung

Der Befehl **CREATE SELECTION FROM ARRAY** erstellt die temporäre Auswahl *Auswahlname*:

- entweder aus einem Array der absoluten Datensatznummern *DatensatzArray* aus *Tabellenname*,
- oder aus einem Array mit Booleans. In diesem Fall geben die Werte des Array für jeden Datensatz an, ob er in *Tabellenname* zu *Auswahlname* gehört (**True**) oder nicht (**False**).

Übergeben Sie den Parameter *Auswahlname* nicht oder einen leeren String, arbeitet der Befehl mit der aktuellen Auswahl, die dann aktualisiert wird.

Verwenden Sie mit diesem Befehl ein Array vom Typ Lange Ganzzahl, entsprechen die Nummern des Array den Datensatznummern in *Auswahlname*. Ist eine Nummer nicht korrekt (nicht erstellter Datensatz), wird der Fehler -10503 erzeugt.

Hinweis: Stellen Sie sicher, dass das Array keine Zeilen enthält, die denselben Wert haben, denn dann erhalten Sie als Ergebnis eine nicht korrekte Auswahl.

Verwenden Sie mit diesem Befehl ein Array vom Typ Boolean, gibt das N-te Element des Array an, ob der Datensatz mit der Nummer N in *Auswahlname* (True) ist oder nicht (False) ist. Die Anzahl der Elemente in *DatensatzArray* muss mit der Anzahl der Datensätze in *Tabellenname* übereinstimmen. Ist das Array kleiner als die Anzahl der Datensätze, können nur die im Array definierten Datensätze für die Auswahl verwendet werden.

Hinweis: Bei einem Array mit Booleans verwendet der Befehl Elemente der Nummern 0 bis N-1.

Warnung: Eine temporäre Auswahl wird im Speicher erstellt und geladen. Sorgen Sie deshalb vor Ausführen dieses Befehls für ausreichend Speicherplatz.

Fehlerverwaltung

Bei einer ungültigen Datensatznummer (Datensatz wird nicht erstellt) wird der Fehler -10503 generiert. Sie können ihn mit einer Methode abfangen, die über den Befehl **ON ERR CALL** aufgerufen wird.

DELETE SELECTION

DELETE SELECTION {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe

Beschreibung

Der Befehl **DELETE SELECTION** löscht alle Datensätze in der aktuellen Auswahl der Tabelle *Tabellenname*. Ist die aktuelle Auswahl leer, hat er keine Auswirkung. Wurden die Datensätze gelöscht, ist die aktuelle Auswahl von *Tabellenname* leer. Datensätze, die während einer Transaktion gelöscht werden, sind für andere Benutzer und Prozesse gesperrt, bis die Transaktion bestätigt oder abgebrochen wird.
Datensätze, die nicht gelöscht werden konnten, weil sie von einem anderen Prozess gesperrt waren, legt 4D in die Menge "LockedSet".

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich **DELETE SELECTION** auf die Haupttabelle.

Warnung: Das Löschen einer Datensatzauswahl sollte mit Bedacht verwendet werden, denn die Operation ist dauerhaft und lässt sich nicht mehr rückgängig machen.

Ist die Option **Datensätze vollständig löschen** im Inspektorfenster nicht markiert, erhöht das die Löschgeschwindigkeit beim Verwenden von **DELETE SELECTION**. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus* im Abschnitt **Datensätze definitiv löschen**.

Beispiel 1

Folgendes Beispiel zeigt alle Datensätze aus der Tabelle [People] und lässt den Anwender wählen, welche er löschen möchte. Das Beispiel ist in zwei Teile gegliedert. Der erste Teil ist eine Methode, die Datensätze anzeigt, der zweite eine Objektmethode für die Schaltfläche **Löschen**.

Die Methode:

```
ALL RECORDS([People])  ` Wähle alle Datensätze
FORM SET OUTPUT([People];"Listing")  ` Setze Formular
DISPLAY SELECTION([People])  ` Zeige alle Datensätze an
```

Die Objektmethode für die Schaltfläche **Löschen**, die im Fußteil des Ausgabeformulars erscheint. Sie verwendet die vom Benutzer ausgewählten Datensätze (die Menge *UserSet*) zum Löschen der Auswahl. Hat der Benutzer keine Auswahl getroffen, hat **DELETE SELECTION** keine Auswirkung.

```
  ` Bestätige, dass der Benutzer die Datensätze wirklich löschen will
CONFIRM("Sie haben "+String(Records in set("UserSet"))+" Personen zum Löschen gewählt."
+Char(13)+"Klicken Sie zum Löschen auf OK.")
If(OK=1)
   USE SET("UserSet")  ` Verwende die vom Benutzer gewählten Datensätze
   DELETE SELECTION([People])  ` Lösche Datensätze in Auswahl
End if
ALL RECORDS([People])  ` Wähle alle Datensätze
```

Beispiel 2

Datensätze, die während der Ausführung von **DELETE SELECTION** gesperrt sind, werden nicht gelöscht. Sie werden in der Menge *LockedSet* abgelegt. Mit dieser Menge können Sie anschließend prüfen, ob Datensätze gesperrt waren. Die folgende Schleife wird ausgeführt, bis alle Datensätze gelöscht sind:

```
Repeat  ` Wiederhole für alle gesperrten Datensätze
   DELETE SELECTION([ThisTable])
   If(Records in set("LockedSet")#0)  ` Gibt es gesperrte Datensätze
      USE SET("LockedSet")  ` Wähle nur die gesperrten Datensätze
   End if
Until(Records in set("LockedSet")=0)  ` Bis es keine gesperrten Datensätze mehr gibt
```

DISPLAY SELECTION

DISPLAY SELECTION ({Tabellenname}{; AuswModus}{; EingListe}{; *}{; *})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe
AuswModus	Lange Ganzzahl	➡	Auswahlmodus
EingListe	Boolean	➡	Option Eingabe in Liste erlauben
*		➡	Ausgabeformular für einzelnen Datensatz verwenden und Rollbalken im Eingabeformular ausblenden
*		➡	Rollbalken im Eingabeformular anzeigen (überschreibt die 2. Option des 1. Parameters *)

Beschreibung

Der Befehl **DISPLAY SELECTION** zeigt die Auswahl der Tabelle *Tabellenname* im aktuellen Ausgabeformular an. Die Datensätze erscheinen in einer scrollbaren Liste ähnlich wie die Liste in der Benutzerumgebung. Klickt der Anwender einen Datensatz zweimal an, wird dieser standardmäßig im aktuellen Eingabeformular dargestellt. Die Liste erscheint im vordersten Fenster. Wollen Sie eine Auswahl anzeigen und auch einen Datensatz im aktuellen Eingabeformular nach Doppelklicken ändern (wie im Fenster der Designumgebung), verwenden Sie den Befehl **MODIFY SELECTION** anstatt **DISPLAY SELECTION**.
Alle nachfolgenden Informationen gelten für beide Befehle. Nur für Ändern von Datensätzen gibt es Unterschiede.

Nach Ausführen von **DISPLAY SELECTION** gibt es u.U. keinen aktuellen Datensatz mehr. Verwenden Sie einen Befehl wie **FIRST RECORD** oder **LAST RECORD**, um wieder einen aktuellen Datensatz auszuwählen.

In *AuswModus* setzen Sie den Auswahlmodus für die Datensätze. Hier können Sie eine der Konstanten unter dem Thema **Formularoptionen** übergeben:

Konstante	Typ	Wert	Kommentar
Multiple selection	Lange Ganzzahl	2	Der Benutzer kann mehrere Datensätze auf einmal auswählen. Für zusammenhängende Datensätze den ersten Datensatz anklicken und mit gedrückter Umschalttaste den letzten Datensatz für die Auswahl anklicken. Für nicht zusammenhängende Datensätze jeden Datensatz einzeln anklicken, unter Windows mit gedrückter **Strg-Taste**, auf Mac OS mit gedrückter **Befehlstaste**.
No selection	Lange Ganzzahl	0	In der Liste lässt sich kein Datensatz auswählen.
Single selection	Lange Ganzzahl	1	Nur ein Datensatz ist auf einmal auswählbar.

Standardmäßig, d.h. ohne den Parameter *AuswModus*, wird die Option multiple Auswahl verwendet.

In *EingListe* ermöglichen Sie in der angezeigten Liste die Option “Eingabe in Liste”. Der Benutzer kann dann die Werte des Datensatzes direkt im Ausgabeformular auswählen und ändern. Mit Wahr ist die Eingabe zugelassen, mit Falsch ist sie nicht zugelassen. Standardmäßig, d.h. ohne den Parameter *EingListe* ist diese Option nicht zugelassen.
Beachten Sie, dass mit diesem Parameter nur möglich ist, die Werte in der Liste auszuwählen und nicht, sie zu ändern. **DISPLAY SELECTION** lädt nämlich die Datensätze der aktuellen Auswahl im aktuellen Prozess im Nur-Lesen Modus. Die Eingabe von Werten ist nur mit dem Befehl **MODIFY SELECTION** möglich.

Hinweis: Mit dem Befehl **OBJECT SET ENTERABLE** können Sie die Option “Eingabe in Liste” sofort aktivieren oder deaktivieren.

Für die optionalen Parameter * gelten folgende Regeln:

- Enthält die Auswahl nur einen Datensatz und wird der optionale Parameter * nicht angegeben, erscheint der Datensatz im Eingabeformular anstatt im Ausgabeformular.
- Wird der optionale Parameter * angegeben und zeigt der Benutzer den Datensatz im Eingabeformular durch Doppelklick an, werden die Rollbalken im Eingabeformular ausgeblendet. Um diesen Effekt wieder umzukehren, übergeben Sie einen zweiten optionalen*.

Die selbsterstellten Schaltflächen können im Fuß- oder Kopfteil des Ausgabeformulars liegen, um die Ausführung von **DISPLAY SELECTION** zu bestimmen. Sie können zum Verlassen die automatischen Schaltflächen **Bestätigen** oder **Abbrechen** verwenden bzw. eine Objektmethode, die die Befehle **ACCEPT** oder **CANCEL** aufruft. Ein über **DISPLAY SELECTION** aufgerufenes Ausgabeformular hat keine Schaltflächen. Sie können es nur über die **Escape-Taste** unter Windows, **Esc** auf Mac OS verlassen.

Während und nach dem Ausführen von **DISPLAY SELECTION** werden die vom Benutzer ausgewählten Datensätze in der Menge “UserSet” abgelegt. “UserSet” ist in der Auswahlanzeige für Objektmethoden verfügbar, wenn auf eine Schaltfläche geklickt oder ein Menüeintrag gewählt wird. Diese Menge ist auch in der Projektmethode verfügbar, die **DISPLAY SELECTION** nach Ausführen des Befehls aufgerufen hat.

Beispiel 1

Folgendes Beispiel wählt alle Datensätze der Tabelle [People]. Sie werden mit **DISPLAY SELECTION** angezeigt. Der Benutzer kann nun die Datensätze auswählen, die er drucken möchte. Die Datensätze werden mit **USE SET** ausgewählt und dann mit **PRINT SELECTION** gedruckt:

```
ALL RECORDS([People]) ` Wähle alle Datensätze
DISPLAY SELECTION([People];*) ` Zeige die Datensätze an
USE SET("UserSet") ` Verwende nur die vom Benutzer markierten Datensätze
PRINT SELECTION([People]) ` Drucke die markierten Datensätze
```

Beispiel 2

Siehe Beispiel 6 unter der Funktion **Form event**. Es zeigt alle Tests, die während dem Ablaufen von **DISPLAY SELECTION** zur kompletten Steuerung der Ereignisse notwendig sind.

Beispiel 3

Wollen Sie zum Beispiel die Funktionsweise des Menüs Suchen der Benutzerumgebung mit **DISPLAY SELECTION** oder **MODIFY SELECTION** in der Runtime-Umgebung reproduzieren, gehen Sie folgendermaßen vor:

1. Erstellen Sie in der Designumgebung eine Menüleiste mit den gewünschten Befehlen, z.B. Alle zeigen, Suchen und Sortieren
2. Weisen Sie diese Menüleiste mit einer negativen Menüleistennummer dem Ausgabeformular zu, das Sie mit **DISPLAY SELECTION** oder **MODIFY SELECTION** verwenden.
3. Den Menübefehlen weisen Sie folgende Projektmethoden zu:

```
` M_Alle zeigen (dem Menübefehl Alle zeigen zugeordnet)
$vpCurTable:=Current form table
ALL RECORDS($vpCurTable->)

` M_Suchen (Dem Menübefehl Suchen zugeordnet)
$vpCurTable:=Current form table
QUERY($vpCurTable->)

` M_Sortieren (dem Menübefehl Sortieren zugeordnet)
$vpCurTable:=Current form table
ORDER BY($vpCurTable->)
```

Sie können natürlich auch andere Befehle verwenden, wie **PRINT SELECTION**, **QR REPORT**, um andere "Standard" Optionen im Menü der Runtime-Umgebung nachzubauen, die Sie beim Anzeigen oder Ändern einer Auswahl benötigen. Durch die Funktion **Current form table** sind diese Methoden generisch, d.h. die entsprechende Menüleiste kann jedem Ausgabeformular einer beliebigen Tabelle zugewiesen werden.

Displayed line number

Displayed line number -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Nummer der angezeigten Zeile

Beschreibung

Die Funktion **Displayed line number** funktioniert nur mit dem Formularereignis *On display detail*. Sie gibt die Nummer der Zeile in Bearbeitung an, während die Liste der Datensätze oder Listbox-Zeilen auf dem Bildschirm angezeigt wird. Wird **Displayed line number** in einem anderen Zusammenhang als Anzeigen einer Liste bzw. Listbox aufgerufen, gibt sie 0 (Null) zurück.

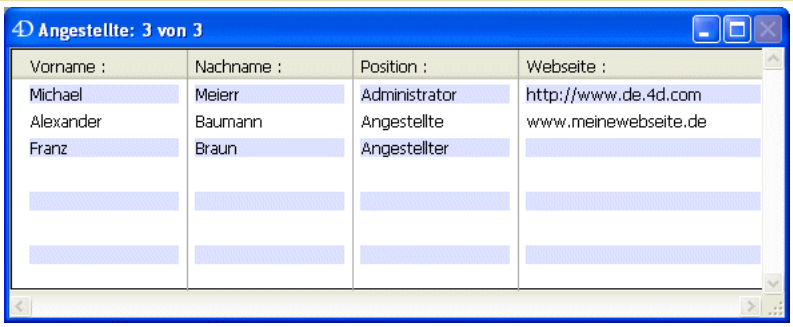
Ist bei einer Datensatzliste die angezeigte Zeile nicht leer, d.h. mit einem Datensatz verknüpft, ist der zurückgegebene Wert identisch mit dem Wert, den die Funktion **Selected record number** zurückgibt.

Displayed line number startet wie bei **Selected record number** bei 1. Diese Funktion ist hilfreich, wenn Sie jede Zeile einer Liste im Formular oder eine Listbox bearbeiten möchten, die auf dem Bildschirm erscheint, inkl. leerer Zeilen.

Beispiel

Die folgende Anweisung setzt für eine Liste im Formular, die auf dem Bildschirm erscheint, eine wechselnde Farbe. Das ist sogar für Zeilen ohne Datensätze möglich.

```
  ` Formularmethode Liste
If(Form event=On Display Detail)
   If(Displayed line number% 2=0)
   `Schwarz auf Weiss für Text in Zeilen mit gerader Nummer
      OBJECT SET RGB COLORS([Table 1]Field1;-1;0x00FFFFFF);
   Else
   `Schwarz auf Hellblau für Text in Zeilen mit ungerader Nummer
      OBJECT SET RGB COLORS([Table 1]Field1;-1;0x00E0E0FF)
   End if
End if
```



End selection

End selection {(Tabellenname)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe
Funktionsergebnis	Boolean	⇐	Ja (TRUE) oder Nein (FALSE)

Beschreibung

Die Funktion **End selection** gibt *TRUE* zurück, wenn **NEXT RECORD** über den letzten Datensatz der aktuellen Auswahl der Tabelle *Tabellenname* hinaus gelesen hat. Ist die aktuelle Auswahl leer, gibt sie den Wert *TRUE* zurück.

Mit den Befehlen **FIRST RECORD**, **LAST RECORD** oder **GOTO SELECTED RECORD** können Sie dann wieder einen aktuellen Datensatz bestimmen, ohne die Auswahl verändern zu müssen.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich die Funktion auf die Haupttabelle.

Wird der letzte Fußteil eines Berichts gedruckt, gibt **End selection** ebenfalls *TRUE* zurück. Mit folgendem Code können Sie den letzten Fußteil prüfen und einen speziellen Fußteil für die letzte Seite drucken:

```
  ` Formularmethode als Ausgabeformular für einen Summenbericht
$vpFormTable:=Current form table
Case of
  ` ...
   :(Form event=On Printing Footer)
  ` Es wird gleich ein Fußteil gedruckt
      If(End selection($vpFormTable->))
  ` Code für den letzten Fußteil
      Else
  ` Code für einen Fußteil
      End if
End case
```

Beispiel

Diese Formularmethode wird während dem Drucken eines Berichts verwendet. Sie setzt die Variable *vFooter*, um den Fußteil auf der letzten Seite zu drucken:

```
  ` Formularmethode [Finances];"Übersicht"
Case of
  ` ...
   :(Form event=On Printing Footer)
      If(End selection([Finances]))
         vFooter:="©1998 Acme Corp."  ` Setze Fußteil für die letzte Seite
      Else
         vFooter:=""  ` Lösche Fußteil für alle anderen Seiten
      End if
End case
```

FIRST RECORD

FIRST RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe

Beschreibung

Der Befehl **FIRST RECORD** bestimmt den ersten Datensatz der aktuellen Auswahl der Tabelle *Tabellenname* als aktuellen Datensatz und lädt ihn in den Hauptspeicher. Alle Befehle für Suche, Auswahl und Sortieren machen auch den ersten Datensatz zum aktuellen Datensatz.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich **FIRST RECORD** auf die Haupttabelle.

Ist kein Datensatz in der Auswahl oder ist der aktuelle Datensatz bereits der erste, wird **FIRST RECORD** nicht ausgeführt.

FIRST RECORD wird oft nach dem Befehl **USE SET** eingesetzt, damit das Durchlaufen einer Auswahl mit dem ersten Datensatz beginnt. Sie können ihn aber auch von einer Unterroutine aus aufrufen, wenn Sie nicht sicher sind, ob der aktuelle Datensatz auch der erste ist.

Beispiel

Folgendes Beispiel macht den ersten Datensatz der Tabelle [Customers] zum ersten Datensatz:

```
FIRST RECORD([Customers])
```

GET HIGHLIGHTED RECORDS

GET HIGHLIGHTED RECORDS ({Tabellenname ;} Mengenname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, in der die markierten Datensätze gelesen werden. Ohne Angabe Tabelle des aktuellen Formulars
Mengenname	String	➡	Menge, in der die markierten Datensätze gespeichert werden.

Beschreibung

Der Befehl **GET HIGHLIGHTED RECORDS** speichert in *Mengenname* die markierten Datensätze (z.B. die von einem Benutzer im Listenformular markierten Datensätze) in *Tabellenname*. Ist der Parameter *Tabellenname* nicht angegeben, wird die Tabelle des aktuellen Formulars oder Unterformulars verwendet.

Im Anwendungsmodus oder beim Ausführen der Befehle **DISPLAY SELECTION** /**MODIFY SELECTION** lässt sich dieser Befehl ersetzen, und zwar durch Aufruf der Systemmenge UserSet. Diese wird von 4D automatisch verwaltet. Da **GET HIGHLIGHTED RECORDS** die Tabelle nimmt, welche markierte Datensätze empfängt, kann der Befehl auch Datensatzauswahlen in Unterformularen verwalten. In diesem Fall kann die Auswahl im Unterformular auch aus anderen Tabellen stammen. Weitere Informationen dazu finden Sie im Kapitel **Mengen**.

GET HIGHLIGHTED RECORDS lässt sich auch in einem Kontext ohne Formular aufrufen, dann ist die zurückgegebene Menge jedoch leer.
Die in *Mengenname* definierte Menge kann vom Typ lokal/Client, Prozess oder Interprozess sein.

Hinweis: In eingebundenen Unterformularen gibt **GET HIGHLIGHTED RECORDS** eine leere Menge zurück, wenn Unterformulare nicht als Eigenschaft **Auswahlmodus Mehrfach** haben. In diesem Fall müssen Sie die Funktion **Selected record number** verwenden, um die gewählte Zeile herauszufinden.

Beispiel

Nachfolgende Methode zeigt an, wieviel Datensätze im Unterformular ausgewählt sind, das die Datensätze der Tabelle [CDs] anzeigt:

```
GET HIGHLIGHTED RECORDS([CDs];"$highlight")
ALERT(String(Records in set("$highlight"))"+"selected records.")
CLEAR SET("$highlight")
```

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable auf OK gesetzt, sonst auf 0 (Null).

GOTO SELECTED RECORD

GOTO SELECTED RECORD ({Tabellenname ;} AbsoluteNr)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe
AbsoluteNr	Lange Ganzzahl	⇒	Position des ausgewählten Datensatzes

Beschreibung

Der Befehl **GOTO SELECTED RECORD** bestimmt den Datensatz mit Nummer *AbsoluteNr* der aktuellen Auswahl der Tabelle *Tabellenname* als aktuellen Datensatz für den laufenden Prozess. *AbsoluteNr* ist nicht die Nummer, die durch die Funktion **Record number** zurückgegeben wird. Er gibt die Position des Datensatzes in der aktuellen Auswahl an. Sie hängt davon ab, wie die Auswahl erstellt und sortiert wurde.

GOTO SELECTED RECORD führt nichts aus, wenn

- In der aktuellen Auswahl keine Datensätze vorhanden sind
- *AbsoluteNr* größer als die Zahl der ausgewählten Datensätze ist
- *AbsoluteNr* bereits der aktuelle Datensatz ist

Übergeben Sie 0 (Null) in *AbsoluteNr*, gibt es in *Tabellenname* nicht länger einen aktuellen Datensatz. Ist der Auswahlmodus "single" gewählt, können Sie alle Datensätze in einer Liste abwählen. Das ist besonders hilfreich bei eingebundenen Unterformularen.

Beispiel

Folgendes Beispiel lädt Daten von Datenfeldern aus der Tabelle [People] in ein Array mit Namen *atNames*. Das Array vom Typ Lange Ganzzahl mit Namen *alRecNum* erhält die Nummern dieser Datensätze. Beide Tabellen werden dann sortiert. Diese Tabellen können anschließend benutzt werden, um die Datensätze der Auswahl anzusprechen:

```
  ` Erstelle eine Auswahl für die Tabelle [People]
  ` ...
  ` Erhalte die Namen
SELECTION TO ARRAY([People]Last Name;atNames)
  `Erstelle ein Array für die ausgewählten Datensatznummern
$vlNbRecords:=Size of array(atNames)
ARRAY LONGINT(alRecNum;$vlNbRecords)
For($vlRecord;1;$vlNbRecords)
   alRecNum{$vlRecord}:=$vlRecord
End for
  ` Sortiere beide Arrays in alphabetischer Reihenfolge
SORT ARRAY(atNames;alRecNum;>)
```

Wird das Array *atNames* in einem rollbaren Bereich dargestellt, kann der Benutzer eines der Elemente im Array anklicken. Da beide Arrays synchron sortiert sind, erhält der Name im Array dann die dazugehörige Nummer.

Folgende Objektmethode für *atNames* wählt den richtigen Datensatz in der Auswahl [People] nach dem im rollbaren Bereich ausgewählten Namen:

```
Case of
   :(Form event=On Clicked)
      If(atNames#0)
         GOTO SELECTED RECORD(alRecNum{atNames})
      End if
End case
```

HIGHLIGHT RECORDS

HIGHLIGHT RECORDS ({Tabellenname }{;}{ Mengenname {; *}})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle, in der die Datensätze hervorgehoben werden. Ohne Angabe, Tabelle des aktuellen Formulars
Mengenname	String	⇒	Menge der hervorzuhebenden Datensätze, ohne Angabe Userset
*	Operator	⇒	Deaktiviert automatisches Scrollen der Liste

Beschreibung

Der Befehl **HIGHLIGHT RECORDS** hebt Datensätze in einem Ausgabeformular hervor. Diese Operation ist identisch zur manuellen Auswahl der Datensätze im Ausgabemodus mit der Maus oder über die Tastenkombinationen **Shift+Click** oder **Ctrl+Click** unter Windows bzw. **Befehl+Click** auf Macintosh. Die aktuelle Auswahl wird nicht geändert.

Hinweis: Die Menge der ausgewählten Datensätze wird nach Neuzeichnen der Datensätze aktualisiert; das passiert aber erst nach Ausführen der kompletten aufrufenden Methode — und nicht bereits unmittelbar nach Ausführen von **HIGHLIGHT RECORDS**.

Mit dem Parameter *Tabellenname* bestimmen Sie die Tabelle, in der die Datensätze hervorgehoben werden. Sie können insbesondere die Datensätze von eingebundenen Unterformularen – die nicht zur aktuellen Tabelle gehören – hervorheben.

- Übergeben Sie in *Mengenname* einen gültigen Mengennamen, gilt der Befehl für die Datensätze, definiert in *Tabellenname*.
- Geben Sie den Parameter *Mengenname* nicht an, markiert der Befehl nur die Datensätze der aktuellen Menge *UserSet*. Diese Menge wird nur in der Anwendungsumgebung verwaltet oder bei Aufruf der Befehle **DISPLAY SELECTION** /**MODIFY SELECTION**. Wollen Sie die Datensätze eines Unterformulars markieren, müssen Sie einen Tabellennamen und einen Mengennamen übergeben. Weitere Informationen dazu finden Sie im Kapitel **Mengen**.

Übergeben Sie den optionalen Parameter *, wird die automatische Scroll-Funktion der Liste deaktiviert, wenn die hervorgehobenen Datensätze nicht sichtbar sind. Diese Funktionsweise ermöglicht die angepasste Verwaltung von Scrollen mit dem Befehl **OBJECT SET SCROLL POSITION**.

Hinweis: Der Befehl **HIGHLIGHT RECORDS** führt in eingebundenen Unterformularen nichts aus, wenn für das Unterformular nicht die Eigenschaft **Mehrfach** des Auswahlmodus ausgewählt ist. Verwenden Sie in diesem Fall den Befehl **GOTO SELECTED RECORD**.

Beispiel

Sie möchten, dass der Benutzer in einem Ausgabeformular, das über den Befehl **MODIFY SELECTION** angezeigt wird, Suchläufe ohne Ändern der aktuellen Auswahl durchführen kann. Dazu richten Sie im Formular eine Schaltfläche **Suchen** ein mit folgender Methode:

```
SET QUERY DESTINATION(Into set;"UserSet")
QUERY
SET QUERY DESTINATION(Into current selection)
HIGHLIGHT RECORDS
```

Klickt der Benutzer auf diese Schaltfläche, erscheint der Standarddialog für Suchen. Hat er die Suche bestätigt, werden die gefundenen Datensätze hervorgehoben. Die aktuelle Auswahl wird nicht verändert.

LAST RECORD

LAST RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe

Beschreibung

Der Befehl **LAST RECORD** bestimmt den letzten Datensatz der aktuellen Auswahl aus der Tabelle *Tabellenname* als aktuellen Datensatz und lädt ihn in den Hauptspeicher. Ist die aktuelle Auswahl leer oder ist der letzte Datensatz in der Auswahl bereits der aktuelle Datensatz, hat er keine Auswirkung.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich der Befehl auf die Haupttabelle.

Beispiel

Folgendes Beispiel macht den letzten Datensatz der Tabelle [People] zum aktuellen Datensatz:

```
LAST RECORD([People])
```

MOBILE Return selection

MOBILE Return selection (Tabellenname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Auswahl zurückgegeben werden soll
Funktionsergebnis	Objekt	➲	Für Wakanda kompatible Auswahl

Beschreibung

Die Funktion **MOBILE Return selection** gibt ein JSON *Objekt* mit der aktuellen Auswahl von *Tabellenname* zurück, die in eine für Wakanda geeignete Entity Collection umgewandelt wurde.

Diese Funktion wird im Rahmen einer 4D Mobile Verbindung aufgerufen, normalerweise zwischen Ihrer 4D Applikation und einer Wakanda Applikation (via REST). Ist eine 4D Mobile Verbindung mit passenden Zugriffsrechten eingerichtet, kann eine Wakanda Applikation eine 4D Projektmethode ausführen, die im Parameter *$0* einen Wert zurückgibt.

Die Funktion **MOBILE Return selection** ermöglicht, in *$0* die aktuelle Datensatzauswahl von *Tabellenname* in Form eines Objekts *Entity Collection* im JSON Format zurückzugeben. Dieses Objekt ist kompatibel mit Entity Collections in Wakanda, die eine Auswahl von Datensätzen enthalten (z.B. von *entities*).

Beachten Sie, dass für 4D Mobile Zugriffe in Ihrer 4D Anwendung spezifische Einstellungen erforderlich sind:

- Der Web Server muss gestartet sein,
- In den Datenbank-Eigenschaften muss die Option "Aktiviere 4D Mobile Service" markiert sein,
- Sie müssen eine gültige Lizenz haben,
- Für Tabellen und Felder muss die Option "Mit 4D Mobile Service veröffentlichen" markiert sein (ist standardmäßig markiert).
- Für aufgerufene Methoden muss die Option "Verfügbar per 4D Mobile Aufruf" markiert sein (ist nicht standardmäßig markiert).

Beachten Sie, dass Sie in *Tabellenname* jede gültige Tabelle der Anwendung übergeben können. Das muss nicht zwingend die Tabelle sein, der die Projektmethode in den Eigenschaften zugewiesen wurde. Dieser Parameter wird nur auf der Wakanda Seite zur Definition der Objekte verwendet, für die sich die Methode aufrufen lässt.

Weitere Informationen dazu finden Sie in der Dokumentation zu **4D Mobile**.

Beispiel

Die aktuelle Auswahl zu einer Suche in der Tabelle [Countries] in einem Grid in Wakanda anzeigen.

Sie schreiben folgende 4D Methode:

```
//Projektmethode FindCountries
//FindCountries( string ) -> object

C_TEXT($1)
C_OBJECT($0)
QUERY([Countries];[Countries]ShortName=$1+"@")
$0:=MOBILE Return selection([Countries])
```

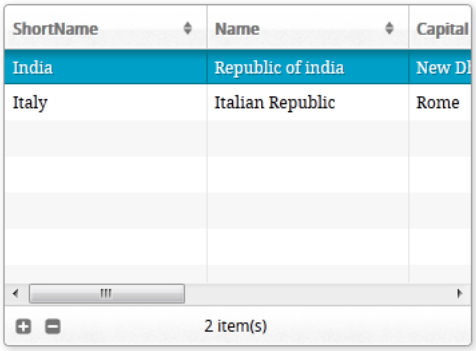
Die zurückgegebene Auswahl lässt sich direkt in Wakanda als gültige Collection verwenden.

Im Wakanda Server Model, das via 4D Mobile mit 4D verbunden ist, haben Sie eine Seite mit einem Grid erstellt, das an die Tabelle 4D Countries gebunden ist. Standardmäßig werden alle Entities aus der 4D Tabelle angezeigt:

Der Code für die Schaltfläche lautet:

```
button1.click = function button1_click (event)  {  sources.countries.FindCountries("i", {  //die 4D Methode aufrufen, "i" wird als $1 übergeben
onSuccess:function(coll){  //Callback Funktion (asynchron), empfängt $0 als Parameter  sources.countries.setEntityCollection(coll.result); //die
aktuelle Entity Collection  // mit der im Objekt coll.result ersetzen  }  });  };
```

Hier als Ergebnis das upgedatete Grid:



Find Countries

MODIFY SELECTION

MODIFY SELECTION ({Tabellenname}{; AuswModus}{; EingListe}{; *}{; *})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Anzeigen oder Ändern, Haupttabelle ohne Angabe
AuswModus	Lange Ganzzahl	⇒	Auswahlmodus
EingListe	Boolean	⇒	Option Eingabe in Liste erlauben
*		⇒	Ausgabeformular für einzelnen Datensatz verwenden und Rollbalken im Eingabeformular ausblenden
*		⇒	Rollbalken im Eingabeformular anzeigen (Überschreibt die 2. Option des 1. Parameters *)

Beschreibung

Der Befehl **MODIFY SELECTION** führt fast dieselben Operationen aus wie **DISPLAY SELECTION**. Es gibt folgende Unterschiede:

1. **DISPLAY SELECTION** und **MODIFY SELECTION** ermöglichen bei Doppelklick auf einen Datensatz, die aktuell gewählten Datensätze im Ausgabe- bzw. Eingabeformular anzuzeigen. Mit dem Befehl **MODIFY SELECTION** können Sie die Felder des Datensatzes im Eingabeformular auch ändern, wenn Sie darauf doppelklicken. Der Datensatz kann jedoch nur geändert werden, wenn er nicht schon von einem anderen Prozess bzw. Benutzer verwendet wird oder wenn der Modus *EingListe* aktiv ist.
2. **DISPLAY SELECTION** lädt die Datensätze im Nur-Lesen Modus in den aktuellen Prozess, d.h. sie sind in den anderen Prozessen nicht im Schreibmodus gesperrt. **MODIFY SELECTION** setzt alle Datensätze der Auswahl in den Lese/Schreib-Modus, d.h. sie sind in anderen Prozessen automatisch im Schreibmodus gesperrt. **MODIFY SELECTION** gibt die Datensätze nach der Ausführung wieder frei.

NEXT RECORD

NEXT RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe

Beschreibung

Der Befehl **NEXT RECORD** setzt den aktuellen Datensatzzeiger auf den nächsten Datensatz in der aktuellen Auswahl von *Tabellenname* für den aktuellen Prozess. Ist die aktuelle Auswahl leer oder haben **Before selection** bzw. **End selection** den Wert *TRUE*, hat der Befehl keine Auswirkung.

Setzt **NEXT RECORD** den aktuellen Datensatzzeiger nach die aktuelle Auswahl, gibt es keinen aktuellen Datensatz. In diesem Fall gibt die Funktion **End selection** den Wert *TRUE* zurück. Mit den Befehlen **FIRST RECORD**, **LAST RECORD** oder **GOTO SELECTED RECORD** können Sie den aktuellen Datensatzzeiger wieder in die aktuelle Auswahl setzen.

Beispiel

Siehe Beispiel zum Befehl **DISPLAY RECORD**.

ONE RECORD SELECT

ONE RECORD SELECT {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe

Beschreibung

ONE RECORD SELECT erstellt für den laufenden Prozess eine neue Auswahl, die nur den aktuellen Datensatz aus *Tabellenname* enthält. Ist kein aktueller Datensatz vorhanden oder wurde der aktuelle Datensatz nicht in den Speicher geladen (Sonderfall), wird der Befehl nicht ausgeführt.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich **ONE RECORD SELECT** auf die Haupttabelle.

Hinweis

Mit diesem Befehl ließ sich ein Datensatz, der beim Ändern der Auswahl in den Datensatzstapel gelegt bzw. daraus entnommen wurde, wieder in die Auswahl zurückholen. Mit **SET QUERY DESTINATION** können Sie eine Suche durchführen, ohne die Auswahl oder den aktuellen Datensatz der Tabelle zu ändern. Deshalb hat **ONE RECORD SELECT** heute weniger Bedeutung, außer Sie wollen die Auswahl einer Tabelle auf den aktuellen Datensatz reduzieren.

PREVIOUS RECORD

PREVIOUS RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe

Beschreibung

Der Befehl **PREVIOUS RECORD** setzt den aktuellen Datensatzzeiger auf den vorherigen Datensatz in in der aktuellen Auswahl von *Tabellenname* für den aktuellen Prozess. Ist die aktuelle Auswahl leer oder haben **Before selection** bzw. **End selection** den Wert *TRUE*, hat **NEXT RECORD** keine Auswirkung.

Setzt **PREVIOUS RECORD** den aktuellen Datensatzzeiger vor die aktuelle Auswahl, gibt es keinen aktuellen Datensatz. In diesem Fall gibt die Funktion **End selection** den Wert *TRUE* zurück. Mit den Befehlen **FIRST RECORD**, **LAST RECORD** oder **GOTO SELECTED RECORD** können Sie den aktuellen Datensatzzeiger zurück in die aktuelle Auswahl setzen.

Records in selection

Records in selection {(Tabellenname)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe
Funktionsergebnis	Lange Ganzzahl	⇐	Datensätze in Auswahl der Tabelle

Beschreibung

Die Funktion **Records in selection** gibt die Zahl der ausgewählten Datensätze der Tabelle *Tabellenname* für den laufenden Prozess zurück.
Im Gegensatz dazu gibt die Funktion **Records in table** die Gesamtanzahl der Datensätze in der Tabelle zurück.

Beispiel

Folgendes Beispiel zeigt eine Schleife, die häufig verwendet wird, um alle Datensätze in einer Auswahl zu durchlaufen. Die gleiche Aktion können Sie auch mit dem Befehl **APPLY TO SELECTION** durchführen:

```
FIRST RECORD([People]) ` Starte bei erstem Datensatz in der Auswahl
For($vlRecord;1;Records in selection([People])) ` Durchlaufe einmal pro Datensatz
   Do Something ` Führe etwas aus mit diesem Datensatz
   NEXT RECORD([People]) ` Gehe zum nächsten Datensatz
End for
```

REDUCE SELECTION

REDUCE SELECTION ({Tabellenname ;} AnzDatensätze)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe
AnzDatensätze	Lange Ganzzahl	⇒	Anzahl der gewünschten Datensätze

Beschreibung

Der Befehl **REDUCE SELECTION** erstellt eine neue Auswahl der Datensätze für *Tabellenname*. Er verringert die Anzahl der Datensätze in der aktuellen Auswahl von *Tabellenname* auf die in *AnzDatensätze* angegebene Anzahl. **REDUCE SELECTION** wird auf die aktuelle Auswahl von *Tabellenname* im aktuellen Prozess angewendet. Der erste Datensatz der neuen Auswahl wird in den Hauptspeicher geladen. Er wird der aktuelle Datensatz.

Tabellenname ist optional. Wird der Parameter nicht angegeben, wird der Befehl auf die Haupttabelle angewandt.

Hinweis:
Nach Ausführen der Anweisung REDUCE SELECTION(Tabellenname ;0) gibt es in *Tabellenname* keine Auswahl mehr und auch keinen aktuellen Datensatz.

Beispiel

Folgendes Beispiel findet zuerst die richtige Statistik zu einer weltweiten Umfrage unter den Händlern in über 20 Ländern. Unter den ersten 100 besten Händlern der Welt werden jeweils die 3 besten Händler jedes Landes mit einem Umsatz über $50.000 mit einem Preis ausgezeichnet. Diese komplexe Anfrage können Sie mit indizierten Suchläufen in wenigen Zeilen Code ausführen:

```
CREATE EMPTY SET([Dealers];"Gewinner") ` Erstelle leere Menge
SCAN INDEX([Dealers]Sales amount;100;<) ` Wähle vom Ende des Index aus
CREATE SET([Dealers];"100 beste Händler") ` Setze ausgewählte Datensätze in Menge
For($Country;1;Records in table([Countries])) ` Suche für jedes Land die Händler
   QUERY([Dealers];[Dealers]Country=[Countries]Name;*) ` ...mit Umsatz über $50.000
   QUERY(&;[Dealers];[Dealers]Sales amount>=50000)
   CREATE SET([Dealers];"GewinnerHändler") ` Setze diese in eine Menge
 ` Sie sollten in der Gruppe der 100 besten Händler sein
   INTERSECTION("GewinnerHändler";"100 beste Händler";"GewinnerHändler")
   USE SET("GewinnerHändler") ` Potentielle Gewinner des Landes
 ` Sortiere sie nach Umsatz in absteigender Reihenfolge
   ORDER BY([Dealers];[Dealers]Sales amount;<)
   REDUCE SELECTION([Dealers];3) ` Nimm die 3 besten Händler
   CREATE SET([Dealers];"GewinnerHändler") ` Gewinner des Landes
 ` Lege Sie in die Liste der weltweiten Gewinner
   UNION("GewinnerHändler";"DieGewinner";"DieGewinner")
End for
CLEAR SET("100 beste Händler") ` Diese Menge wird nicht mehr benötigt
CLEAR SET("GewinnerHändler") ` Diese Menge wird nicht mehr benötigt
USE SET("DieGewinner") ` Hier sind die Gewinner
CLEAR SET("DieGewinner") ` Diese Menge wird nicht mehr benötigt
OUTPUT FORM([Dealers];"Brief für Preis") ` Wähle den Brief
PRINT SELECTION([Dealers]) ` Drucke die Briefe
```

SCAN INDEX

SCAN INDEX (Feldname ; AnzDatensätze {; > oder <})

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Indiziertes Datenfeld
AnzDatensätze	Lange Ganzzahl	➡	Anzahl der ausgewählten Datensätze
> oder <	Operator	➡	> erste Datensätze, < letzte Datensätze sortiert nach dem Index

Beschreibung

Der Befehl **SCAN INDEX** wählt die ersten oder die letzten *AnzDatensätze* der Tabelle, sortiert nach dem Index. Dieser Befehl ist sehr schnell, da er nur auf indizierte Datenfelder angewandt wird.

- Bei > werden die ersten *AnzDatensätze* zurückgegeben.
- Bei < werden die letzten *AnzDatensätze* zurückgegeben.

Hinweis: Die erhaltene Auswahl wird nicht sortiert.

Dieser Befehl funktioniert nur bei indizierten Feldern. Er ändert die aktuelle Auswahl der Tabelle für den laufenden Prozess und lädt den ersten Datensatz der Auswahl als den aktuellen Datensatz.
Geben Sie mehr Datensätze an, als die Tabelle enthält, gibt **SCAN INDEX** alle Datensätze zurück.

Hinweis: Dieser Befehl unterstützt keine Felder vom Typ Objekt.

Beispiel

Folgendes Beispiel schickt Briefe an die 50 schlechtesten und dann an die 50 besten Kunden:

```
SCAN INDEX([Customers]TotalDue;50;<) ` Erhalte die 50 schlechtesten Kunden
ORDER BY([Customers]PLZ;>) ` Sortiere nach Postleitzahl
FORM SET OUTPUT([Customers];"Schimpfbrief")
PRINT SELECTION([Customers]) ` Drucke die Briefe
SCAN INDEX([Customers]TotalDue;50;>) ` Erhalte die 50 besten Kunden
ORDER BY([Customers]PLZ;>) ` Sortiere nach Postleitzahl
FORM SET OUTPUT([Customers];"Dankesbrief")
PRINT SELECTION([Customers]) ` Drucke die Briefe
```

Selected record number

Selected record number {(Tabellenname)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	→	Tabelle zum Bearbeiten oder Haupttabelle ohne Angabe
Funktionsergebnis	Lange Ganzzahl	↩	Ausgewählte Nummer des aktuellen Datensatzes

Beschreibung

Die Funktion **Selected record number** gibt für den laufenden Prozess die ausgewählte Nummer des aktuellen Datensatzes in der aktuellen Auswahl der Tabelle *Tabellenname* zurück.

Ist die Auswahl nicht leer und gibt es einen aktuellen Datensatz in der Auswahl, gibt die Funktion einen Wert zwischen *1* und **Records in selection** zurück. Ist die Auswahl leer und gibt es keinen aktuellen Datensatz in der Auswahl, gibt sie den Wert 0 (Null) zurück.

Die ausgewählte Nummer unterscheidet sich von der Nummer, die **Record number** zurückgibt. Hier wird die absolute Nummer zurückgegeben. Die ausgewählte Nummer richtet sich nach der aktuellen Auswahl und nach dem aktuellen Datensatz.

Beispiel

Folgendes Beispiel sichert die Nummer des ausgewählten aktuellen Datensatzes in einer Variablen:

```
CurSelRecNum:=Selected record number([People])
  ` Erhalte ausgewählte Datensatznummer
```

TRUNCATE TABLE

TRUNCATE TABLE {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle, in der alle Datensätze auf einmal gelöscht werden, oder Haupttabelle ohne Angabe

Beschreibung

Der Befehl **TRUNCATE TABLE** löscht alle Datensätze in *Tabellenname* auf einmal. Nach Aufrufen dieses Befehls gibt es weder eine aktuelle Auswahl noch einen aktuellen Datensatz.

Dieser Befehl arbeitet ähnlich wie die Befehlsfolge **ALL RECORDS** / **DELETE SELECTION**; es gibt jedoch folgende Unterschiede:

- Es wird kein Trigger aufgerufen.
- Die referentielle Integrität der Daten wird nicht geprüft.
- Während der Ausführung dieses Befehls darf keine Transaktion ablaufen. In diesem Fall führt der Befehl nichts aus und die Systemvariable **OK** wird auf 0 (Null) gesetzt.
- Sind ein oder mehrere Datensätze durch einen anderen Prozess gesperrt, schlägt der Befehl fehl: Es wird ein Fehler generiert und die Systemvariable OK wird auf 0 (Null) gesetzt. Es wird keine Systemmenge **LockedSet** erstellt.
- Ist *Tabellenname* bereits leer, führt **TRUNCATE TABLE** nichts aus und setzt die Variable OK auf 1.
- Ist *Tabellenname* im Nur-Lesen Status, führt **TRUNCATE TABLE** nichts aus und setzt die Variable OK auf 0.
- Gibt es ein Logbuch, wird die Operation darin gespeichert.

Verwenden Sie den Befehl **TRUNCATE TABLE** mit Bedacht, da er in bestimmten Fällen nachhaltige Wirkung zeigt, wie z.B. rasches Löschen temporärer Dateien.

Hinweis: Dieser Befehl ähnelt im Konzept und in der Funktionsweise dem SQL Befehl TRUNCATE (TABLE).

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

Backup

- Datenbankmethode On Backup Shutdown
- Datenbankmethode 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*

Datenbankmethode On Backup Shutdown

Die **Datenbankmethode On Backup Shutdown** wird immer aufgerufen, wenn ein Backup der Datenbank endet. Gründe dafür können das Ende der Kopie, Unterbrechung durch den Benutzer oder ein Fehler sein.
Das betrifft alle 4D Umgebungen: 4D im lokalen und im remote Modus, 4D Server, sowie alle 4D Anwendungen mit integrierter 4D Volume Desktop.

Mit der **Datenbankmethode On Backup Shutdown** können Sie prüfen, ob das Backup korrekt durchgeführt wurde. Sie empfängt im Parameter $1 einen Wert, der den Status des abgeschlossenen Backup anzeigt:

- Bei korrekt ausgeführtem Backup hat $1 den Wert 0 (Null).
- Wurde das Backup durch den Benutzer oder aufgrund eines Fehlers unterbrochen, enthält $1 einen anderen Wert als 0 (Null). Wurde das Backup von der **Datenbankmethode On Backup Startup** gestoppt ($0 # 0), enthält $1 den aktuell im Parameter $0 zurückgegebenen Wert. So können Sie ein eigenes Fehlerverwaltungssystem einbauen.
- Wurde das Backup aufgrund eines Fehlers gestoppt, wird in $1 die Fehlernummer zurückgegeben.

In allen Fällen erhalten Sie Informationen zum aufgetretenen Fehler über den Befehl **GET BACKUP INFORMATION**.

Hinweis: Sie müssen den Parameter $1 (Lange Ganzzahl) in der Datenbankmethode deklarieren:

```
C_LONGINT($1)
```

Bitte beachten Sie, dass bei einem Fehler während dem Backup (Festplatte voll, o.ä) die Fehlermeldung nur auf dem 4D Server Monitor oder im MSC angezeigt und in das Backup Logbuch kopiert wird. Es erscheint kein Dialogfenster mit der Meldung und die Variable *Error* wird nicht verändert. Wenn Sie den Administrator informieren wollen, dass ein Fehler aufgetreten ist, insbesondere bei einer Anwendung im Client/Server Modus, müssen Sie die **Datenbankmethode On Backup Shutdown** einsetzen.

Datenbankmethode On Backup Startup

Die **Datenbankmethode On Backup Startup** wird immer aufgerufen, wenn ein Backup der Datenbank gerade startet. Das Backup kann manuell, automatisch festgelegt oder über den Befehl **BACKUP** aufgerufen werden.
Das betrifft alle 4D Umgebungen: 4D in lokalen und im remote Modus, 4D Server und Datenbanken mit einkompilierter 4D Volume Desktop.

Mit der **Datenbankmethode On Backup Startup** können Sie prüfen, ob das Backup gestartet ist. Sie gibt im Parameter $0 einen Wert zurück, der das Backup zulässt oder verweigert.

- Bei $0 = 0 lässt sich das Backup starten.
- Bei $0 # 0 ist kein Backup zugelassen. Der Vorgang wird abgebrochen und ein Fehler zurückgegeben. Sie können den Fehler über den Befehl **GET BACKUP INFORMATION** erhalten.

Über diese Datenbankmethode können Sie die Bedingungen der Backup-Ausführung prüfen, z.B. Benutzer, Datum des letzten Backup.

Hinweis: Sie müssen den Parameter $0 (Lange Ganzzahl) in der Datenbankmethode deklarieren:

```
C_LONGINT($0)
```

BACKUP

BACKUP
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **BACKUP** startet das Backup der Datenbank über die aktuellen Einstellungen des Backup. Es erscheint kein Dialog zum Bestätigen, ein Ablaufbalken auf dem Bildschirm zeigt jedoch das Fortschreiten der Operation an.

Backup-Einstellungen setzen Sie in den Datenbank-Eigenschaften (Einstellungen in 4D v11 SQL). Sie werden auch im Ordner Backup.XML gespeichert, der im Unterordner *Preferences/backup* der Datenbank liegt.

BACKUP ruft die **Datenbankmethode On Backup Startup** zu Beginn der Ausführung und die **Datenbankmethode On Backup Shutdown** am Ende der Ausführung auf. Deshalb sollte der Befehl nicht über diese Datenbankmethoden aufgerufen werden.

4D Server: Bei Aufruf über einen Client-Rechner wird der Befehl als Serverprozedur gewertet, er wird weiterhin auf dem Server ausgeführt.

Systemvariablen und Mengen

Bei korrekt ausgeführtem Backup hat die Systemvariable OK den Wert 1, sonst den Wert 0 (Null).

Fehlerverwaltung

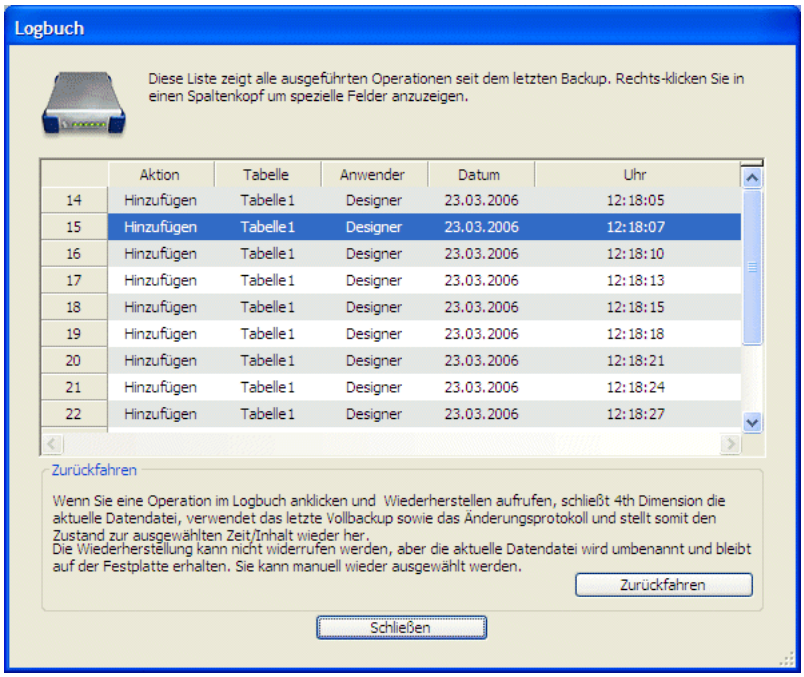
Tritt während dem Backup ein Fehler auf, wird die Information zum Fehler in das Backup Logbuch geschrieben und nur der Fehler auf oberster Ebene wird an die **Datenbankmethode On Backup Shutdown** gesendet. Deshalb ist es besonders wichtig, diese Datenbankmethode zu verwenden, um Fehler während dem Backup per Programmierung zu verwalten.

CHECK LOG FILE

CHECK LOG FILE
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **CHECK LOG FILE** zeigt das Dialogfenster zum Sichten des aktuellen Logbuchs der Datenbank. Sie können auch über das Maintenance- und Security-Center darauf zugreifen.:



Es enthält die Schaltfläche **Zurückfahren**, um in der Datenbank ausgeführte Operationen wieder rückgängig zu machen. Weitere Informationen dazu finden Sie im Kapitel **Maintenance und Security Center** des Handbuchs *4D Designmodus*.

Da die Rollback-Funktion recht leistungsstark ist, empfiehlt es sich, den Zugriff auf den Befehl **CHECK LOG FILE** auf den Datenbankadministrator zu beschränken.
Dieser Befehl ist nur in 4D Anwendungen im Einzelplatz verwendbar. Er ermöglicht insbesondere den Zugriff auf die Rollback-Funktion in Anwendungen mit integrierter 4D Volume Desktop, d.h. auf Anwendungen ohne Designumgebung. Wird er im Client/Server-Betrieb aufgerufen, hat er keine Auswirkung. Es erscheint die Fehlermeldung 1421.

Fehlerverwaltung

- Wird dieser Befehl in einer Datenbankoperation ohne ein Logbuch ausgeführt, führt er nichts aus und der Fehler 1403 wird zurückgegeben.
- Wird dieser Befehl in einer Client/Server Anwendung ausgeführt, führt er nichts aus und der Fehler 1421 wird zurückgegeben.
  Sie können diese Fehler mit einer Fehlerverwaltungsmethode abfangen, die mit dem Befehl **ON ERR CALL** installiert wird.

GET BACKUP INFORMATION

GET BACKUP INFORMATION (Selector ; Info1 ; Info2)

Parameter	Typ		Beschreibung
Selector	Lange Ganzzahl	➡	Typ der gewünschten Information
Info1	Lange Ganzzahl, Datum	⬅	Wert 1 von Selektor
Info2	Zeit, String	⬅	Wert 2 von Selektor

Beschreibung

Der Befehl **GET BACKUP INFORMATION** liefert Informationen zum letzten für die Daten der Datenbank durchgeführten Backup.

In *Selector* geben Sie die Art der gewünschten Information an. Dafür können Sie eine Konstante unter dem Thema **Backup und Wiederherstellen** verwenden:

Konstante	Typ	Wert
Last Backup Date	Lange Ganzzahl	0
Last Backup Status	Lange Ganzzahl	2
Next Backup Date	Lange Ganzzahl	4

Typ und Inhalt der Parameter *Info1* und *Info2* richten sich nach dem Wert von *Selector*.

- Ist *Selector* = 0 (Datum des letzten Backup), gibt *Info1* das Datum und *Info2* die Uhrzeit des letzten Backup zurück.
- Ist *Selector* = 2 (Status des letzten Backup), gibt *Info1* die Zahl und *Info2* den Text zum Status des letzten Backup zurück.
- Ist *Selector* = 4 (Datum des nächsten Backup), gibt *Info1* das Datum und *Info2* die Uhrzeit des nächsten geplanten Backup zurück.

GET RESTORE INFORMATION

GET RESTORE INFORMATION (Selector ; Info1 ; Info2)

Parameter	Typ		Beschreibung
Selector	Lange Ganzzahl	➡	Typ der gewünschten Information
Info1	Lange Ganzzahl, Datum	⬅	Wert 1von Selektor
Info2	String, Zeit	⬅	Wert 2 von Selektor

Beschreibung

Der Befehl **GET RESTORE INFORMATION** liefert die Informationen zur letzten automatischen Wiederherstellung der Datenbank.

In *Selector* geben Sie die Art der gewünschten Information an. Dafür können Sie eine Konstante unter dem Thema **Backup und Wiederherstellen** verwenden:

Konstante	Typ	Wert
Last Restore Date	Lange Ganzzahl	0
Last Restore Status	Lange Ganzzahl	2

Typ und Inhalt der Parameter *Info1* und *Info2* richten sich nach dem Wert von *Selector*.

- Ist *Selector* = 0 (Datum der letzten Wiederherstellung), gibt *Info1* das Datum und *Info2* die Uhrzeit der letzten automatischen Wiederherstellung der Datenbank zurück.
- Ist *Selector* = 2 (Status der letzten Wiederherstellung), gibt *Info1* die Zahl und *Info2* den Text zum Status der letzten automatischen Wiederherstellung der Datenbank zurück.

Hinweis: Dieser Befehl berücksichtigt kein manuelles Wiederherstellen der Datenbank.

INTEGRATE MIRROR LOG FILE

INTEGRATE MIRROR LOG FILE (PfadName ; OperationNr {; Modus {; errObjekt}})

Parameter	Typ		Beschreibung
PfadName	Text	➡	Name oder Pfadname des zu integrierenden Logbuchs
OperationNr	Variable Zahl	➡	Nummer der zuletzt integrierten Operation
		⬅	Neue Nummer der letzten integrierten Operation oder -2 zum Integrieren der gesamten Datei
Modus	Lange Ganzzahl	➡	0=Strikter Modus (Standard), 1=auto-repair Modus
errObjekt	Object variable	⬅	Fehlende Operation(en)

Beschreibung

Vorbemerkung: Dieser Befehl arbeitet nur mit 4D Server. Er lässt sich nur über die Funktion **Execute on server** oder in einer Serverprozedur ausführen.

Der Befehl **INTEGRATE MIRROR LOG FILE** integriert das Journal, das im Parameter *PfadName* angegeben ist, in eine 4D Server Datenbank, optional nach der Operation, angegeben in *OperationNr*. Dieser Befehl erlaubt, jedes Journal in die Datenbank zu integrieren, auch wenn es nicht dem aktuellen Status der Datendatei entspricht. Er eignet sich speziell zur Verwendung im Kontext einer Spiegeldatenbank.

Hinweis: Ab 4D v14 lässt sich ein Journal als Teil einer Spiegeldatenbank verwenden. Die Option "Benutze Logbuch" lässt sich auch in den Datenbank-Eigenschaften eines 4D Server markieren, der als Spiegel-Server verwendet wird. Auf diese Weise ist das Implementieren einer Reihe verschachtelter Spiegel-Server möglich. Weitere Informationen dazu finden Sie im Abschnitt **Logischen Spiegel einrichten** des Handbuchs *4D Server*.

Dieser Befehl ersetzt, im Gegensatz zum Befehl **_o_INTEGRATE LOG FILE** am Ende der Ausführung nicht das aktuelle Journal durch das integrierte: Das aktuelle Journal der Datenbank wird weiter verwendet. So werden alle Änderungen während der Integration im aktuellen Journal gesichert.

In *PfadName* übergeben Sie einen absoluten oder relativen Pfad des Datenbankordners. Übergeben Sie einen leeren String, erscheint ein Standard-Öffnen Dialog, so dass Sie die Datei zum Integrieren wählen können. Wird dieser Dialog abgebrochen, wird keine Datei integriert und die Systemvariable OK auf 0 gesetzt.

Standardmäßig, d.h. ohne den Parameter *OperationNr*, integriert der Befehl alle Operationen des Journals.
Übergeben Sie einen Wert in *OperationNr*, beginnt die Integration mit der Operation mit der nächsthöheren Nummer. Nach der Integration wird der Wert der Variablen *OperationNr* mit der Nummer der zuletzt integrierten Operation aktualisiert. Sie müssen diese Variable sichern und können sie dann als Parameter *OperationNr* für die nächste Operation zum Integrieren verwenden. So können Sie mit **INTEGRATE MIRROR LOG FILE** nachfolgende Logbücher direkt im Anschluss integrieren.

Hinweis zur Kompatibilität: Bisher war der Parameter *OperationNr* optional; ab v15 R4 ist er zwingend, d.h. wird *OperationNr* weggelassen, wird ein Fehler generiert. Wollen Sie die bisherige Funktionsweise wiederherstellen, übergeben Sie -2 als Variablenwert des Parameters.

In *Modus* übergeben Sie den gewünschten Integrationsmodus. Sie können zwischen zwei Konstanten unter dem Thema "**Backup und Wiederherstellen**" wählen:

Konstante	Typ	Wert	Kommentar
Auto repair mode	Lange Ganzzahl	1	Verwendet flexiblen Modus mit auto-repair Aktionen und füllt den Parameter *errObjekt* (falls vorhanden).
Strict mode	Lange Ganzzahl	0	Verwendet strikten Integrationsmodus (Standard).

- Strikter Modus: Wie in bisherigen 4D Versionen stoppt die Integration, wenn ein Fehler auftritt und Sie müssen das MSC aufrufen, um den Fehler nachzuverfolgen. Dieser Modus wird standardmäßig verwendet und bietet sich in den meisten Fällen an.
- Auto repair Modus: Treten in diesem Modus nicht-kritische Fehler auf, werden sie übersprungen und die Integration läuft weiter. Ist der Parameter *errObjekt* übergeben, wird der Fehler protokolliert und kann später analysiert werden.
  Fälle für nicht-kritische Fehler sind:
  - Das Journal meldet, einen Datensatz hinzuzufügen, aber dieser Datensatz ist bereits in den Daten vorhanden.
    Aktion zum Reparieren: 4D aktualisiert den Datensatz
  - Das Journal meldet, einen Datensatz zu aktualisieren, aber dieser Datensatz existiert noch nicht.
    Aktion zum Reparieren: 4D fügt den Datensatz hinzu.
  - Das Journal meldet, einen Datensatz zu löschen, aber dieser Datensatz existiert nicht.
    Aktion zum Reparieren: 4D führt nichts aus.

Tritt eine dieser Unregelmäßigkeiten im auto-repair Modus auf, wird der betroffene Datensatz automatisch "repariert" und die entsprechende Operation wird im Parameter *errObjekt* protokolliert. Nach abgeschlossener Ausführung listet der Parameter *errObjekt* alle reparierten Datensätze. Er enthält ein Array Objekt mit Namen "operations" in folgender Form:

```
{"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"
               }
      },
      {...}
]
```


Warnung: Der auto-repair Modus darf nur in spezifischen Fällen verwendet werden, da er die 4D Kontrollfunktionen zur internen Datenintegration übergeht. Ein solcher Fall wäre z.B., wenn ein dazwischen liegendes Logbuch verloren ging oder beschädigt ist, und Sie möglichst viele Operationen wiederherstellen wollen. In diesem Modus müssen Sie besonders auf die Datenintegrität achten.

Die aktuelle Liste der verfügbaren Eigenschaften richtet sich nach der Art der Operation (z.B. Datensatz erstellen, Datensatz löschen, Datensatz ändern, Blob erstellen). Hier die Haupteigenschaften:

- *operationType*: interner Code der Operation
- *operationName*: Art der Operation, z.B. "Datensatz erstellen," "Datensatz ändern"
- *operationNumber*: interne Nummer der Operation im Logbuch
- *contextID*: ID des Ausführungskontexts
- *timeStamp*: Zeitstempel der Operation im Logbuch
- *dataLen*: interne Größe der Daten
- *recordNumber*: interne Datensatznummer
- *tableID*: interne ID der Tabelle
- *tableName*: Name der Tabelle
- *fields*: Objekt mit der Liste der Feldnummern (oder Feldnamen) mit ihren Werten. Jedes Feld mit einem geänderten Wert wird protokolliert.
  Bei Werten vom Typ Blob oder Bild wird je nach Speicherort unterschiedliche Information geliefert:
  - Beim Speichern innerhalb der Datendatei lautet die Eigenschaft "BlobID:"+ eine interne Blob Nummer, z.B.: "BlobID:1"
  - Beim Speichern außerhalb der Datendatei lautet die Eigenschaft "BlobPfad:" + Datenpfad, z.B.: "BlobPath: Table 1/Field 6/Data_EE12D091535F9748BCE62EDE972A4BA2.jpg"
- *extraData*: Daten des Benutzerkontexts mit Benutzername und ID, Task-Name und ID, Name des Host Rechners und Client Version
- *sequenceNumber*: Aktuelle Nummer in der automatisch durchnummerierten Sequenz
- *primaryKey*: Wert des Primärschlüssels

Beispiel

Ein Spiegel Logbuch auf dem Server im auto-repair Modus integrieren:

```
 //Auf dem Server ausführen
C_OBJECT($err)
C_LONGINT($num)  //-2 zum Integrieren aller Operationen
INTEGRATE MIRROR LOG FILE("c:\\mirror\\logNew.journal";$num;Auto repair mode;$err)
```

Systemvariablen und Mengen

Bei korrekt ausgeführter Integration wird die Systemvariable OK auf 1 gesetzt; andernfalls auf 0 (Null).

Log File

Log File -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	String	➲	Langer Name des Logbuchs der Datenbank

Beschreibung

Die Funktion **Log File** gibt den langen Namen des aktuellen Logbuchs der geöffneten Datenbank zurück, z.B. den vollständigen Pfadnamen der Datei inkl. Dateiname.

Arbeitet die Datenbank ohne ein Logbuch, gibt der Befehl einen leeren String zurück. Die Systemvariable OK wird auf 0 (Null) gesetzt.
Läuft beim Arbeiten mit der Datenbank ein Logbuch mit, wird die Systemvariable OK auf 1 gesetzt. Der von der Funktion zurückgegebene Pfadname wird mit der Syntax der aktuellen Plattform angezeigt.

Warnung: Führen Sie diesen Befehl von einem remote Rechner aus, wird nur der Name des Logbuchs zurückgegeben, nicht der vollständige Pfadname.

Systemvariablen und Mengen

- Läuft die Datenbank ohne Logbuch, wird die Systemvariable OK auf 0 (Null) gesetzt; andernfalls auf 1.
- Ist das Logbuch während einer Arbeitssitzung aus irgendwelchen Gründen nicht verfügbar, wird der Fehler 1274 generiert und 4D Server erlaubt Benutzern nicht mehr, Daten zu schreiben. Ist das Logbuch wieder verfügbar, muss ein Backup gemacht werden.

LOG FILE TO JSON

LOG FILE TO JSON (ZielordnerPfad {; maxGröße {; LogPfad {; FeldAtt}}})

Parameter	Typ		Beschreibung
ZielordnerPfad	Text	➡	Pfad des Zielordners für Export
maxGröße	Lange Ganzzahl	➡	Maximale Größe der JSON Datei zum Erstellen (bytes)
LogPfad	Text	➡	Pfadname des Logbuchs zum Exportieren, ohne Angabe aktuelles Logbuch
FeldAtt	Lange Ganzzahl	➡	Feldbeschreibung Attribut: 1 = Nummer verwenden (Standard), 2 = Name verwenden

Beschreibung

Der Befehl LOG FILE TO JSON sichert das aktuelle bzw. das angegebene Journal (Logbuch) im JSON Format.

Ist ein Journal (binäre Datei) in JSON gesichert, lässt sich sein Inhalt vom Datenbankadministrator oder einem anderen Nutzer lesen, um z.B. die Operationen in der Datenbank zu analysieren.

In *ZielordnerPfad* geben Sie den Pfad des Ordners zum Speichern der JSON Datei an. Sie lautet **JournalExport.json**.

Die maximale Größe (in bytes) für die exportierte Datei JSON ist standardmäßig 10 MB. Ist dieses Limit erreicht, wird die Datei geschlossen und eine weitere Datei geöffnet. Die Größenbegrenzung für eine JSON Datei reduziert die Speicheranforderung zum Analysieren der Dateien. Wollen Sie die maximale Größe verändern, können Sie einen Wert im Parameter *maxGröße* setzen. Übergeben Sie 0, um auf die Standardgröße zurückzusetzen. Übergeben Sie einen negativen Wert, gibt es keine Größenbegrenzung

Standardmäßig, also ohne den Parameter *LogPfad*, sichert der Befehl das aktuelle Journal. Wollen Sie ein bestimmtes Journal exportieren, übergeben Sie seinen Pfad im Parameter *LogPfad*. Das Journal muss die Endung .journal haben. Um ein archiviertes Logbuch (Endung .4bl) zu exportieren, müssen Sie es zuerst über den Befehl **RESTORE** konvertieren.

Sie können einen leeren String ("") übergeben, um den Standard Öffnen-Dialog anzuzeigen, damit der Benutzer das Logbuch zum Sichern auswählen kann. Der gewählte Logbuch-Pfad wird in der Systemvariable **Document** zurückgegeben.

Hinweis: Sichert der Befehl das aktuelle Logbuch, ist die Anwendung nicht gesperrt, d.h. es lassen sich neue Operationen ausführen, während die Datei auf die Festplatte geschrieben wird - diese Operationen werden nicht in die gesicherte Datei aufgenommen.

Beim Exportieren des aktuellen Logbuchs können Sie mit dem Parameter *FeldAtt* definieren, wie die Felder im exportierten Attribut beschrieben werden: als Zahl (Standard) oder als Name. Sie können eine der folgenden Konstanten unter dem Thema **Backup und Wiederherstellen** übergeben:

Konstante	Typ	Wert	Kommentar
Field attribute with name	Lange Ganzzahl	2	Felder werden über ihren Namen identifiziert. Beispiel: {"Nachname":"Jones"}
Field attribute with number	Lange Ganzzahl	1	Felder werden über ihre Nummer identifiziert (Standard, wenn weggelassen). Beispiel: {"5":"Jones"}.

Hinweis: Beim Exportieren eines externen Logbuchs werden Felder immer über ihre Nummer identifiziert.

Die gesicherte JSON Datei enthält alle Operationen, die im Journal protokolliert sind, in Form eines Array mit JSON Objekten. Jedes Objekt enthält mehrere Eigenschaften, die die Operation beschreiben. Hier ein Beispiel:


```
[
  {
    "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"
    }
]
```


Hinweis: Übergeben Sie Field attribute with name im Parameter *FeldAtt*, erscheinen die Objekte "fields" wie folgt:


```
...
        "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"
        },...
```


Die aktuelle Liste der verfügbaren Eigenschaften richtet sich nach der Art der Operation (z.B. Datensatz erstellen, Datensatz löschen, Datensatz ändern, Blob erstellen). Hier die Haupteigenschaften:

- *operationType*: interner Code der Operation
- *operationName*: Art der Operation, z.B. "Datensatz erstellen," "Datensatz ändern"
- *operationNumber*: interne Nummer der Operation im Logbuch
- *contextID*: ID des Ausführungskontexts, Details zum Kontext im Bereich *extraData*
- *timeStamp*: Zeitstempel der Operation im Logbuch
- *dataLen*: Größe der Daten
- *recordNumber*: interne Datensatznummer
- *tableID*: interne ID der Tabelle
- *tableName*: Name der Tabelle
- *fields*: Objekt mit der Liste der Feldnummern (oder Feldnamen) mit ihren Werten. Jedes Feld mit einem geänderten Wert wird protokolliert.
  Bei Werten vom Typ Blob oder Bild wird je nach Speicherort unterschiedliche Information geliefert:
  - Beim Speichern innerhalb der Datendatei lautet die Eigenschaft "BlobID:"+ eine interne Blob Nummer, z.B.: "BlobID:1"
  - Beim Speichern außerhalb der Datendatei lautet die Eigenschaft "BlobPfad:" + Datenpfad, z.B.: "BlobPath: Table 1/Field 6/Data_EE12D091535F9748BCE62EDE972A4BA2.jpg"
- *extraData*: Daten des Benutzerkontexts mit Benutzername und ID, Task-Name und ID, Name des Host Rechners und Client Version
- *sequenceNumber*: Aktuelle Nummer in der automatisch durchnummerierten Sequenz
- *primaryKey*: Wert des Primärschlüssels

Beispiel

Das aktuelle Journal in JSON sichern:

```
LOG FILE TO JSON("c:\\4Dv15\\ExportLogs")
```

Ein bestimmtes Journal mit Feldnamen in JSON sichern:

```
LOG FILE TO JSON("c:\\4Dv15\\ExportLogs";0;"c:\\4Dv15\\Backup\\old_myDB.journal";Field attribute with name)
```

Systemvariablen und Mengen

Der Befehl **LOG FILE TO JSON** ändert den Wert der Variablen *OK* und *Document*: Wurde die JSON Datei korrekt gesichert, wird OK auf 1 gesetzt und *Document* enthält den Pfadnamen des Jounrals. Haben Sie "" in *LogPfad* übergeben und annulliert der Benutzer das Dialogfenster zum Auswählen der Datei, wird OK auf 0 gesetzt und *Document* enthält einen leeren String. Wählt der Benutzer eine ungültige Datei aus, wird OK auf 0 gesetzt und *Document* enthält den Pfadnamen der ungültigen Datei.

New log file

New log file -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Text	Voller Pfadname des geschlossenen Logbuchs

Beschreibung

Hinweis: Diese Funktion arbeitet nur mit 4D Server. Er lässt sich nur über die Funktion **Execute on server** oder in einer Serverprozedur ausführen.

Die Funktion **New log file** schließt das aktuelle Logbuch, ändert seinen Namen und erstellt ein neues mit demselben Namen und am selben Ort wie das vorige. Sie wird zum Einrichten eines Backup-Systems über logisches Spiegeln verwendet. Weitere Informationen dazu finden Sie im *4D Server Handbuch* im Abschnitt **Logischen Spiegel einrichten**.
Die Funktion gibt den vollständigen Pfadnamen (Zugriffspfad + Name) des gerade geschlossenen Logbuchs zurück (genannt "Segment"). Die Datei wird an derselben Stelle wie das aktuelle Logbuch gespeichert. Das legen Sie in den Einstellungen der Datenbank unter dem Thema Backup fest. Der Befehl bearbeitet die gesicherte Datei nicht, sie wird also weder komprimiert noch segmentiert. Es erscheint kein Dialogfenster.
Die Datei wird mit den aktuellen Backup-Nummern der Datenbank und des Logbuchs bezeichnet. Der Name ist untergliedert in *DatenbankName[BackupNum-LogBackupNum].journal*.

Nachfolgend ein Beispiel:

- Wurde MeineDatenbank.4DD 4 mal gesichert, hat die letzte Backup-Datei den Namen *MeineDatenbank[0004].4BK*. Der Name des ersten Segments des Logbuchs lautet dann *MeineDatenbank[0004-0001].journal*.
- Wurde *MeineDatenbank.4DD* 3 mal gesichert und das Logbuch inzwischen 5 mal, lautet der Name des 6. Backup des Logbuchs *MeineDatenbank[0003-0006].journal*.

Fehlerverwaltung

Im Falle eines Fehlers generiert der Befehl einen Code, der über den Befehl **ON ERR CALL** abgefangen werden kann.

RESTORE

RESTORE {(ArchivPfad {; ZielordnerPfad})}

Parameter	Typ		Beschreibung
ArchivPfad	Text	➡	Pfadname des wiederherzustellenden Archivs
ZielordnerPfad	Text	➡	Pfadname des Zielordners

Beschreibung

Der Befehl **RESTORE** dient zum Wiederherstellen der Dateien, die in einem 4D Archiv enthalten sind. Dieser Befehl ist hilfreich zum Verwalten von Backups bei selbst gestalteten Oberflächen.

Übergeben Sie nicht den Parameter *ArchivPfad*, zeigt der Befehl den Standard-Öffnen Dialog, so dass der Benutzer das Archiv zum Wiederherstellen wählen kann

Im Parameter *ArchivPfad* geben Sie den Pfadnamen der Archivdatei an, die wiederhergestellt werden soll. Dieser Pfadname muss in der Syntax des Betriebssystems ausgedrückt werden. Sie können einen absoluten Pfadnamen oder einen in Bezug auf die Strukturdatei der Datenbank übergeben.
Dann erscheint (ohne Angabe des Parameters *ZielordnerPfad*) der standardmäßige Wiederherstellungsdialog mit dem vorausgewählten Archiv, so dass der Benutzer den Zielordner angeben kann. Ist der Vorgang abgeschlossen, erscheint eine Meldung und der Ordner mit den wiederhergestellten Elementen wird angezeigt.
Sie können den Parameter *ZielordnerPfad* auch mit dem Pfadnamen des Zielordners der wiederhergestellten Elemente übergeben. Dieser Pfadname muss in der Syntax des Betriebssystems ausgedrückt werden. Sie können einen absoluten Pfadnamen oder einen in Bezug auf die Strukturdatei der Datenbank übergeben. Übergeben Sie diesen Parameter, erscheint ein vorkonfiguriertes Dialogfenster zum Wiederherstellen, so dass der Benutzer die Operation zum Wiederherstellen starten oder abbrechen kann. Ist sie abgeschlossen, schließt sich das Fenster ohne weitere Informationen.

RESTORE ändert den Wert der Variablen Document und OK: Wurde das Wiederherstellen korrekt ausgeführt, wird OK auf 1 gesetzt und Document enthält den Pfad des Wiederherstellungsordners.
Bricht der Benutzer das Wiederherstellen ab oder tritt ein Fehler auf, wird OK auf 0 gesetzt und Document enthält einen leeren String. Sie können den Fehler mit einer Methode abfangen, die über den Befehl **ON ERR CALL** installiert wird.

Hinweis: In einer kompilierten 4D Anwendung mit integrierter 4D Volume Desktop zeigt **RESTORE** einen standardmäßigen Öffnen-Dialog, der alle Dateien mit der Endung “4BK” anzeigt.

SELECT LOG FILE

SELECT LOG FILE (Logbuchname)

Parameter	Typ		Beschreibung
Logbuchname	Operator, String	⇒	Name der Logbuchdatei oder "*" zum Schließen der aktuellen Logbuchdatei

Beschreibung

Der Befehl **SELECT LOG FILE** erstellt oder schließt die Logbuchdatei gemäß dem in *Logbuchname* übergebenen Wert.

Hinweis: Aufrufen dieses Befehls bewirkt dasselbe wie Aktivieren bzw. Deaktivieren der Option **Benutze Logbuch** auf der Seite **Backup**/**Konfiguration** in den Einstellungen der Datenbank.

In *Logbuchname* übergeben Sie den Namen bzw. den kompletten Pfadnamen des Logbuchs, das erstellt werden soll. Übergeben Sie nur den Namen, wird das Logbuch im Ordner Logs angelegt, der neben der Strukturdatei der Datenbank liegt. Ist *Logbuchname* ein leerer Text, zeigt **SELECT LOG FILE** einen Dialog an, in dem der Benutzer Name und Ort des zu erstellenden Logbuchs wählen kann.
Wird die Datei korrekt angelegt, hat die Systemvariable *OK* den Wert 1. Sonst hat sie den Wert Null. Klickt der Benutzer auf die Schaltfläche **Abbrechen** oder kann die Datei nicht angelegt werden, hat *OK* den Wert Null.

Hinweis: Das neue Logbuch wird nicht sofort nach Ausführen des Befehls angelegt, sondern erst nach dem nächsten Backup. Der Parameter wird in der Datendatei gehalten und berücksichtigt, selbst wenn die Datenbank inzwischen geschlossen ist. Sie können den Befehl **BACKUP** aufrufen, um das Erstellen des Logbuch auszulösen.

Übergeben Sie in *Logbuchname* den Parameter *, schließt **SELECT LOG FILE** die aktuelle Logbuchdatei der Datenbank. *OK* hat dann den Wert 1.

Verwenden Sie diesen Befehl zum Erstellen einer Logbuchdatei, bevor ein Voll-Backup ausgeführt wurde und enthält die Datei bereits Datensätze, generiert 4D den Fehler *-4447,* den Sie mit der Methode **ON ERR CALL** abfangen können.

Systemvariablen und Mengen

Die Systemvariable **OK** hat den Wert 1, wenn die Logbuchdatei korrekt erstellt oder geschlossen wird.

Fehlerverwaltung

Der Fehler -4447 wird generiert, wenn die Operation nicht ausgeführt werden kann, da von der Datenbank kein Backup erstellt wurde. Sie können den Fehler mit der Methode **ON ERR CALL** abfangen.

_o_INTEGRATE LOG FILE

_o_INTEGRATE LOG FILE (PfadName)

Parameter	Typ		Beschreibung
PfadName	Text	➡	Name oder Pfadname des zu integrierenden Logbuchs

Hinweis zur Kompatibilität

Der Befehl **_o_INTEGRATE LOG FILE** ist ab 4D Version 16 überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Das Backup Feature logisches Spiegeln basiert nun allein auf dem Befehl **INTEGRATE MIRROR LOG FILE**. Er wurde optimiert und bietet größere Flexibilität.

Benutzer und Gruppen

- 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 (Benutzer)

Parameter	Typ		Beschreibung
Benutzer	BLOB	➡	BLOB (verschlüsselt) mit Benutzerkonten der Datenbank, erstellt und gesichert vom Administrator

Beschreibung

Der Befehl **BLOB TO USERS** ersetzt die Benutzerkonten und Gruppen, die der Administrator in der Datenbank erstellt hat, mit den im BLOB *Benutzer* gefundenen Benutzern Das BLOB *Benutzer* ist verschlüsselt und muss über den Befehl **USERS TO BLOB** erstellt sein.

Nur der Administrator oder Designer der Datenbank können diesen Befehl verwenden. Versucht ein anderer Benutzer, ihn einzusetzen, führt er nichts aus. Der Fehler -9949 (kein Zugriffsrecht) wird erzeugt.

Dieser Befehl ersetzt alle vorhandenen Benutzerkonten und Gruppen, die der Administrator in der Datenbank angelegt hat. Enthält das BLOB *Benutzer* gültige Daten, führt der Befehl folgende Operationen aus:

- Alle in der Datenbank definierten Benutzer und Gruppen mit negativer Referenznummer (vom Administrator angelegte Gruppen und Benutzer) werden aus der Struktur entfernt.
- Alle im BLOB *Benutzer* vorhandenen Benutzer und Gruppen werden in der Struktur hinzugefügt.

Hinweis zur Kompatibilität: Dateien mit Benutzern und Gruppen mit der Endung .4UG, die mit dem Menübefehl **Save Groups** in früheren 4D Versionen erstellt wurden, lassen sich mit folgender Anweisung in 4D übernehmen (Bei sehr alten Versionen sind u.U. Zwischenversionen erforderlich.):

```
DOCUMENT TO BLOB(mydoc;blob)
BLOB TO USERS(blob)
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl, wird die Systemvariable OK auf 1 gesetzt, andernfalls auf 0.

CHANGE CURRENT USER

CHANGE CURRENT USER {(Benutzer ; Kennwort)}

Parameter	Typ		Beschreibung
Benutzer	String, Lange Ganzzahl	➡	Name oder einmalige Benutzernummer
Kennwort	String	➡	Kennwort (unverschlüsselt)

Beschreibung

Der Befehl **CHANGE CURRENT USER** ändert die Identität des aktuellen Benutzers in der Datenbank, ohne sie zu beenden. Die Benutzer können ihre Identität selbst ändern, entweder über den Datenbank Verbindungsdialog (wenn der Befehl ohne Parameter aufgerufen wird) oder direkt über den Befehl. Ändert ein Benutzer seine Identität, gibt er alle bisherigen Zugriffsberechtigungen zugunsten der des gewählten Benutzers auf.

Wird **CHANGE CURRENT USER** ohne Parameter verwendet, erscheint der Verbindungsdialog für die Datenbank. Der Benutzer muss dann einen gültigen Namen und Kennwort eingeben bzw. auswählen, um die Datenbank zu öffnen. Der Inhalt des Dialogs richtet sich nach den Optionen, die in den Datenbank-Eigenschaften auf der Seite **Sicherheit** definiert wurden.

Hinweis: Für diesen Befehl muss das Zugriffskontrollsystem aktiviert sein, z.B. muss dem Designer ein Kennwort zugewiesen sein. Sonst hat **CHANGE CURRENT USER** keine Auswirkung und zeigt nicht das Standardfenster zum Wechseln von Benutzern.

Sie können auch die beiden optionalen Parameter *Benutzer* und *Kennwort* übergeben, um per Programmierung das neue Account zu definieren.
Im Parameter *Benutzer* übergeben Sie den Namen oder die einmalige Benutzernummer (*BenutzerRef*) des zu verwendenden Accounts. Die Benutzernamen und -nummern erhalten Sie über den Befehl **GET USER LIST**. Folgende Kennziffern sind für Benutzer möglich:

Kennziffer	Benutzer
1	Designer
2	Administrator
3 bis 15000	Vom Designer angelegte Benutzer (der erste Benutzer hat die Nummer #3, der zweite die Nummer #4, usw.)
-11 bis -15010	Vom Administrator angelegte Benutzer (der erste Benutzer hat die Nummer -11, der zweite die Nummer -12, usw.)

Ist das Benutzerkonto nicht vorhanden oder wurde es gelöscht, wird Fehler -9979 generiert. Sie können diesen mit einer Fehlerverwaltungsmethode abfangen, die über den Befehl **ON ERR CALL** installiert wurde. Sie können aber auch vor Aufrufen dieses Befehls das Benutzerkonto über die Funktion **Is user deleted** testen.
Im Parameter *Kennwort* übergeben Sie das unverschlüsselte Kennwort des Benutzerkontos. Passt es nicht zum Benutzer, gibt der Befehl die Fehlermeldung -9978 zurück und führt nichts aus.

Der Befehl wird zeitverzögert ausgeführt, um "Überschwemmung" (flooding) durch massive Hackerattacken zu verhindern, das sind Versuche über vielfache Kombinationen Name/Kennwort. Das Ergebnis ist, dass der 4. Aufruf des Befehls erst nach 10 Sekunden ausgeführt wird. Diese Zeitspanne zieht sich durch die gesamte Workstation.

Eigene Dialogfenster für Zugriffsverwaltung

Über den Befehl **CHANGE CURRENT USER** können Sie eigene Dialogfenster zum Eingeben von Name und Kennwort einrichten, inkl. Eingabe- und Ausschlusskriterien. Sie haben die gleichen Vorteile wie das Zugriffssystem von 4D.

Funktionsweise:

1. Die Datenbank wird direkt im Modus "Standardbenutzer" geöffnet, es erscheint kein Dialogfenster.
2. Die **Datenbankmethode On Startup** zeigt ein angepasstes Dialogfenster zum Eingeben von Benutzername und Kennwort. Sie können folgende Operationen einrichten:
   - Mit dem Befehl **GET USER LIST** können Sie – wie im standardmäßigen Zugriffsdialog von 4D – die Liste der Datenbankbenutzer anzeigen.
   - Das Eingabefeld für Kennwort kann verschiedene Beschränkungen enthalten, welche die Gültigkeit der eingegebenen Zeichen steuern, z.B. Mindestanzahl der Zeichen, Einmaligkeit u.ä.
   - Damit die Zeichen der eingegebenen Kennwörter auf dem Bildschirm verschlüsselt erscheinen, können Sie den Befehl **FILTER KEYSTROKE** mit der Spezialschrift *%password* verwenden.
   - Beim Bestätigen des Dialogfensters können Ausschlussregeln zur Anwendung kommen, z.B. Ablaufdatum, erzwungene Änderung auf Anfangsverbindung, Sperren des Accounts nach mehreren inkorrekten Eingaben, Speichern bereits verwendeter Kennwörter.
3. Beim Bestätigen der Eingabe wird die erforderliche Information (Benutzername und Kennwort) im Befehl **CHANGE CURRENT USER** übergeben, um die Datenbank mit den Zugriffsrechten des Benutzer-Accounts zu öffnen.

Beispiel

Das folgende Beispiel zeigt den Anmeldedialog:

```
CHANGE CURRENT USER
```

CHANGE PASSWORD

CHANGE PASSWORD (Kennwort)

Parameter	Typ		Beschreibung
Kennwort	String	➡	Neues Kennwort

Beschreibung

Der Befehl **CHANGE PASSWORD** ändert das Kennwort für den aktuellen Benutzer. Dieser Befehl ersetzt das aktuelle Kennwort durch das neue Kennwort, das im Parameter *Kennwort* übergeben wurde.

Warnung: Kennwörter berücksichtigen Groß- und Kleinschreibung.

Beispiel

Im folgenden Beispiel kann der Benutzer das Kennwort ändern.

```
CHANGE CURRENT USER  ` Zeige Benutzer mit Kennwortdialog
If(OK=1)
   $pw1:=Request("Gib neues Kennwort ein für "+Current user)
 ` Kennwort sollte mindestens 5 Zeichen lang sein.
   If(((OK=1)&($pw1#""))&(Length($pw1)>5))
 ` Prüfen Sie, ob Kennwort richtig eingegeben wurde
      $pw2:=Request("Gib Kennwort erneut ein")
      If((OK=1)&($pw1=$pw2))
         CHANGE PASSWORD($pw2)  ` Ändere Kennwort
      End if
   End if
End if
```

Current user

Current user -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	String	Name des aktuellen Benutzers

Beschreibung

Die Funktion **Current user** gibt den Namen des aktuellen Benutzers zurück.

Beispiel

Siehe Beispiel zur Funktion **User in group**.

DELETE USER

DELETE USER (BenutzerNr)

Parameter	Typ		Beschreibung
BenutzerNr	Lange Ganzzahl	⇒	Kennziffer des zu löschenden Benutzers

Beschreibung

Der Befehl **DELETE USER** löscht den Benutzer mit der einmaligen Kennziffer *BenutzerNr*. Sie müssen eine gültige Kennziffer übergeben, die der Befehl **GET USER LIST** zurückgibt.

Gibt es das Benutzerkonto nicht oder wurde es bereits gelöscht, erhalten Sie den Fehler -9979. Sie können diesen Fehler mit dem Befehl **ON ERR CALL** in einer Methode zur Fehlerverwaltung ausfindig machen.

Nur Designer und Administrator können Benutzer löschen. Der Administrator kann jedoch keinen Benutzer löschen, den der Designer angelegt hat.
Gelöschte Benutzernamen erscheinen nicht länger im Kennwortdialog, der beim Aufrufen von **EDIT ACCESS** angezeigt wird, bzw. im Designmodus. Beachten Sie, dass Kennziffern gelöschter Benutzer wieder neu zugewiesen werden, wenn neue Benutzerkonten angelegt werden.

Fehlerverwaltung

Haben Sie keine Zugriffsberechtigung zum Aufrufen von **DELETE USER** oder wird bereits von einem anderen Prozess aus auf das Kennwortsystem zugegriffen, erhalten Sie eine Fehlermeldung. Sie können diesen Fehler mit dem Befehl **ON ERR CALL** in einer Methode zur Fehlerverwaltung ausfindig machen.

EDIT ACCESS

EDIT ACCESS
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **EDIT ACCESS** zeigt den Dialog mit den verfügbaren Benutzern und Benutzergruppen wie im Designmodus. Der Gruppeneigentümer, Administrator oder Designer können darin die Benutzer für diese Gruppe ändern, beispielsweise in der Gruppe einen Benutzer hinzufügen oder löschen. Designer und Administrator können alle Gruppen bearbeiten, Eigentümer nur die Gruppe, die ihnen gehören.
Der Befehl hat keine Auswirkung, wenn keine Gruppen definiert wurden.

Hinweis: Dieser Befehl öffnet ein modales Fenster. Sie dürfen ihn also nicht in einem anderen modalen Fenster aufrufen, denn dann führt er nichts aus.

Beispiel

Folgendes Beispiel zeigt das Dialogfenster der Kennwörter an:

```
EDIT ACCESS
```

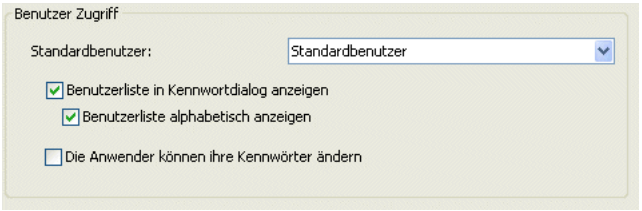
Get default user

Get default user -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Lange Ganzzahl	Einmalige Kennziffer für Benutzer

Beschreibung

Die Funktion **Get default user** gibt die einmalige Kennziffer des Benutzers zurück, der in den Einstellungen der Datenbank als Standardbenutzer definiert wurde:



Folgende Kennziffern sind für Benutzer möglich:

Kennziffer	Benutzer
1	Designer
2	Administrator
3 bis 15000	Vom Designer angelegte Benutzer (der erste Benutzer hat die Nummer 3, der zweite Nummer 4, usw.)
-11 bis -15010	Vom Administrator angelegte Benutzer (der erste Benutzer hat die Nummer -11, der zweite die Nummer -12, usw.)

Wurde kein Standardbenutzer eingerichtet, gibt die Funktion den Wert 0 (Null) zurück.

GET GROUP LIST

GET GROUP LIST (GruppenName ; GruppenNr)

Parameter	Typ		Beschreibung
GruppenName	Array String	⇐	Gruppennamen, wie sie im Fenster Kennworteditor erscheinen
GruppenNr	Array Lange Ganzzahl	⇐	Entsprechende einmalige Gruppenkennziffern

Beschreibung

Der Befehl **GET GROUP LIST** füllt die Arrays *GruppenName* und *GruppenNr* mit den Namen und einmaligen Kennziffern der Gruppen, wie sie im Fenster Kennworteditor erscheinen.

Das Array *GruppenNr* ist mit *GruppenName* synchronisiert. Es enthält die dazugehörigen einmaligen Gruppenkennziffern. Sie können folgende Bereiche haben:

Gruppenkennziffer	Beschreibung
15001 bis 32767	Gruppe, angelegt vom Designer oder dem zugeordneten Gruppeneigentümer (Gruppennr. 15001 ist die erste Gruppe, Gruppennr. 15002 die zweite usw.)
-15001 bis -32768	Gruppe, angelegt vom Administrator oder dem zugeordneten Gruppeneigentümer (Gruppennr. -15001 ist die erste Gruppe, Gruppennr. -15002 die zweite usw.)

Fehlerverwaltung

Haben Sie keine Zugriffsberechtigung zum Aufrufen von **GET GROUP LIST** oder wird bereits von einem anderen Prozess aus auf das Kennwortsystem zugegriffen, erhalten Sie eine Fehlermeldung. Sie können diesen Fehler mit dem Befehl **ON ERR CALL** in einer Methode zur Fehlerverwaltung ausfindig machen.

GET GROUP PROPERTIES

GET GROUP PROPERTIES (GruppenNr ; Auswahlname ; Eigentümer {; Mitglieder})

Parameter	Typ		Beschreibung
GruppenNr	Lange Ganzzahl	⇒	Einmalige Gruppenkennziffer
Auswahlname	String	⇐	Name der Gruppe
Eigentümer	Lange Ganzzahl	⇐	Kennziffer des Gruppeneigentümers
Mitglieder	Array Lange Ganzzahl	⇐	Gruppenmitglieder

Beschreibung

Der Befehl **GET GROUP PROPERTIES** gibt die Eigenschaften der Gruppe mit der einmaligen Gruppenkennziffer *GruppenNr* zurück. Sie müssen eine gültige Kennziffer übergeben, die der Befehl **GET GROUP LIST** zurückgibt. Gruppenkennziffern können folgende Werte oder Bereiche haben:

Gruppenkennziffer	Beschreibung
15001 bis 32767	Gruppe, angelegt vom Designer oder dem zugeordneten Gruppeneigentümer (Gruppennr. 15001 ist die erste Gruppe Gruppennr. 15002 die zweite usw.)
-15001 bis -32768	Gruppe, angelegt vom Administrator oder dem zugeordneten Gruppeneigentümer (Gruppennr. -15001 ist die erste Gruppe Gruppennr. -15002 die zweite usw.)

Übergeben Sie eine ungültige Gruppenkennziffer, gibt **GET GROUP PROPERTIES** leere Parameter zurück.

Nach dem Aufruf finden Sie Name und Eigentümer der Gruppe in den Parametern *Name* und *Eigentümer*.

Mit dem optionalen Parameter *Mitglieder* werden die einmaligen Kennziffern der Benutzer und Gruppen zurückgegeben, die zu der Gruppe gehören. Die Kennziffern der Mitglieder können folgende Bereiche haben:

Benutzerkennziffer	Beschreibung
1	Designer
2	Administrator
3 bis 15000	Benutzer, die der Datenbank-Designer angelegt hat (Benutzernr. 3 ist der erste Benutzer, Benutzernr. 4 der zweite, usw.)
-11 bis -15000	Benutzer, die der Datenbank-Administrator angelegt hat (Benutzernr. -11 ist der erste Benutzer, Benutzernr. -12 der zweite, usw.)
15001 bis 32767	Gruppe, angelegt vom Designer oder dem zugeordneten Gruppeneigentümer (Gruppennr. 15001 ist die erste Gruppe Gruppennr. 15002 die zweite usw.).
-15001 bis -32768	Gruppe, angelegt vom Administrator oder dem zugeordneten Gruppeneigentümer (Gruppennr. -15001 ist die erste Gruppe Gruppennr. -15002 die zweite usw.)

Fehlerverwaltung

Haben Sie keine Zugriffsberechtigung zum Aufrufen von **GET GROUP PROPERTIES** oder wird bereits von einem anderen Prozess aus auf das Kennwortsystem zugegriffen, erhalten Sie eine Fehlermeldung. Sie können diesen Fehler mit dem Befehl **ON ERR CALL** in einer Methode zur Fehlerverwaltung ausfindig machen.

Get plugin access

Get plugin access (PlugIn) -> Funktionsergebnis

Parameter	Typ		Beschreibung
PlugIn	Lange Ganzzahl	⇒	Plug-In Nummer
Funktionsergebnis	String	⇐	Gruppenname für Plug-In

Beschreibung

Die Funktion **Get plugin access** gibt den Namen der Benutzergruppe zurück, die das in *PlugIn* übergebene Plug-In benutzen darf. Wurde der Gruppe kein Plug-In zugeordnet, gibt die Funktion einen leeren String ("") zurück.

Im Parameter *PlugIn* übergeben Sie die Nummer des Plug-In, für das Sie die zugeordnete Benutzergruppe erhalten wollen. Plug-In Lizenzen enthalten Web und Web Client Lizenzen. Sie können eine der Konstanten unter dem Thema **Is license available** verwenden:

Konstante	Typ	Wert
4D Client SOAP license	Lange Ganzzahl	808465465
4D Client Web license	Lange Ganzzahl	808465209
4D for OCI license	Lange Ganzzahl	808465208
4D ODBC Pro license	Lange Ganzzahl	808464946
4D View license	Lange Ganzzahl	808465207
4D Write license	Lange Ganzzahl	808464697

GET USER LIST

GET USER LIST (BenutzerName ; BenutzerNr)

Parameter	Typ		Beschreibung
BenutzerName	Array String	⇐	Benutzernamen, wie sie im Fenster Kennworteditor erscheinen
BenutzerNr	Array Lange Ganzzahl	⇐	Dazugehörige einmalige Benutzerkennziffer

Beschreibung

Der Befehl **GET USER LIST** füllt die Arrays *BenutzerName* und *BenutzerNr* mit den Namen und einmaligen Kennziffern der Benutzer, so wie sie im Fenster Kennworteditor erscheinen. Das Array *BenutzerName* enthält die Benutzernamen, die im Kennwortdialog angezeigt werden. Dazu gehören auch die deaktivierten Benutzer. Sie erscheinen im Kennwortdialog in Grün.

Hinweis: Verwenden Sie die Funktion **Is user deleted**, um gelöschte Benutzer zu finden.

Das Array *BenutzerNr* ist mit *BenutzerName* synchronisiert. Es enthält die dazugehörigen einmaligen Benutzerkennziffern. Sie können folgende Werte haben:

Benutzerkennziffer	Beschreibung
1	Designer
2	Administrator
3 bis 15000	Benutzer, die der Datenbank-Designer angelegt hat (Benutzernr. 3 ist der erste Benutzer Benutzernr. 4 der zweite, usw.)
-11 bis -15000	Benutzer, die der Datenbank-Administrator angelegt hat (Benutzernr. -11 ist der erste Benutzer Benutzernr. -12 der zweite, usw.)

Fehlerverwaltung

Haben Sie keine Zugriffsberechtigung zum Aufrufen von **GET USER LIST** oder wird bereits von einem anderen Prozess aus auf das Kennwortsystem zugegriffen, erhalten Sie eine Fehlermeldung. Sie können diesen Fehler mit dem Befehl **ON ERR CALL** in einer Methode zur Fehlerverwaltung ausfindig machen.

GET USER PROPERTIES

GET USER PROPERTIES (BenutzerNr ; Auswahlname ; Startup ; Kennwort ; AnzLogin ; letztesLogin {; Gruppen {; GrEigentümer}})

Parameter	Typ		Beschreibung
BenutzerNr	Lange Ganzzahl	➡	Einmalige Benutzerkennziffer
Auswahlname	String	⬅	Name des Benutzers
Startup	String	⬅	Name der Startup-Methode
Kennwort	String	⬅	Immer ein leerer String
AnzLogin	Lange Ganzzahl	⬅	Anzahl Benutzungen der Datenbank
letztesLogin	Datum	⬅	Datum letzte Benutzung der Datenbank
Gruppen	Array Lange Ganzzahl	⬅	Gruppenkennziffern, zu denen der Benutzer gehört
GrEigentümer	Lange Ganzzahl	⬅	Kennziffer der Benutzergruppe Eigentümer

Beschreibung

Der Befehl **GET USER PROPERTIES** gibt Informationen über den Benutzer mit der einmaligen Kennziffer, übergeben in *BenutzerNr*, zurück. Sie müssen eine gültige Kennziffer übergeben, die der Befehl **GET USER LIST** zurückgibt.

Gibt es den Benutzer nicht oder wurde er gelöscht, erscheint der Fehler -9979. Sie können diesen Fehler mit **ON ERR CALL** in einer Fehlerverwaltungsmethode ausfindig machen. Sie können aber auch vor dem Aufrufen von **GET USER PROPERTIES** mit der Funktion **Is user deleted** prüfen, ob der Benutzer existiert. Benutzerkennziffern können folgende Werte bzw. Bereiche haben:

Benutzerkennziffer	Beschreibung
1	Designer
2	Administrator
3 bis 15000	Benutzer, die der Datenbank-Designer angelegt hat (Benutzernr. 3 ist der erste Benutzer Benutzernr. 4 der zweite, usw.)
-11 bis -15000	Benutzer, die der Datenbank-Administrator angelegt hat (Benutzernr. -11 ist der erste Benutzer Benutzernr. -12 der zweite, usw.)

Nach dem Aufrufen von **GET USER PROPERTIES** finden Sie den Benutzernamen, die Startup-Methode, einen leeren String für das Kennwort, die Anzahl der Benutzungen sowie Datum der letzten Benutzung der Datenbank in den entsprechenden Parametern.

Hinweis: Der Parameter *Kennwort* ist überholt, es wird immer ein leerer String zurückgegeben. Wollen Sie das Kennwort eines Benutzers prüfen, rufen Sie die Funktion **Validate password** auf.

Mit dem optionalen Parameter *GrEigentümer* erhalten Sie die Kennziffer der Benutzergruppe "Eigentümer" der Benutzergruppe, z.B. die Standardgruppe "Eigentümer" für Objekte, die dieser Benutzer angelegt hat.

Mit dem optionalen Parameter *Gruppen* werden die einmaligen Kennziffern der Gruppen, zu denen der Benutzer gehört, zurückgegeben. Gruppenkennziffern können folgende Bereiche haben:

Gruppenkennziffer	Beschreibung
15001 bis 32767	Gruppe, angelegt vom Designer oder dem zugeordneten Gruppeneigentümer (Gruppennr. 15001 ist die erste Gruppe Gruppennr. 15002 die zweite usw.)
-15001 bis -32768	Gruppe, angelegt vom Administrator oder dem zugeordneten Gruppeneigentümer (Gruppennr. -15001 ist die erste Gruppe Gruppennr. -15002 die zweite usw.)

Fehlerverwaltung

Haben Sie keine Zugriffsberechtigung zum Aufrufen von **GET USER PROPERTIES** oder wird bereits von einem anderen Prozess aus auf das Kennwortsystem zugegriffen, erhalten Sie eine Fehlermeldung. Sie können diesen Fehler mit dem Befehl **ON ERR CALL** in einer Methode zur Fehlerverwaltung ausfindig machen.

Is user deleted

Is user deleted (BenutzerNr) -> Funktionsergebnis

Parameter	Typ		Beschreibung
BenutzerNr	Lange Ganzzahl	⇒	Nummer des Benutzers
Funktionsergebnis	Boolean	⇐	Wahr = Benutzerkonto ist gelöscht oder existiert nicht; Falsch = Benutzerkonto ist aktiv

Beschreibung

Die Funktion **Is user deleted** prüft den Benutzer mit der einmaligen Kennziffer *BenutzerNr*.
Existiert der Benutzer nicht oder wurde er gelöscht, gibt **Is user deleted** den Wert *Wahr* zurück, sonst den Wert *Falsch*.

Fehlerverwaltung

Haben Sie keine Zugriffsberechtigung zum Aufrufen von **Is user deleted** oder wird bereits von einem anderen Prozess aus auf das Kennwortsystem zugegriffen, erhalten Sie eine Fehlermeldung. Sie können diesen Fehler mit dem Befehl **ON ERR CALL** in einer Methode zur Fehlerverwaltung ausfindig machen.

Set group properties

Set group properties (GruppenNr ; Auswahlname ; Eigentümer {; Mitglieder}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
GruppenNr	Lange Ganzzahl	→	Einmalige Gruppenkennziffer oder -1 für neue Designer-Gruppe oder -2 für neue Administrator-Gruppe
Auswahlname	String	→	Neuer Gruppenname
Eigentümer	Lange Ganzzahl	→	Kennziffer des neuen Gruppeneigentümers
Mitglieder	Array Lange Ganzzahl	→	Neue Gruppenmitglieder
Funktionsergebnis	Lange Ganzzahl	←	Einmalige Kennziffer der neuen Gruppe

Beschreibung

Die Funktion **Set group properties** ändert die Eigenschaften einer bestehenden Gruppe mit der einmaligen Kennziffer *GruppenNr* oder fügt eine neue Gruppe hinzu, die dem Designer oder dem Administrator zugewiesen ist.

Ändern Sie die Eigenschaften einer bestehenden Gruppe, müssen Sie eine gültige Gruppenkennziffer übergeben, die der Befehl **GET GROUP LIST** zurückgibt. Gruppenkennziffern können folgende Werte oder Bereiche haben:

Gruppenkennziffer	Beschreibung
15001 bis 32767	Gruppe, angelegt vom Designer oder dem zugeordneten Gruppeneigentümer (Gruppennr. 15001 ist die erste Gruppe Gruppennr. 15002 die zweite usw.)
-15001 bis -32768	Gruppe, angelegt vom Administrator oder dem zugeordneten Gruppeneigentümer (Gruppennr. -15001 ist die erste Gruppe Gruppennr. -15002 die zweite usw.)

Wollen Sie eine neue Gruppe hinzufügen, die dem Designer zugeordnet ist, übergeben Sie in *GruppenNr* -1. Wollen Sie eine neue Gruppe hinzufügen, die dem Administrator zugeordnet ist, übergeben Sie in *GruppenNr* -2.

Übergeben Sie weder -1, -2 noch eine gültige Kennziffer, hat **Set group properties** keine Auswirkung.

Vor dem Aufrufen übergeben Sie für die Gruppe den neuen Namen und Eigentümer in den entsprechenden Parametern. Wollen Sie nicht alle Eigenschaften der Gruppe ändern (mit Ausnahme der Mitglieder, Näheres siehe unten), rufen Sie zuerst **GET GROUP PROPERTIES** auf und übergeben Sie die erhaltenen Werte für die Eigenschaften, die nicht geändert werden sollen.

Übergeben Sie nicht den optionalen Parameter *Mitglieder*, bleibt die aktuelle Mitgliedsliste der Gruppe bestehen. Übergeben Sie beim Hinzufügen einer Gruppe keine Mitglieder, hat die Gruppe keine Mitglieder.

Hinweis: Der Gruppeneigentümer ist nicht automatisch Mitglied seiner Gruppe. Sie können ihn je nach Wunsch über den Parameter *Mitglieder* in die Gruppe einfügen.

Mit dem optionalen Parameter *Mitglieder* ändern Sie alle Mitgliederlisten für die Gruppe. Sie müssen das Array *Mitglieder* vor dem Aufrufen von **Set group properties** mit den Kennziffern der Benutzer und Gruppen füllen, die Mitglied der Gruppen sein sollen. Mitgliedskennziffern können folgende Bereiche haben:

Gruppenkennziffer	Beschreibung
1	Designer
2	Administrator
3 bis 15000	Benutzer, die der Datenbank-Designer angelegt hat (Benutzernr. 3 ist der erste Benutzer, Benutzernr. 4 der zweite, usw.)
-11 bis -15000	Benutzer, die der Datenbank-Administrator angelegt hat (Benutzernr. -11 ist der erste Benutzer, Benutzernr. -12 der zweite, usw.)
15001 bis 32767	Gruppe, angelegt vom Designer oder dem zugeordneten Gruppeneigentümer (Gruppennr. 15001 ist die erste Gruppe Gruppennr. 15002 die zweite usw.)
15001 bis 32767	Gruppe, angelegt vom Designer oder dem zugeordneten Gruppeneigentümer (Gruppennr. 15001 ist die erste Gruppe Gruppennr. 15002 die zweite usw.)
-15001 bis -32768	Gruppe, angelegt vom Administrator oder dem zugeordneten Gruppeneigentümer (Gruppennr. -15001 ist die erste Gruppe Gruppennr. -15002 die zweite usw.)

Um alle Mitglieder aus einer Gruppe zu entfernen, übergeben Sie ein leeres Array *Mitglieder*.

Fehlerverwaltung

Haben Sie keine Zugriffsberechtigung zum Aufrufen von **Set group properties** oder wird bereits von einem anderen Prozess aus auf das Kennwortsystem zugegriffen, erhalten Sie eine Fehlermeldung. Sie können diesen Fehler mit dem Befehl **ON ERR CALL** in einer Methode zur Fehlerverwaltung ausfindig machen.

SET PLUGIN ACCESS

SET PLUGIN ACCESS (PlugIn ; Gruppe)

Parameter	Typ		Beschreibung
PlugIn	Lange Ganzzahl	➡	Plug-in Nummer
Gruppe	String	➡	Plug-In zuzuordnender Gruppenname

Beschreibung

Der Befehl **SET PLUGIN ACCESS** legt per Programmierung die Benutzergruppe fest, welche alle in der Datenbank lizenzierten Plug-Ins benutzen darf. Auf diese Weise können Sie die Verwendung der Plug-In Lizenzen steuern.

Hinweis: Diese Operation lässt sich auch über den Gruppeneditor in der Designumgebung ausführen.

Im Parameter *PlugIn* übergeben Sie die Nummer des Plug-In, das Sie der Benutzergruppe zuordnen wollen. Plug-In Lizenzen enthalten Web und Web Client Lizenzen. Sie können eine der Konstanten unter dem Thema **Is license available** verwenden:

Konstante	Typ	Wert
4D Client SOAP license	Lange Ganzzahl	808465465
4D Client Web license	Lange Ganzzahl	808465209
4D for OCI license	Lange Ganzzahl	808465208
4D ODBC Pro license	Lange Ganzzahl	808464946
4D View license	Lange Ganzzahl	808465207
4D Write license	Lange Ganzzahl	808464697

In *Gruppe* übergeben Sie den Namen der Gruppe, deren Benutzer das Plug-In einsetzen dürfen.

Hinweis: Es kann immer nur eine Gruppe gleichzeitig ein Plug-In verwenden. Wird **SET PLUGIN ACCESS** ausgeführt, wenn eine andere Gruppe Zugriff auf das Plug-In hat, verliert sie dieses Recht.

Set user properties

Set user properties (BenutzerNr ; Auswahlname ; Startup ; Kennwort ; AnzLogin ; letztesLogin {; Gruppen {; GrEigentümer}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
BenutzerNr	Lange Ganzzahl	➡	Einmalige Kennziffer des Benutzers, oder -1 für dem Designer zugeordnete Benutzer, oder -2 für dem Administrator zugeordnete Benutzer
Auswahlname	String	➡	Neuer Benutzername
Startup	String	➡	Name der neuen Startup-Methode des Benutzers
Kennwort	String	➡	Neues (unverschlüsseltes) Kennwort oder * für unverändertes Kennwort
AnzLogin	Lange Ganzzahl	➡	Neue Anzahl Benutzungen der Datenbank
letztesLogin	Datum	➡	Neues Datum letzte Benutzung der Datenbank
Gruppen	Array Lange Ganzzahl	➡	Gruppenkennziffern, zu denen der Benutzer gehört
GrEigentümer	Lange Ganzzahl	➡	Referenznummer des Gruppeneigentümers
Funktionsergebnis	Lange Ganzzahl	↩	Einmalige Kennziffer des neuen Benutzers

Beschreibung

Die Funktion **Set user properties** ermöglicht, die Eigenschaften eines bestehenden Benutzers mit der in *BenutzerNr* übergebenen Kennziffer zu ändern oder einen neuen Benutzer hinzufügen, der dem Designer oder Administrator zugewiesen ist.

Wollen Sie die Eigenschaften eines bestehenden Benutzers ändern, müssen Sie eine gültige Kennziffer übergeben, die der Befehl **GET USER LIST** zurückgibt.

Gibt es den Benutzer nicht oder wurde er gelöscht, erscheint der Fehler -9979. Sie können diesen Fehler mit **ON ERR CALL** in einer Fehlerverwaltungsmethode ausfindig machen. Sie können aber auch vor dem Aufrufen von **Set user properties** mit der Funktion **Is user deleted** prüfen, ob der Benutzer existiert. Benutzerkennziffern können folgende Werte bzw. Bereiche haben:

Benutzerkennziffer	Beschreibung
1	Designer
2	Administrator
3 bis 15000	Benutzer, die der Datenbank-Designer angelegt hat (Benutzernr. 3 ist der erste Benutzer, Benutzernr. 4 der zweite, usw.).
-11 bis -15000	Benutzer, die der Datenbank-Administrator angelegt hat (Benutzernr. -11 ist der erste Benutzer, Benutzernr. -12 der zweite, usw.).

Konnte der Benutzer er folgreich angelegt oder geändert werden, enthält der Parameter *BenutzerNr* die einmalige Kennummer des Benutzers.

Wollen Sie einen neuen Benutzer hinzufügen, der dem Designer zugeordnet ist, übergeben Sie in *BenutzerNr* -1. Wollen Sie einen neuen Benutzer hinzufügen, der dem Administrator zugeordnet ist, übergeben Sie in *BenutzerNr* -2. In beiden Fällen sucht 4D zuerst nach dem ersten verfügbaren inaktiven (gelöschter) Benutzereintrag. Ein neuer Eintrag wird nur erstellt, wenn kein inaktiver (gelöschter) Eintrag vorhanden ist. Wurde der Benutzer korrekt hinzugefügt, erscheint nach dem Aufrufen von **Set user properties** in *BenutzerNr* die dazugehörige einmalige Kennziffer.

Übergeben Sie weder -1, -2 noch eine gültige Kennziffer, hat **Set user properties** keine Auswirkung.

Vor dem Aufrufen übergeben Sie für den Benutzer den neuen Namen, die Startup Methode, das Kennwort, die Anzahl der Benutzungen und das Datum der letzten Benutzung in den entsprechenden Parametern. In *Kennwort* übergeben Sie ein unverschlüsseltes Kennwort, das 4D vor dem Sichern verschlüsselt. Mit dem optionalen Symbol * können Sie das bisherige Kennwort beibehalten.
Wollen Sie nicht alle Eigenschaften des Benutzers ändern (mit Ausnahme von Gruppen, Näheres siehe unten), rufen Sie zuerst **GET USER PROPERTIES** auf und übergeben Sie die erhaltenen Werte für die Eigenschaften, die nicht geändert werden sollen.

Übergeben Sie nicht den optionalen Parameter *Gruppen*, bleibt die aktuelle Gruppenzugehörigkeit der Benutzer bestehen. Fügen Sie einen Benutzer ohne den Parameter *Gruppen* hinzu, gehört der Benutzer zu keiner Gruppe.

Mit dem optionalen Parameter *Gruppen* ändern Sie alle Gruppenzugehörigkeiten für den Benutzer. Sie müssen das Array *Gruppen* vor dem Aufrufen von **Set user properties** mit den Kennziffern der Gruppen füllen, zu denen der Benutzer gehört.

Mit dem optionalen Parameter *GrEigentümer* setzen Sie die Kennziffer der Benutzergruppe "Eigentümer", z.B. die Standardgruppe "Eigentümer" für Objekte, die dieser Benutzer angelegt hat. Gruppenkennziffern können folgende Bereiche haben:

Gruppenkennziffer	Beschreibung
15001 bis 32767	Gruppe, angelegt vom Designer oder dem zugeordneten Gruppeneigentümer (Gruppennr. 15001 ist die erste Gruppe Gruppennr. 15002 die zweite usw.)
-15001 bis -32768	Gruppe, angelegt vom Administrator oder dem zugeordneten Gruppeneigentümer (Gruppennr. -15001 ist die erste Gruppe Gruppennr. -15002 die zweite usw.)

Um alle Gruppenzugehörigkeiten aufzuheben, übergeben Sie ein leeres Array *Gruppen*.

Fehlerverwaltung

Haben Sie keine Zugriffsberechtigung zum Aufrufen von **Set user properties** oder wird bereits von einem anderen Prozess aus auf das Kennwortsystem zugegriffen, erhalten Sie eine Fehlermeldung. Sie können diesen Fehler mit dem Befehl **ON ERR CALL** in einer Methode zur Fehlerverwaltung ausfindig machen.

User in group

User in group (Benutzer ; Gruppe) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Benutzer	String	➡	Name des Benutzers
Gruppe	String	➡	Name der Gruppe
Funktionsergebnis	Boolean	⮌	true = Benutzer ist in Gruppe, false = Benutzer ist nicht in Gruppe

Beschreibung

Die Funktion **User in group** gibt den Wert *Wahr* zurück, wenn der Benutzer zu *Gruppe* gehört.

Beispiel

Folgendes Beispiel sucht nach bestimmten Rechnungen. Gehört der Benutzer zur Gruppe Ausführung, kann er auf Formulare mit vertraulichen Informationen zugreifen. Sonst erscheint ein anderes Formular:

```
QUERY([Invoices];[Invoices]Retail>100)
If(User in group(Current user;"Ausführung"))
   FORM SET OUTPUT([Invoices];"Ausführung Ausgabe")
   FORM SET INPUT([Invoices];"Ausführung Eingabe")
Else
   FORM SET OUTPUT([Invoices];"Standard Ausgabe")
   FORM SET INPUT([Invoices];"Standard Eingabe")
End if
MODIFY SELECTION([Invoices];*)
```

USERS TO BLOB

USERS TO BLOB (Benutzer)

Parameter	Typ		Beschreibung
Benutzer	BLOB	⇒	BLOB mit den Benutzern
		⇐	Benutzerkonten (verschlüsselt)

Beschreibung

Der Befehl **USERS TO BLOB** speichert im BLOB *Benutzer* die Liste aller vom Administrator angelegten Benutzerkonten und Datenbankgruppen.

Nur der Administrator oder Designer der Datenbank können diesen Befehl verwenden. Versucht ein anderer Benutzer, ihn einzusetzen, führt er nichts aus. Der Fehler -9949 (kein Zugriffsrecht) wird erzeugt.

Das erstellte BLOB ist automatisch verschlüsselt und nur über den Befehl **BLOB TO USERS** lesbar. Sie können dieses BLOB in einer Datei auf Ihrer Festplatte oder in einem Datenfeld speichern. Dieser Befehl entspricht dem Speichern von Gruppen und Benutzern über die Werkzeugleiste. Der einzige Unterschied ist, dass er Benutzerkonten auch in ein Feld vom Typ BLOB und nicht nur in eine Datei speichern kann.

Auf diese Weise können Sie ein Backup der Benutzer in der Datenbank bewahren und einen Backup-Mechanismus sowie ein System einrichten, das die Benutzer beim Aktualisieren der Strukturdatei der Datenbank automatisch lädt. Informationen zum Benutzerkonto speichert 4D nämlich in der Strukturdatei der Datenbank.

Validate password

Validate password (BenutzerNr ; Kennwort {; Digest}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
BenutzerNr	Lange Ganzzahl, String	➡	Einmalige BenutzerNr
Kennwort	String	➡	Unverschlüsseltes Kennwort
Digest	Boolean	➡	Digest Kennwort = Wahr Volltext Kennwort (Standard)= Falsch
Funktionsergebnis	Boolean	⮌	True = gültiges Kennwort, False = ungültiges Kennwort

Beschreibung

Validate password gibt Wahr zurück, wenn der in *Kennwort* übergebene String das Kennwort für das Benutzerkonto ist mit der Nummer, die in *BenutzerNr* übergeben wurde.

Der optionale Parameter *Digest* gibt an, ob der Parameter *Kennwort* ein unverschlüsseltes Kennwort oder ein verschlüsseltes Kennwort (Digest Modus) enthält:

- Übergeben Sie **Wahr**, enthält *Kennwort* ein verschlüsseltes Kennwort (Digest Modus)
- Übergeben Sie **Falsch** oder lassen diesen Parameter weg, enthält *Kennwort* ein unverschlüsseltes Kennwort

Dieser Parameter ist besonders hilfreich beim Verwenden der **Datenbankmethode On 4D Mobile Authentication**.

Die Funktion wird verzögert ausgeführt, um Flooding (brute force attacks) zu verhindern, also Versuche mit multiplen Benutzername/Kennwort Kombinationen. Ruft 4D diese Funktion auf, läuft sie erst nach 10 Sekunden. Diese Verzögerung gilt für die gesamte Arbeitsstation.

Beispiel 1

Dieses Beispiel prüft, ob das Kennwort des Benutzers “Hardy” oder “Laurel” ist:

```
GET USER LIST(atBenutzerName;alBenutzerID)
$vlElem:=Find in array(atBenutzerName;"Hardy")
If($vlElem>0)
   If(Validate password(alBenutzerID{$vlElem};"Laurel"))
      ALERT("Yep!")
   Else
      ALERT("Fehlanzeige!")
   End if
Else
   ALERT("Name unbekannt ")
End if
```

Beispiel 2

In der **Datenbankmethode On 4D Mobile Authentication** eine Verbindungsanfrage testen, die 4D Benutzer der Anwendung verwendet. Dazu schreiben Sie folgende Anweisung

```
$0:=Validate password($1;$2;$3)
```

Benutzerformulare

- Einführung in Benutzerformulare
- CREATE USER FORM
- DELETE USER FORM
- EDIT FORM
- LIST USER FORMS

Einführung in Benutzerformulare

In 4D können Entwickler Benutzern die Möglichkeit geben, angepasste Formulare zu erstellen oder zu ändern. Diese Benutzerformulare lassen sich dann wie jedes andere 4D Formular verwenden.

Einleitung

Benutzerformulare basieren auf standardmäßigen 4D Formularen, die der Entwickler in der Designumgebung angelegt hat. Solche Formulare heißen Quell- oder Entwicklerformulare, für welche die Eigenschaft **Vom Benutzer editierbar** aktiviert wurde. Der Benutzer kann über einen vereinfachten Formulareditor, der mit dem Befehl **EDIT FORM** aufgerufen wird, die Darstellung des Formulars und der Grafikelemente verändern, Elemente ausblenden, etc. Der Entwickler kann steuern, welche Aktionen zugelassen sind.

Benutzerformulare lassen sich auf folgende Weise einsetzen:

- Der Benutzer ändert das Quellformular und passt es über den Befehl **EDIT FORM** an seine Bedürfnisse an. Das Benutzerformular bleibt lokal erhalten und wird automatisch anstelle des Originalformulars verwendet.
  Dieses Verhalten kommt dem Entwickler entgegen, denn er kann so Parameter in Dialogfenstern einrichten, wenn er beim Kunden vor Ort ist; z.B. ein Firmenlogo in Formularen hinzufügen, unnötige Felder ausblenden, etc.
- Die Quelldatei dient als Vorlage, welche der Benutzer frei duplizieren kann. Über den Befehl **CREATE USER FORM** kann er beliebig viele Kopien erstellen. Jede Kopie lässt sich über den Befehl **EDIT FORM** frei einrichten (Inhalt, Name, etc). Der Name jedes Formulars muss jedoch einmalig sein. Um diese Kopien zu verwenden, müssen Sie die Befehle **FORM SET INPUT** und **FORM SET OUTPUT** verwenden, die jetzt den Namen eines Benutzerformulars als Parameter akzeptieren.
  Mit dieser Funktionsweise kann der Entwickler z.B. für Benutzer angepasste Berichte erstellen.

Benutzerformulare speichern und verwalten

Benutzerformulare funktionieren in kompilierten und interpretierten Datenbanken, mit 4D im lokalen Modus, 4D Server oder 4D Desktop. Im Client/Server-Betrieb sind vom Benutzer veränderte Formulare auf allen Rechnern verfügbar.
4D verwaltet Änderungen in Formularen automatisch. Ein Formular mit der Eigenschaft **Vom Benutzer editierbar** wird in der Designumgebung gesperrt, d.h. es wird mit dem Symbol Vorhängeschloss gekennzeichnet. Der Entwickler muss erst die Sperre aufheben, wenn er auf Objekte in diesem Formular zugreifen will. Die damit verknüpften Benutzerformulare werden ungültig und müssen wieder generiert werden. Wird ein Quellformular gelöscht, werden alle dazugehörigen Benutzerformulare auch gelöscht.

Benutzerformulare werden in einer eigenen Datei mit der Endung .4DA gespeichert. Sie liegt auf derselben Ebene wie die Strukturdatei (.4DB/.4DC). Das Verhalten dieser Datei ist transparent: 4D verwendet ein Benutzerformular, sofern vorhanden. Der Befehl **LIST USER FORMS** findet jederzeit gültige Benutzerformulare. In dieser Datei suchen auch die Befehle **FORM SET INPUT** und **FORM SET OUTPUT** nach Benutzerformularen. Überflüssige Benutzerformulare werden entfernt, 4D verwendet dann standardmäßig das Quellformular.

Im Client/Server-Betrieb wird die Datei .4DA nach denselben Regeln wie die Hauptstrukturdatei verteilt.
Auf diese Weise müssen Benutzerformulare nicht neu generiert werden, wenn der Entwickler die Strukturdatei aktualisiert.

Fehlermeldungen

Beim Verwenden von Befehlen zur Verwaltung von Benutzerformularen können spezifische Fehler auftreten. Sie liegen im Bereich -9750 bis -9759. Weitere Informationen dazu finden Sie im Abschnitt **Fehler der Datenbank (-10602 -> 4004)**.

Benutzerformulare und Projektformulare

Die Arbeitsweise von Benutzerformularen ist nicht länger kompatibel mit Projektformularen. Von daher sind die Befehle unter dem Thema "Benutzerformulare" nicht für Projektformulare verwendbar.

CREATE USER FORM

CREATE USER FORM (Tabellenname ; Formularname ; Benutzerformular)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Quellformular Tabelle
Formularname	String	➡	Quellformular Name
Benutzerformular	String	➡	Name des neuen Benutzerformulars

Beschreibung

Der Befehl **CREATE USER FORM** dupliziert das 4D Tabellenformular *Formular*, dessen Tabelle und Name als Parameter übergeben wurden und erstellt ein neues Benutzerformular mit Namen *Benutzerformular*.

Ein einmal erstelltes Benutzerformular lässt sich über den Befehl **EDIT FORM** verändern. Damit können Sie aus einem einzigen Quellformular beliebig viele Benutzerformulare erstellen, z.B. verschiedene Berichtformulare.

Systemvariablen und Mengen

Bei korrekt ausgeführter Operation wird die Variable OK auf 1 gesetzt.

Fehlerverwaltung

Ein Fehler wird generiert, wenn

- *Formularname* bereits ein Benutzerformular ist,
- Der Name von *BenutzerFormular* genauso lautet wie der Name des Quellformulars oder eines bereits vorhandenen Benutzerformulars.
- Der Benutzer nicht auf das Formular zugreifen kann, weil er nicht dazu berechtigt ist.

Sie können diese Fehler mit der Fehlerverwaltungsmethode abfangen, die über den Befehl **ON ERR CALL** installiert wird.

DELETE USER FORM

DELETE USER FORM (Tabellenname ; Formularname ; Benutzerformular)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle Benutzerformular
Formularname	String	⇒	Quelle Formularname
Benutzerformular	String	⇒	Name Benutzerformular

Beschreibung

Der Befehl **DELETE USER FORM** entfernt das Benutzerformular, das mit den Parametern *Tabellenname*, *Formular* und *Benutzerformular* eingerichtet wurde.

Systemvariablen und Mengen

Wurde das Benutzerformular korrekt entfernt, gibt die Variable OK den Wert 1 zurück. Andernfalls wird OK wird auf 0 (Null) gesetzt.

Fehlerverwaltung

Ein Fehler wird generiert, wenn

- Das Benutzerformular nicht existiert oder *Benutzerformular* einen leeren String enthält (-9757)
- Der Benutzer keine Zugangsberechtigung hat, um das Formulars zu entfernen. Sie können diesen Fehler abfangen mit der Fehlerverwaltungsmethode, installiert über den Befehl **ON ERR CALL**.

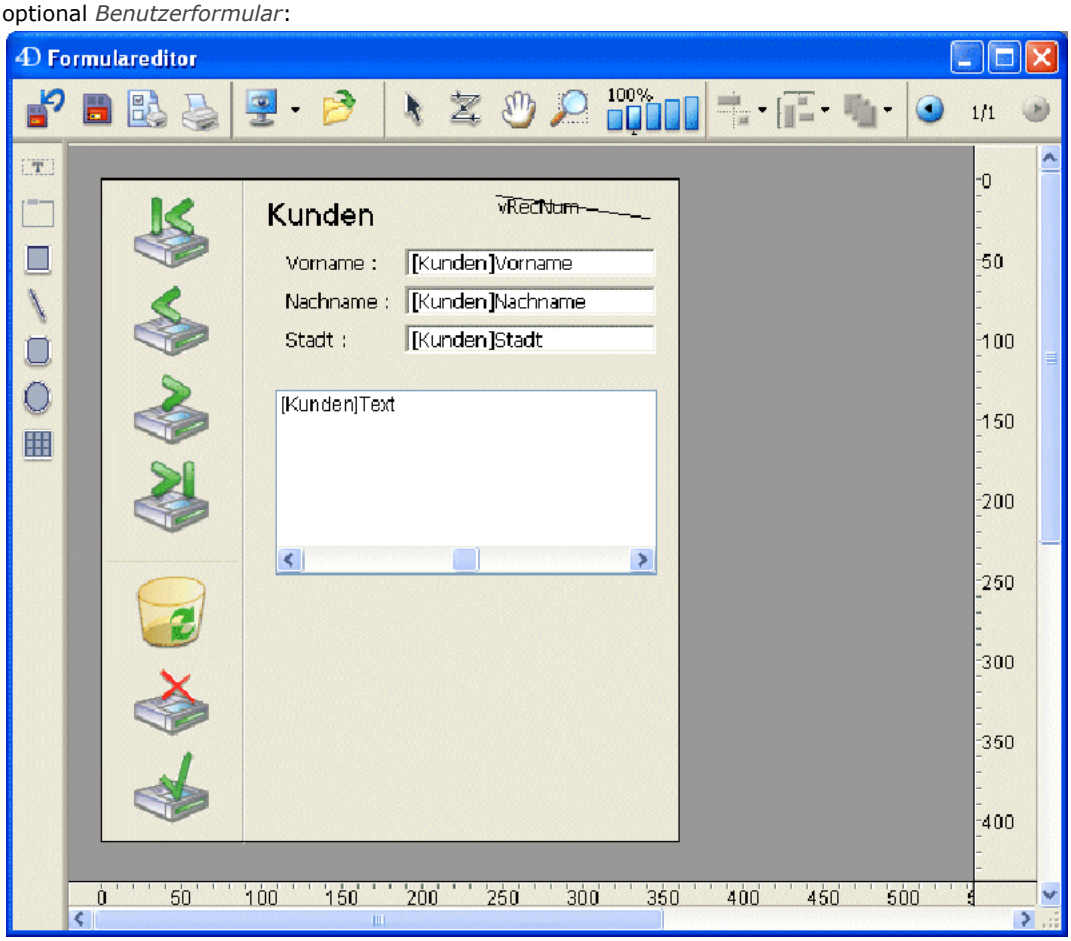
EDIT FORM

EDIT FORM (Tabellenname ; Formularname {; Benutzerformular {; Bibliothek}})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle des zu ändernden Formulars
Formularname	String	➡	Name des zu ändernden Formulars
Benutzerformular	String	➡	Name des zu ändernden Benutzerformulars
Bibliothek	String	➡	Pfadname der verwendbaren Objektbibliothek

Beschreibung

Der Befehl **EDIT FORM** öffnet das Tabellenformular im Benutzerformular-Editor über die Parameter *Tabellenname*, *Formular* und optional *Benutzerformular*:



Hinweis: Das Editorfenster öffnet sich nur, wenn es das erste Fenster im Prozess ist. Folglich müssen Sie einen neuen Prozess öffnen, damit der Editor angezeigt wird.

Übergeben Sie im Parameter *Benutzerformular* einen leeren String und ist mit Formular noch kein Benutzerformular verknüpft, erscheint das Quellformular im Editor. Das geänderte Formular wird dann in die Benutzerstrukturdatei (.4DA) kopiert und zum Ersetzen in Formular verwendet.
Wurde über **EDIT FORM** bereits ein Benutzerformular generiert, erscheint das Benutzerformular im Editor. Wollen Sie vom Quellformular aus starten, müssen Sie zuerst über den Befehl **DELETE USER FORM** das Benutzerformular löschen.

Über den Parameter *Benutzerformular* können Sie ein änderbares Benutzerformular setzen, das über **CREATE USER FORM** erstellt wurde. In diesem Fall erscheint das Formular im Editor.

In *Bibliothek* übergeben Sie den kompletten Pfadnamen für die Objektbibliothek, die zum eigenen Gestalten des Formulars zugelassen ist. Im Client/Server-Betrieb muss die Bibliothek im Ordner **Resources** der Datenbank auf derselben Ebene wie der Ordner **PlugIns** liegen, damit er für alle Client-Rechner verfügbar ist. Ist die Bibliothek gültig, wird sie mit dem Formularfenster geöffnet. Weitere Informationen dazu finden Sie im Abschnitt **Objektbibliothek verwenden** des Handbuchs *4D Designmodus*.

Beispiel

Der Benutzer kann eine Bibliothek wählen und dann ein Dialogformular ändern:

```
MAP FILE TYPES("4DLB";4IL";"4D Library
$vAlib:=Select document(1;"4DLB";"Eine Bibliothek auswählen";0)
If(OK=1)
  `Es wurde eine Bibliothek ausgewählt
   $vALibPath=Document
Else
```

```
    $vALibPath=""
End if
```

```
EDIT FORM([Dialogs];"Willkommen";"Lib_Logos.4il")
If(OK=1)
    Anzeige des geänderten Formulars
    DIALOG([Dialogs];"Willkommen")
End if
```

Systemvariablen und Mengen

Speichert der Benutzer die im Formular ausgeführten Änderungen, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf, wird OK auf 0 (Null) gesetzt.

Fehlerverwaltung

Ein Fehler wird generiert, wenn:

- Der Benutzer das Formular in der Designumgebung nicht ändern kann, oder es kein Formular gibt,
- Das Formular bereits geöffnet ist und in einem anderen Prozess geändert wurde,
- Der Benutzer nicht auf das Formular zugreifen kann, weil er nicht dazu berechtigt ist.
  Sie können diesen Fehler abfangen mit einer Fehlerverwaltungsmethode, die über den Befehl **ON ERR CALL** installiert wird.

LIST USER FORMS

LIST USER FORMS (Tabellenname ; Formularname ; ArrayBenutzerFormular)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Quellformular Tabelle
Formularname	String	⇒	Quellformular Name
ArrayBenutzerFormular	Array String	⇐	Namen der Benutzerformulare, die vom Quellformular stammen

Beschreibung

Der Befehl **LIST USER FORMS** füllt *ArrayBenutzerFormular* mit den Namen der Benutzerformulare, die vom Entwicklerformular (Tabellenformular) stammen, das mit den Parametern *Tabellenname* und *Formular* gesetzt wurde.

Wurde das Benutzerformular direkt mit dem Befehl **EDIT FORM** erstellt, enthält *ArrayBenutzerFormular* als einzigen Eintrag einen leeren String (""). Das Array ist leer, wenn es zum angegebenen Entwicklerformular keine Benutzerformulare gibt.

Benutzeroberfläche

- 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
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **BEEP** erzeugt auf PC oder Macintosh ein Beep. Je nach dem Wert, der im Kontrollfeld *Ton* eingestellt ist, kann Ihr Rechner auch andere Töne spielen.

Warnung: Rufen Sie **BEEP** nicht innerhalb eines Prozesses einer Web-Verbindung auf, da der Beep auf dem Web-Server Rechner von 4D aufgerufen wird und nicht auf dem Web-Client Rechner.

Beispiel

Im folgendes Beispiel wird ein Beep ausgegeben, wenn im Suchlauf keine Datensätze gefunden werden. Anschließend erscheint eine Meldung:

```
QUERY([Customers];[Customers]Name=$vsNameToLookFor)
If(Records in selection([Customers])=0)
   BEEP
   ALERT("Es gibt keinen Kunden mit diesem Namen.")
End if
```

Caps lock down

Caps lock down -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Status der Feststelltaste

Beschreibung

Die Funktion **Caps lock down** gibt bei gedrückter **Feststelltaste** den Wert *Wahr* zurück.

Beispiel

Siehe Beispiel zur Funktion **Shift down**.

Focus object

Focus object -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Zeiger		Zeiger auf das Objekt mit Fokus

Hinweis zur Kompatibilität

Dieser Befehl wird zur Wahrung der Kompatibilität beibehalten. Ab 4D Version 12 empfehlen wir, den Befehl **OBJECT Get pointer** zu verwenden.

Beschreibung

Die Funktion **Focus object** gibt einen Zeiger auf das Objekt mit Fokus im aktuellen Formular zurück. Hat kein Objekt Fokus, gibt der Befehl **Is nil pointer** zurück. Mit **Focus object** können Sie eine Aktion in einem Formularbereich ausführen, ohne zu wissen, welches Objekt aktuell ausgewählt ist. Vor Ausführen einer Aktion sollten Sie jedoch mit **Type** prüfen, ob das Objekt den korrekten Datentyp hat.

Hinweis: Bei Verwendung mit einer Listbox gibt die Funktion **Focus object** je nach Kontext einen Zeiger auf die Listbox oder die Spalte der Listbox mit Fokus zurück. Weitere Informationen dazu finden Sie im Abschnitt **Einführung in Listboxen**.

Diese Funktion lässt sich nicht in Feldern von Unterformularen benutzen.

Hinweis: Die Funktion ist nur bei der Dateneingabe sinnvoll, sonst wird ein Fehler zurückgegeben.

Beispiel

Folgendes Beispiel ist eine Objektmethode für eine Schaltfläche. Sie ändert die Daten im aktuellen Objekt um in Großbuchstaben. Das Objekt muss vom Typ String oder Text sein (Typ 0 oder 24):

```
$vp :=Focus object  ` Sichere Zeiger auf den letzten Bereich
Case of
   :(Nil($pointer))  ` Kein Objekt hat den Fokus
      ...
   :((Type($vp->)=ls alpha field)|(Type($vp->)=ls string var))
 ` Ist es ein String oder Textbereich
      $vp->:=Uppercase($vp->)  ` Ändere Bereich um in Großbuchstaben
End case
```

GET FIELD TITLES

GET FIELD TITLES (Tabellenname ; TitelFelder ; NumFelder)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle, deren Feldnamen Sie herausfinden wollen.
TitelFelder	Array Text	⇐	Aktuelle Feldnamen
NumFelder	Array Lange Ganzzahl	⇐	Feldnummern

Beschreibung

Der Befehl **GET FIELD TITLES** füllt die Arrays *TitelFelder* und *NumFelder* mit den Namen und Nummern der Datenfelder für *Tabellenname*. Der Inhalt der beiden Arrays wird aufeinander abgestimmt.

Wird **SET FIELD TITLES** während einer Sitzung aufgerufen, gibt **GET FIELD TITLES** nur die Feldnamen und Nummern zurück, die dieser Befehl definiert hat.
Sonst gibt **GET FIELD TITLES** die Namen aller Felder der Datenbank zurück, die im Strukturfenster definiert wurden.
Der Befehl gibt in beiden Fällen keine ausgeblendeten Tabellen zurück.

GET MOUSE

GET MOUSE (MausX ; MausY ; Maustaste {; *})

Parameter	Typ		Beschreibung
MausX	Zahl	⇐	Horizontale Koordinate der Maus
MausY	Zahl	⇐	Vertikale Koordinate der Maus
Maustaste	Lange Ganzzahl	⇐	Status der Maustaste: 0 = Taste oben, 1 = Taste unten, 2 = Rechte Taste unten, 3 = Beide Tasten unten
*	Operator	⇒	Mit * wird das globale Koordinatensystem verwendet, ohne * das lokale Koordinatensystem

Beschreibung

Der Befehl **GET MOUSE** gibt den aktuellen Status der Maus zurück.

MausX und *MausY* geben die horizonalen und vertikalen Koordinaten zurück. Mit dem optionalen Parameter * werden sie in Bezug auf den Hauptbildschirm angegeben (macOS und SDI Modus unter Windows) oder in Bezug auf das Anwendungsfenster (MDI Modus unter Windows); ohne den optionalen Parameter * in Bezug auf das aktuelle Formularfenster (falls vorhanden) des aktuellen Prozesses.
Der Parameter *Maustaste* gibt den Status der Maustaste an, wie oben aufgelistet.

Hinweis: Die Werte 2 und 3 können auf OS X ab Version 10.2.5 zurückgegeben werden.

Beispiel

Siehe Beispiel zur Funktion **Pop up menu**.

GET TABLE TITLES

GET TABLE TITLES (TitelTabellen ; NumTabellen)

Parameter	Typ		Beschreibung
TitelTabellen	Array Text	⇐	Aktuelle Tabellennamen
NumTabellen	Array Lange Ganzzahl	⇐	Tabellennummern

Beschreibung

Der Befehl **GET TABLE TITLES** füllt die Arrays *TitelTabellen* und *NumTabellen* mit den Namen und Nummern der Tabellen, die im Strukturfenster oder über den Befehl **SET TABLE TITLES** definiert wurden. Der Inhalt der beiden Arrays wird aufeinander abgestimmt.

Wird **SET TABLE TITLES** während einer Sitzung aufgerufen, gibt **GET TABLE TITLES** nur die Tabellennamen und Nummern zurück, die dieser Befehl definiert hat.
Sonst gibt **GET TABLE TITLES** die Namen aller Tabellen der Datenbank zurück, die im Strukturfenster definiert wurden.
Der Befehl gibt in beiden Fällen keine ausgeblendeten Tabellen zurück.

HIDE MENU BAR

HIDE MENU BAR
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **HIDE MENU BAR** blendet die Menüleiste aus.
Ist die Menüleiste bereits ausgeblendet, hat der Befehl keine Auswirkung.

Beispiel

Folgende Methode zeigt einen Datensatz bis zum Mausklick auf dem ganzen Bildschirm (Macintosh):

```
HIDE TOOL BAR
HIDE MENU BAR
Open window(-1;-1;1+Screen width;1+Screen height;Alternate dialog box)
FORM SET INPUT([Paintings];"Full Screen 800")
DISPLAY RECORD([Paintings])
Repeat
   GET MOUSE($vlX;$vlY;$vlButton)
Until($vlButton#0)
CLOSE WINDOW
SHOW MENU BAR
SHOW TOOL BAR
```

Hinweis: Unter Windows richtet sich die Größe nach dem Anwendungsfenster.

Macintosh command down

Macintosh command down -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Status der Befehlstaste auf Macintosh (Strg-Taste unter Windows)

Beschreibung

Die Funktion **Macintosh command down** gibt bei gedrückter **Befehlstaste** auf Macintosh den Wert *Wahr* zurück.
Rufen Sie **Macintosh command down** unter Windows auf, gibt die Funktion den Wert *Wahr* bei gedrückter **Strg-Taste** zurück.

Beispiel

Siehe Beispiel zum Befehl **Shift down**.

Macintosh control down

Macintosh control down -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		Status der ctrl-Taste auf Macintosh

Beschreibung

Die Funktion **Macintosh control down** gibt bei gedrückter **ctrl-Taste** auf Macintosh den Wert *Wahr* zurück.
Rufen Sie **Macintosh control down** unter Windows auf, gibt die Funktion immer den Wert *Falsch* zurück, da es unter Windows keine Entsprechung gibt.

Beispiel

Siehe Beispiel zur Funktion **Shift down**.

Macintosh option down

Macintosh option down -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		Status der Wahltaste auf Macintosh (alt-Taste unter Windows)

Beschreibung

Die Funktion **Macintosh option down** gibt bei gedrückter **Wahltaste** auf Macintosh den Wert *Wahr* zurück.
Rufen Sie **Macintosh option down** unter Windows auf, gibt die Funktion den Wert *Wahr* bei gedrückter **alt-Taste** zurück.

Beispiel

Siehe Beispiel zum Befehl **Shift down**.

PLAY

PLAY (ObjektName {; async})

Parameter	Typ		Beschreibung
ObjektName	String	➡	Name oder Pfad von Tondatei oder Systemton, leerer String zum Stoppen bei asynchronem Abspielen
async	Lange Ganzzahl	➡	(Windows) mit Angabe asynchrone Ausführung; ohne Angabe synchrone Ausführung

Beschreibung

Der Befehl PLAY spielt Ton- oder Multimedia-Dateien ab. In *ObjektName* übergeben Sie den kompletten Pfadnamen der Datei, die Sie abspielen möchten. Auf OS X ist der Befehl auch zum Abspielen eines Systemtons verwendbar.

- Um eine Datei abzuspielen, übergeben Sie ihren Namen und Pfadnamen im *ObjektName*. Sie können einen vollen Pfadnamen oder einen Namen in Bezug auf die Strukturdatei der Anwendung angeben.
  Unterstützt werden die gängigen Ton- und Multimedia Dateiformate wie .WAV, .MP3, AIFF (OS X). Auf OS X unterstützt der Befehl insbesondere Core Audioformate.
- (nur OS X): Um einen Systemton zu spielen, übergeben Sie den Namen direkt im Parameter *ObjektName*.

Hinweis: 'snd' Ressourcen, wie auf Mac OS 9 und älter verwendet, werden nicht mehr unterstützt

Der Parameter *async* gibt unter Windows an, asynchron zu spielen. Synchron bedeutet, dass die Bearbeitungsprozesse gestoppt werden, bis der Ton ganz abgespielt ist. Asynchron bedeutet, dass die Bearbeitungsprozesse nicht gestoppt werden und der Ton im Hintergrund läuft. Ist *asnc* übergeben und hat den Wert 0 (oder eine andere Lange Ganzzahl), läuft der Ton asynchron. Ist *asnc* nicht übergeben, läuft der Ton synchron.

Hinweis: Auf OS X wird der Ton immer asynchron abgespielt, unabhängig ob mit oder ohne den Parameter *async*

Folgende Anweisung stoppt das Abspielen von asynchronem Ton:

```
PLAY("";0)
```

Beispiel 1

Folgendes Beispiel zeigt, wie Sie eine WAV Datei auf Windows spielen:

```
$DocRef :=Open document("";"WAV";Read Mode)
If(OK=1)
   CLOSE DOCUMENT($DocRef)
   PLAY(Document;0)  //asynchron spielen
End if
```

Beispiel 2

Folgende Anweisung spielt einen Systemton auf OS X:

```
PLAY("Submarine.aiff")
```

Pop up menu

Pop up menu (Inhalt {; Standard {; xKoord ; yKoord}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Inhalt	Text	➡	Definition des Menütextes
Standard	Lange Ganzzahl	➡	Standardmäßig gewählte Menüzeilennummer
xKoord	Lange Ganzzahl	➡	X Koordinate der oberen linken Ecke
yKoord	Lange Ganzzahl	➡	Y Koordinate der oberen linken Ecke
Funktionsergebnis	Lange Ganzzahl	↩	Nummer der gewählten Menüzeile

Beschreibung

Die Funktion **Pop up menu** zeigt ein PopUp-Menu an der aktuellen Mausposition.

Um unter anderem die Regeln für die Benutzeroberfläche zu befolgen, rufen Sie diese Funktion normalerweise als Antwort auf einen Mausklick bei noch gedrückter Maustaste auf.

Sie definieren die Zeilen des PopUp-Menüs mit dem Parameter *Inhalt* folgendermaßen:

- Trennen Sie die Zeilen durch Strichpunkt (*;*) voneinander.
  Beispiel, *"ZeileText1;ZeileText2;ZeileText3"*.
- Deaktivieren Sie eine Zeile durch eine offene Klammer (*(*) im Zeilentext.
- Übergeben Sie für eine Trennungslinie als Zeilentext die Zeichen "-" oder "(-".
- Setzen Sie für die Zuordnung einer Schriftart in einer Zeile im Zeilentext ein Kleiner-als-Zeichen (<) gefolgt von einem dieser Buchstaben:
  - <B Fett
  - <I Kursiv
  - <U Unterstrichen
  - <O Kontur (nur auf Macintosh)
  - <S Schatten (nur auf Macintosh)
- Setzen Sie für die Zuordnung einer Marke vor einer Zeile im Zeilentext ein Ausrufezeichen (*!*) gefolgt von dem gewünschten Zeichen.
  - Unter Windows wird die Marke unabhängig vom übergebenen Zeichen angezeigt
  - Auf Macintosh erscheint das eingegebene Zeichen direkt. Um eine Standardmarke, unabhängig von der Systemversion oder Sprache anzuzeigen, wählen Sie die Anweisung *Char (18)*.
- Setzen Sie für die Zuordnung eines Icon für eine Zeile einen Circumflex Akzent (^) gefolgt von einem Zeichen. Sein Code minus 48 plus 256 bzw. plus 208 ist die Ressourcen ID einer auf Mac OS-basierenden Icon Ressource.
- Setzen Sie für die Zuordnung eines Tastaturkürzels für eine Zeile einen rechtsgerichteten Schrägstrich (*/*) gefolgt von dem Kürzel für die Zeile. Beachten Sie, dass diese letzte Option rein informativ ist; ein Tastaturkürzel aktiviert nicht das PopUp-Menü. Es bietet sich jedoch an, ein Tastaturkürzel zu integrieren, wenn es für die Zeile des PopUp-Menüs eine Entsprechung in der Hauptmenüleiste Ihrer Anwendung gibt.

Tipp: Sie können die Operation zum Interpretieren von Sonderzeichen (!, /, etc.) im PopUp-Menü deaktivieren, um diese Zeichen im Text einfügen zu können. Dazu müssen Sie lediglich den *Inhalt* mit der Anweisung *Char(1)* beginnen lassen und diese Anweisung als Trenner verwenden. Zum Beispiel:Inhalt:=Char(1)+"1/4"+Char(1)+1/2+Char(1)+3/4"

Beachten Sie, dass Sie nach Ausführen dieser Anweisung dem PopUp-Menü nicht mehr Stilarten oder Tastenkürzel zuweisen können.

Mit dem optionalen Parameter *Standard* können Sie die Standardmenüzeile festlegen, die beim Anzeigen des PopUp-Menüs ausgewählt wird. Übergeben Sie einen Wert zwischen 1 und der Anzahl der Menüzeilen. Geben Sie diesen Parameter nicht an, wählt die Funktion als Standard die erste Menüzeile.

Über die optionalen Parameter *xKoord* und *yKoord* können Sie die Position des anzuzeigenden PopUp-Menüs bestimmen. Sie übergeben jeweils die horizontale und vertikale Koordinate der oberen linken Ecke des Menüs. Die Angabe erfolgt in Pixel im lokalen Koordinatensystem des aktuellen Formulars. Sie müssen beide Parameter angeben; bei nur einem Parameter wird die Einstellung ignoriert.
Setzen Sie *xKoord* und *yKoord* ein, wird der Parameter *Standard* ignoriert. In diesem Fall liegt die Maus nicht unbedingt auf der Ebene des PopUp-Menüs.
Diese Parameter sind besonders hilfreich, um 3D Schaltfläche mit einem zugeordneten PopUp-Menü zu verwalten.

Wählen Sie eine Menüzeile, gibt die Funktion deren Nummer an; sonst gibt sie den Wert Null (0) zurück.

Hinweis: Verwenden Sie PopUp-Menüs mit einer vernünftigen Anzahl Zeilen. Bei mehr als 50 Zeilen sollten Sie anstelle eines PopUp-Menüs einen rollbaren Bereich verwenden.

Beispiel

Die Projektmethode *MY SPEED MENU* durchläuft ein Kontextmenü:

```
` Projektmethode MY SPEED MENU
GET MOUSE($vlMouseX;$vlMouseY;$vlButton)
If(Macintosh control down|($vlButton=2))
   $vtItems:="Über diese Datenbank...<I;(-;!-Andere Optionen;(-"
   For($vlTable;1;Get last table number)
      If(Is table number valid($vlTable))
         $vtItems:=$vtItems+";"+Table name($vlTable)
```

```
        End if
    End for
    $vlUserChoice:=Pop up menu($vtItems)
    Case of
        :($vlUserChoice=1)
    ` Zeige Information
        :($vlUserChoice=3)
    ` Zeige Optionen
        Else
            If($vlUserChoice>0)
    ` Gehe zu Tabelle mit der Nummer $vlUserChoice-4
            End if
    End case
End if
```

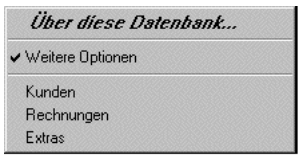
Sie können diese Projektmethode aufrufen in:

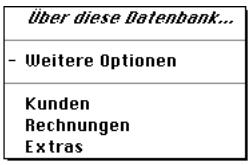
- Der Methode eines Formularobjekts, das auf einen Mausklick reagiert, ohne das Loslassen der Maustaste abzuwarten (z.B. eine unsichtbare Schaltfläche)
- Einem Prozess, der nach Ereignissen "spioniert" und mit anderen Prozessen kommuniziert.
- Einer mit **ON ERR CALL** installierten Methode zur Verwaltung von Ereignissen.

In den beiden letzten Fällen muss der Mausklick nicht in einem Formularobjekt auftreten. Das ist einer der Vorteile der Funktion **Pop up menu**. Zum Anzeigen von PopUp-Menüs verwenden Sie im allgemeinen Formularobjekte. Mit **Pop up menu** lässt sich das Menü überall anzeigen.

Unter Windows erscheint das PopUp-Menü, wenn Sie die rechte Maustaste drücken; auf Macintosh mit der Tastenkombination **Ctrl+Klick**. Beachten Sie jedoch, dass die Methode nicht überprüft, ob ein Mausklick stattgefunden hat; das prüft die aufrufende Methode.

Im Folgenden sehen Sie links das PopUp-Menü unter Windows, rechts das PopUp-Menü auf Macintosh. Beachten Sie unter Windows die Standardmarke vor der Zeile.





POST CLICK

POST CLICK (MausX ; MausY {; Prozess} {; *})

Parameter	Typ		Beschreibung
MausX	Lange Ganzzahl	➡	Horizontale Koordinate
MausY	Lange Ganzzahl	➡	Vertikale Koordinate
Prozess	Lange Ganzzahl	➡	Referenznummer des Zielprozesses, ohne Angabe Ereignisschleife der Anwendung oder 0
*		➡	ohne Angabe wird das lokale Koordinatensystem verwendet, mit Angabe das globale System

Beschreibung

Der Befehl **POST CLICK** simuliert einen Mausklick. Er hat denselben Effekt wie das Betätigen der Maustaste.

In *MausX* und *MausY* übergeben Sie die horizontalen und vertikalen Koordinaten des Klicks.
Mit dem optionalen Parameter * werden sie in Bezug auf den Bildschirm angegeben, ohne den optionalen Parameter * in Bezug auf das vorderste Fenster des Prozesses mit der in *Prozess* übergebenen Nummer.

Geben Sie den Parameter *Prozess* an, wird der Klick zum Prozess mit der in *Prozess* übergebenen Nummer gesendet. Übergeben Sie den Wert Null (*0*) oder keinen Parameter, wird der Klick in die Anwendungsebene gesendet. Der 4D Kernel übermittelt ihn dann an den obersten Prozess.

POST EVENT

POST EVENT (Art ; Meldung ; Zeit ; MausX ; MausY ; Zusatztaste {; Prozess})

Parameter	Typ		Beschreibung
Art	Lange Ganzzahl	➡	Ereignisart
Meldung	Lange Ganzzahl	➡	Ereignismeldung
Zeit	Lange Ganzzahl	➡	Ereigniszeit in Ticks
MausX	Lange Ganzzahl	➡	Horizontale Koordinate der Maus
MausY	Lange Ganzzahl	➡	Vertikale Koordinate der Maus
Zusatztaste	Lange Ganzzahl	➡	Status der Zusatztasten
Prozess	Lange Ganzzahl	➡	Referenznummer des Zielprozesses, ohne Angabe Ereignisschleife der Anwendung oder 0

Beschreibung

Der Befehl **POST EVENT** simuliert ein Ereignis der Tastatur oder der Maus. Er hat denselben Effekt wie eine Aktion des Benutzers auf der Tastatur oder mit der Maus.

In *Art* übergeben Sie eine der folgenden Konstanten:

Konstante	Typ	Wert
Auto key event	Lange Ganzzahl	5
Key down event	Lange Ganzzahl	3
Key up event	Lange Ganzzahl	4
Mouse down event	Lange Ganzzahl	1
Mouse up event	Lange Ganzzahl	2

Bezieht sich das Ereignis auf die Maus, übergeben Sie in *Meldung* den Wert Null (*0*). Bezieht es sich auf die Tastatur, übergeben Sie in *Meldung* den Code des simulierten Zeichens.

Normalerweise übergeben Sie in *Zeit* den von **Tickcount** zurückgegebenen Wert.

Bezieht sich das Ereignis auf die Maus, übergeben Sie in *MausX* und *MausY* jeweils die horizontale und die vertikale Koordinate.

Mit dem Parameter *Zusatztaste* können Sie eine oder mehrere Konstanten unter dem Thema **Ereignisse (Zusatztasten)** übergeben:

Konstante	Typ	Wert	Kommentar
Activate window bit	Lange Ganzzahl	0	
Activate window mask	Lange Ganzzahl	1	
Caps lock key bit	Lange Ganzzahl	10	Windows und OS X
Caps lock key mask	Lange Ganzzahl	1024	Windows und OS X
Command key bit	Lange Ganzzahl	8	Ctrl Taste unter Windows, Befehlstaste auf OS X
Command key mask	Lange Ganzzahl	256	Strg-Taste unter Windows, Befehlstaste auf OS X
Control key bit	Lange Ganzzahl	12	Ctrl-Taste auf OS X, oder rechter Mausklick unter Windows und OS X
Control key mask	Lange Ganzzahl	4096	Ctrl-Taste auf OS X, oder rechter Mausklick unter Windows und OS X
Mouse button bit	Lange Ganzzahl	7	
Mouse button mask	Lange Ganzzahl	128	
Option key bit	Lange Ganzzahl	11	Alt Taste (Wahltaste unter OS X)
Option key mask	Lange Ganzzahl	2048	Windows = Alt-Taste, Mac OS = Wahltaste
Right control key bit	Lange Ganzzahl	15	
Right control key mask	Lange Ganzzahl	32768	
Right option key bit	Lange Ganzzahl	14	
Right option key mask	Lange Ganzzahl	16384	
Right shift key bit	Lange Ganzzahl	13	
Right shift key mask	Lange Ganzzahl	8192	
Shift key bit	Lange Ganzzahl	9	Windows und OS X
Shift key mask	Lange Ganzzahl	512	Windows und Mac OS

Geben Sie den Parameter *Prozess* an, wird das Ereignis zum Prozess mit der in *Prozess* übergebenen Nummer gesendet. Übergeben Sie den Wert Null (0) oder keinen Parameter, wird der Klick in die Anwendungsebene gesendet. Der 4D Kernel übermittelt ihn dann an den entsprechenden Prozess.

POST KEY

POST KEY (Code {; Zusatztaste {; Prozess}})

Parameter	Typ		Beschreibung
Code	Lange Ganzzahl	⇒	ASCII Code für Zeichen oder Funktionstaste
Zusatztaste	Lange Ganzzahl	⇒	Status der Zusatztasten
Prozess	Lange Ganzzahl	⇒	Referenznummer des Zielprozesses, Ohne Angabe Ereignisschleife der Anwendung oder 0

Beschreibung

Der Befehl **POST KEY** simuliert eine Eingabe auf der Tastatur. Er hat denselben Effekt wie das Eingeben eines Zeichens über Tastatur.

In *Code* übergeben Sie den Code des Zeichens.

Mit dem Parameter *Zusatztaste* können Sie eine oder mehrere Konstanten für Ereignisse (Zusatztasten) übergeben. Wollen Sie zum Beispiel die Umschalttaste simulieren, übergeben Sie Shift key mask. Ohne den Parameter *Zusatztaste* werden keine Zusatztasten simuliert.

Mit dem Parameter *Prozess* wird die Eingabe zum Prozess mit der in *Prozess* übergebenen Prozessnummer gesendet. Übergeben Sie den Wert Null (0) oder keinen Parameter, wird die Eingabe in die Anwendungsebene gesendet. Der 4D Kernel übermittelt diese dann an den obersten Prozess.

Beispiel

Siehe Beispiel zum Befehl **Process number**.

REDRAW

REDRAW (Objekt)

Parameter	Typ		Beschreibung
Objekt	Formularobjekt	⇒	Tabelle zum Neuzeichnen des Unterformulars; Feld bzw. Variable zum Neuzeichnen des Bereichs; Listbox zum Aktualisieren

Beschreibung

In einer Methode, die den Wert eines Datenfelds in einem Unterformular verändert, müssen Sie die Funktion **REDRAW** ausführen, damit das Formular aktualisiert wird. .

Wird eine **REDRAW** Anweisung auf ein Objekt vom Typ Listbox angewendet, wenn diese im Auswahlmodus ist, werden die im Objekt angezeigten Daten neu gezeichnet. Diese Anweisung muss typischerweise aufgerufen werden, wenn in den Datensätzen der Auswahl Änderungen eingetreten sind.

SET ABOUT

SET ABOUT (NeuerText ; Methodenname)

Parameter	Typ		Beschreibung
NeuerText	String	⇒	Neuer Text der Menüzeile Über...
Methodenname	String	⇒	Name der auszuführenden Methode bei Aufruf der Menüzeile

Beschreibung

Der Befehl **SET ABOUT** ändert die Menüzeile **Über 4D** im Menü **Hilfe** (Windows) oder im **Anwendungsmenü** (Mac OS X) in *NeuerText*.
Wählt ein Benutzer diese Menüzeile dann im Design- oder Anwendungsmodus auf, wird *Methode* aufgerufen, die das Fenster für die Anzeige des Über-Dialogs öffnet.

Dieser Befehl lässt sich mit 4D im lokalen und im remote Modus, 4D Desktop und 4D Server verwenden. Bei Ausführung auf einem Server-Rechner wird ein neuer Prozess erstellt.

Beispiel 1

Folgendes Beispiel ersetzt den Menübefehl **Über 4D** mit dem Befehl **Über Terminplaner.** Die Methode ***ABOUT*** zeigt ein eigenes Fenster **Über** an:

```
SET ABOUT("Über Terminplaner···";"Über")
```

Beispiel 2

Folgendes Beispiel setzt den Befehl wieder auf **Über 4D** zurück:

```
SET ABOUT("Über 4D";"")
```

SET CURSOR

SET CURSOR {(Cursor)}

Parameter	Typ		Beschreibung
Cursor	Lange Ganzzahl	➡	Ressourcennummer für Cursor

Beschreibung

Der Befehl **SET CURSOR** ändert den Maus-Cursor auf den System-Cursor mit der Kennnummer, übergeben in *Cursor*.
Er muss im Kontext von **Form event**=On Mouse Move aufgerufen werden.

Geben Sie diesen Parameter nicht an, wird der standardmäßige Maus-Cursor angezeigt.

Hier die Symbole, die Sie im Parameter *Cursor* übergeben können:

1
2
4
9000
9001
9003
9004
9005
9006
9021
351
9010
9011
9013
9014
9015
9016
9017
9019
9020
559
560[#/table]

Beispiel

Wenn die Maus über einen Variablenbereich im Formular geht, soll der Cursor angezeigt werden. In der Objektmethode der Variable schreiben Sie folgenden Code:

```
If(Form event=On Mouse Move)
   SET CURSOR(9019)
End if
```

SET FIELD TITLES

SET FIELD TITLES (Tabellenname ; FeldTitel ; FeldNr {; *})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle für Feldtitel
FeldTitel	Array String	⇒	Feldnamen, wie sie in Dialogfenstern erscheinen sollen
FeldNr	Array Lange Ganzzahl	⇒	Aktuelle Feldnummern
*		⇒	Die eigenen Namen im Formeleditor verwenden

Beschreibung

Der Befehl **SET FIELD TITLES** kann Felder in *Tabellenname* ausblenden, umbenennen und umsortieren, wenn sie in den 4D Standarddialogfenstern erscheinen, z.B. im Sucheditor in der Anwendungsumgebung (insbesondere, wenn die Editoren über Befehle der 4D Programmiersprache aufgerufen werden).
Mit diesem Befehl können Sie Feldnamen in Ihren Formularen "on the fly" umbenennen, wenn Sie dynamische Namen verwendet haben. Weitere Informationen dazu finden Sie im Abschnitt **Referenzen in statischem Text verwenden** des Handbuchs *4D Designmodus*.

Die Arrays *FeldTitel* und *FeldNr* müssen synchronisiert sein. Im Array *FeldTitel* übergeben Sie die gewünschten Namen. Wollen Sie ein bestimmtes Feld nicht anzeigen, vergeben Sie dafür keinen Namen im Array. Die Felder erscheinen in der im Array festgelegten Reihenfolge. Sie übergeben in jedem Element des Array *FeldNr* die aktuelle Feldnummer des dazugehörigen Feldnamens bzw. des neuen Titels, der in *FeldTitel* übergeben wurde.

Sie haben zum Beispiel eine Datenbank mit den Feldern F, G und H in dieser Reihenfolge. Diese Felder sollen als M, N und O erscheinen, wobei Feld N nicht angezeigt werden soll. Außerdem soll die Reihenfolge O und dann M sein. Übergeben Sie dazu O und M in einem Array *FeldTitel* aus zwei Elementen und 3 und 1 in einem Array *FeldNr* aus zwei Elementen.

Mit dem optionalen Parameter * können Sie angeben, ob mit diesem Befehl in 4D Formeln selbst definierte Namen (virtuelle Strukturen) verwendbar sind.

- Standardmäßig. d.h. ohne diesen Parameter, können Formeln, die in 4D ausgeführt werden, nicht die selbst definierten Namen verwenden; es müssen die tatsächlichen Tabellennamen verwendet werden. Eigene Namen bieten mehr Spielraum beim Benennen von Feldern, da der Programmiersprache-Interpreter eigene Namen nicht verarbeitet.
- Ist der Parameter * übergeben, können Formeln, die in 4D ausgeführt werden, die selbst definierten Namen verwenden. **Beachten Sie in diesem Fall**, dass die eigenen Namen keine Zeichen enthalten dürfen, die der Interpreter der 4D Programmiersprache nicht zulässt, wie -?*! (Weitere Informationen dazu finden Sie im Abschnitt **Namenskonventionen**).

Hinweis: Bietet Ihre Anwendung Zugriff auf den Formeleditor, z.B. über den Schnellberichteditor, müssen Sie den Parameter * übergeben, damit die Konsistenz der Oberfläche gewährleistet ist.

SET FIELD TITLES verändert NICHT die aktuelle Struktur Ihrer Datenbank. Der Befehl beeinflusst nur den späteren Gebrauch der Standarddialogfenster von 4D und Formulare mit dynamischen Namen, wenn sie über Befehle der 4D Programmiersprache aufgerufen werden. Die reale Struktur der Datenbank erscheint, wenn der Editor oder das Formular über einen Menübefehl im Designmodus aufgerufen wird. **SET FIELD TITLES** gilt während einer Arbeitssitzung. Ein Vorteil im Client/Server-Betrieb ist, dass mehrere Arbeitsstationen gleichzeitig Ihre Datenbank auf unterschiedliche Art “sehen” können. Sie können **SET FIELD TITLES** beliebig oft aufrufen.

Verwenden Sie den Befehl **SET FIELD TITLES**, wenn Sie:

- Eine Datenbank dynamisch lokalisieren wollen,
- Felder in eigener Darstellung anzeigen wollen, unabhängig von der in der Datenbank festgelegten Definition.
- Felder je nach dem Benutzer oder den zugewiesenen Privilegien anders anzeigen wollen.

Hinweise:

- **SET FIELD TITLES** überschreibt NICHT die Eigenschaft *Unsichtbar* eines Feldes. Ist einem Feld im Designmodus Ihrer Anwendung die Eigenschaft *Unsichtbar* zugewiesen, erscheint es nicht, auch wenn es im Aufruf von **SET FIELD TITLES** enthalten ist.
- Plug-Ins greifen immer auf die virtuelle Struktur zu, die durch diesen Befehl spezifiziert ist.
- Wird **SET FIELD TITLES** ohne einen Parameter aufgerufen, wird die Programmierspracheumgebung zurückgesetzt und die virtuelle Struktur, inkl. eigene Tabellen- und Feldnamen, wird komplett entfernt.

Beispiel

Siehe Beispiel zum Befehl **SET TABLE TITLES**.

SET TABLE TITLES

SET TABLE TITLES {(TabellenTitel ; TabellenNr {; *})}

Parameter	Typ		Beschreibung
TabellenTitel	Array String	➡	Tabellennamen, wie sie im Dialog erscheinen sollen
TabellenNr	Array Lange Ganzzahl	➡	Aktuelle Tabellennummern
*		➡	Verwende die eigenen Namen im Formeleditor

Beschreibung

Der Befehl **SET TABLE TITLES** kann Tabellen Ihrer Datenbank ausblenden, umbenennen und neu ordnen, die in 4D Standarddialogfenstern in der Anwendungsumgebung erscheinen (insbesondere, wenn die Editoren über Befehle der 4D Programmiersprache aufgerufen werden). Dieser Befehl kann z.B. die Anzeige von Tabellen im Sucheditor in der Anwendungsumgebung ändern.
Mit diesem Befehl können Sie auch Tabellennamen in Ihren Formularen "on the fly" umbenennen, wenn Sie dynamische Namen verwendet haben. Weitere Informationen dazu finden Sie im Abschnitt **Referenzen in statischem Text verwenden** des Handbuchs *4D Designmodus*.

Die Arrays *TabellenTitel* und *TabellenNr* müssen synchronisiert sein. Im Array *TabellenTitel* übergeben Sie die entsprechenden Namen. Wollen Sie eine bestimmte Tabelle nicht anzeigen, übernehmen Sie deren Namen oder neuen Titel nicht in das Array. Die Tabellen erscheinen in der im Array festgelegten Reihenfolge. Sie übergeben in jedem Element des Array *TabellenNr* die aktuelle Tabellennummer des dazugehörigen Tabellennamens bzw. des neuen Titels, der unter derselben Elementnummer in *TabellenTitel* übergeben wurde.

Sie haben zum Beispiel eine Datenbank mit den Tabellen A, B und C in dieser Reihenfolge. Diese Tabellen sollen als X, Y und Z erscheinen, wobei Tabelle B nicht angezeigt werden soll. Außerdem soll die Reihenfolge Z und dann X sein. Übergeben Sie dazu Z und X in einem Array *TabellenTitel* aus zwei Elementen und 3 und 1 in einem Array *TabellenNr* aus zwei Elementen.

Mit dem optionalen Parameter * können Sie angeben, ob mit diesem Befehl in 4D Formeln selbst definierte Namen (virtuelle Strukturen) verwendbar sind.

- Ist dieser Parameter nicht angegeben, können Formeln, die in 4D ausgeführt werden, standardmäßig nicht die selbst definierten Namen verwenden; es müssen die tatsächlichen Tabellennamen verwendet werden. Das Verwenden eigener Namen bietet mehr Spielraum bei Tabellennamen, da der Programmiersprache-Interpreter eigene Namen nicht verarbeitet.
- Ist der Parameter * angegeben, können Formeln, die in 4D ausgeführt werden, die selbst definierten Namen verwenden. **Beachten Sie in diesem Fall**, dass die eigenen Namen keine Zeichen enthalten dürfen, die der Interpreter der 4D Programmiersprache nicht zulässt, wie -?*%! So kann z.B. in einer Formel nicht der Name "Rate_in_%" verwendet werden (Weitere Informationen dazu finden Sie im Abschnitt **Namenskonventionen**).

Hinweis: Bietet Ihre Anwendung Zugriff auf den Formeleditor, z.B. über den Schnellberichteditor, müssen Sie den Parameter * übergeben, damit die Konsistenz der Oberfläche gewährleistet ist.

SET TABLE TITLES verändert NICHT die aktuelle Struktur Ihrer Datenbank. Der Befehl beeinflusst nur den späteren Gebrauch der Standarddialogfenster von 4D und Formulare mit dynamischen Namen, wenn sie über Befehle der 4D Programmiersprache aufgerufen werden. Die reale Struktur der Datenbank erscheint, wenn der Editor oder das Formular über einen Menübefehl im Designmodus aufgerufen wird. **SET TABLE TITLES** gilt während einer Arbeitssitzung. Ein Vorteil im Client/Server-Betrieb ist, dass mehrere Arbeitsstationen gleichzeitig Ihre Datenbank auf unterschiedliche Art "sehen" können. Sie können **SET TABLE TITLES** beliebig oft aufrufen.

Verwenden Sie den Befehl **SET TABLE TITLES**, wenn Sie:

- Eine Datenbank dynamisch lokalisieren wollen.
- Tabellen in eigener Darstellung anzeigen wollen, unabhängig von der in der Datenbank festgelegten Definition.
- Tabellen je nach dem Benutzer oder den zugewiesenen Privilegien anders anzeigen wollen.

Hinweise:

- **SET TABLE TITLES** überschreibt NICHT die Eigenschaft *Unsichtbar* einer Tabelle. Ist einer Tabelle im Designmodus Ihrer Datenbank die Eigenschaft *Unsichtbar* zugewiesen, erscheint sie nicht im Anwendungsmodus, auch wenn sie im Aufruf von **SET TABLE TITLES** enthalten ist.
- Plug-Ins greifen immer auf die virtuelle Struktur zu, die durch diesen Befehl spezifiziert ist.
- Wird **SET TABLE TITLES** ohne einen Parameter aufgerufen, wird die Programmierspracheumgebung zurückgesetzt und die virtuelle Struktur, inkl. eigene Tabellen- und Feldnamen, wird komplett entfernt.

Beispiel 1

- Sie entwickeln eine 4D Anwendung für den internationalen Gebrauch. Dafür müssen Sie die jeweilige Lokalisierung mit Bedacht planen. Sie können die erforderliche Lokalisierung über eine Tabelle [Translations] und einige Projektmethoden steuern, die lokalisierte Felder für beliebig viele Länder erstellen und einsetzen.
- Fügen Sie in Ihrer Datenbank folgende Tabelle hinzu:

- Dann erstellen Sie die nachfolgende Projektmethode ***TRANSLATE TABLES AND FIELDS***. Diese Methode durchläuft die aktuelle Struktur Ihrer Datenbank und erstellt alle notwendigen [Translations]Datensätze für die Lokalisierung gemäß der als Parameter übergebenen Sprache.

```
` Projektmethode TRANSLATE TABLES AND FIELDS
` TRANSLATE TABLES AND FIELDS ( Text)
` TRANSLATE TABLES AND FIELDS ( LanguageCode )

C_TEXT($1) ` Sprachcode
C_LONGINT($vlTable;$vlField)
C_TEXT($Language)

For($vlTable;1;Get last table number) ` Durchlaufe die Tabellen
   If($vlTable#(Table(->[Translations]))) `Übersetze nicht die Tabelle Translations
` Prüfe, ob es eine Übersetzung des Tabellennamen für die angegebene Sprache gibt.
      QUERY([Translations];[Translations]LanguageCode=$Language;*) `gewünschte Sprache
      QUERY([Translations]; & ;[Translations]TableID=$vlTable;*) `Tabellennummer
      QUERY([Translations]; & ;[Translations]FieldID=0) `Feldnummer = 0 heißt, es ist ein Tabellenname
      If(Is table number valid($vlTable)) `Prüfe, ob die Tabelle noch existiert
         If(Records in selection([Translations])=0) ` Andernfalls lege den Datensatz an.
            CREATE RECORD([Translations])
            [Translations]LanguageCode:=$Language
            [Translations]TableID:=$vlTable
            [Translations]FieldID:=0 `Der Name der übersetzten Tabelle muss eingegeben werden.
            [Translations]Translation:=Table name($vlTable)+" in "+$Language
            SAVE RECORD([Translations])
         End if

         For($vlField;1;Get last field number($vlTable))
`Prüfe, ob es eine Übersetzung des Feldnamen für die angegebene Sprache gibt.
            QUERY([Translations[Translations]LanguageCode=$Language;*) `gewünschte Sprache
            QUERY([Translations]; & ;[Translations]TableID=$vlTable;*) `Tabellennummer
            QUERY([Translations]; & ;[Translations]FieldID=$vlField) `Feldnummer
            If(Is field number valid($vlTable;$vlField))
               If(Records in selection([Translations])=0) `Andernfalls lege den Datensatz an.
                  CREATE RECORD([Translations])
                  [Translations]LanguageCode:=$Language
                  [Translations]TableID:=$vlTable
                  [Translations]FieldID:=$vlField `Der Name des übersetzten Feldes muss eingegeben werden.
                  [Translations]Translation:=Field name($vlTable;$vlField)+" in "+$Language
                  SAVE RECORD([Translations])
               End if
            Else
               If(Records in selection([Translations])#0)
`Existiert das Feld nicht mehr, entferne die Übersetzung.
                  DELETE RECORD([Translations])
               End if
            End if
         End for
      Else
         If(Records in selection([Translations])#0)
`Existiert die Tabelle nicht mehr, entferne die Übersetzung.
            DELETE RECORD([Translations])
         End if
      End if
   End if
End for
```

- Führen Sie an dieser Stelle folgende Zeile aus, können Sie soviel Datensätze erstellen, wie Sie für eine Lokalisierung der Tabellen- und Feldtitel in Spanisch benötigen.

```
TRANSLATE TABLES AND FIELDS("Spanisch")
```

- Nach Durchführen dieses Aufrufs können Sie für jeden neu erstellten Datensatz *[Translations]Translated Name* eingeben.
- Damit Sie je nach Bedarf Ihre 4D Standarddialogfenster in Spanisch anzeigen können, schreiben Sie folgenden Code:

```
LOCALIZED TABLES AND FIELDS("Spanisch")
```

mit der Projektmethode ***LOCALIZED TABLES AND FIELDS***:

```
  ` Globale Methode LOCALIZED TABLES AND FIELDS
  ` LOCALIZED TABLES AND FIELDS ( Text )
  ` LOCALIZED TABLES AND FIELDS ( LanguageCode )

C_TEXT($1) `Language code
C_LONGINT($vlTable;$vlField)
C_TEXT($Language)
C_LONGINT($vlTableNum;$vlFieldNum)
$Language:=$1

  `Aktualisiere Tabellennamen
ARRAY TEXT($asNames;0) ` Initialisiere Arrays für SET TABLE TITLES und SET FIELD TITLES
ARRAY INTEGER($aiNumbers;0)
QUERY([Translations];[Translations]LanguageCode=$Language;*)
QUERY([Translations]; & ;[Translations]FieldID=0) `also Tabellennamen
SELECTION TO ARRAY([Translations]Translation;$asNames;[Translations]TableID;$aiNumbers)
SET TABLE TITLES($asNames;$aiNumbers)

  `Aktualisiere Feldnamen
$vlTableNum:=Get last table number `Erhalte Anzahl der Tabellen in der Datenbank.
For($vlTable;1;$vlTableNum) `Durchlaufe jede Tabelle
   If(Is table number valid($vlTable))
      QUERY([Translations];[Translations]LanguageCode=$Language;*)
      QUERY([Translations]; & ;[Translations]TableID=$vlTable;*)
      QUERY([Translations]; & ;[Translations]FieldID#0)
  `Vermeide Null, da es als Tabellenname dient.
      SELECTION TO ARRAY([Translations]Translation;$asNames;[Translations]FieldID;$aiNumbers)
      SET FIELD TITLES(Table($vlTable)->;$asNames;$aiNumbers)
   End if
End for
```

Beachten Sie, dass Sie neue Lokalisierungen ohne Ändern oder Rekompilieren des Code hinzufügen können.

Beispiel 2

Sie wollen alle selbst definierten Feld- und Tabellennamen entfernen:

```
SET TABLE TITLES //alle eigenen Namen entfernen
```

Shift down

Shift down -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Status der Umschalttaste

Beschreibung

Die Funktion **Shift down** gibt bei gedrückter **Umschalttaste** den Wert *Wahr* zurück.

Beispiel

Folgende Objektmethode für die Schaltfläche *bAnyButton* führt verschiedene Aktionen aus, je nachdem welche Zusatztasten bei Anklicken der Schaltfläche gedrückt sind:

```
  ` Objektmethode bAnyButton
Case of
  ` Hier können Sie auch andere Tastenkombinationen prüfen
  ` ...
  :(Shift down  &  Windows Ctrl down)
  ` Umschalt- und Ctrl-Taste (unter Windows) bzw. Befehlstaste
      (auf Macintosh)sind gedrückt
      DO ACTION1
  ` ...
  :(Shift down)
  ` Nur Umschalttaste ist gedrückt
      DO ACTION2
  ` ...
  :(Windows Ctrl down)
  ` Nur Ctrl-Taste unter Windows bzw. Befehlstaste auf Macintosh ist gedrückt
      DO ACTION3
  ` ...
  ` Hier können Sie auch andere Einzeltasten prüfen
  ` ...
End case
```

SHOW MENU BAR

SHOW MENU BAR
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **SHOW MENU BAR** blendet die Menüleiste ein.
Ist die Menüleiste bereits sichtbar, hat der Befehl keine Auswirkung.

Beispiel

Siehe Beispiel zum Befehl **HIDE MENU BAR**.

Windows Alt down

Windows Alt down -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Status der alt-Taste unter Windows (Wahltaste auf Macintosh)

Beschreibung

Die Funktion **Windows Alt down** gibt unter Windows bei gedrückter **alt-Taste** den Wert Wahr zurück.
Hinweis: Rufen Sie **Windows Alt down** auf Macintosh auf, gibt die Funktion den Wert *Wahr* bei gedrückter **Wahltaste** zurück.

Beispiel

Siehe Beispiel zum Befehl **Shift down**.

Windows Ctrl down

Windows Ctrl down -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Status der Strg-Taste unter Windows (Befehlstaste auf Macintosh)

Beschreibung

Die Funktion **Windows Ctrl down** gibt bei gedrückter **Strg-Taste** unter Windows den Wert *Wahr* zurück.
Hinweis: Rufen Sie **Windows Ctrl down** auf Macintosh auf, gibt die Funktion den Wert *Wahr* bei gedrückter **Befehlstaste** zurück.

Beispiel

Siehe Beispiel zum Befehl **Shift down**.

_o_Get platform interface

_o_Get platform interface -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Derzeit benutzte Plattformoberfläche

Hinweis zur Kompatibilität

Die Plattformoberfläche wird jetzt automatisch von 4D verwaltet. Dieser Befehl ist ohne Wirkung und sollten nicht mehr verwendet werden.

_o_INVERT BACKGROUND

_o_INVERT BACKGROUND ({* ;} TextObjekt)

Parameter	Typ		Beschreibung
*	Operator	➡	Erlaubt die Eingabe eines Variablen- oder Objektnamens
TextObjekt	Variable, Feld	➡	Zu invertierende Textvariable oder Feld

Beschreibung

Dieser Befehl wird in 4D nicht mehr unterstützt.

_o_SET PLATFORM INTERFACE

_o_SET PLATFORM INTERFACE (Oberfläche)

Parameter	Typ		Beschreibung
Oberfläche	Lange Ganzzahl	➡	Einstellung für neue Plattformoberfläche: -1=Automatisch, 0=MacOS (System7), 1=Windows NT 3.51, 2=Windows 95, 3=Platinum, 4=Mac Theme

Hinweis zur Kompatibilität

Die Plattformoberfläche wird von 4D automatisch verwaltet. Dieser Befehl ist ohne Wirkung und sollte nicht mehr verwendet werden.

Berechnungen

Überblick über Berechnungen

Die Funktionen dieses Kapitels führen Berechnungen auf eine Folge von Werten aus.

Die Funktionen **Average**, **Max**, **Min**, **Sum**, **Sum squares**, **Std deviation** und **Variance** sind auf Datenfelder oder Arrays anwendbar:

- Bei Anwendung auf Datenfelder verwenden sie die aktuelle Auswahl der Datensätze
- Bei Anwendung auf Arrays verwenden sie Elemente der Arrays

Beachten Sie, dass sich die Funktionen **Sum squares**, **Std deviation** und **Variance** bei Anwendung auf Datenfelder nur beim Drucken verwenden lassen.

Diese Funktionen funktionieren nur mit numerischen Daten und geben jeweils einen numerischen Wert zurück.

Statistische Funktionen ohne Drucken verwenden

Setzen Sie **Average**, **Max**, **Min** oder **Sum** für ein Datenfeld außerhalb des Druckvorgangs ein, muss für eine Berechnung jeder Datensatz in die aktuelle Auswahl geladen werden. Das nimmt bei vielen Datensätzen einige Zeit in Anspruch. Um die Prozesszeit zu verringern, können Sie die Datenfelder indizieren.

Hinweis: Dauert die Operation länger, erscheint ein Ablaufbalken. Er hat eine Schaltfläche **Stopp**, über die der Benutzer die Operation abbrechen kann. Klickt er auf die Schaltfläche **Stopp**, wird die Variable OK auf 0 gesetzt. Wurde die Operation erfolgreich abgeschlossen, wird die Variable OK auf 1 gesetzt.

Statistische Funktionen in gedrucktem Bericht verwenden

In Berichten verhalten sich diese Funktionen auf spezifische Weise, da der Bericht selbst jeden Datensatz laden muss. Verwenden Sie diese Funktionen in einem Formular oder einer Objektmethode, wenn Sie zum Drucken den Befehl **PRINT SELECTION** oder in der Designumgebung im Menü **Datei/Ablage** den Befehl **Drucken** auswählen.

In einem Bericht sind diese Funktionen am Berichtende sinnvoll, d.h. wenn alle Datensätze bearbeitet wurden. Die zurückgegebenen Werte erscheinen nur in der Umbruchebene 0 und nur, wenn der Umbruch für den Bericht aktiviert ist.

Benutzen Sie diese Funktionen ausschließlich in einer Objektmethode für einen nicht eingebbaren Bereich in der Umbruchebene U0.

Achten Sie darauf, dass das als Parameter übergebene Datenfeld numerisch ist.

Average

Average (Werte {; AttributPfad}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Werte	Feld, Array	⇒	Werte, deren Durchschnitt ermittelt werden soll
AttributPfad	Text	⇒	Pfad des Attributs, dessen Durchschnitt zurückgegeben werden soll
Funktionsergebnis	Zahl	⇐	Arithmetisches Mittel (Durchschnitt) von Objekt

Beschreibung

Die Funktion **Average** gibt den Durchschnitt von *Werte* zurück. Ist *Werte* ein indiziertes Feld, wird der Index für die Berechnung verwendet.

Im Parameter *Werte* können Sie ein Array (ein- oder zweidimensional) übergeben. Das Array muss vom Typ Ganzzahl, Lange Ganzzahl oder Zahl sein.

Diese Funktion erlaubt den optionalen Parameter *AttributPfad* vom Typ Text, wenn *Werte* ein Feld oder eine Variable vom Typ Objekt ist. Hier können Sie den Pfad des Attributs zum Berechnen übergeben. Verwenden Sie die Standard Notation mit Punkt für Pfade zu eingebetteten Attributen, z.B. "company.address.number". Beachten Sie, dass Attributnamen vom Typ Objekt zwischen Groß- und Kleinschreibung unterscheiden.
Nur Attributwerte vom Typ Zahl werden berechnet. Gibt es im Attributpfad Werte, die nicht vom Typ Zahl sind, werden sie ignoriert.

Bei korrekt ausgeführter Funktion wird die Systemvariable OK auf 1 gesetzt. Wird sie unterbrochen, z.B. weil der Benutzer auf die Schaltfläche **Stopp** des Ablaufbalkens geklickt hat, wird die Variable OK auf 0 gesetzt.

Beispiel 1

Folgendes Beispiel legt die Variable *vAverage* im Umbruchteil U0 eines Ausgabeformulars fest. Die Codezeile ist die Objektmethode für *vAverage*. Sie wird erst bei der Umbruchebene U0 ausgeführt:

```
vAverage:=Average([Employees] Salary)
```

Folgende Methode druckt die Datensätze in der Auswahl und aktiviert den Umbruch:

```
ALL RECORDS([Employees])
ORDER BY([Employees];[Employees]LastNm;>)
BREAK LEVEL(1)
ACCUMULATE([Employees]Salary)
FORM SET OUTPUT([Employees];"PrintForm")
PRINT SELECTION([Employees])
```

Hinweis: Der Parameter für den Befehl **BREAK LEVEL** sollte gleich sein mit der Anzahl der Umbrüche in Ihrem Bericht. Weitere Informationen dazu finden Sie im Kapitel **Drucken**.

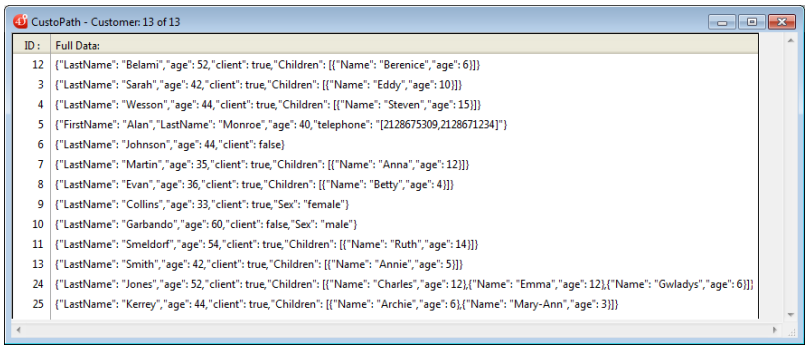
Beispiel 2

Dieses Beispiel erhält den Durchschnitt der 15 ersten "Grades" in der Auswahl:

```
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)
```

Beispiel 3

Ihre Tabelle [Customer] enthält ein Objektfeld "full_Data" mit folgenden Daten:



Sie können folgende Berechnungen durchführen:

```
C_REAL($vAvg)
ALL RECORDS([Customer])
$vAvg:=Average([Customer]full_Data;"age")
 //$vAvg ist 44,46

C_LONGINT($vTot)
$vTot:=Sum([Customer]full_Data;"Children[].age")
 //$vTot ist 105
```

Max

Max (Werte {; AttributPfad}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Werte	Feld, Array	➡	Werte, deren Maximum ermittelt werden soll
AttributPfad	Text	➡	Pfad des Attributs, dessen Maximalwert zurückgegeben werden soll
Funktionsergebnis	Datum, Zahl	⮌	Maximalwert von Objekt

Beschreibung

Die Funktion **Max** gibt den Maximalwert von *Werte* zurück. Ist *Werte* ein indiziertes Feld, wird der Index für die Berechnung verwendet.

Im Parameter *Werte* können Sie ein Array (ein- oder zweidimensional) übergeben. Das Array muss vom Typ Ganzzahl, Lange Ganzzahl, Zahl oder Datum sein.

Diese Funktion erlaubt den optionalen Parameter *AttributPfad* vom Typ Text, wenn *Werte* ein Feld oder eine Variable vom Typ Objekt ist. Hier können Sie den Pfad des Attributs zum Berechnen übergeben. Verwenden Sie die Standard Notation mit Punkt für Pfade zu eingebetteten Attributen, z.B. "company.address.number". Beachten Sie, dass Attributnamen vom Typ Objekt zwischen Groß- und Kleinschreibung unterscheiden.
Nur Attributwerte vom Typ Zahl werden berechnet. Gibt es im Attributpfad Werte, die nicht vom Typ Zahl sind, werden sie ignoriert.

Ist die Auswahl *Werte* leer, gibt **Max** 0 zurück.

Bei korrekt ausgeführter Funktion wird die Systemvariable OK auf 1 gesetzt. Wird sie unterbrochen, z.B. weil der Benutzer auf die Schaltfläche **Stopp** des Ablaufbalkens geklickt hat, wird die Variable OK auf 0 gesetzt.

Beispiel 1

Folgendes Beispiel zeigt eine Objektmethode für die Variable *vMax* in der Umbruchebene 0 des Formulars. Die Variable wird am Berichtende gedruckt. Die Objektmethode weist der Variablen den Maximalwert des Feldes zu, der dann im letzten Umbruch des Berichts gedruckt wird.

```
vMax:=Max([Employees] Salary)
```

Hinweis: Stellen Sie sicher, dass für die Variable das Formularereignis "On printing break" ausgewählt ist.

Folgende Methode druckt die Datensätze in der Auswahl und aktiviert den Umbruch:

```
ALL RECORDS([Employees])
ORDER BY([Employees];[Employees]Company;>)
BREAK LEVEL(1)
ACCUMULATE([Employees]Salary)
FORM SET OUTPUT([Employees];"PrintForm")
PRINT SELECTION([Employees])
```

Hinweis: Der Parameter für den Befehl **BREAK LEVEL** sollte gleich sein mit der Anzahl der Umbrüche in Ihrem Bericht. Weitere Informationen dazu finden Sie im Kapitel **Drucken**.

Beispiel 2

Dieses Beispiel erhält den höchsten Wert im Array:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vMax:=Max($ArrGrades)
```

Beispiel 3

Ein Beispiel zum Berechnen eines Attributs Objektfeld finden Sie im 3. Beispiel unter dem Befehl **Average**.

Min

Min (Werte {; AttributPfad}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Werte	Feld, Array	⇒	Werte, deren Minimum ermittelt werden soll
AttributPfad	Text	⇒	Pfad des Attributs, dessen Minimalwert zurückgegeben werden soll
Funktionsergebnis	Datum, Zahl	⇐	Minimalwert von Werte

Beschreibung

Die Funktion **Min** gibt den Minimalwert von *Werte* zurück. Ist *Werte* ein indiziertes Feld, wird der Index für die Berechnung verwendet.

Im Parameter *Werte* können Sie ein Array (ein- oder zweidimensional) übergeben. Das Array muss vom Typ Ganzzahl, Lange Ganzzahl, Zahl oder Datum sein.

Diese Funktion erlaubt den optionalen Parameter *AttributPfad* vom Typ Text, wenn *Werte* ein Feld oder eine Variable vom Typ Objekt ist. Hier können Sie den Pfad des Attributs zum Berechnen übergeben. Verwenden Sie die Standard Notation mit Punkt für Pfade zu eingebetteten Attributen, z.B. "company.address.number". Beachten Sie, dass Attributnamen vom Typ Objekt zwischen Groß- und Kleinschreibung unterscheiden.
Nur Attributwerte vom Typ Zahl werden berechnet. Gibt es im Attributpfad Werte, die nicht vom Typ Zahl sind, werden sie ignoriert.

Ist die Auswahl *Werte* leer, gibt **Min** den Wert 0 zurück.

Bei korrekt ausgeführter Funktion wird die Systemvariable OK auf 1 gesetzt. Wird sie unterbrochen, z.B. weil der Benutzer auf die Schaltfläche **Stopp** des Ablaufbalkens geklickt hat, wird die Variable OK auf 0 gesetzt.

Beispiel 1

Folgendes Beispiel zeigt eine Objektmethode für die Variable *vMin* in der Umbruchebene 0 eines Formulars. Die Variable wird am Berichtende gedruckt. Die Objektmethode weist der Variablen den Mindestwert des Feldes zu, der dann im letzten Umbruch des Berichts gedruckt wird:

```
vMin:=Min([Employees]Salary)
```

Hinweis: Stellen Sie sicher, dass für die Variable das Formularereignis "On printing break" ausgewählt ist.

Folgende Methode druckt die Datensätze in der Auswahl und aktiviert den Umbruch:

```
ALL RECORDS([Employees])
ORDER BY([Employees];[Employees]Company;>)
BREAK LEVEL(1)
ACCUMULATE([Employees]Salary)
FORM SET OUTPUT([Employees];"PrintForm")
PRINT SELECTION([Employees])
```

Hinweis: Der Parameter für den Befehl **BREAK LEVEL** sollte gleich sein mit der Anzahl der Umbrüche in Ihrem Bericht. Weitere Informationen dazu finden Sie im Kapitel **Drucken**.

Beispiel 2

Folgendes Beispiel findet die niedrigste Verkaufssumme eines Angestellten und zeigt das Ergebnis in einer Meldung an. Die Verkaufssummen werden im Unterdatenfeld [Employees]SalesDollars abgespeichert:

```
ALERT("Minimum sale = "+String(Min([Employees]SalesDollars)))
```

Beispiel 3

Dieses Beispiel erhält den niedrigsten Wert im Array:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vMin:=Min($ArrGrades)
```

Beispiel 4

Ein Beispiel zum Berechnen eines Attributs Objektfeld finden Sie im 3. Beispiel unter dem Befehl **Average**.

Std deviation

Std deviation (Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Objekt	Feld, Array	➡	Objekt, dessen Standardabweichung ermittelt werden soll
Funktionsergebnis	Zahl	⮌	Standardabweichung von Objekt

Beschreibung

Die Funktion **Std deviation** gibt die Standardabweichung einer Anzahl von Werten von *Objekt* zurück. Die Standardabweichung ist die Streuung oder Abweichung der einzelnen Werte von einem Mittelwert. Ist *Objekt* indiziert, wird der Index für die Berechnung verwendet.

Im Parameter *Objekt* können Sie ein Array (ein- oder zweidimensional) übergeben. Das Array muss vom Typ Ganzzahl, Lange Ganzzahl oder Zahl sein.

Beispiel 1

Folgendes Beispiel zeigt eine Objektmethode für die Variable *vDeviate*. Die Objektmethode weist *vDeviate* die Standardabweichung zu:

```
vDeviate:=Std deviation([Table1]DataSeries)
```

Folgende Methode druckt die Datensätze in der Auswahl und aktiviert den Umbruch:

```
ALL RECORDS([Table1])
ORDER BY([Table1];[Table1]DataSeries;>)
BREAK LEVEL(1)
ACCUMULATE([Table1]DataSeries)
OUTPUT FORM([Table1];"PrintForm")
PRINT SELECTION([Table1])
```

Hinweis: Der Parameter für den Befehl **BREAK LEVEL** sollte gleich sein mit der Anzahl der Umbrüche in Ihrem Bericht. Weitere Informationen dazu finden Sie im Kapitel **Drucken**.

Beispiel 2

Dieses Beispiel erhält die Standardabweichung einer Berechnung von Werten aus einem 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 (Werte {; AttributPfad}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Werte	Feld, Array	➡	Werte, deren Summe ermittelt werden soll
AttributPfad	Text	➡	Pfad des Attributs, dessen Summe zurückgegeben werden soll
Funktionsergebnis	Zahl	⮌	Summe von Werte

Beschreibung

Die Funktion **Sum** gibt die Summe von *Werte* zurück. Ist *Werte* ein indiziertes Feld, wird der Index für die Berechnung verwendet.

Im Parameter *Werte* können Sie ein Array (ein- oder zweidimensional) übergeben. Das Array muss vom Typ Ganzzahl, Lange Ganzzahl oder Zahl sein.

Diese Funktion erlaubt den optionalen Parameter *AttributPfad* vom Typ Text, wenn *Werte* ein Feld oder eine Variable vom Typ Objekt ist. Hier können Sie den Pfad des Attributs zum Berechnen übergeben. Verwenden Sie die Standard Notation mit Punkt für Pfade zu eingebetteten Attributen, z.B. "company.address.number". Beachten Sie, dass Attributnamen vom Typ Objekt zwischen Groß- und Kleinschreibung unterscheiden.

Nur Attributwerte vom Typ Zahl werden berechnet. Gibt es im Attributpfad Werte, die nicht vom Typ Zahl sind, werden sie ignoriert.

Bei korrekt ausgeführter Funktion wird die Systemvariable OK auf 1 gesetzt. Wird sie unterbrochen, z.B. weil der Benutzer auf die Schaltfläche **Stopp** des Ablaufbalkens geklickt hat, wird die Variable OK auf 0 gesetzt.

Beispiel 1

Folgendes Beispiel zeigt eine Objektmethode für die Variable *vTotal* in einem Formular. Die Objektmethode weist *vTotal* die Summe aller Gehälter zu:

```
vTotal:=Sum([Employees]Salary)
```

Folgende Methode druckt die Datensätze in der Auswahl und aktiviert den Umbruch:

```
ALL RECORDS([Employees])
ORDER BY([Employees];[Employees]LastNm;>)
BREAK LEVEL(1)
ACCUMULATE([Employees]Salary)
OUTPUT FORM([Employees];"PrintForm")
PRINT SELECTION([Employees])
```

Hinweis: Der Parameter für den Befehl **BREAK LEVEL** sollte gleich sein mit der Anzahl der Umbrüche in Ihrem Bericht. Weitere Informationen dazu finden Sie im Kapitel **Drucken**.

Beispiel 2

Dieses Beispiel erhält die Summe aller Werte im Array:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vSum:=Sum($ArrGrades)
```

Beispiel 3

Ein Beispiel zum Berechnen eines Attributs Objektfeld finden Sie im 3. Beispiel unter dem Befehl **Average**.

Sum squares

Sum squares (Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Objekt	Feld, Array	➡	Objekt, dessen Quadratsumme ermittelt werden soll
Funktionsergebnis	Zahl	⮌	Quadratsumme von Feldname

Beschreibung

Die Funktion **Sum squares** gibt die Quadratsumme der Werte aus *Objekt* zurück. Ist *Objekt* indiziert, wird der Index für die Berechnung verwendet.

In *Objekt* können Sie ein Array (ein- oder zweidimensional) übergeben. Es muss vom Typ Ganzzahl, Lange Ganzzahl oder Zahl sein.

Beispiel 1

Folgendes Beispiel zeigt eine Objektmethode für die Variable *vSquares*. Die Objektmethode weist *vSquares* die Quadratsumme zu, die im letzten Umbruch des Berichts gedruckt wird:

```
vSquares:=Sum squares([Table1]DataSeries)
```

Folgende Methode druckt die Datensätze in der Auswahl und aktiviert den Umbruch:

```
ALL RECORDS([Table1])
ORDER BY([Table1];[Table1]DataSeries;>)
BREAK LEVEL(1)
ACCUMULATE([Table1]DataSeries)
OUTPUT FORM([Table1];"PrintForm")
PRINT SELECTION([Table1])
```

Hinweis: Der Parameter für den Befehl **BREAK LEVEL** sollte gleich sein mit der Anzahl der Umbrüche in Ihrem Bericht. Weitere Informationen dazu finden Sie im Kapitel **Drucken**.

Beispiel 2

Dieses Beispiel erhält die Quadratsumme der Werte im 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 (Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Objekt	Feld, Array	➡	Objekt, dessen Varianz ermittelt werden soll
Funktionsergebnis	Zahl	⮌	Varianz von Objekt

Beschreibung

Die Funktion **Variance** gibt den Schätzwert der Varianz für eine Anzahl von Werten von *Objekt* zurück. Ist *Objekt* indiziert, wird der Index für die Berechnung verwendet.

In *Objekt* können Sie ein Array (ein- oder zweidimensional) übergeben. Es muss vom Typ Ganzzahl, Lange Ganzzahl oder Zahl sein.

Varianz ist das arithmetische Mittel der quadratischen Abweichung von einem Mittelwert. 4D verwendet dafür folgende Formel:

Variance(x) = Sum (x-m)*(x-m)/(n-1)
m = Mittelwert
n = Anzahl Werte

Werden die Werte nicht als Ganzes gesehen, wird der zurückgegebene Wert mit (n-1)n multipliziert.

Beispiel 1

Folgendes Beispiel zeigt eine Objektmethode für die Variable *var*. Die Objektmethode weist *var* die Varianz zu:

```
var:=Variance([Students]Grades)
```

Folgende Methode druckt die Datensätze in der Auswahl und aktiviert den Umbruch:

```
ALL RECORDS([Students])
ORDER BY([Students];[Students]Class;>)
BREAK LEVEL(1)
ACCUMULATE([Students]Grades)
OUTPUT FORM([Students];"PrintForm")
PRINT SELECTION([Students])
```

Hinweis: Der Parameter für den Befehl **BREAK LEVEL** sollte gleich sein mit der Anzahl der Umbrüche in Ihrem Bericht. Weitere Informationen dazu finden Sie im Kapitel **Drucken**.

Beispiel 2

Dieses Beispiel erhält die Varianz der Werte im Array:

```
ARRAY REAL($ArrGrades;0)
QUERY([Exams];[Exams]Exam_Date=!01/07/11!)
SELECTION TO ARRAY([Exams]Exam_Grade;$ArrGrades)
vVariance:=Variance($ArrGrades)
```

Bilder

- Überblick über Bilder
- 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*

Überblick über Bilder

Native Unterstützung von Formaten

4D enthält die native Verwaltung von Bildformaten, d.h. die Bilder werden in ihrem Originalformat angezeigt und gespeichert, ohne jegliche Interpretation durch 4D. Die spezifischen Merkmale der verschiedenen Formate, wie Schatten, transparente Bereiche, o.ä. werden beim Kopieren und Einsetzen beibehalten und ohne Veränderung angezeigt. Die native Verwaltung gilt für alle in 4D gespeicherten Bilder: Bilder aus der Objektbibliothek, im Designmodus in Formulare eingefügte Bilder, im Anwendungsmodus in Datenfelder oder Variablen eingefügte Bilder, etc.

4D verwendet native APIs, um Bilder (Felder und Variablen) unter Windows und auf Mac OS zu codieren bzw. decodieren. Diese Implementationen bieten Zugriff auf zusätzliche native Formate, inkl. RAW, dem gängigen Format für Digitalkameras.

- **Unter Windows** verwendet 4D WIC (Windows Imaging Component). WIC unterstützt nativ folgende Formate: BMP, PNG, ICO (nur Decodierung), JPEG, GIF, TIFF und WDP (Microsoft Windows Digital Photo).
  Es sind auch zusätzliche Formate wie JPEG-2000 verwendbar, wenn Sie WIC Codecs von Drittherstellern installieren.
- **Auf Mac OS** verwendet 4D ImageIO. Alle ImageIO Codecs werden so nativ zum Decodieren (Lesen) und Codieren (Schreiben) unterstützt:

Decodierung	Codierung
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 (überholt)
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 (überholt)	
com.apple.icns	
com.apple.pict (überholt)	
com.apple.macpaint-image	
com.kodak.flashpix-image	
public.xbitmap-image	
com.ilm.openexr-image	
public.radiance	

Die unterstützten Formate variieren unter Windows und auf Mac OS je nach Betriebssystem und den eigenen Codecs, die auf den Rechnern installiert sind. Um herauszufinden, welche Codecs verfügbar sind, müssen Sie den Befehl **PICTURE CODEC LIST** einsetzen.

Hinweis: WIC und ImageIO erlauben auch die Verwendung von Metadaten in Bildern. Für diesen Zweck können Sie die beiden Befehle **SET PICTURE METADATA** und **GET PICTURE METADATA** verwenden.

Bild Codec Kennung

Der Befehl **PICTURE CODEC LIST** gibt von 4D erkannte Bildformate gibt als Codec Kennung für Bilder zurück. Es gibt folgende Formen:

- Als Endung (zum Beispiel ".gif")
- Als Mime Typ (zum Beispiel "image/jpeg")

Die vom Befehl zurückgegebene Form richtet sich nach der Art, wie der Codec auf Ebene des Betriebssystems gespeichert ist. Die meisten 4D Befehle zur Bildverwaltung können eine Codec Kennung als Parameter empfangen. Deshalb sollte die vom Befehl **PICTURE CODEC LIST** zurückgegebene ID des Systems verwendet werden.

Nicht darstellbares Bildformat

Für Bilder, die von dem verwendeten Computer nicht darstellbar sind, erscheint ein Ersatzsymbol. Die Endung des benutzten Bildformats wird darunter angezeigt:

Dieses Symbol wird beim Anzeigen von Bildern automatisch dargestellt:

FirstName :	LastName :	Photo :
Elizabeth	Smith	
Gerry	Mc Namara	
Henry	Portier	

Es gibt an, dass das Bild nicht angezeigt bzw. konvertiert werden kann -- es lässt sich aber speichern und ggf. auf anderen Rechnern anzeigen. Das ist z.B. bei PDF Bildern auf einer Windows Plattform, PICT-basierten Bildern auf einem 64-bit 4D Server auf OS X oder QuickTime komprimierten Bildern der Fall.

Aktivierung von QuickTime (Kompatibilität)

4D unterstützt ab Version 14 standardmäßig nicht mehr Bild-Codecs, die sich auf QuickTime beziehen.

Zur Wahrung der Kompatibilität können Sie über die Option QuickTime support des Befehls **SET DATABASE PARAMETER** QuickTime in Ihrer Anwendung reaktivieren. Wir empfehlen jedoch, QuickTime Codecs nicht mehr zu verwenden.

Hinweis: Die Option zum Reaktivieren von QuickTime wird in 64-bit Versionen von 4D ignoriert (keine Unterstützung von QuickTime).

Mauskoordinaten für Klicks auf ein Bild

4D ermöglicht, die lokalen Koordinaten eines Klicks oder Darüberziehen mit der Maus auf ein Datenfeld vom Typ Bild oder Variable herauszufinden, auch wenn das Bild gescrollt oder gezoomt wurde. Die Funktionsweise ist ähnlich zu einer Bildkarte und lässt sich z.B. zum Verwalten scrollbarer Schaltflächenleisten oder für die Oberfläche kartographischer Software verwenden.

Die Koordinaten werden in den **Systemvariablen** *MouseX* und *MouseY* übergeben. Sie werden in Pixel ausgedrückt, ausgehend von der oberen linken Ecke (0,0). Ist die Maus außerhalb der Koordinaten des Bildes, wird -1 in *MouseX* und *MouseY* zurückgegeben.

Den Wert dieser Variablen müssen Sie als Teil der Formularereignisse On Clicked, On Double Clicked, On Mouse up, On Mouse Enter oder On Mouse Move erhalten.

Bildoperatoren

4D ermöglicht mit 4D Bildern Operationen wie Zusammenfügen, Übereinanderlegen, etc. auszuführen. Weitere Informationen dazu finden Sie im Abschnitt **Bildoperatoren**.

BLOB TO PICTURE

BLOB TO PICTURE (BildBlob ; Bild {; Codec})

Parameter	Typ		Beschreibung
BildBlob	BLOB	➡	BLOB, das ein Bild enthält
Bild	Bild	⬅	Bild aus BLOB
Codec	String	➡	Codec ID des Bildes

Beschreibung

Der Befehl **BLOB TO PICTURE** setzt in eine 4D Variable bzw. ein 4D Feld vom Typ Bild unabhängig vom Originalformat ein Bild, das in einem BLOB gespeichert ist.

Dieser Befehl arbeitet wie der Befehl **READ PICTURE FILE**, einziger Unterschied ist, dass er nicht auf eine Datei, sondern auf ein BLOB angewandt wird. Damit können Sie Bilder in einem native Format in BLOBs anzeigen. Sie können z.B. mit den Befehlen **DOCUMENT TO BLOB** oder **PICTURE TO BLOB** ein Bild in ein BLOB laden.

Im Parameter *BildBlob* wird eine BLOB Variable oder ein BLOB Feld mit einem Bild übergeben. Das Bild kann in jedem Format sein, das von 4D native unterstützt wird. Über den Befehl **PICTURE CODEC LIST** erhalten Sie die Liste der verfügbaren Formate. Übergeben Sie den optionalen Parameter *Codec*, verwendet 4D den hier angegebenen Wert zum Decodieren des BLOB (Beschreibung dieses Parameters siehe unten).

Im Parameter *Bild* übergeben Sie das 4D Feld bzw. die Variable vom Typ Bild, die das Bild enthalten sollte.

Hinweis: Das interne Bildformat wird in die 4D Variable bzw. das 4D Feld gespeichert.

Wurde der Befehl ausgeführt und das BLOB korrekt decodiert, enthält der Parameter *Bild* das anzuzeigende Bild.

Über den optionalen Parameter *Codec* können Sie eine Codec Kennung zur Decodierung des BLOB angeben. Wird die übergebene Codec Kennung von 4D erkannt (zurückgegeben über den Befehl **PICTURE CODEC LIST**), wird sie auf das BLOB angewandt und das Bild im Parameter *Bild* als Bildfeld oder Variable zurückgegeben.
Übergeben Sie in *Codec* einen unbekannten, eigenen Codec, den 4D nicht erkennt, wird dynamisch ein neuer Codec mit der übergebenen Kennung angelegt. 4D gibt dann ein Objekt vom Typ Bild mit eingebundenem BLOB zurück und die Variable OK wird auf 1 gesetzt. In diesem Fall müssen Sie zum Wiederauslesen des BLOB den Befehl **PICTURE TO BLOB** mit derselben eigenen Kennung verwenden.

Diese spezielle Vorgehensweise lässt sich für folgende Anforderungen verwenden:

- Einbindung eines BLOB (das kein Bild ist) in ein Bild,
- Bild ohne Einsatz eines Codec laden.

Auf diese Weise ist es möglich, "BLOB Arrays" über Bild Arrays zu erstellen. Verwenden Sie diese Technik jedoch mit Bedacht, da die Arrays vollständig in den Speicher geladen werden. Das kann bei umfangreichen BLOBs die Arbeitsweise der Anwendung beeinträchtigen.

Hinweis: Ein BLOB, das über den Befehl **VARIABLE TO BLOB** erstellt wurde, wird automatisch verwaltet; es ist nicht notwendig, zur Einbindung einen Codec zu übergeben, da das BLOB "signiert" ist. In diesem Fall müssen Sie für die entgegengesetzte Operation im Befehl **PICTURE TO BLOB** ".4DVarBlob" als Codec Kennung übergeben.

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable **OK** auf 1 gesetzt. Wurde die Konvertierung nicht durchgeführt (QuickTime 4 ist nicht installiert oder das BLOB enthält kein lesbares Bild), wird **OK** auf 0 (Null) gesetzt, Variable oder Feld vom Typ Bild bleiben leer.

COMBINE PICTURES

COMBINE PICTURES (Ergebnisbild ; Bild1 ; Operator ; Bild2 {; horVersatz ; vertVersatz})

Parameter	Typ		Beschreibung
Ergebnisbild	Bild	⇐	Bild, das sich aus der Kombination ergibt
Bild1	Bild	⇒	1. Bild für Kombination
Operator	Lange Ganzzahl	⇒	Art der Kombination
Bild2	Bild	⇒	2. Bild für Kombination
horVersatz	Lange Ganzzahl	⇒	Horizontaler Versatz für Überlappung
vertVersatz	Lange Ganzzahl	⇒	Vertikaler Versatz für Überlappung

Beschreibung

Der Befehl **COMBINE PICTURES** ermöglicht, die Bilder in *Bild1* und *Bild2* über den Parameter *Operator* zu kombinieren, um daraus ein 3. Ergebnisbild zu erstellen. Das Ergebnisbild ist vom kombinierten Typ und behält alle Merkmale der Ursprungsbilder bei.

Hinweis: Dieser Befehl erweitert die Funktionalitäten der klassischen Operatoren zur Bildkombination (+/, etc, siehe **Bildoperatoren**). Diese bleiben in 4D v11 weiterhin verwendbar.

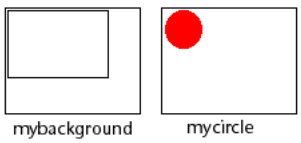
In *Operator* übergeben Sie die Art der Kombination. Es gibt drei Arten, die Sie über die Konstanten unter dem Thema **Bildtransformation** vergeben können:

- Horizontal concatenation (1): *Bild2* wird an *Bild1* angehängt, die obere linke Ecke von *Bild2* stimmt mit der oberen rechten Ecke von *Bild1* überein.
- Vertical concatenation (2): *Bild2* wird an *Bild1* angehängt, die obere linke Ecke von *Bild2* stimmt mit der unteren linken Ecke von *Bild1* überein.
- Superimposition (3): *Bild2* wird über *Bild1* gelegt, die obere linke Ecke von *Bild2* stimmt mit der oberen linken Ecke von *Bild1* überein. Bei Verwendung der optionalen Parameter *horVersatz* und *vertVersatz* erfolgt vor dem Übereinanderlegen eine Verschiebung von *Bild2*. Die Werte in *horVersatz* und *vertVersatz* müssen Pixel entsprechen. Übergeben Sie positive Werte für den Versatz nach rechts bzw. unten, negative Werte für den Versatz nach links oder nach oben.

Hinweis: Das Übereinanderlegen durch den Befehl **COMBINE PICTURES** unterscheidet sich vom Übereinanderlegen durch die gängigen Operatoren & und | (exklusive und inklusive Übereinanderlegen). Während **COMBINE PICTURES** im Ergebnisbild die Merkmale der einzelnen Ursprungsbilder beibehält, bearbeiten die Operatoren & und | jedes Pixel und erstellen dann immer ein Bitmap-Bild. Diese Operatoren dienten in der Vergangenheit für Schwarz/Weiß-Bilder und sind jetzt überholt.

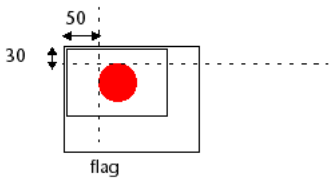
Beispiel

Wir gehen von folgenden Bildern aus:




```
COMBINE PICTURES(flag;mybackground;Superimposition;mycircle;50;30)
```

Ergebnis:



CONVERT PICTURE

CONVERT PICTURE (Bild ; Codec {; Komprimierung})

Parameter	Typ		Beschreibung
Bild	Bild	➡	Zu konvertierendes Bild
		⬅	Konvertiertes Bild
Codec	String	➡	Codec Kennummer des Bildes
Komprimierung	Zahl	➡	Komprimierungsstufe

Beschreibung

Der Befehl **CONVERT PICTURE** konvertiert *Bild* in einen neuen Typ.

Der Parameter *Codec* gibt die Art des zu erstellenden Bildes an. *Codec* kann eine Endung (z.B. ".gif") oder ein Mime Typ (z.B. "image/jpeg") sein. Über den Befehl **PICTURE CODEC LIST** können Sie die Liste der verfügbaren Codecs aufrufen.

Ist der Parameter *Bild* ein zusammengesetzter Typ, z.B. das Ergebnis einer Copy-Paste Aktion, bleibt im Ergebnisbild nur die Information zum Codec Typ erhalten.

Hinweis: Ist der in *Codec* angeforderte Typ der gleiche wie der Originaltyp von *Bild*, wird keine Konvertierung ausgeführt und das Bild wird "wie gehabt" zurückgegeben, außer es wird ein Parameter *Komprimierung* verwendet (siehe unten).

Mit dem optionalen Parameter *Komprimierung* können Sie die Qualität der Komprimierung für das Ergebnisbild angeben, wenn ein kompatibler Codec verwendet wird. In *Komprimierung* übergeben Sie einen Wert zwischen 0 und 1, wobei 0 die niedrigste Qualität (höchste Komprimierung) und 1 die beste Qualität (geringe Komprimierung) ist. Dieser Parameter wird nur berücksichtigt, wenn der Codec Komprimierung unterstützt (z.B. JPEG oder HDPhoto) und von WIC und ImageIO APIs unterstützt wird. Weitere Informationen zur APIs Bildverwaltung in 4D finden Sie im Abschnitt **Überblick über Bilder**. Standardmäßig, d.h. ohne den optionalen Parameter, wird die höchste Qualität angewandt, d.h. Komprimierung = 1.

Hinweis: Wollen Sie **CONVERT PICTURE** mit einem Bildtyp aufrufen, der in 4D 64-bit Versionen nicht unterstützt wird (wie PICT), achten Sie darauf, dass die Konvertierung in einer 4D 32-bit Version durchgeführt wird, wo dieser Typ unterstützt wird. Weitere Informationen dazu finden Sie im Abschnitt **Von 32-bit Versionen auf 64-bit Versionen wechseln**.

Beispiel 1

Das Bild vpPhoto in das Format jpeg umwandeln:

```
CONVERT PICTURE(vpPhoto;".jpg")
```

Beispiel 2

Bild mit 60% Qualität konvertieren:

```
CONVERT PICTURE(vPicture;".JPG";0.6)
```

CREATE THUMBNAIL

CREATE THUMBNAIL (Quelle ; Ziel {; Breite {; Höhe {; Modus {; Tiefe}}}})

Parameter	Typ		Beschreibung
Quelle	Bild	⇒	4D Feld oder Variable vom Typ Bild zur Konvertierung in Minivorschau
Ziel	Bild	⇐	Resultierende Minivorschau
Breite	Ganzzahl	⇒	Breite der Minvorschau in Pixel, Standardwert = 48
Höhe	Ganzzahl	⇒	Höhe der Minvorschau in Pixel, Standardwert = 48
Modus	Ganzzahl	⇒	Erstellungsmodus für Minvorschau Standardwert = Scaled to fit prop centered (6)
Tiefe	Ganzzahl	⇒	>>Überholt, nicht mehr verwenden!

Beschreibung

Der Befehl **CREATE THUMBNAIL** gibt eine Minivorschau (thumbnail) des vorhandenen Ursprungsbilds zurück. Thumbnails werden normalerweise in Multimedia-Software oder Web Sites als Bildvorschau verwendet.

Im Parameter *Quelle* übergeben Sie die 4D Variable bzw. das Feld mit dem Bild, das Sie auf Daumennagelgröße verkleinern wollen. Im Parameter *Ziel* übergeben Sie die 4D Variable bzw. das Feld vom Typ Bild, das die Minivorschau aufnehmen soll.

Über die optionalen Parameter *Breite* und *Höhe* definieren Sie eigene Maße für die Minivorschau (in Pixel). Geben Sie diese Parameter nicht an, ist die Minivorschau standardmäßig 48 x 48 Pixel groß.

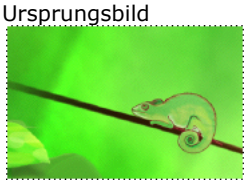
Über den optionalen Parameter *Modus* definieren Sie die Art der Skalierung für die Minivorschau. 4D bietet dafür unter dem Thema **Bild Anzeigeformate** drei vordefinierte Konstanten:

Konstante	Typ	Wert
Scaled to fit	Lange Ganzzahl	2
Scaled to fit prop centered	Lange Ganzzahl	6
Scaled to fit proportional	Lange Ganzzahl	5

Hinweis: Mit dem Befehl **CREATE THUMBNAIL** können Sie nur diese drei Konstanten verwenden. Die anderen Konstanten unter diesem Thema sind hierfür nicht verwendbar.

Geben Sie keinen Parameter an, wird standardmäßig der Modus *Scaled to fit prop centered (6)* verwendet. Im folgenden sehen Sie die Auswirkung der verschiedenen Modi:

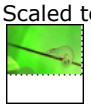
Ursprungsbild



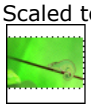
Thumbnails (48x48)

- Scaled to fit = 2

- Scaled to fit proportional = 5



- Scaled to fit prop centered = 6 (Standard)



Hinweis: Bei "Scaled to fit proportional" und "Scaled to fit prop centered" erscheint der freie Platz in weiß. Bei der Anwendung auf ein Feld bzw. eine Variable vom Typ Bild in 4D Formularen erscheint der freie Platz transparent.

Der optionale Parameter *Tiefe* wird ignoriert und muss weggelassen werden. Der Befehl verwendet immer die aktuelle Bildschirmauflösung (Anzahl der Farben).

Equal pictures

Equal pictures (Bild1 ; Bild2 ; Maske) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bild1	Bildfeld, Bildvariable	➡	Quellbild
Bild2	Bildfeld, Bildvariable	➡	Bild zum Vergleichen
Maske	Bildfeld, Bildvariable	⬅	Ergebnismaske
Funktionsergebnis	Boolean	↩	Wahr, wenn beide Bilder identisch sind; sonst Falsch

Beschreibung

Die Funktion **Equal pictures** vergleicht genau Ausmaße und Inhalt von zwei Bildern.
In *Bild1* übergeben Sie das Quellbild, in *Bild2* das Bild, welches sie damit vergleichen wollen.

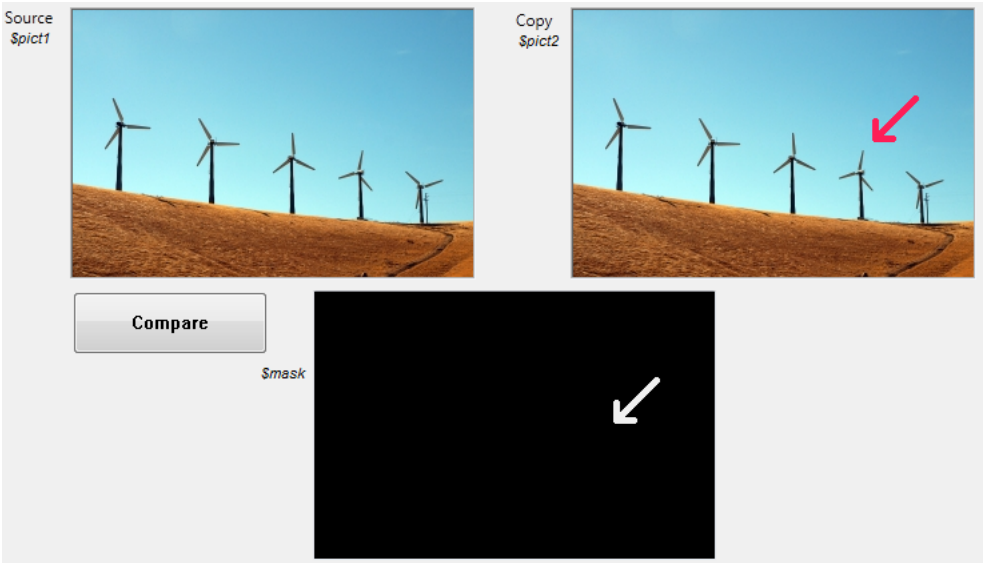
- Haben die Bilder nicht die gleichen Ausmaße, gibt die Funktion **Falsch** zurück und der Parameter *Maske* enthält ein leeres Bild.
- Haben die Bilder die gleichen Ausmaße, jedoch unterschiedlichen Inhalt, gibt die Funktion **Falsch** zurück und der Parameter *Maske* enthält die Bildmaske, die sich aus dem Vergleich von zwei Bildern ergibt. Der Vergleich wird Pixel für Pixel durchgeführt. Jedes nicht-passende Pixel erscheint weiß auf schwarzem Hintergrund.
- Sind beide Bilder genau gleich, gibt die Funktion **Wahr** zurück und der Parameter *Maske* enthält ein komplett schwarzes Bild.

Systemvariablen und Mengen

Wurde der Befehl erfolgreich ausgeführt (die beiden Bilder werden verglichen), wird die Systemvariable *OK* auf 1 gesetzt. Bei Problemen, insbesondere, wenn eins der Bilder nicht initialisiert ist (weißes Bild), wird die Variable *OK* auf 0 (Null) gesetzt.

Beispiel

Folgendes Beispiel vergleicht zwei Bilder (pict1 und pict2) und zeigt die Ergebnismaske:



Hier ist der Code für die Schaltfläche **Compare**:

```
$equal :=Equal pictures($pict1;$pict2;$mask)
```

Get picture file name

Get picture file name (Bild) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bild	Bildfeld, Bildvariable	➡	Bild, dessen Standardname ermittelt werden soll
Funktionsergebnis	Text	⮌	Standardname der Bilddatei

Beschreibung

Die Funktion **Get picture file name** gibt den aktuellen Standardnamen des als Parameter übergebenen Bildes zurück.

Der Standardname wird verwendet, wenn das Bild in eine Datei der Festplatte exportiert wird.
Der Name wird u.U. automatisch gesetzt, basierend auf dem Originalnamen der Bilddatei, die in das Datenfeld vom Typ Bild oder Variable importiert wird, oder über den Befehl **SET PICTURE FILE NAME**. Weitere Informationen dazu finden Sie im Abschnitt des Handbuchs *4D Designmodus*.

Hat *Bild* keinen Standardnamen, gibt die Funktion einen leeren String zurück.

GET PICTURE FORMATS

GET PICTURE FORMATS (Bild ; CodecIDs)

Parameter	Typ		Beschreibung
Bild	Bild	➡	Bildfeld oder -variable zum Analysieren
CodecIDs	Array Text	⬅	Codec Kennung für Bilder

Beschreibung

Der Befehl **GET PICTURE FORMATS** gibt ein Array mit einer Auflistung aller Bildformate zurück, die im Parameter *Bild* enthalten sind. Ein 4D Bildfeld bzw. Bildvariable kann dasselbe Bild in verschiedenen Formaten, wie PNG, BMP, GIF enthalten.

Im Parameter *Bild* übergeben Sie ein Feld oder eine Variable vom Typ Bild, dessen vorhandene Formate im Array *CodecIDs* zurückgegeben werden sollen.

Die zurückgegebenen Codec Kennungen werden von 4D in exakt derselben Weise wie für **PICTURE CODEC LIST** erstellt. Es gibt folgende Formen:

- Als Endung (zum Beispiel ".gif")
- Als Mime Typ (zum Beispiel "image/jpeg")
- Als 4-stelliger QuickTime Code

Hinweise:

- Folgende Codecs, von 4D intern verwaltet, werden immer als Endungen zurückgegeben: JPEG, PNG, TIFF, GIF, BMP, SVG, PDF, EMF.
- 4-stellige QuickTime Codes werden in Anwendungen zurückgegeben, für die die Option zur Kompatibilität QuickTime support gesetzt wurde (über den Befehl **SET DATABASE PARAMETER**). Da QuickTime in 4D nicht mehr unterstützt wird, empfehlen wir jedoch, QuickTime Codecs nicht mehr zu verwenden.

Weitere Informationen dazu finden Sie im Abschnitt **Überblick über Bilder**.

Beispiel

Die Bildformate erhalten, die in einem Feld für den aktuellen Datensatz gespeichert sind:

```
ARRAY TEXT($aTPictureFormats;0)
 //Alle gespeicherten Formate erhalten
GET PICTURE FORMATS([Employees]Photo;$aTPictureFormats)
```

GET PICTURE FROM LIBRARY

GET PICTURE FROM LIBRARY (BildRef | BildName ; Bild)

Parameter	Typ		Beschreibung
BildRef | BildName	Lange Ganzzahl, String	➡	Referenznummer oder Name von Bild aus Bildbibliothek
Bild	Bildvariable	⬅	Bild aus Bildbibliothek

Beschreibung

Der Befehl **GET PICTURE FROM LIBRARY** gibt im Parameter *Bild* das Bild aus der Bildbibliothek mit der Referenznummer *BildRef* oder mit dem Bildnamen *Bildname* zurück.
Gibt es kein Bild mit dieser Referenznummer, bleibt der Parameter *Bild* unverändert.

Beispiel 1

Folgendes Beispiel gibt in *vgMyPicture* das in der lokalen Variable *$vlPicRef* gespeicherte Bild zurück:

```
GET PICTURE FROM LIBRARY($vlPicRef;vgMyPicture)
```

Beispiel 2

Folgendes Beispiel gibt in *$DDcom_Prot_MyPicture* das Bild mit Namen "DDcom_Prot_Button1" aus der Bildbibliothek zurück.

```
GET PICTURE FROM LIBRARY("DDcom_Prot_Button1";$DDcom_Prot_MyPicture)
```

Beispiel 3

Siehe drittes Beispiel zum Befehl **PICTURE LIBRARY LIST**.

Systemvariablen und Mengen

Gibt es die Bildbibliothek, nimmt die Systemvariable **OK** den Wert 1 an, andernfalls den Wert Null.

Fehlerverwaltung

Reicht der Speicher nicht aus, um das Bild zu laden, wird Fehler -108 erzeugt. Sie können diesen Fehler mit einer Methode zur Fehlerverwaltung ausfindig machen.

GET PICTURE KEYWORDS

GET PICTURE KEYWORDS (Bild ; arrkeywords {; *})

Parameter	Typ		Beschreibung
Bild	Bildfeld, Bildvariable	➡	Bild, dessen zugeordnete Schlüsselwörter ermittelt werden sollen
arrkeywords	Array Text	⬅	Array mit den entnommenen Schlüsselwörtern
*	Operator	➡	Mit Stern: Unterschiedliche Werte

Beschreibung

Der Befehl **GET PICTURE KEYWORDS** gibt im Array *arrkeywords* die Liste der Schlüsselwörter für das als Parameter übergebene Bild zurück.

Es werden nur über IPTC/Keywords Metadaten gesetzte Schlüsselwörter berücksichtigt. Andere Arten von Metadaten werden ignoriert. Der Befehl arbeitet nur mit Bildtypen, die diese Art Metadaten unterstützen (JPEG, TIFF, etc.).

Hinweis: Metadaten vom Typ IPTC/Keywords lassen sich in 4D v13 indizieren (siehe Abschnitt des Handbuchs *4D Designmodus*).

Übergeben Sie den Parameter *, gibt der Befehl nur unterschiedliche Werte zurück, d.h. eine Liste ohne Duplikate.
Enthält das Bild keine Schlüsselwörter oder IPTC/Keywords Metadaten, gibt der Befehl ein leeres Array zurück und es wird kein Fehler erzeugt.

Hinweis: Dieser Befehl kann je nach der eingestellten Option "Nur nicht alphanumerische Zeichen als Schlüsselwörter ansehen" in den Datenbank-Eigenschaften unterschiedliche Ergebnisse liefern (siehe Abschnitt **Seite Datenbank/ Datenspeicherung** des Handbuchs *4D Designmodus*).

GET PICTURE METADATA

GET PICTURE METADATA (Bild ; metaName ; metaInhalt {; metaName2 ; metaInhalt2 ; ... ; metaNameN ; metaInhaltN})

Parameter	Typ		Beschreibung
Bild	Bild	➡	Bild, dessen Metadaten Sie erhalten wollen
metaName	Text	➡	Name oder Pfad des zu lesenden Blocks
metaInhalt	Variable	⬅	Metadaten Inhalt

Beschreibung

Der Befehl **GET PICTURE METADATA** liest den Inhalt der Metadaten oder Meta-Tags von *Bild* (4D Feld vom Typ Bild oder Variable). Weitere Informationen zu Metadaten finden Sie unter dem Befehl **SET PICTURE METADATA**.

Im Parameter *metaName* übergeben Sie einen String, der den Typ der zu findenden Metadaten angibt. Sie können folgendes übergeben:

- Eine Konstante unter dem Thema **Bild Metadaten Namen** mit einem Tag Pfad,
- Den Namen eines kompletten Blocks mit Metadaten ("TIFF", "EXIF", "GPS" oder "IPTC"),
- Einen leeren String ("").

Im Parameter *metaInhalt* übergeben Sie die Variable, welche die Metadaten empfangen soll

- Haben Sie in *metaName* einen Tag Pfad übergeben, enthält der Parameter *metaInhalt* direkt den zu lesenden Wert. Der Wert wird in den Typ der Variablen konvertiert. Variablen vom Typ Text werden in XML (XMP Standard) formatiert. Sie können ein Array verwenden, wenn Metadata mehr als einen Wert enthält. Das ist insbesondere für die Tags IPTC Keywords der Fall.
- Haben Sie in *metaName* einen Blocknamen oder einen leeren String übergeben, muss der Parameter *metaInhalt* eine gültige Referenz auf ein XML DOM Element sein. In diesem Fall wird der Inhalt des angegebenen Blocks (oder alle Blöcke, wenn Sie in *metaName* einen leeren String übergeben haben) in das referenzierte Element kopiert.

Beispiel 1

DOM Baum Strukturen verwenden:

```
$xml:=DOM Create XML Ref("Root")\Einen XML DOM Baum erstellen

 // Empfangen von TIFF metadata
$_Xml_TIFF:=DOM Create XML element($xml;"/Root/TIFF")
GET PICTURE METADATA(vPicture;"TIFF";$_Xml_TIFF)

 // Empfangen von GPS metadata
$_Xml_GPS:=DOM Create XML element($xml;"/Root/GPS")
GET PICTURE METADATA(vPicture;"GPS";$_Xml_GPS)
```

Beispiel 2

Variablen verwenden

```
C_DATE($dateAsDate)
GET PICTURE METADATA(vPicture;TIFF date time;$dateAsDate)
 //gibt nur das Datum zurück, da $dateAsDate vom Typ Datum ist

C_TEXT($dateAsText)
GET PICTURE METADATA(vPicture;TIFF date time;$dateAsText)
 //gibt nur das Datum zurück, jedoch in XML Format

C_INTEGER($urgency)
GET PICTURE METADATA(vPicture;IPTC urgency;$urgency)
```

Beispiel 3

Tags mit mehrfachen Werten in einer Textvariablen empfangen:

```
ARRAY TEXT($tTkeywords;0)
GET PICTURE METADATA(vPicture;"IPTC/Keywords";$tTkeywords)
```

Nach Ausführen des Befehls enthält arrTkeywords zum Beispiel:

```
$arrTkeywords{1}="Frankreich"
$arrTkeywords{2}="Europa"
```

Beispiel 4

Tags mit mehrfachen Werten in einer Textvariablen empfangen

```
C_TEXT($vTwords;0)
GET PICTURE METADATA(vPicture;IPTC/Keywords;$vTwords)
```

Nach Ausführen des Befehls enthält vTwords z.B. "Frankreich;Europa".

Systemvariablen und Mengen

Die Systemvariable *OK* gibt 1 zurück, wenn das Auffinden der Metadaten korrekt abgelaufen ist. 0, wenn ein Fehler auftritt oder mindestens einer der Tags nicht gefunden wird. In allen Fällen werden alle Werte, die gelesen werden können, zurückgegeben.

Is picture file

Is picture file (Dateipfad {; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Dateipfad	Text	⇒	Pfadname der Datei
*	Operator	⇒	Daten bestätigen
Funktionsergebnis	Boolean	⇐	Wahr = Dateipfad gibt eine Bilddatei an; andernfalls Falsch

Beschreibung

Die Funktion **Is picture file** testet die im Parameter *Dateipfad* angegebene Datei und gibt Wahr zurück, wenn es eine gültige Bilddatei ist. Die Funktion gibt *Falsch* zurück, wenn die Datei nicht vom Typ Bild ist oder nicht gefunden wird.

In *Dateipfad* übergeben Sie den Pfadnamen der zu prüfenden Bilddatei. Der Pfad muss in der Syntax des Betriebssystems ausgedrückt werden. Sie können einen absoluten bzw. relativen Pfadnamen zur Strukturdatei der Datenbank übergeben. Übergeben Sie einen leeren String (""), gibt die Funktion *Falsch* zurück.

Ohne den optionalen Parameter * überprüft die Funktion die Datei nur anhand ihrer Endung mit den in der Liste vorhandenen Codecs. Wollen Sie ermöglichen, Dateien ohne Endung zu prüfen oder eine weitergehende Prüfung ausführen, übergeben Sie den Parameter *. Dann macht die Funktion zusätzliche Tests: Sie lädt und prüft den Kopfteil der Datei und sucht die Codecs, um das Bild zu bestätigen. Beachten Sie jedoch, dass diese Syntax die Ausführung der Funktion verlangsamt.

Hinweis: Die Funktion gibt für PDF Dateien unter Windows und EMF Dateien auf Mac OS *Wahr* zurück, obwohl die Bilder nicht auf dieser Plattform angezeigt werden können.

PICTURE CODEC LIST

PICTURE CODEC LIST (CodecArray {; ArrayName}{; *})

Parameter	Typ		Beschreibung
CodecArray	Array String	⇐	Kennung der verfügbaren Bild Codecs
ArrayName	Array String	⇐	Namen der Bild Codecs
*	Operator	⇒	Liste der lesenden (decoding) Codecs zurückgeben

Beschreibung

Der Befehl **PICTURE CODEC LIST** füllt das Array *CodecArray* mit der Liste der Bild Codec Kennung, die auf dem ausführenden Rechner verfügbar sind. Diese Liste enthält die Codec Kennung der Bildformate, die 4D nativ verwaltet.

Die Codec Kennung kann im Array *CodecArray* in verschiedenen Formen zurückgegeben werden:

- Als Endung (zum Beispiel ".gif")
- Als Mime Typ (zum Beispiel, "image/jpg")

Hinweis zur Kompatibilität: Ist QuickTime in der Datenbank aktiviert (siehe Abschnitt **Überblick über Bilder**), lassen sich auch 4-stellige QuickTime Codes zurückgeben (zum Beispiel "PNTG").

Die vom Befehl zurückgegebene Form richtet sich nach der Art, wie das Codec auf Ebene des Betriebssystems gespeichert ist. Über das optionale Array *ArrayName* können Sie den Namen der einzelnen Codecs wiederfinden. Diese Namen sind aussagekräftiger als die Kennuung. Über dieses Array können Sie z.B. ein Menü mit den verfügbaren Codecs einrichten und anzeigen.
Standardmäßig, d.h. ohne den Parameter *, gibt der Befehl nur die Codecs zurück, die zum Codieren (Schreiben) der Bilder verwendet werden können. Sie können diese Kennungn im Parameter *Format* für die Befehle **WRITE PICTURE FILE** und **PICTURE TO BLOB** übergeben, die zum Exportieren von Bildern verwendet werden.

Mit dem Parameter * gibt der Befehl auch die Liste der Codecs zurück, die zum Decodieren (Lesen) der Bilder verwendet werden. Die beiden Listen schließen sich nicht gegenseitig aus, einige lesende und schreibende Codec sind identisch. Codecs zum Codieren von Bildern werden in der Regel zum Decodieren verwendet. Dagegen lassen sich decodierende Codecs nicht unbedingt zum Codieren verwenden. So steht z.B. das Codec ".jpg" in beiden Listen, während ".xbmp" nur in der Liste der lesenden (decoding) Codecs vorhanden ist.

PICTURE LIBRARY LIST

PICTURE LIBRARY LIST (BildRefs ; BildNamen)

Parameter	Typ		Beschreibung
BildRefs	Array Lange Ganzzahl	⬅	Referenznummern der Bilder in Bildbibliothek
BildNamen	Array String	⬅	Namen der Bilder in Bildbibliothek

Beschreibung

Der Befehl **PICTURE LIBRARY LIST** gibt die Referenznummern und -namen der Bilder zurück, die in der Bildbibliothek der Datenbank gespeichert sind.

Nach dem Aufruf finden Sie die Referenznummer im Array *BildRefs*, die Namen im Array *BildNamen*. Beide Arrays sind synchronisiert: Das n-te Element von *BildRefs* entspricht dem n-ten Element von *BildNamen*.

Bei Bedarf erstellt der Befehl automatisch die Arrays *BildRefs* und *BildNamen*.

Die Bildnamen in der Bildbibliothek können bis zu 255 Zeichen lang sein. Gibt es keine Bilder in der Bildbibliothek, erhalten Sie leere Arrays.

Mit der Funktion **Size of array** erhalten Sie die Größe eines der Arrays und können so feststellen, wieviel Bilder in der Bildbibliothek gespeichert sind.

Beispiel 1

Folgender Code gibt den Katalog der Bildbibliothek in den Arrays *alPicRef* und *asPicName* zurück:

```
PICTURE LIBRARY LIST(alPicRef;asPicName)
```

Beispiel 2

Folgendes Beispiel prüft, ob die Bildbibliothek leer ist oder nicht:

```
PICTURE LIBRARY LIST(alPicRef;asPicName)
If(Size of array(alPicRef)=0)
   ALERT("Die Bildbibliothek ist leer.")
Else
   ALERT("Die Bildbibliothek enthält "+String(Size of array(alPicRef))+" Bilder.")
End if
```

Beispiel 3

Folgendes Beispiel exportiert die Bildbibliothek in ein Dokument auf der Festplatte:

```
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(" Bildbibliothek wurde nicht exportiert, mehr Speicher zuweisen.")
```

```
            DELETE DOCUMENT(Document)
        End if
    End if
Else
    ALERT("Die Bildbibliothek ist leer.")
End if
```

PICTURE PROPERTIES

PICTURE PROPERTIES (Bild ; Breite ; Höhe {; hOffset {; vOffset {; Modus}}})

Parameter	Typ		Beschreibung
Bild	Bild	⇒	Bild, dessen Eigenschaften bestimmt werden
Breite	Lange Ganzzahl	⇐	Breite des Bildes in Pixel
Höhe	Lange Ganzzahl	⇐	Höhe des Bildes in Pixel
hOffset	Lange Ganzzahl	⇐	Horizontaler Versatz bei Anzeige auf Hintergrund
vOffset	Lange Ganzzahl	⇐	Vertikaler Versatz bei Anzeige auf Hintergrund
Modus	Lange Ganzzahl	⇐	Übertragungsmodus bei Anzeige auf Hintergrund

Beschreibung

Der Befehl **PICTURE PROPERTIES** gibt Information über das in *Bild* übergebene Bild.

Die Parameter *Breite* und *Höhe* geben die Breite und Höhe des Bildes zurück.

Die Parameter *hOffset*, *vOffset* und *Modus* geben die horizontale und vertikale Position sowie den Übertragungsmodus des Bildes an, wenn es in einem Formular im Hintergrund angezeigt wird.

Picture size

Picture size (Bild) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bild	Bild	⇒	Bilddatenfeld oder Variable
Funktionsergebnis	Lange Ganzzahl	⇐	Größe des Bildes in Bytes

Beschreibung

Die Funktion **Picture size** gibt die Größe von *Bild* in Bytes zurück.

PICTURE TO BLOB

PICTURE TO BLOB (Bild ; BildBlob ; Codec)

Parameter	Typ		Beschreibung
Bild	Bild	➡	Feld oder Variable vom Typ Bild
BildBlob	BLOB	⬅	BLOB für das konvertierte Bild
Codec	String	➡	Codec Kennung für Bild

Beschreibung

Der Befehl **PICTURE TO BLOB** konvertiert ein Bild, das in einer 4D Variablen bzw. einem Feld gespeichert ist, in ein anderes Format und legt das neue Bild in ein BLOB.

Im Parameter *Bild* wird ein 4D Feld bzw. eine Variable übergeben. Im Parameter *BildBlob* wird eine BLOB Variable oder ein Feld übergeben, welches das konvertierte Bild enthalten sollte.

Im Parameter *Codec* übergeben Sie das Konvertierungsformat in einem String.

Codec kann eine Endung, z.B. gif oder ein Mime Typ, z.B. "image/jpg" sein. Die Liste der verfügbaren Codecs finden Sie unter dem Befehl **PICTURE CODEC LIST**.

Sobald der Befehl ausgeführt wurde, enthält *BildBlob* das Bild im festgelegten Format.

Bei erfolgreicher Konvertierung wird die Systemvariable OK auf 1 gesetzt. War die Konvertierung nicht erfolgreich (Konverter nicht verfügbar), hat OK den Wert 0 (Null), das erstellte BLOB ist leer (0 Byte).

Beispiel

Ein Bild vom Ausgangsformat in GIF-Format konvertieren und auf einer statischen Web Seite anzeigen:

```
C_PICTURE($picture)
C_BLOB($BLOB)
C_TEXT($path)

$path:=Get 4D folder(Current resources folder)+"Bilder"+Folder separator+"Sunrise.psd"  //Bild im Ordner Bilder innerhalb des Ordners "Resources" finden
READ PICTURE FILE($path;$picture)  //Das Bild in die Bildvariable setzen

PICTURE TO BLOB($picture;$BLOB;".gif")  //Bild in ".gif" Format konvertieren
WEB SEND BLOB($BLOB;"image/gif")
```

READ PICTURE FILE

READ PICTURE FILE (DateiName ; Bild {; *})

Parameter	Typ		Beschreibung
DateiName	String	➡	Name oder ganzer Pfadname der zu lesenden Datei oder leerer String
Bild	Bild	⬅	Feld oder Variable, die das Bild aufnimmt
*	Operator	➡	Mit Stern = akzeptiere jeden Dateityp

Beschreibung

Der Befehl **READ PICTURE FILE** öffnet das in *DateiName* gesicherte Bild und lädt es in das 4D Datenfeld oder die Variable *Bild*.

In *DateiName* können Sie nur den Dateinamen oder den kompletten Pfadnamen für die einzulesende Datei übergeben. Übergeben Sie nur den Dateinamen, sollten Sie die Datei neben die Strukturdatei der Datenbank legen. Unter Windows müssen Sie die Dateiendung angeben.
Wird in *DateiName* ein leerer String ("") übergeben, erscheint der Standard-Öffnen Dialog. Hier kann der Benutzer für die einzulesende Datei Name, Position und die vorhandenen Formate wählen.

Die Liste der verfügbaren Formate erhalten Sie über den Befehl **PICTURE CODEC LIST**.

In *Bild* übergeben Sie die Variable bzw. das Feld vom Typ Bild, welche das gelesene Bild aufnimmt.

Hinweis: Das interne Bildformat wird in der 4D Variablen bzw. im 4D Feld gespeichert.

Übergeben Sie den optionalen Parameter *, akzeptiert der Befehl jeden Dateityp. So können Sie auch mit Bildern ohne passenden Codec arbeiten. Weitere Informationen dazu finden Sie unter dem Befehl **BLOB TO PICTURE**.

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, enthält die Systemvariable *Document* den vollen Pfadnamen zum Öffnen der Datei und die Systemvariable *OK* wird auf 1 gesetzt. Sonst hat *OK* den Wert 0 (Null).

REMOVE PICTURE FROM LIBRARY

REMOVE PICTURE FROM LIBRARY (BildRef | BildName)

Parameter	Typ		Beschreibung
BildRef | BildName	Lange Ganzzahl, String	➡	Referenznummer oder Name von Bild aus Bildbibliothek

Beschreibung

Der Befehl **REMOVE PICTURE FROM LIBRARY** entfernt aus der Bildbibliothek das Bild mit der Referenznummer *BildRef* bzw. dem Bildnamen *BildName*.

Gibt es kein Bild mit dieser Referenznummer bzw. Namen, hat der Befehl keine Auswirkung.

4D Server: REMOVE PICTURE FROM LIBRARY kann nicht in einer Methode verwendet werden, die auf dem Serverrechner ausgeführt wird (Serverprozedur oder Trigger). Der Befehl hat auf dem Server keine Auswirkung.

Warnung: Designobjekte, wie z.B. Einträge in hierarchischen Listen, Menüeinträge, etc. können sich auf Bilder der Bildbibliothek beziehen. Seien Sie also vorsichtig, wenn Sie Bilder der Bildbibliothek per Programmierung entfernen.

Beispiel 1

Folgendes Beispiel löscht das Bild #4444 in der Bildbibliothek.

```
REMOVE PICTURE FROM LIBRARY(4444)
```

Beispiel 2

Folgendes Beispiel löscht in der Bildbibliothek alle Bilder, die mit dem Dollarzeichen beginnen (*$*):

```
PICTURE LIBRARY LIST($alPicRef;$asPicName)
For($vlPicture;1;Size of array($alPicRef))
   If($asPicName{$vlPicture}="$@")
      REMOVE PICTURE FROM LIBRARY($alPicRef{$vlPicture})
   End if
End for
```

SET PICTURE FILE NAME

SET PICTURE FILE NAME (Bild ; Dateiname)

Parameter	Typ		Beschreibung
Bild	Bildfeld, Bildvariable	➡	Bild, dessen Standardname gesetzt wird
Dateiname	Text	➡	Standard Bildname

Beschreibung

Der Befehl **SET PICTURE FILE NAME** fügt den standardmäßigen Dateinamen für das als Parameter übergebene Bild hinzu oder verändert ihn.

Der Name wird u.U. automatisch gesetzt, basierend auf dem Originalnamen der Bilddatei, die in das Bildfeld bzw. die Variable importiert wird oder während einem vorangegangenen Aufruf von **SET PICTURE FILE NAME**.

Dieser Standardname wird verwendet, wenn das Bild in eine Datei der Festplatte exportiert wird. Wird der Inhalt des Feldes in eine Variable oder in ein anderes Datenfeld kopiert, wird der Standardname ebenfalls kopiert. Weitere Informationen dazu finden Sie im Abschnitt des Handbuchs *4D Designmodus*.

SET PICTURE METADATA

SET PICTURE METADATA (Bild ; metaName ; metaInhalt {; metaName2 ; metaInhalt2 ; ... ; metaNameN ; metaInhaltN})

Parameter	Typ		Beschreibung
Bild	Bild	➡	Bild, dessen Metadaten Sie setzen wollen
metaName	Text	➡	Name oder Pfad des zu schreibenden Blocks
metaInhalt	Variable	➡	Metadaten Inhalt

Beschreibung

Der Befehl **SET PICTURE METADATA** schreibt oder ändert den Inhalt der Metadaten bzw. Meta-Tags, die in *Bild* (4D Feld vom Typ Bild oder Variable) gefunden werden, wenn sie änderbar sind.

Metadaten sind zusätzliche Information, die in Bilder eingefügt ist. 4D ermöglicht vier Arten von standardmäßigen Metadaten zu verwalten: EXIF, GPS, IPTC und TIFF.

Hinweis: Eine ausführliche Beschreibung der Metadaten Typen finden Sie unter http://www.iptc.org/std/IIM/4.1/specification/IIMV4.1.pdf (IPTC) und http://exif.org/Exif2-2.PDF (TIFF, EXIF und GPS).

Im Parameter *metaName* übergeben Sie einen String, der den Typ der Metadaten, die zu schreiben oder zu ändern sind, angibt. Sie können folgendes übergeben:

- Eine Konstante aus dem Thema **Bild Metadaten Namen**. Es enthält alle Tags, die 4D unterstützt. Jede Konstante enthält ein Tag Pfad, z.B."TIFF/DateTime"),
- Den Namen des kompletten Blocks von Metadaten ("TIFF", "EXIF", "GPS" oder "IPTC"),
- Einen leeren String ("").

Im Parameter *metaInhalt* übergeben Sie die neuen Werte der Metadaten:

- Haben Sie in *metaName* eine Konstante mit Tag Pfad übergeben, können Sie im Parameter *metaInhalt* den zu schreibenden Wert direkt übergeben oder eine passende Konstante aus dem Thema **Bild Metadaten Werte**. Der Wert kann je nach den festgelegten Metadaten vom Typ Text, Lange Ganzzahl, Zahl, Datum oder Zeit sein. Sie können ein Array verwenden, wenn Metadata mehr als einen Wert enthält. Übergeben Sie einen String, muss er in XML (XMP Standard) formatiert sein. Sie können einen leeren String ("") übergeben, um bereits vorhandene Metadaten zu löschen.
- Haben Sie in *metaName* einen Blocknamen oder einen leeren String übergeben, können Sie im Parameter *metaInhalt* eine XML DOM Referenz des Elements mit den zu schreibenden Metadaten übergeben. Bei einem leeren String werden alle Metadaten geändert.

Warnung: Bestimmte Metadaten sind nur lesbar und lassen sich deshalb nicht mit **SET PICTURE METADATA** verändern. Das gilt zum Beispiel für TIFF XResolution/TIFF YResolution, EXIF Color Space oder EXIF Pixel X Dimension/EXIF Pixel Y Dimension.

Tritt unter Windows ein Fehler während der Ausführung des Befehls auf, wird die Systemvariable OK auf 0 gesetzt. Beachten Sie, dass auf Mac OS das Betriebssystem (Image I0) keine Fehler beim Schreiben von Metadaten ausweist. Deshalb ändert dieser Befehl auf Mac OS die Systemvariable OK nicht.

Hinweise:

Nur bestimmte Bildformate (insbesondere JPEG und TIFF) unterstützen Metadaten. Umgekehrt akzeptieren Formate wie GIF oder BMP keine Metadaten. Konvertieren Sie ein Bild mit Metadaten in ein Format, das dies nicht unterstützt, geht die Information verloren.

Unter OS X Version 10.7 (Lion) kann ein Fehler in einem nativen Framework dazu führen, das das Encoden bzw Decoden von Bild-Metadaten zu Genauigkeitsverlusten in GPS Koordinaten führen. Wir empfehlen daher dringend auf OS X 10.8 (Moutain Lion) oder 10.9 (Maverick) zu aktualisieren.

Beispiel 1

Mehrere Werte der Metadaten "Keywords" über Arrays setzen:

```
ARRAY TEXT($arrTkeywords;2)
$arrTkeywords{1}:="Frankreich"
$arrTkeywords{2}:="Europa"
SET PICTURE METADATA(vPicture;IPTC keywords;$arrTkeywords)
```

Beispiel 2

GPS Block über eine DOM Referenz schreiben:

```
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)
 //hier zeigt $gpsRef aktuell auf das GPS Element
   ...
```

```
End if
DOM CLOSE XML($domMetas)
```

Hinweis

Werden alle Metadaten über Referenz auf ein DOM Element verwaltet, werden die Tags als Attribute gespeichert, angehängt an ein Element (Kind des referenzierten Elements), dessen Name der Blockname (TIFF, IPTC, etc.) ist. Wird ein bestimmter Metablock bearbeitet, werden die Block Tags als Attribute gespeichert, die über den Befehl direkt an das referenzierte Element angehängt werden.

SET PICTURE TO LIBRARY

SET PICTURE TO LIBRARY (Bild ; BildRef ; BildName)

Parameter	Typ		Beschreibung
Bild	Bild	⇒	Neues Bild
BildRef	Lange Ganzzahl	⇒	Referenznummer von Bild aus Bilbbibliothek
BildName	String	⇒	Neuer Name des Bildes

Beschreibung

Der Befehl **SET PICTURE TO LIBRARY** erstellt ein neues Bild oder ersetzt ein Bild in der Bildbibliothek.
Übergeben Sie vor dem Aufruf:

- in *BildRef* die Referenznummer des Bildes (Bereich 1...32767)
- In *Bild* das Bild selbst.
- In *BildName* den Namen des Bildes (max. 255 Zeichen lang).

Gibt es in der Bildbibliothek bereits ein Bild mit der gleichen Referenznummer, wird der Bildinhalt ersetzt und das Bild mit den in *Bild* und *BildName* übergebenen Werten umbenannt.
Existiert die Referenznummer noch nicht, wird ein neues Bild in der Bildbibliothek hinzugefügt.
4D Server: SET PICTURE TO LIBRARY kann nicht in einer Methode verwendet werden, die auf dem Server ausgeführt wird. (Serverprozedur oder Trigger). Der Befehl hat auf dem Server keine Auswirkung.
Warnung: Designobjekte, wie z.B. Einträge in hierarchischen Listen, Menüeinträge, etc. können sich auf Bilder der Bildbibliothek beziehen. Seien Sie also vorsichtig, wenn Sie Bilder der Bildbibliothek per Programmierung ändern.
Hinweis: Übergeben Sie in *Bild* ein leeres Bild oder in *BildRef* einen negativen Wert bzw. die Ziffer Null, hat der Befehl keine Auswirkung.

Beispiel 1

Folgendes Beispiel sucht - unabhängig vom aktuellen Inhalt der Bildbibliothek - zuerst eine einmalige Referenznummer für ein Bild und fügt dann ein neues Bild in der Bildbibliothek hinzu:

```
PICTURE LIBRARY LIST($alPicRef;$asPicNames)
Repeat
   $vlPicRef:=1+Abs(Random)
Until(Find in array($alPicRef;$vlPicRef)<0)
SET PICTURE TO LIBRARY(vgPicture;$vlPicRef;"New Picture")
```

Beispiel 2

Folgendes Beispiel importiert Bilder aus einem Dokument auf der Festplatte in die Bildbibliothek, die mit dem dritten Beispiel zum Befehl **PICTURE LIBRARY LIST** erstellt wurde:

```
SET CHANNEL(10;"")
If(OK=1)
   RECEIVE VARIABLE($vsTag)
   If($vsTag="4DV6PICTURELIBRARYEXPORT")
      RECEIVE VARIABLE($vlNbPictures)
      If($vlNbPictures>0)
         For($vlPicture;1;$vlNbPictures)
            RECEIVE VARIABLE($vlPicRef)
            If(OK=1)
               RECEIVE VARIABLE($vsPicName)
            End if
            If(OK=1)
               RECEIVE VARIABLE($vgPicture)
            End if
            If(OK=1)
               SET PICTURE TO LIBRARY($vgPicture;$vlPicRef;$vsPicName)
            Else
               $vlPicture:=$vlNbPictures+1
               ALERT("Diese Datei ist evtl. beschädigt.")
            End if
         End for
      Else
         ALERT("Diese Datei ist evtl. beschädigt.")
      End if
```

```
Else
    ALERT("Datei “"+Dokument+"” ist keine Bildbibliothek Exportdatei.")
End if
SET CHANNEL(11)
End
```

Fehlerverwaltung

Reicht der Speicher nicht aus, um das Bild der Bildbibliothek hinzuzufügen, wird Fehler -108 erzeugt. Es können auch E/A Fehler auftreten, wie z.B. die Strukturdatei ist gesperrt. Sie können diese Fehler mit einer Methode zur Fehlerverwaltung ausfindig machen.

TRANSFORM PICTURE

TRANSFORM PICTURE (Bild ; Operator {; Param1 {; Param2 {; Param3 {; Param4}}}})

Parameter	Typ		Beschreibung
Bild	Bild	➡	Quellbild zum Transformieren
		⬅	Bild nach Transformation
Operator	Lange Ganzzahl	➡	Art der Transformation
Param1	Zahl	➡	Parameter der Transformation
Param2	Zahl	➡	Parameter der Transformation
Param3	Zahl	➡	Parameter der Transformation
Param4	Zahl	➡	Parameter der Transformation

Beschreibung

Der Befehl **TRANSFORM PICTURE** führt eine Transformation vom Typ *Operator* für das Bild im Parameter *Bild* durch.

Hinweis: Dieser Befehl erweitert die Funktionalitäten der herkömmlichen Operatoren für Bildtransformation, wie +/, etc. (siehe **Bildoperatoren**). Diese Operatoren sind in 4D weiterhin verwendbar.

Das Quellbild wird direkt nach Ausführen des Befehls verändert. Beachten Sie, dass die Operationen bis auf "Beschnitt" und "In Graustufen umwandeln" durch Ausführen der entgegengesetzten Operation oder über die Operation "Reset" wieder umkehrbar sind. So erhält zum Beispiel ein Bild, das auf 1% reduziert wurde, durch anschließendes Vergrößern mit dem Faktor 100 wieder ohne Beeinträchtigung die Originalgröße. Transformationen verändern nicht den ursprünglichen Bildtyp. So bleibt ein Vektor-Bild nach der Transformation weiterhin ein Vektor-Bild.

In *Operator* übergeben Sie die Nummer der auszuführenden Operation, in *Param1..4* die dafür benötigten Parameter. Ihre Anzahl richtet sich nach der Operation. Sie können eine der Konstanten unter dem Thema **Bildtransformation** verwenden. Die folgende Tabelle beschreibt die Operatoren und dazugehörigen Parameter:

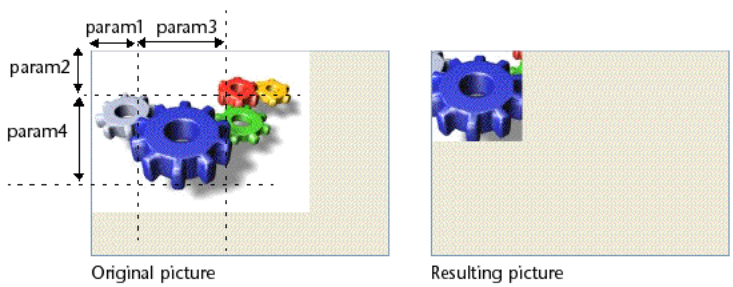
Operator (Wert)	Param1	Param2	Param3	Param4	Werte	Rücksetzbar
Reset (0)	-	-	-	-	-	-
Scale (1)	Weite	Höhe	-	-	Faktoren	Ja
Translate (2)	X Achse	Y Achse	-	-	Pixel	Ja
Flip horizontally (3)	-	-	-	-		Ja
Flip vertically (4)	-	-	-	-	-	Ja
Crop (100)	X Orig.	Y Orig.	Breite	Höhe	Pixel	Nein
Fade to grey scale (101)	-	-	-	-	-	Nein
Transparency (102)	RGB Farbe	-	-	-	Hexadezimal	Nein

- Reset: Alle am Bild ausgeführten Matrix Operationen wie Skalieren, Spiegeln, usw. werden rückgängig gemacht.
- Scale: Das Bild wird gemäß den Werten in *Param1* und *Param2* horizontal und vertikal angepasst. Diese Werte sind Faktoren. Wollen Sie z.B. die Breite um 50% erweitern, übergeben Sie 1,5 in *Param1*, wollen Sie die Höhe um 50% verringern, übergeben Sie 0,5 in *Param2*.
- Translate: Das Bild wird durch *Param1* horizontal, durch *Param2* vertikal in Pixel bewegt. Übergeben Sie einen positiven Wert zum Verschieben nach rechts oder nach unten; einen negativen Wert zum Verschieben nach links oder nach oben.
- Flip horizontally and Flip vertically: Das Originalbild wird gespiegelt. Alles zuvor ausgeführte Bewegen wird nicht berücksichtigt.
- Crop: Das Bild wird beschnitten ausgehend vom Punkt der Koordinaten in *Param1* und *Param2* (in Pixel). Breite und Höhe des neuen Bildes wird durch *Param3* und *Param4* bestimmt.
  Diese Umwandlung lässt sich nicht rückgängig machen.
- Fade to grey scale: Das Bild wird in Graustufen umgewandelt. Dafür ist kein Parameter erforderlich. Diese Umwandlung lässt sich nicht rückgängig machen.
- Transparency: Auf das Bild wird Transparenz gemäß der in *Param1* angegebenen Farbe angewandt. Übergeben Sie z.B. 0x00FFFFFF (weiß) in *Param1*, werden alle weißen Pixel im Originalbild im umgewandelten Bild transparent. Diese Operation lässt sich auf Bitmap- oder Vektor-Bilder anwenden. Standardmäßig, d.h. ohne *Param1*, wird die Farbe Weiß (0x00FFFFFF) als Zielfarbe gesetzt. Diese Funktion dient speziell zum Verwalten von Transparenz in Bildern, die aus dem überholten PICT Format konvertiert wurden. Es lässt sich aber auch für Bilder in anderen Formaten verwenden. Die Umwandlung lässt sich nicht zurücksetzen.

Beispiel 1

Dieses Beispiel beschneidet ein Bild. Es wird im Format "Abgeschnitten (nicht-zentriert)"angezeigt:

```
TRANSFORM PICTURE($vpGears;Crop;50;50;100;100)
```

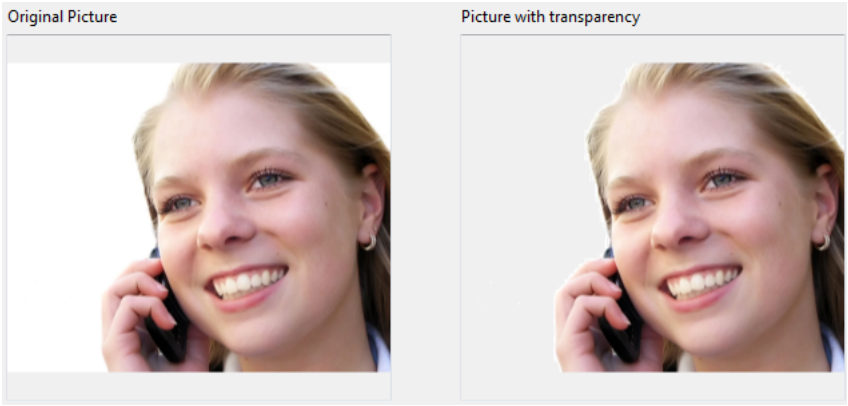


Beispiel 2

Mit folgendem Code können Sie die weißen Teile eines Bildes auf transparent setzen:

```
TRANSFORM PICTURE(Pict1;Transparency;0x00FFFFFF) //0x00FFFFFF ist weiß
```

Sie erhalten folgendes Ergebnis:



WRITE PICTURE FILE

WRITE PICTURE FILE (DateiName ; Bild {; Codec})

Parameter	Typ		Beschreibung
DateiName	Alpha	➡	Name oder voller Pfadname der zu schreibenden Datei oder leerer String
Bild	Bild	➡	Zu schreibendes 4D Bildfeld bzw. Variable
Codec	String	➡	Codec ID des Bildes

Beschreibung

Der Befehl **WRITE PICTURE FILE** speichert das im Parameter *Bild* übergebene Bild mit der vorgegebenen Codec ID auf der Festplatte.

In *DateiName* können Sie den vollen Pfadnamen für die zu erstellende Datei übergeben oder nur den Dateinamen selbst. Übergeben Sie nur den Dateinamen, wird die Datei neben die Strukturdatei der Datenbank gesetzt. Sie müssen die Dateiendung angeben.

Wird in *DateiName* ein leerer String ("") übergeben, erscheint der Standarddialog zum Sichern. Hier kann der Benutzer Name, Position und Format für die zu erstellende Datei eintragen. Ist dem Datenfeld vom Typ Bild ein Standardname zugeordnet, erscheint er im Dialogfenster. Weitere Informationen dazu finden Sie unter dem Befehl **SET PICTURE FILE NAME**.

In *Bild* übergeben Sie die Variable bzw. das Datenfeld vom Typ Bild mit dem Bild, das auf der Festplatte gesichert werden soll.

Der optionale Parameter *Codec* definiert das Format, in welchem das Bild gesichert wird. Codec kann eine Endung, z.B. .gif, ein Mime-Typ, z.B. image/jpeg sein, z.B. PNTG. Die Liste der verfügbaren Codecs erhalten Sie über den Befehl **PICTURE CODEC LIST**.

Lassen Sie den Parameter *Codec* weg, versucht der Befehl, den Codec gemäß der Dateiendung, übergeben in DateiName zu bestimmen. Bei der folgenden Anweisung:

```
WRITE PICTURE FILE("c:\folder\photo.jpg";myphoto)
```

... verwendet der Befehl den Codec JPEG, um das Bild zu speichern.

Entspricht die Endung keinem verfügbaren Codec, wird die Datei nicht gesichert und die Systemvariable OK wird auf 0 gesetzt. Übergeben Sie weder ein Codec noch eine Dateiendung, wird die Bilddatei im PICT Format gesichert.

Hinweis: Ist das Schreibformat von *Bild* (definiert über die Endung von *DateiName* oder *Codec*) das gleiche wie sein Originaltyp und wurde keine Transformierung ausgeführt, wird das Bild "wie gehabt" geschrieben, d.h. ohne Änderung.

Bei erfolgreich ausgeführtem Befehl enthält die Systemvariable *Document* den vollen Pfadnamen der erstellten Datei und wird auf 1 gesetzt. Sonst hat OK den Wert 0 (Null).

_o_PICTURE TO GIF

_o_PICTURE TO GIF (Pict ; BlobGIF)

Parameter	Typ		Beschreibung
Pict	Bild	➡	Feld oder Variable vom Typ Bild
BlobGIF	BLOB	⬅	BLOB mit Bild vom Typ GIF

Hinweis zur Kompatibilität

Dieser Befehl ist überholt. Sie können zum Konvertieren den Befehl **PICTURE TO BLOB** verwenden.

_o_PICTURE TYPE LIST

_o_PICTURE TYPE LIST (FormatArray {; NameArray})

Parameter	Typ		Beschreibung
FormatArray	Array String	⇐	QuickTime Codes für die verfügbaren Import/Export Formate
NameArray	Array String	⇐	Formatnamen

Hinweis zur Kompatibilität

Dieser Befehl benötigt QuickTime und bietet keinen Zugriff auf Formate, die 4D nativ verwaltet. So ist er nur begrenzt einsetzbar und sollte durch den neuen Befehl **PICTURE CODEC LIST** ersetzt werden.

_o_QT COMPRESS PICTURE

_o_QT COMPRESS PICTURE (Bild ; Methodenname ; Qualität)

Parameter	Typ		Beschreibung
Bild	Bild	➡	Zu komprimierendes Bild
		⬅	Komprimiertes Bild
Methodenname	String	➡	Komprimierungsverfahren (4 Zeichen)
Qualität	Lange Ganzzahl	➡	Qualität der Komprimierung (1..1000)

Hinweis zur Kompatibilität

Dieser Befehl ruft überholte Mechanismen auf und muss mit dem Befehl **CONVERT PICTURE** ersetzt werden.

_o_QT COMPRESS PICTURE FILE

_o_QT COMPRESS PICTURE FILE (DokRef ; Methodenname ; Qualität)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenznummer der Datei
Methodenname	String	➡	Komprimierungsverfahren (4 Zeichen)
Qualität	Lange Ganzzahl	➡	Qualität der Komprimierung (1..1000)

Hinweis zur Kompatibilität

Dieser Befehl ist überholt und muss mit dem Befehl **WRITE PICTURE FILE** bzw. **PICTURE TO BLOB** ersetzt werden.[#/descv]

_o_QT LOAD COMPRESS PICTURE FROM FILE

_o_QT LOAD COMPRESS PICTURE FROM FILE (DokRef ; Methodenname ; Qualität ; Bild)

Parameter	Typ		Beschreibung
DokRef	DokRef	⇒	Referenznummer des Dokuments
Methodenname	String	⇒	Komprimierungsverfahren (4 Zeichen)
Qualität	Lange Ganzzahl	⇒	Qualität der Komprimierung (1..1000)
Bild	Bild	⇐	Zu komprimierendes Bild

Hinweis zur Kompatibilität

Dieser Befehl ruft überholte Mechanismen auf und muss durch die Befehle **READ PICTURE FILE** und **CONVERT PICTURE** ersetzt werden.

_o_SAVE PICTURE TO FILE

_o_SAVE PICTURE TO FILE (DokRef ; Bild)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenznummer der Datei
Bild	Bild	➡	Zu sicherndes Bild

Hinweis zur Kompatibilität

Dieser Befehl ruft überholte Mechanismen auf und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen, stattdessen den Befehl **WRITE PICTURE FILE** zu verwenden.

BLOB

- BLOB Befehle
- 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

BLOB Befehle

Definition

4D unterstützt den Datentyp BLOB (Binary Large OBjects).
Sie können BLOB Datenfelder und BLOB Variablen definieren:

- Für ein BLOB Datenfeld wählen Sie im Dialogfenster **Feldeigenschaften** in der Dropdown-Liste den Typ BLOB aus.
- Für eine BLOB Variable wählen Sie den Compiler-Befehl **C_BLOB**. Sie können lokale, Prozess-und Interprozessvariablen vom Typ BLOB erstellen.
- Für ein BLOB Array wählen Sie den Befehl **ARRAY BLOB**.

Ein BLOB sind in 4D miteinander zusammenhängende Bytes unterschiedlicher Länge, die Sie sowohl als eine Einheit als auch als einzelne Bytes ansprechen können. Ein BLOB kann leer sein (Länge Null) oder bis zu 2147483647 Bytes enthalten (2 GB).

BLOBs und Speicher

Ein BLOB wird als Ganzes in den Speicher geladen. Eine BLOB Variable oder ein BLOB Array existieren nur im Speicher. Ein BLOB Datenfeld wird von der Festplatte in den Speicher geladen sowie der Rest des Datensatzes, zu dem es gehört.
Analog zu anderen Feldtypen, die eine umfangreiche Datenanzahl enthalten können, wie der Typ Bild, werden BLOB-Felder beim Ändern des Datensatzes nicht im Speicher dupliziert. Folglich ist das Ergebnis, das von den Funktionen **Old** und **Modified** bei Anwendung auf ein BLOB-Feld zurückgegeben wird, nicht signifikant.

BLOBs anzeigen

Ein BLOB kann Daten jeglicher Art enthalten. Von daher gibt es keine Standarddarstellung auf dem Bildschirm. Zeigen Sie ein BLOB Feld bzw. eine BLOB Variable in einem Formular an, erscheinen sie unabhängig vom Inhalt leer.

BLOB Felder

In BLOB Feldern können Sie Daten jeglicher Art bis zu 2 GB speichern. Ein BLOB Feld können Sie nicht indizieren. Wollen Sie Datensätze zu Werten suchen, die in einem BLOB gespeichert sind, müssen Sie eine Formel einsetzen.

Parameterübergabe, Zeiger und Funktionsergebnisse

Sie können BLOBs von 4D als Parameter für 4D Befehle oder Plug-In Routinen übergeben, die derartige Parameter erwarten. Sie können BLOBS auch als Parameter für eine Benutzermethode übergeben oder als Funktionsergebnis zurückgeben.
Wollen Sie ein BLOB für eigene Methoden übergeben, definieren Sie einen Zeiger auf das BLOB und übergeben den Zeiger als Parameter.
Beispiele:

```
  ` Deklariere eine Variable vom Typ BLOB
C_BLOB(anyBlobVar)
  ` Das BLOB wird als Parameter für einen 4D Befehl übergeben
SET BLOB SIZE(anyBlobVar;1024*1024)
  ` Das BLOB wird als Parameter für eine externe Routine übergeben
$errCode:=Do Something With This BLOB(anyBlobVar)
  ` Das BLOB wird als Parameter auf eine Methode übergeben, die ein BLOB zurückgibt
C_BLOB(retrieveBlob)
retrieveBlob:=Fill_Blob(anyBlobVar)
  ` Ein Zeiger auf das BLOB wird als Parameter auf eine Benutzermethode übergeben
COMPUTE BLOB(->anyBlobVar)
```

Hinweis für Plug-In Entwickler: Ein BLOB Parameter wird als "&O" deklariert (der Buchstabe "O", nicht die Ziffer "0").

Zuweisung

Sie können BLOBs gegenseitig zuweisen.
Beispiel:

```
  ` Deklariere zwei Variablen vom Typ BLOB
C_BLOB(vBlobA;vBlobB)
  ` Setze die Größe des ersten BLOB auf 10 K
SET BLOB SIZE(vBlobA;10*1024)
  ` Weise das erste BLOB dem zweiten zu
vBlobB:=vBlobA
```

Sie können jedoch keinen Operator auf BLOBs anwenden; es gibt keinen Ausdruck vom Typ BLOB.

BLOB Inhalt ansprechen

Sie können jedes Byte eines BLOB über geschweifte Klammern {...} individuell ansprechen. Bytes in einem BLOB werden von 0 zu N-1 nummeriert, wobei N die Größe des BLOB ist. Beispiel:

```
  ` Deklariere eine Variable vom Typ BLOB
C_BLOB(vBlob)
  ` Setze Größe des BLOB auf 256 Bytes
SET BLOB SIZE(vBlob;256)
  ` Die u.a. Schleife initialisiert die 256 Bytes des BLOB auf Null
For(vByte;0;BLOB size(vBlob)-1)
   vBlob{vByte}:=0
End for
```

Da alle Bytes eines BLOB individuell ansprechbar sind, können Sie in einem Feld bzw. einer Variablen vom Typ BLOB speichern, was Sie möchten.

BLOB Befehle von 4D

4D bietet folgende Befehle zum Arbeiten mit BLOBS:

- **SET BLOB SIZE** verändert die Größe eines BLOB Feldes bzw. einer BLOB Variablen.
- **BLOB size** gibt die Größe eines BLOB zurück.
- Mit **DOCUMENT TO BLOB** und **BLOB TO DOCUMENT** laden bzw. schreiben Sie ein ganzes Dokument in bzw. aus einem BLOB (optional, Data- und Ressource-Forks auf Macintosh).
- Mit **VARIABLE TO BLOB** und **BLOB TO VARIABLE** sowie **LIST TO BLOB** und **BLOB to list** speichern und finden Sie 4D Variablen in BLOBs wieder.
- Mit **COMPRESS BLOB**, **EXPAND BLOB** und **BLOB PROPERTIES** arbeiten Sie mit komprimierten BLOBs
- Mit den Funktionen/Befehlen **BLOB to integer**, **BLOB to longint**, **BLOB to real**, **BLOB to text**, **INTEGER TO BLOB**, **LONGINT TO BLOB**, **REAL TO BLOB** und **TEXT TO BLOB** bearbeiten Sie beliebige strukturierte Daten von der Festplatte, dem Betriebssystem,, von Ressourcen, etc..
- Mit **DELETE FROM BLOB**, **INSERT IN BLOB** und **COPY BLOB** verwalten Sie große Datenpakete innerhalb von BLOBs.
- Mit **ENCRYPT BLOB** und **DECRYPT BLOB** können Sie Daten in einer 4D Anwendung verschlüsseln und entschlüsseln.

Diese Befehle werden auf den folgenden Seiten beschrieben.

Darüberhinaus gibt es noch folgende Befehle:

- **C_BLOB** deklariert eine Variable vom Typ BLOB. Weitere Informationen dazu finden Sie im Kapitel **Compiler**.
- **ARRAY BLOB** erstellt oder passt ein BLOB vom Typ Array an. Weitere Informationen dazu finden Sie im Abschnitt **Arrays**.
- Mit **GET PASTEBOARD DATA** und **APPEND DATA TO PASTEBOARD** verwalten Sie beliebige Datentypen, die in der Zwischenablage gespeichert sind. Weitere Informationen dazu finden Sie im Abschnitt **Pasteboards verwalten**.
- Mit **GET RESOURCE** und **_o_SET RESOURCE** arbeiten Sie mit beliebigen Datentypen und Ressourcen, die auf der Festplatte gespeichert sind. Weitere Informationen dazu finden Sie im Abschnitt **Einführung in Ressourcen**.
- Mit **WEB SEND BLOB** können Sie jede Art von Daten zu einem Web Browser senden. Weitere Informationen dazu finden Sie im Kapitel **Web Server**.
- Mit **PICTURE TO BLOB**, **BLOB TO PICTURE** und **_o_PICTURE TO GIF** können Sie Bilder öffnen und konvertieren. Weitere Informationen dazu finden Sie im Kapitel **Bilder**.
- **GENERATE ENCRYPTION KEYPAIR** und **GENERATE CERTIFICATE REQUEST** sind Befehle zur Verschlüsselung, die vom SSL (Secured Socket Layer) Protokoll für sichere Verbindungen verwendet wird. Weitere Informationen dazu finden Sie im Abschnitt **TLS Protokoll verwenden (HTTPS)**.

BLOB PROPERTIES

BLOB PROPERTIES (BLOB ; Komprimiert {; GrößeUnkomprimiert {; AktuelleGröße}})

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	BLOB, über das Angaben benötigt werden
Komprimiert	Lange Ganzzahl	⬅	0 = BLOB unkomprimiert, 1 = BLOB kompakt komprimiert, 2 = BLOB schnell komprimiert
GrößeUnkomprimiert	Lange Ganzzahl	⬅	Größe des unkomprimierten BLOB (in Bytes)
AktuelleGröße	Lange Ganzzahl	⬅	Aktuelle Größe des BLOB (in Bytes)

Beschreibung

Der Befehl **BLOB PROPERTIES** gibt Informationen über *BLOB* zurück.

Der Parameter *Komprimiert* zeigt an, ob und wie ein BLOB komprimiert bzw. unkomprimiert ist und gibt einen der folgenden Werte zurück. 4D bietet folgende vordefinierten Konstanten:

Konstante	Typ	Wert	Kommentar
Compact compression mode	Lange Ganzzahl	1	So kompakt wie möglich komprimieren (Standardmodus). Verlangsamt die Komprimierungsgeschwindigkeit und die Operationen zur Entkomprimierung.
Fast compression mode	Lange Ganzzahl	2	So schnell wie möglich komprimieren (wird auch schnellstmöglich entkomprimiert), verringert die Komprimierungsrate, d.h. das komprimierte BLOB wird größer.
GZIP best compression mode	Lange Ganzzahl	-1	Kompakteste GZIP Komprimierung
GZIP fast compression mode	Lange Ganzzahl	-2	Schnellste GZIP Komprimierung
Is not compressed	Lange Ganzzahl	0	

Der Parameter *GrößeUnkomprimiert* gibt, unabhängig vom Status des BLOB die Größe des unkomprimierten BLOB an.

Der Parameter *AktuelleGröße* gibt die aktuelle Größe des BLOB zurück. Bei komprimiertem BLOB ist das der Wert *AktuelleGröße* minus *GrößeUnkomprimiert*. Bei unkomprimiertem BLOB erhalten Sie immer den Wert *AktuelleGröße* ist gleich *GrößeUnkomprimiert*.

Beispiel 1

Siehe Beispiele zu den Befehlen **COMPRESS BLOB** und **EXPAND BLOB**.

Beispiel 2

Wurde ein BLOB komprimiert, erhält folgende Projektmethode den durch die Komprimierung gewonnenen Platz in Prozent:

```
` Projektmethode Space saved by compression
` Durch Komprimierung gesparter Platz (Zeiger {; Zeiger } ) -> Lange Ganzzahl
` Durch Komprimierung gesparter Platz ( -> BLOB {; -> gewonnene Bytes } ) -> Prozent

C_POINTER($1;$2)
C_LONGINT($0;$vlCompressed;$vlExpandedSize;$vlCurrentSize)

BLOB PROPERTIES($1->;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
If($vlExpandedSize=0)
   $0:=0
   If(Count parameters>=2)
      $2->:=0
   End if
Else
   $0:=100-(($vlCurrentSize/$vlExpandedSize)*100)
   If(Count parameters>=2)
      $2->:=$vlExpandedSize-$vlCurrentSize
   End if
End if
```

Diese Methode in Ihrer Anwendung können Sie folgendermaßen einsetzen:

```
` ...
COMPRESS BLOB(vxBlob)
$vlPercent:=Space saved by compression(->vxBlob;->vlBlobSize)
ALERT("Die Komprimierung spart "+String(vlBlobSize)+" Bytes, das sind "+String($vlPercent;"#0%")")
```

BLOB size

BLOB size (BLOB) -> Funktionsergebnis

Parameter	Typ		Beschreibung
BLOB	BLOB	→	BLOB Feld oder Variable
Funktionsergebnis	Lange Ganzzahl	←	Größe des BLOB in Bytes

Beschreibung

Die Funktion **BLOB size** gibt die Größe von *BLOB* in Bytes zurück.

Beispiel

Die Codezeile fügt dem BLOB *MeinBlob* 100 Bytes hinzu:

```
SET BLOB SIZE(MeinBlob;BLOB size(MeinBlob)+100)
```

BLOB TO DOCUMENT

BLOB TO DOCUMENT (Dokument ; BLOB {; *})

Parameter	Typ		Beschreibung
Dokument	String	➡	Name des Dokuments
BLOB	BLOB	➡	Neuer Inhalt für das Dokument
*	Operator	➡	*** überholt, nicht verwenden ***

Beschreibung

Der Befehl **BLOB TO DOCUMENT** schreibt den gesamten Inhalt von *Dokument* mit den in *BLOB* gespeicherten Daten neu. Sie können in *Dokument* den Namen eines vorhandenen bzw. noch nicht vorhandenen Dokuments übergeben. Gibt es das Dokument noch nicht, legt der Befehl es an. Übergeben Sie den Namen eines bestehenden Dokuments, achten Sie darauf, dass es noch nicht geöffnet ist, da sonst ein Fehler generiert wird. Wollen Sie den Benutzer das Dokument auswählen lassen, verwenden Sie die Funktionen **Open document** oder **Create document** und die Prozessvariable *Document* (siehe Beispiel).

Hinweis zur Kompatibilität: Der optionale Parameter * (Verwaltung von Resource Fork in älteren Versionen auf Mac OS) wird ab 4D v16 nicht mehr unterstützt. Weitere Informationen dazu finden Sie im Abschnitt **Mac Ressourcen**.

Beispiel

Sie schreiben ein Informationsystem, mit dem Sie Dokumente schnell speichern bzw. wieder finden können. Im Eingabeformular legen Sie eine Schaltfläche an, mit der Sie ein Dokument mit den in ein BLOB Feld geladenen Daten sichern können. Die dazugehörige Methode lautet:

```
$vhDocRef:=Create document("")  ` Sichere Dokument Ihrer Wahl
If(OK=1)  ` Wurde ein Dokument erstellt
   CLOSE DOCUMENT($vhDocRef)  ` Es muss nicht länger offen bleiben
   BLOB TO DOCUMENT(Document;[YourTable]YourBLOBField)
 ` Schreibe Dokumentinhalt
   If(OK=0)
 ` Verwalte Fehler
   End if
End if
```

Systemvariablen und Mengen

OK wird auf 1 gesetzt, wenn das Dokument korrekt geschrieben wurde, andernfalls auf 0. Es wird ein Fehler generiert.

Fehler verwalten

- Versuchen Sie ein BLOB in ein Dokument zu schreiben, das nicht vorhanden ist oder bereits von einem anderen Prozess bzw. einer anderen Anwendung geöffnet wurde, erhalten Sie einen entsprechenden OS Systemfehler.
- Beim Schreiben können Eingabe-/Ausgabefehler auftreten.
- Unter Umständen reicht der Speicher nicht aus, um den neuen Inhalt des Dokuments zu schreiben.

Sie können die Fehler in allen Fällen mit der Unterbrechungsmethode **ON ERR CALL** ausfindig machen.

BLOB to integer

BLOB to integer (BLOB ; ByteAnordnung {; Offset}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	BLOB zum Erhalten des Wertes vom Typ Ganzzahl
ByteAnordnung	Lange Ganzzahl	➡	0=Native Byte Anordnung, 1=Macintosh Byte Anordnung, 2=PC Byte Anordnung
Offset	Variable	➡	Versatz im BLOB (in Bytes)
		⬅	Neuer Versatz nach Lesen
Funktionsergebnis	Ganzzahl	↩	2-byte Wert vom Typ Zahl

Beschreibung

Die Funktion **BLOB to integer** gibt einen aus *BLOB* gelesenen 2-byte Wert vom Typ Ganzzahl zurück.

Der Parameter *ByteAnordnung* legt die Byte-Anordnung des zu lesenden 2-Byte Wertes vom Typ Ganzzahl fest. Übergeben Sie eine der vordefinierten Konstanten von 4D:

Konstante	Typ	Wert
Macintosh byte ordering	Lange Ganzzahl	1
Native byte ordering	Lange Ganzzahl	0
PC byte ordering	Lange Ganzzahl	2

Hinweis zur Plattformunabhängigkeit: Bei diesem Befehl müssen Sie sich selbst um den Austausch von Bytes zwischen den Plattformen kümmern.

Geben Sie den optionalen Parameter *Offset* an, wird der 2-byte Wert vom Typ Zahl im BLOB am Versatz gelesen. Geben Sie den optionalen Parameter *Offset* nicht an, werden die beiden ersten Bytes des BLOB gelesen.

Hinweis: Übergeben Sie als Versatz einen Wert (in Bytes) zwischen *0* (Null) und der BLOB-Größe minus 2. Andernfalls wird ein Fehler -111 generiert.

Nach dem Aufruf wird die Variable um die Anzahl der gelesenen Bytes erhöht. Von daher können Sie dieselbe Variable mit einem anderen BLOB Befehl zum Schreiben eines anderen Wertes verwenden.

Beispiel

Folgendes Beispiel liest 20 Werte vom Typ Ganzzahl aus einem BLOB, beginnend mit dem Versatz 0x200:

```
$vlOffset:=0x200
For($viLoop;0;19)
   $viValue:=BLOB to integer(vxSomeBlob;PC byte ordering;$vlOffset)
  ` Führe etwas aus mit $viValue
End for
```

BLOB to list

BLOB to list (BLOB {; Offset}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	BLOB mit einer hierarchischen Liste
Offset	Lange Ganzzahl	➡	Versatz im BLOB (in Bytes)
		⬅	Neuer Versatz nach Lesen
Funktionsergebnis	ListRef	↩	Referenz auf die neu erstellte Liste

Beschreibung

Der Befehl **BLOB to list** erstellt eine neue hierarchische Liste mit den in *BLOB* gespeicherten Daten an der mit *Offset* festgelegten Position (beginnend mit Null) und gibt für diese neue Liste eine Listenreferenznummer zurück.

Die BLOB Daten müssen mit dem Befehl übereinstimmen. Im Normalfall verwenden Sie BLOBs, die Sie zuvor über den Befehl **LIST TO BLOB** gefüllt haben.

Geben Sie den optionalen Parameter *Offset* nicht an, wird die Liste ab Beginn des BLOB gelesen. Bei einem BLOB mit mehreren Variablen bzw. Listen müssen Sie den Parameter *Offset* und zusätzlich eine numerische Variable übergeben. Setzen Sie vor dem Aufruf diese numerische Variable auf den geeigneten Versatz. Nach dem Aufruf gibt dieselbe Variable den Versatz der nächsten im BLOB gespeicherten Variablen zurück.

Hinweis zur Plattformunabhängigkeit: BLOB to list und **LIST TO BLOB** verwalten die in BLOBs gespeicherten Listen in einem internen 4D Format. Beim Einsetzen dieser beiden Befehle müssen Sie sich deshalb nicht um den Austausch von Bytes zwischen den Plattformen kümmern. Mit anderen Worten, Sie können ein unter Windows erstelltes BLOB auf Macintosh wiederverwenden und umgekehrt.

Beispiel

In diesem Beispiel entnimmt die Formularmethode für ein Dateneingabeformular eine Liste aus einem BLOB Feld, bevor das Formular auf dem Bildschirm erscheint und speichert es wieder in dem BLOB Feld, sobald die Dateneingabe gespeichert wird:

```
` [Dinge zu erledigen];"Eingabe" Formularmethode

Case of

   :(Form event=On Load)
      hList:=BLOB to list([Dinge zu erledigen]Weitere Ideen)
      If(OK=0)
         hList:=New list
      End if

   :(Form event=On Unload)
      CLEAR LIST(hList;*)

   :(bValidate=1)
      LIST TO BLOB(hList;[Dinge zu erledigen]Weitere Ideen)

End case
```

Systemvariablen und Mengen

Wurde die Liste erfolgreich gespeichert, wird die OK Variable auf 1 gesetzt, andernfalls auf 0 (zum Beispiel, wenn der Speicher nicht ausreicht.).

BLOB to longint

BLOB to longint (BLOB ; ByteAnordnung {; Offset}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	BLOB zum Entnehmen des Wertes Lange Ganzzahl
ByteAnordnung	Lange Ganzzahl	➡	0=Native Byte Anordnung, 1=Macintosh Byte Anordnung, 2=PC Byte Anordnung
Offset	Variable	➡	Versatz im BLOB (in Bytes)
		⬅	Neuer Versatz nach dem Lesen
Funktionsergebnis	Lange Ganzzahl	↩	4-byte Wert vom Typ Lange Ganzzahl

Beschreibung

Die Funktion **BLOB to longint** gibt einen aus *BLOB* gelesenen 4-byte Wert vom Typ Lange Ganzzahl zurück.

Der Parameter *ByteAnordnung* legt die Byte-Anordnung des zu lesenden 4-Byte Wertes vom Typ Lange Ganzzahl fest. Übergeben Sie eine der folgenden vordefinierten 4D Konstanten:

Konstante	Typ	Wert
Macintosh byte ordering	Lange Ganzzahl	1
Native byte ordering	Lange Ganzzahl	0
PC byte ordering	Lange Ganzzahl	2

Hinweis zur Plattformunabhängigkeit: Bei diesem Befehl müssen Sie sich selbst um den Austausch von Bytes zwischen den Plattformen kümmern.

Geben Sie den optionalen Variablenparameter *Offset* an, wird der 4-byte Wert vom Typ Lange Ganzzahl im BLOB am Versatz gelesen (beginnend bei Null). Geben Sie den optionalen Variablenparameter *Offset* nicht an, werden die ersten vier Bytes des BLOB gelesen.

Hinweis: Übergeben Sie als Versatz einen Wert (in Bytes) zwischen *0* (Null) und der BLOB-Größe minus 4. Andernfalls wird ein Fehler -111 generiert.

Nach dem Aufruf wird die Variable um die Anzahl der gelesenen Bytes erhöht. Von daher können Sie dieselbe Variable mit einem anderen BLOB Befehl zum Schreiben eines anderen Wertes verwenden.

Beispiel

Folgendes Beispiel liest 20 Werte vom Typ Lange Ganzzahl aus einem BLOB, beginnend mit dem Versatz 0x200:

```
$vlOffset:=0x200
For($viLoop;0;19)
   $vlValue:=BLOB to longint(vxSomeBlob;PC byte ordering;$vlOffset)
  ` Führe etwas aus mit $vlValue
End for
```

BLOB to real

BLOB to real (BLOB ; ZahlenFormat {; Offset}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	BLOB zum Entnehmen des Wertes Zahl
ZahlenFormat	Lange Ganzzahl	➡	0=Native Zahlenformat, 1=Erweitertes Zahlenformat, 2=Macintosh Doppeltes Zahlenformat, 3=Windows Doppeltes Zahlenformat
Offset	Variable	➡	Versatz im BLOB (in Bytes)
		⬅	Neuer Versatz nach dem Lesen
Funktionsergebnis	Zahl	➲	Zahlenwert

Beschreibung

Die Funktion **BLOB to real** gibt einen aus *BLOB* gelesenen Wert vom Typ Zahl zurück.

Der Parameter *ZahlenFormat* legt die Byte-Anordnung des zu lesenden Wertes vom Typ Zahl fest. Übergeben Sie eine der folgenden vordefinierten 4D Konstanten:

Konstante	Typ	Wert
Extended real format	Lange Ganzzahl	1
Macintosh double real format	Lange Ganzzahl	2
Native real format	Lange Ganzzahl	0
PC double real format	Lange Ganzzahl	3

Hinweis zur Plattformunabhängigkeit: Bei diesem Befehl müssen Sie sich selbst um den Austausch von Bytes zwischen den Plattformen kümmern.

Geben Sie den optionalen Variablenparameter *Offset* an, wird der Wert vom Typ Zahl im BLOB am Versatz gelesen (bei Null beginnend). Geben Sie den optionalen Variablenparameter *Offset* nicht an, werden die ersten 8 bzw. 10 Bytes des BLOB gelesen.

Hinweis: Übergeben Sie als Versatz einen Wert (in Bytes) zwischen *0* (Null) und der BLOB-Größe minus 8 bzw. 10. Andernfalls wird ein Fehler -111 generiert.

Nach dem Aufruf wird die Variable um die Anzahl der gelesenen Bytes erhöht. Von daher können Sie dieselbe Variable mit einem anderen BLOB Befehl zum Schreiben eines anderen Wertes verwenden.

Beispiel

Folgendes Beispiel liest 20 Werte vom Typ Zahl aus einem BLOB, beginnend mit dem Versatz 0x200:

```
$vlOffset:=0x200
For($viLoop;0;19)
   $vrValue:=BLOB to real(vxSomeBlob;PC byte ordering;$vlOffset)
  ` Führe etwas aus mit $vrValue
End for
```

BLOB to text

BLOB to text (BLOB ; TextFormat {; Offset {; TextLänge}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	BLOB zum Entnehmen des Textes
TextFormat	Lange Ganzzahl	➡	Format und Zeichensatz des Textes
Offset	Variable	➡	Versatz im BLOB (in Bytes)
		⬅	Neuer Versatz nach dem Lesen
TextLänge	Lange Ganzzahl	➡	Anzahl der zu lesenden Zeichen
Funktionsergebnis	Text	↩	Textwert

Beschreibung

Die Funktion **BLOB to text** gibt einen aus *BLOB* gelesenen Textwert zurück.

Der Parameter *TextFormat* legt das interne Format und den Zeichensatz des zu lesenden Textwertes fest. In Datenbanken, die mit 4D v11 erstellt wurden, verwendet 4D standardmäßig den Unicode Zeichensatz zur Textverwaltung. Zur Wahrung der Kompatibilität kann dieser Befehl die Umwandlung in den Mac Roman Zeichensatz erzwingen, der in früheren Versionen verwendet wurde. Der Zeichensatz wird im Parameter *TextFormat* definiert. Übergeben Sie eine der folgenden vordefinierten Konstanten unter dem Thema **BLOB**:

Konstante	Typ	Wert
Mac C string	Lange Ganzzahl	0
Mac Pascal string	Lange Ganzzahl	1
Mac text with length	Lange Ganzzahl	2
Mac text without length	Lange Ganzzahl	3
UTF8 C string	Lange Ganzzahl	4
UTF8 text with length	Lange Ganzzahl	5
UTF8 text without length	Lange Ganzzahl	6

Hinweise:

- Die “UTF8” Konstanten sind nur verwendbar, wenn die Anwendung im Unicode Modus arbeitet.
- Die “Mac “ Konstanten können nicht mit Texten größer als 32 KB arbeiten.
- Wollen Sie mit anderen Zeichensätzen als UTF8 arbeiten, verwenden Sie die Funktion **Convert to text**.

Weitere Informationen zu diesen Formaten finden Sie in der Beschreibung zum Befehl **TEXT TO BLOB**.

Warnung: Die Anzahl der zu lesenden Zeichen wird durch den Parameter *TextFormat* festgelegt, mit Ausnahme der Formate Mac Text without length und UTF8 Text without length. Hier müssen Sie im Parameter *TextLänge* die Anzahl der Zeichen festlegen, da sonst der komplette Inhalt gelesen wird. Dieser Parameter wird für die anderen Formate ignoriert und Sie können ihn weglassen.

Geben Sie den optionalen Parameter *Offset* an, wird der Textwert im BLOB am Versatz gelesen (beginnend bei Null). Geben Sie den optionalen Parameter *Offset* nicht an, wird der Anfang des Blob gemäß dem in *TextFormat* übergebenen Wert gelesen. Beachten Sie, dass Sie beim Lesen von Text ohne Länge den Parameter *Offset* übergeben müssen.

Hinweis: Übergeben Sie als Versatz einen Wert (in Bytes) zwischen *0* (Null) und der BLOB-Größe minus der zu lesenden Textgröße. Andernfalls ist das Funktionsergebnis nicht vorhersehbar.

Nach dem Aufruf wird die Variable um die Anzahl der gelesenen Bytes erhöht. Von daher können Sie dieselbe Variable mit einem anderen BLOB Befehl zum Schreiben eines anderen Wertes verwenden.

BLOB TO VARIABLE

BLOB TO VARIABLE (BLOB ; Variable {; Offset})

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	BLOB mit 4D Variablen
Variable	Variable	⬅	Mit BLOB Inhalt zu schreibende Variable
Offset	Lange Ganzzahl	➡	Position der Variablen im BLOB
		⬅	Position der folgenden Variable im BLOB

Beschreibung

Der Befehl **BLOB TO VARIABLE** schreibt die Variable *Variable* mit den in *BLOB* gespeicherten Daten an der durch *Offset* festgelegten Position (bei Null beginnend). Die BLOB Daten müssen mit der Zielvariablen übereinstimmen. Im Normalfall verwenden Sie BLOBs, die Sie zuvor über den Befehl **VARIABLE TO BLOB** gefüllt haben. Geben Sie den optionalen Parameter *Offset* nicht an, wird der Variableninhalt ab Beginn des BLOB gelesen. Bei einem BLOB mit mehreren Variablen müssen Sie den Parameter *Offset* übergeben und zwar als numerische Variable. Setzen Sie vor dem Aufruf diese numerische Variable auf den geeigneten Versatz. Nach dem Aufruf gibt dieselbe Variable den Versatz der nächsten im BLOB gespeicherten Variablen zurück.

Hinweis: BLOB TO VARIABLE unterstützt Objekt- und Collection-Variablen der Typen **C_OBJECT** und **C_COLLECTION**. Weitere Informationen dazu finden Sie unter dem Befehl **VARIABLE TO BLOB**.

Wurde die Variable nach dem Aufruf erfolgreich neu geschrieben, hat die Variable OK den Wert 1. Konnte die Operation nicht ausgeführt werden, hat sie den Wert 0; zum Beispiel, wenn der Speicher nicht ausreicht.

Hinweis zur Plattformunabhängigkeit: BLOB TO VARIABLE und **VARIABLE TO BLOB** verwalten die in BLOBs gespeicherten Variablen in einem internen 4D Format. Beim Einsetzen dieser beiden Befehle müssen Sie sich deshalb nicht um den Austausch von Bytes zwischen den Plattformen kümmern. Mit anderen Worten, Sie können ein unter Windows erstelltes BLOB auf macOS wiederverwenden und umgekehrt.

Beispiel

Siehe Beispiele zum Befehl **VARIABLE TO BLOB**.

Systemvariablen und Mengen

Wurde die Variable erfolgreich gespeichert, wird die OK Variable auf 1 gesetzt, andernfalls auf 0.

COMPRESS BLOB

COMPRESS BLOB (BLOB {; Komprimierung})

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	Zu komprimierendes BLOB
Komprimierung	Lange Ganzzahl	➡	Wenn angegeben: 1=komprimiert so kompakt wie möglich, 2=komprimiert so schnell wie möglich

Beschreibung

Der Befehl **COMPRESS BLOB** komprimiert *BLOB* mit dem internen Komprimierungsalgorithmus von 4D. Dieser Befehl komprimiert nur BLOBs, die größer als 255 Bytes sind.

Mit dem optionalen Parameter *Komprimierung* legen Sie fest, wie das BLOB komprimiert wird. 4D bietet dafür folgende vordefinierten Konstanten:

Konstante	Typ	Wert	Kommentar
Compact compression mode	Lange Ganzzahl	1	So kompakt wie möglich komprimieren (Standardmodus). Verlangsamt die Komprimierungsgeschwindigkeit und die Operationen zur Entkomprimierung.
Fast compression mode	Lange Ganzzahl	2	So schnell wie möglich komprimieren (wird auch schnellstmöglich entkomprimiert), verringert die Komprimierungsrate, d.h. das komprimierte BLOB wird größer.
GZIP best compression mode	Lange Ganzzahl	-1	Kompakteste GZIP Komprimierung
GZIP fast compression mode	Lange Ganzzahl	-2	Schnellste GZIP Komprimierung

Übergeben Sie einen anderen Wert oder lassen Sie den Parameter *Komprimierung* weg, wird der Komprimierungsmodus 1 verwendet (kompakteste interne Komprimierung).

Nach dem Aufruf hat die OK Variable den Wert 1, wenn das BLOB erfolgreich komprimiert wurde. Andernfalls hat die OK Variable den Wert 0. Dann gibt es zwei Möglichkeiten:

1. Der Fehler ist unwichtig (Speicherproblem, Blobgröße). Es wird kein Fehler generiert. Die aufrufende Methode wird weiter ausgeführt.
2. Der Fehler ist wichtig, d.h. BLOB ist beschädigt. Der Fehler -10600 wird generiert. Sie können ihn mit dem Befehl **ON ERR CALL** verwalten.

Mit dem Befehl **EXPAND BLOB** können Sie ein komprimiertes BLOB entkomprimieren.

Mit dem Befehl **BLOB PROPERTIES** können Sie feststellen, ob ein BLOB komprimiert ist.

Warnung: Ein komprimiertes BLOB ist weiterhin ein BLOB, Sie können also weiterhin dessen Inhalt verändern. Beachten Sie jedoch, dass der Befehl **EXPAND BLOB** das BLOB dann nicht korrekt entkomprimieren kann.

Beispiel 1

Dieses Beispiel prüft, ob das BLOB *vxMyBlob* komprimiert ist, und komprimiert es gegebenenfalls:

```
BLOB PROPERTIES(vxMyBlob;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
If($vlCompressed=ls not compressed)
   COMPRESS BLOB(vxMyBlob)
End if
```

Wenden Sie den Befehl **COMPRESS BLOB** auf ein bereits komprimiertes BLOB an, erkennt er das und führt nichts aus.

Beispiel 2

Dieses Beispiel wählt ein Dokument aus und komprimiert es:

```
$vhDocRef :=Open document("")
If(OK=1)
   CLOSE DOCUMENT($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
```

Beispiel 3

HTTP Rohdaten senden, komprimiert mit 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 ;*)
```

Systemvariablen und Mengen

Wurde das BLOB erfolgreich komprimiert, hat die OK Variable den Wert 1; andernfalls den Wert 0.

COPY BLOB

COPY BLOB (srcBLOB ; dstBLOB ; srcOffset ; dstOffset ; len)

Parameter	Typ		Beschreibung
srcBLOB	BLOB	➡	Ausgangs-BLOB
dstBLOB	BLOB	➡	Ziel-BLOB
srcOffset	Lange Ganzzahl	➡	Ausgangsposition für die Kopie
dstOffset	Lange Ganzzahl	➡	Zielposition für die Kopie
len	Lange Ganzzahl	➡	Anzahl der zu kopierenden Bytes

Beschreibung

Der Befehl **COPY BLOB** kopiert die Anzahl der in *len* angegebenen Bytes aus dem BLOB *srcBLOB* in das BLOB *dstBLOB*.

Die Kopie beginnt an der in *srcOffset* festgelegten Position (in Bezug auf den Anfang des Ausgangs-BLOB) und wird an der in *dstOffset* festgelegten Position ausgeführt (in Bezug auf den Anfang des Ziel-BLOB).

Hinweis: Das Ziel-BLOB kann bei Bedarf vergrößert werden.

DECRYPT BLOB

DECRYPT BLOB (ZuEntschlüsseln ; SendePubKey {; EmpfPrivKey})

Parameter	Typ		Beschreibung
ZuEntschlüsseln	BLOB	⇒	Daten zum Entschlüsseln
		⇐	Entschlüsselte Daten
SendePubKey	BLOB	⇒	Public key des Senders
EmpfPrivKey	BLOB	⇒	Private key des Empfängers

Beschreibung

Der Befehl **DECRYPT BLOB** entschlüsselt den Inhalt des BLOB mit dem Parameter *ZuEntschlüsseln* über den öffentlichen Schlüssel *SendePubKey* des Senders und optional den privaten Schlüssel *EmpfPrivKey* des Empfängers.

Das BLOB mit dem öffentlichen Schlüssel des Senders wird im Parameter *SendePubKey* übergeben. Diesen Schlüssel hat der Sender mit dem Befehl **GENERATE ENCRYPTION KEYPAIR** erstellt. Er muss an den Empfänger gesendet werden.

Das BLOB mit dem privaten Schlüssel des Empfängers kann im optionalen Parameter *EmpfPrivKey* übergeben werden. In diesem Fall muss der Empfänger über den Befehl **GENERATE ENCRYPTION KEYPAIR** ein Schlüsselpaar erstellen und seinen öffentlichen Schlüssel an den Sender schicken. Das auf Schlüsselpaaren basierende Verschlüsselungssystem garantiert, dass die Meldung nur vom Sender verschlüsselt wurde und nur vom Empfänger entschlüsselt werden kann. Weitere Informationen dazu finden Sie unter dem Befehl **ENCRYPT BLOB**.

Der Befehl **DECRYPT BLOB** verwendet eine Prüfsumme, um jegliche Änderung (bewusst oder unbewusst) des BLOB Inhalts zu verhindern. Wird ein verschlüsseltes BLOB beschädigt oder geändert, führt der Befehl nichts aus. Es wird eine Fehlermeldung zurückgegeben.

Beispiel

Beispiele dazu finden Sie unter dem Befehl **ENCRYPT BLOB**.

DELETE FROM BLOB

DELETE FROM BLOB (BLOB ; Offset ; len)

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	BLOB, aus dem Bytes gelöscht werden sollen
Offset	Lange Ganzzahl	➡	Position, ab der Bytes gelöscht werden sollen
len	Lange Ganzzahl	➡	Anzahl der zu löschenden Bytes

Beschreibung

Der Befehl **DELETE FROM BLOB** löscht die Anzahl der in *len* angegebenen Bytes aus *BLOB* an der in *Offset* angegebenen Position (in Bezug auf den Anfang des BLOB). Das BLOB wird dann um *len* Bytes kleiner.

DOCUMENT TO BLOB

DOCUMENT TO BLOB (Dokument ; BLOB {; *})

Parameter	Typ		Beschreibung
Dokument	String	➡	Name des Dokuments
BLOB	BLOB	➡	BLOB Feld oder Variable für das Dokument
		⬅	Inhalt des Dokuments
*	Operator	➡	*** überholt, nicht verwenden ***

Beschreibung

Der Befehl **DOCUMENT TO BLOB** lädt den gesamten Inhalt von *Dokument* in *BLOB*. Sie müssen den Namen eines bestehenden Dokuments übergeben, das noch nicht geöffnet ist, da sonst ein Fehler generiert wird. Verwenden Sie die Funktion **Open document** und die Prozessvariable *Dokument*, kann der Benutzer das Dokument wählen, das in das BLOB geladen werden soll (siehe Beispiel).

Hinweis zur Kompatibilität: Der optionale Parameter * (Verwaltung von Resource Fork in älteren Versionen auf Mac OS) wird ab 4D v16 nicht mehr unterstützt. Weitere Informationen dazu finden Sie im Abschnitt **Mac Ressourcen**.

Beispiel

Sie schreiben ein Informationsystem, mit dem Sie Dokumente schnell speichern bzw. wieder finden können. Im Eingabeformular legen Sie eine Schaltfläche an, mit der Sie ein Dokument in ein BLOB Feld laden können. Die dazugehörige Methode lautet:

```
$vhDocRef:=Open document("") ` Wähle das gewünschte Dokument aus
If(OK=1) ` Wurde ein Dokument ausgewählt
   CLOSE DOCUMENT($vhDocRef) ` Das Dokument muss nicht länger offen bleiben
   DOCUMENT TO BLOB(Document;[YourTable]YourBLOBField)
 ` Lade das Dokument
   If(OK=0)
 ` Verwalte Fehler
   End if
End if
```

Systemvariablen und Mengen

OK wird auf 1 gesetzt, wenn das Dokument korrekt geladen wurde, andernfalls auf 0. Es wird ein Fehler generiert.

Fehler verwalten

- Versuchen Sie ein Dokument in ein BLOB zu laden, das nicht vorhanden ist oder bereits von einem anderen Prozess bzw. einer anderen Anwendung geöffnet wurde, erhalten Sie einen entsprechenden OS Systemfehler.
- Ein Eingabe-/Ausgabefehler kann auftreten, wenn das Dokument gesperrt ist, auf einem gesperrten Volume liegt oder beim Lesen des Dokuments ein Problem auftritt.
- Reicht der Speicher nicht aus, um das Dokument zu laden, wird ein Fehler -108 generiert.

Sie können den Fehler in allen Fällen mit der Unterbrechungsmethode **ON ERR CALL** ausfindig machen.

ENCRYPT BLOB

ENCRYPT BLOB (ZuVerschlüsseln ; SendePrivKey {; EmpfPubKey})

Parameter	Typ		Beschreibung
ZuVerschlüsseln	BLOB	⇒	Daten zum Verschlüsseln
		⇐	Verschlüsselte Daten
SendePrivKey	BLOB	⇒	Privater Schlüssel des Senders
EmpfPubKey	BLOB	⇒	Öffentlicher Schlüssel des Empfängers

Beschreibung

Der Befehl **ENCRYPT BLOB** verschlüsselt den Inhalt des BLOB mit dem Parameter *ZuVerschlüsseln* mit dem privaten Schlüssel *SendePrivKey* des Senders sowie optional auch den öffentlichen Schlüssel *EmpfPubKey* des Empfängers. Eine Verschlüsselung (öffentliche und private Schlüssel) erhalten Sie über den Befehl **GENERATE ENCRYPTION KEYPAIR**.

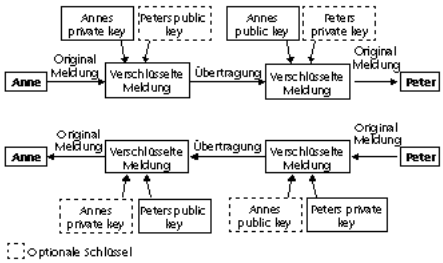
Hinweis: Dieser Befehl verwendet Algorithmus und Verschlüsselungsfunktionen des TLS Protokolls. Achten Sie deshalb darauf, dass alle dafür notwendigen Komponenten korrekt auf Ihrem Rechner installiert sind — selbst wenn Sie TLS nicht für 4D Web Server Verbindungen einsetzen. Ausführliche Informationen dazu finden Sie im Abschnitt **TLS Protokoll verwenden (HTTPS)**.

- Wird nur ein Schlüssel (der private Schlüssel des Senders) für die Verschlüsselung verwendet, können Personen, denen der öffentliche Schlüssel bekannt ist, die Information lesen. Dieses System garantiert nur, dass der Sender selbst die Information verschlüsselt hat.
- Der simultane Gebrauch des privaten Schlüssels vom Sender und des öffentlichen Schlüssels vom Empfänger garantiert, dass nur der Empfänger die Information lesen kann.

Das BLOB mit den Schlüsseln hat ein internes PEM (Privacy Enhanced Mail) Format. Über dieses plattformunabhängige Format können Sie mühelos Schlüssel in einem E-Mail oder in einer Textdatei per Kopieren/Einsetzen austauschen bzw. verwalten.

Sobald der Befehl läuft, enthält das BLOB mit dem Parameter *ZuVerschlüsseln* die verschlüsselten Daten. Sie lassen sich nur über den Befehl **DECRYPT BLOB** entschlüsseln, mit dem öffentlichen Schlüssel des Senders als Parameter.
Wurde darüberhinaus zum Verschlüsseln der Information der optionale öffentliche Schlüssel des Empfängers verwendet, ist zum Entschlüsseln auch der private Schlüssel des Empfängers notwendig.

Verschlüsselter Nachrichtenaustausch zwischen den Personen "Anne" und "Peter":



Hinweis: Der Befehl verwendet eine Prüfsumme, um jegliche Änderung (bewusst oder unbewusst) des BLOB Inhalts zu verhindern. Von daher sollten Sie ein verschlüsseltes BLOB nicht verändern, es könnte dann möglicherweise nicht entschlüsselbar sein.

Verschlüsselungsbefehle optimieren

Eine Verschlüsselung der Daten verlangsamt die Ausführung Ihrer Anwendungen, insbesondere wenn ein Schlüsselpaar verwendet wird. Es gibt jedoch bestimmte Maßnahmen zum Optimieren:

- Der Befehl wird, je nach dem aktuell verfügbaren Speicher, synchron oder asynchron ausgeführt.
  Der asynchrone Modus ist schneller, da er die anderen Prozesse nicht einfriert. Dieser Modus wird automatisch verwendet, wenn der verfügbare Speicherplatz mindestens doppelt so hoch ist wie die zu verschlüsselnden Daten.
  Sonst wird zur Sicherheit der synchrone Modus verwendet. Er ist langsamer, da der die anderen Prozesse einfriert.
- Bei umfangreichen BLOBs können Sie nur einen kleinen strategisch wichtigen Teil des BLOB verschlüsseln. Das verringert die Prozesszeit und die Größe der zu bearbeitenden Daten.

Beispiel

Einen Schlüssel verwenden
Eine Firma möchte die Daten in einer 4D Datenbank firmenintern halten. Sie muss diese Information über Dateien regelmäßig via Internet an die Niederlassungen senden. Die Niederlassungen wiederum müssen sicher sein, dass die Informationen wirklich von der Mutterfirma stammen.

1) Die Mutterfirma kann mit dem Befehl **GENERATE ENCRYPTION KEYPAIR** ein Schlüsselpaar generieren:

```
`Methode GENERATE_KEYS_TXT
C_BLOB($BPublicKey;$BPrivateKey)
GENERATE ENCRYPTION KEYPAIR($BPrivateKey;$BPublicKey)
BLOB TO DOCUMENT("PublicKey.txt";$BPublicKey)
BLOB TO DOCUMENT("PrivateKey.txt";$BPrivateKey)
```

2) Die Mutterfirma behält den privaten Schlüssel und sendet an jede Niederlassung eine Kopie des Dokuments mit dem öffentlichen Schlüssel. Zur maximalen Sicherheit sollte der Schlüssel auf Diskette kopiert werden, die den Niederlassungen

persönlich übergeben wird.
3) Dann kopiert die Firma die firmeninterne Information (die z.B. in einem Textfeld gespeichert ist) in BLOBs, die mit dem privaten Schlüssel verschlüsselt werden:

```
  `Methode ENCRYPT_INFO
C_BLOB($vbEncrypted;$vbPrivateKey)
C_TEXT($vtEncrypted)

$vtEncrypted:=[Private]Info
VARIABLE TO BLOB($vtEncrypted;$vbEncrypted)
DOCUMENT TO BLOB("PrivateKey.txt";$vbPrivateKey)
If(OK=1)
   ENCRYPT BLOB($vbEncrypted;$vbPrivateKey
   BLOB TO DOCUMENT("Update.txt";$vbEncrypted)
End if
```

4) Die Update-Dateien können dann über einen ungesicherten Kanal wie z.B. Internet an die Niederlassungen gesandt werden.
5) Jede Niederlassung kann das Dokument mit dem öffentlichen Schlüssel entschlüsseln:

```
  `Methode DECRYPT_INFO
C_BLOB($vbEncrypted;$vbPublicKey)
C_TEXT($vtDecrytped)
C_TIME($vtDocRef)

ALERT("Bitte das verschlüsselte Dokument wählen.")
$vtDocRef:=Open document("")  `Wähle Update.txt
If(OK=1)
   CLOSE DOCUMENT($vtDocRef)
   DOCUMENT TO BLOB(Document;$vbEncrypted)
   DOCUMENT TO BLOB("PublicKey.txt";$vbPublicKey)
   If(OK=1)
      DECRYPT BLOB($vbEncrypted;$vbPublicKey)
      BLOB TO VARIABLE($vbEncrypted;$vtDecrypted)
      CREATE RECORD([Private])
      [Private]Info:=$vtDecrypted
      SAVE RECORD([Private])
   End if
End if
```

Wenn die Daten von einer anderen Quelle stammen (die den privaten Schlüssel nicht kennen kann), gibt es eine Fehlermeldung, da der öffentliche Schlüssel nicht passt.

Ein Schlüsselpaar verwenden
Eine Firma möchte Informationen via Internet austauschen. Jede Niederlassung empfängt firmeninterne Information und sendet Information an die Mutterfirma. Es gibt also zwei Anforderungen:

- Der berechtigte Empfänger soll die Meldung nur lesen können,
- Der Empfänger muss einen Beweis erhalten, dass der Sender selbst die Meldung gesendet hat.

1) Die Mutterfirma und jede Niederlassung generieren über die Methode ***GENERATE_KEYS_TXT*** ihre eigenen Schlüsselpaare.

2) Der private Schlüssel wird von beiden Seiten geheimgehalten. Die Mutterfirma und jede Niederlassung senden sich gegenseitig jeweils die öffentlichen Schüssel. Sie müssen nicht über einen gesicherten Kanal gesendet werden, da der öffentliche Schlüssel nicht zum Entschlüsseln ausreicht.

3) Um eine zu sendende Information zu verschlüsseln, führt die Niederlassung bzw. Mutterfirma die Methode ***ENCRYPT_INFO_2*** aus. Diese verwendet zum Verschlüsseln den privaten Schlüssel des Senders und den öffentlichen Schlüssel des Empfängers:

```
  `Methode ENCRYPT_INFO_2
C_BLOB($vbEncrypted;$vbPrivateKey;$vbPublicKey)
C_TEXT($vtEncrypt)
C_TIME($vtDocRef)

$vtEncrypt:=[Private]Info
VARIABLE TO BLOB($vtEncrypt;$vbEncrypted)
  ` Ihr eigener privater Schlüssel wird geladen...
DOCUMENT TO BLOB("PrivateKey.txt";$vbPrivateKey)
If(OK=1)
  ` ...ebenso der öffentliche Schlüssel des Empfängers
   ALERT("Bitte den öffentlichen Schlüssel des Empfängers wählen.")
   $vhDocRef:=Open document("")  `Zu ladender öffentlicher Schlüssel
   If(OK=1)
      CLOSE DOCUMENT($vtDocRef)
      DOCUMENT TO BLOB(Document;$vbPublicKey)
```

```
    `BLOB Verschlüsselung mit dem beiden Schlüsseln als Parameter
        ENCRYPT BLOB($vbEncrypted;$vbPrivateKey;$vbPublicKey)
        BLOB TO DOCUMENT("Update.txt";$vbEncrypted)
    End if
End if
```

4) Die verschlüsselte Datei kann nun via Internet an den Empfänger gesendet werden. Eine dritte Person kann die verschlüsselte Meldung nicht entschlüsseln, selbst wenn sie die öffentlichen Schlüssel des Absenders kennt. Zum Entschlüsseln ist nämlich auch der private Schlüssel des Empfängers notwendig.

5) Jeder Empfänger kann das Dokument mit dem eigenen privaten und dem öffentlichen Schlüssel des Senders entschlüsseln:

```
    `Methode DECRYPT_INFO_2
C_BLOB($vbEncrypted;$vbPublicKey;$vbPrivateKey)
C_TEXT($vtDecrypted)
C_TIME($vhDocRef)

ALERT("Bitte das verschlüsselte Dokument wählen.")
$vhDocRef:=Open document("")  `Wähle die Datei Update.txt
If(OK=1)
    CLOSE DOCUMENT($vhDocRef)
    DOCUMENT TO BLOB(Document;$vbEncrypted)
  `Ihr eigener privater Schlüssel wird geladen
    DOCUMENT TO BLOB("PrivateKey.txt";$vbPrivateKey)
    If(OK=1)
  ` ...ebenso der öffentliche Schlüssel des Senders
        ALERT("Bitte öffentlichen Schlüssel des Senders wählen")
        $vhDocRef:=Open document("")  `Zu ladender öffentlicher Schlüssel
        If(OK=1)
            CLOSE DOCUMENT($vhDocRef)
            DOCUMENT TO BLOB(Document;$vbPublicKey)
  `Das BLOB mit den beiden Schlüsseln als Parameter entschlüsseln
            DECRYPT BLOB($vbEncrypted;$vbPublicKey;$vbPrivateKey)
            BLOB TO VARIABLE($vbEncrypted;$vtDecrypted)
            CREATE RECORD([Private])
            [Private]Info:=$vtDecrypted
            SAVE RECORD([Private])
        End if
    End if
End if
```

EXPAND BLOB

EXPAND BLOB (BLOB)

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	Zu erweiterndes BLOB

Beschreibung

Der Befehl **EXPAND BLOB** entkomprimiert *BLOB*, das zuvor mit dem Befehl **COMPRESS BLOB** komprimiert wurde.
Die Variable OK hat nach dem Aufruf den Wert 1, wenn das BLOB entkomprimiert wurde.
Konnte keine Entkomprimierung durchgeführt werden, hat die Variable OK den Wert 0. Dann gibt es zwei Möglichkeiten:

1. Der Fehler ist unwichtig (Speicherproblem, Blobgröße). Es wird kein Fehler generiert. Die aufrufende Methode wird weiter ausgeführt.
2. Der Fehler ist wichtig, d.h. BLOB ist beschädigt. Der Fehler -10600 wird generiert. Sie können ihn mit dem Befehl **ON ERR CALL** verwalten.

Mit dem Befehl **BLOB PROPERTIES** können Sie feststellen, ob ein BLOB komprimiert wurde.

Beispiel 1

Dieses Beispiel prüft, ob das BLOB *vxMyBlob* komprimiert ist und entkomprimiert es gegebenenfalls:

```
BLOB PROPERTIES(vxMyBlob;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
If($vlCompressed#Is not compressed)
   EXPAND BLOB(vxMyBlob)
End if
```

Beispiel 2

Dieses Beispiel wählt ein Dokument aus und entkomprimiert es, sofern es komprimiert ist:

```
$vhDocRef :=Open document("")
If(OK=1)
   CLOSE DOCUMENT($vhDocRef)
   DOCUMENT TO BLOB(Document;vxBlob)
   If(OK=1)
      BLOB PROPERTIES(vxBlob;$vlCompressed;$vlExpandedSize;$vlCurrentSize)
      If($vlCompressed#Is not compressed)
         EXPAND BLOB(vxBlob)
         If(OK=1)
            BLOB TO DOCUMENT(Document;vxBlob)
         End if
      End if
   End if
End if
```

Systemvariablen und Mengen

Die Variable OK hat den Wert 1, wenn das BLOB erfolgreich entkomprimiert wurde, andernfalls hat sie den Wert 0.

INSERT IN BLOB

INSERT IN BLOB (BLOB ; Offset ; len {; Füller})

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	BLOB zum Einfügen von Bytes
Offset	Lange Ganzzahl	➡	Startposition zum Einfügen von Bytes
len	Lange Ganzzahl	➡	Anzahl der einzufügenden Bytes
Füller	Lange Ganzzahl	➡	Standard-Byte Wert (0x00..0xFF) 0x00 ohne Angabe

Beschreibung

Der Befehl **INSERT IN BLOB** fügt in *BLOB* die in *len* angegebene Anzahl Bytes an der in *Offset* angegebenen Position ein. Das BLOB wird dann um *len* länger.

Geben Sie den optionalen Parameter *Füller* nicht an, werden die in das BLOB eingefügten Bytes auf *0x00* gesetzt. Andernfalls werden die Bytes auf den in *Füller* übergebenen Wert gesetzt (Modulo 256 — 0..255).

Vor Aufruf dieses Befehls übergeben Sie im Parameter *Offset* die Einfügestelle in Bezug auf den Anfang des BLOB.

INTEGER TO BLOB

INTEGER TO BLOB (Ganzzahl ; BLOB ; ByteAnordnung {; Offset | *})

Parameter	Typ		Beschreibung
Ganzzahl	Lange Ganzzahl	➡	Wert vom Typ Ganzzahl zum Schreiben ins BLOB
BLOB	BLOB	➡	BLOB zum Empfangen des Wertes vom Typ Ganzzahl
ByteAnordnung	Lange Ganzzahl	➡	0=Native Byte Anordnung, 1=Macintosh Byte Anordnung, 2=PC Byte Anordnung
Offset | *	Variable, Operator	➡	Versatz in Bytes im BLOB oder *, um Wert anzuhängen
		⬅	Neuer Versatz nach Schreiben ohne *

Beschreibung

Der Befehl **INTEGER TO BLOB** schreibt den 2-byte Wert *Ganzzahl* in *BLOB*.

Der Parameter *ByteAnordnung* legt die Anordnung des zu schreibenden 2-byte Wertes *Ganzzahl* fest. Sie übergeben eine der nachfolgenden vordefinierten Konstanten von 4D:

Konstante	Typ	Wert
Macintosh byte ordering	Lange Ganzzahl	1
Native byte ordering	Lange Ganzzahl	0
PC byte ordering	Lange Ganzzahl	2

Hinweis zur Plattformunabhängigkeit: Bei diesem Befehl müssen Sie sich selbst um den Austausch von Bytes zwischen den Plattformen kümmern.

Geben Sie den optionalen Parameter * an, wird der 2-byte Wert vom Typ Ganzzahl an das BLOB angefügt und die Größe des BLOB entsprechend erweitert. Mit dem optionalen Parameter * können Sie jede Zahl vom Typ *Ganzzahl*, *Lange Ganzzahl*, *Zahl* oder *Text* (siehe andere BLOB Befehle) sequentiell in einem BLOB speichern, solange der Speicher für das BLOB ausreicht.

Geben Sie den optionalen Parameter * oder den Variablenparameter *Offset* nicht an, wird der 2-byte Wert vom Typ Ganzzahl am Anfang des BLOB gespeichert, der bisherige Inhalt wird überschrieben. Die Größe des BLOB wird entsprechend angepasst.

Übergeben Sie den Variablenparameter *Offset*, wird der 2-byte Wert vom Typ Ganzzahl im BLOB am Byte-Versatz geschrieben (von Null ausgehend). Egal, wo Sie den Wert schreiben, die Größe des BLOB wird gemäß der übergebenen Position erhöht (bei Bedarf bis zu 2 Bytes). Neu zugewiesene Bytes, die Sie nicht schreiben, werden auf Null initialisiert.

Nach dem Aufruf wird der Variablenparameter *Offset* zurückgegeben, erhöht um die Anzahl der geschriebenen Bytes. Von daher können Sie dieselbe Variable mit einem anderen BLOB Befehl zum Schreiben eines anderen Wertes verwenden.

Beispiel 1

Nach Ausführen dieses Code:

```
INTEGER TO BLOB(0x0206;vxBlob;Native byte ordering)
```

- Ist die Größe von *vxBlob* 2 Bytes
- Gilt auf PowerPC Plattform: *vxBLOB{0}* = *$02* und *vxBLOB{1}* = *$06*
- Gilt auf Intel Plattform: *vxBLOB{0}* = *$06* und *vxBLOB{1}* = *$02*

Beispiel 2

Nach Ausführen dieses Code:

```
INTEGER TO BLOB(0x0206;vxBlob;Macintosh byte ordering)
```

- Ist die Größe von *vxBlob* 2 Bytes
- Gilt auf allen Plattformen *vxBLOB{0}* = *$02* und *vxBLOB{1}* = *$06*

Beispiel 3

Nach Ausführen dieses Code:

```
INTEGER TO BLOB(0x0206;vxBlob;PC byte ordering)
```

- Ist die Größe von *vxBlob* ist 2 Bytes
- Gilt auf allen Plattformen *vxBLOB{0}* = *$06* und *vxBLOB{1}* = *$02*

Beispiel 4

Nach Ausführen dieses Code:

```
SET BLOB SIZE(vxBlob;100)
INTEGER TO BLOB(0x0206;vxBlob;PC byte ordering;*)
```

- Ist die Größe von *vxBlob* 102 Bytes
- Gilt auf allen Plattformen *vxBLOB{100}* = *$06* und *vxBLOB{101}* = *$02*
- Bleiben die anderen Bytes des BLOB unverändert

Beispiel 5

Nach Ausführen dieses Code:

```
SET BLOB SIZE(vxBlob;100)
vlOffset:=50
INTEGER TO BLOB(518;vxBlob;Macintosh byte ordering;vlOffset)
```

- Ist die Größe von *vxBlob* 100 Bytes
- Gilt auf allen Plattformen *vxBLOB{50}* = *$02* und *vxBLOB{51}* = *$06*
- Bleiben die anderen Bytes des BLOB unverändert
- Wurde die Variable *vlOffset* um 2 erhöht (und ist nun gleich 52)

LIST TO BLOB

LIST TO BLOB (Liste ; BLOB {; *})

Parameter	Typ		Beschreibung
Liste	ListRef	➡	Hierarchische Liste zum Speichern im BLOB
BLOB	BLOB	➡	BLOB zum Empfangen der hierarchischen Liste
*	Operator	➡	* zum Wert anfügen

Beschreibung

Der Befehl **LIST TO BLOB** speichert die hierarchische Liste *Liste* in *BLOB*.

Geben Sie den optionalen Parameter * an, wird die hierarchische Liste an das BLOB angefügt und die Größe des BLOB entsprechend erweitert. Mit dem optionalen Parameter * können Sie beliebig viele Variablen oder Listen sequentiell in einem BLOB speichern (siehe andere BLOB Befehle), solange der Speicher dafür ausreicht.
Geben Sie den optionalen Parameter * nicht an, wird die hierarchische Liste am Anfang des BLOB gespeichert, der bisherige Inhalt wird überschrieben; die Größe des BLOB wird entsprechend angepasst.
Wird eine hierarchische Liste gespeichert, wird die Größe des BLOB je nach dem angegebenen Ort erhöht (bei Bedarf bis zur Größe der Liste). Geänderte Bytes (nicht die, welche Sie soeben gesetzt haben) werden auf 0 (Null) gesetzt.

Warnung: Verwenden Sie zum Speichern von Listen ein BLOB, müssen Sie dann den Inhalt des BLOB mit dem Befehl **BLOB to list** auslesen, da Listen in BLOBs mit einem internen 4D Format gespeichert werden.
Wurde die Liste erfolgreich gespeichert, wird die OK Variable auf 1 gesetzt, andernfalls auf 0; zum Beispiel, wenn der Speicher nicht ausreicht.

Hinweis zur Plattformunabhängigkeit: LIST TO BLOB und **BLOB to list** verwalten die in BLOBs gespeicherten Listen in einem internen 4D Format. Beim Einsetzen dieser beiden Befehle müssen Sie sich deshalb nicht um den Austausch von Bytes zwischen den Plattformen kümmern. Mit anderen Worten, Sie können ein unter Windows erstelltes BLOB auf Macintosh wiederverwenden und umgekehrt.

Beispiel

Siehe Beispiel zum Befehl **BLOB to list**.

LONGINT TO BLOB

LONGINT TO BLOB (LangeGanzzahl ; BLOB ; ByteAnordnung {; Offset I *})

Parameter	Typ		Beschreibung
LangeGanzzahl	Lange Ganzzahl	➡	Lange Ganzzahl zum Schreiben ins BLOB
BLOB	BLOB	➡	BLOB zum Empfangen der Langen GanzzahI
ByteAnordnung	Lange Ganzzahl	➡	0=Native Byte Anordnung, 1=Macintosh Byte Anordnung, 2=PC Byte Anordnung
Offset I *	Variable, Operator	➡	Versatz im BLOB (in Bytes) oder *, um Wert anzufügen
		⬅	Neuer Versatz nach Schreiben ohne *

Beschreibung

Der Befehl **LONGINT TO BLOB** schreibt den 4-byte Wert *LangeGanzzahl* in *BLOB*.

Der Parameter *ByteAnordnung* legt die Anordnung des zu schreibenden 4-byte Wertes *LangeGanzzahl* fest. Sie übergeben eine der nachfolgenden vordefinierten Konstanten von 4D:

Konstante	Typ	Wert
Macintosh byte ordering	Lange Ganzzahl	1
Native byte ordering	Lange Ganzzahl	0
PC byte ordering	Lange Ganzzahl	2

Hinweis zur Plattformunabhängigkeit: Bei diesem Befehl müssen Sie sich selbst um den Austausch von Bytes zwischen den Plattformen kümmern.

Geben Sie den optionalen Parameter * an, wird der 4-byte Wert vom Typ Lange Ganzzahl an das BLOB angefügt und die Größe des BLOB entsprechend erweitert. Mit dem optionalen Parameter * können Sie jede Zahl vom Typ *Ganzzahl*, *Lange Ganzzahl*, *Zahl* oder *Text* (siehe andere BLOB Befehle) sequentiell in einem BLOB speichern, solange der Speicher für das BLOB ausreicht.

Geben Sie den optionalen Parameter * oder den Variablenparameter *Offset* nicht an, wird der 4-byte Wert vom Typ Ganzzahl am Anfang des BLOB gespeichert, der bisherige Inhalt wird überschrieben. Die Größe des BLOB wird entsprechend angepasst.

Übergeben Sie den Variablenparameter *Offset*, wird der 4-byte Wert vom Typ Lange Ganzzahl im BLOB am Byte-Versatz geschrieben (von Null ausgehend). Egal, wo Sie den Wert schreiben, die Größe des BLOB wird gemäß der übergebenen Position erhöht (bei Bedarf bis zu 4 Bytes). Neu zugewiesene Bytes, die Sie nicht schreiben, werden auf Null initialisiert.

Nach dem Aufruf wird der Variablenparameter *Offset* zurückgegeben, erhöht um die Anzahl der geschriebenen Bytes. Von daher können Sie dieselbe Variable mit einem anderen BLOB Befehl zum Schreiben eines anderen Wertes verwenden.

Beispiel 1

Nach Ausführen dieses Code:

```
LONGINT TO BLOB(0x01020304;vxBlob;Native byte ordering)
```

- Ist die Größe von *vxBlob* 4 Bytes
- Gilt auf PowerPC: *vxBLOB{0}=$01*, *vxBLOB{1}=$02*, *vxBLOB{2}=$03*, *vxBLOB{3}=$04*
- Gilt auf Intel: *vxBLOB{0}=$04*, *vxBLOB{1}=$03*, *vxBLOB{2}=$02*, *vxBLOB{3}=$01*

Beispiel 2

Nach Ausführen dieses Code:

```
LONGINT TO BLOB(0x01020304;vxBlob;Macintosh byte ordering)
```

- Ist die Größe von *vxBlob* 4 Bytes
- Gilt auf allen Plattformen *vxBLOB{0}=$01*, *vxBLOB{1}=$02*, *vxBLOB{2}=$03*, *vxBLOB{3}=$04*

Beispiel 3

Nach Ausführen dieses Code:

```
LONGINT TO BLOB(0x01020304;vxBlob;PC byte ordering)
```

- Ist die Größe von *vxBlob* 4 Bytes
- Gilt auf allen Plattformen *vxBLOB{0}=$04*, *vxBLOB{1}=$03*, *vxBLOB{2}=$02*, *vxBLOB{3}=$01*

Beispiel 4

Nach Ausführen dieses Code:

```
SET BLOB SIZE(vxBlob;100)
LONGINT TO BLOB(0x01020304;vxBlob;PC byte ordering;*)
```

- Ist die Größe von *vxBlob* 104 Bytes
- Gilt auf allen Plattformen *vxBLOB{100}=$04*, *vxBLOB{101}=$03*, *vxBLOB{102}=$02*, *vxBLOB{103}=$01*
- Bleiben die anderen Bytes des BLOB unverändert.

Beispiel 5

Nach Ausführen dieses Code:

```
SET BLOB SIZE(vxBlob;100)
vlOffset:=50
LONGINT TO BLOB(0x01020304;vxBlob;Macintosh byte ordering;vlOffset)
```

- Ist die Größe von *vxBlob* 100 Bytes
- Gilt auf allen Plattformen *vxBLOB{50}=$01*, *vxBLOB{51}=$02*, *vxBLOB{52}=$03*, *vxBLOB{53}=$04*
- Bleiben die anderen Bytes des BLOB unverändert.
- Wurde die Variable *vlOffset* um 4 erhöht (und ist nun gleich 54)

REAL TO BLOB

REAL TO BLOB (Zahl ; BLOB ; ZahlenFormat {; Offset I *})

Parameter	Typ		Beschreibung
Zahl	Zahl	➡	Wert vom Typ Zahl zum Schreiben ins BLOB
BLOB	BLOB	➡	BLOB zum Empfangen des Wertes vom Typ Zahl
ZahlenFormat	Lange Ganzzahl	➡	0=Native Zahlenformat, 1=Erweitertes Zahlenformat, 2=Macintosh doppeltes Zahlenformat, 3=Windows doppeltes Zahlenformat
Offset I *	Variable, Operator	➡	Versatz im BLOB (in Bytes) oder * zum Anfügen des Wertes
		⬅	Neuer Versatz nach Schreiben ohne *

Beschreibung

Der Befehl **REAL TO BLOB** schreibt den Wert *Zahl* in *BLOB*.

Der Parameter *ZahlenFormat* legt das interne Format und die Byte Anordnung des zu schreibenen Wertes *Zahl* fest. Übergeben Sie eine der folgenden vordefinierten Konstanten von 4D:

Konstante	Typ	Wert
Extended real format	Lange Ganzzahl	1
Macintosh double real format	Lange Ganzzahl	2
Native real format	Lange Ganzzahl	0
PC double real format	Lange Ganzzahl	3

Hinweis zur Plattformunabhängigkeit: Bei diesem Befehl müssen Sie sich selbst um den Austausch von Bytes zwischen den Plattformen kümmern.

Geben Sie den optionalen Parameter * an, wird der Wert vom Typ Zahl an das BLOB angefügt und die Größe des BLOB entsprechend erweitert. Mit dem optionalen Parameter * können Sie jede Zahl vom Typ *Ganzzahl*, *Lange Ganzzahl*, *Zahl* oder *Text* (siehe andere BLOB Befehle) sequentiell in einem BLOB speichern, solange der Speicher für das BLOB ausreicht.

Geben Sie den optionalen Parameter * oder den Variablenparameter *Offset* nicht an, wird der Wert vom Typ Zahl am Anfang des BLOB gespeichert, der bisherige Inhalt wird überschrieben. Die Größe des BLOB wird entsprechend angepasst.

Übergeben Sie den Variablenparameter *Offset*, wird der Wert vom Typ Zahl im BLOB am Byte-Versatz geschrieben (von Null ausgehend). Egal, wo Sie den Wert schreiben, die Größe des BLOB wird gemäß der übergebenen Position erhöht (bei Bedarf bis zu 8 bzw. 10 Bytes). Neu zugewiesene Bytes, die Sie nicht schreiben, werden auf Null initialisiert.

Nach dem Aufruf wird der Variablenparameter *Offset* zurückgegeben, erhöht um die Anzahl der geschriebenen Bytes. Von daher können Sie dieselbe Variable mit einem anderen BLOB Befehl zum Schreiben eines anderen Wertes verwenden.

Beispiel 1

Nach Ausführen dieses Code:

```
C_REAL(vrValue)
vrValue:=...
REAL TO BLOB(vrValue;vxBlob;Native real format)
```

Ist die Größe von *vxBlob* auf allen Plattformen 8 Bytes.

Beispiel 2

Nach Ausführen dieses Code:

```
C_REAL(vrValue)
vrValue:=...
REAL TO BLOB(vrValue;vxBlob;Extended real format)
```

Ist die Größe von *vxBlob* auf allen Plattformen 10 Bytes.

Beispiel 3

Nach Ausführen dieses Code:

```
C_REAL(vrValue)
vrValue:=...
REAL TO BLOB(vrValue;vxBlob;Macintosh double real format)
  ` oder Windows doppeltes Zahlenformat
```

Ist die Größe von *vxBlob* auf allen Plattformen 8 Bytes

Beispiel 4

Nach Ausführen dieses Code:

```
SET BLOB SIZE(vxBlob;100)
C_REAL(vrValue)
vrValue:=...
INTEGER TO BLOB(vrValue;vxBlob;PC double real format)
  ` oder Macintosh doppeltes Zahlenformat
```

Ist die Größe von *vxBlob* auf allen Plattformen 8 Bytes

Beispiel 5

Nach Ausführen dieses Code:

```
SET BLOB SIZE(vxBlob;100)
REAL TO BLOB(vrValue;vxBlob;Extended real format;*)
```

- Ist die Größe von *vxBlob* auf allen Plattformen 110 Bytes
- Wird der Wert vom Typ Zahl auf allen Plattformen bei den Bytes #100 to #109 gespeichert
- Bleiben alle anderen Bytes des BLOB unverändert

Beispiel 6

Nach Ausführen dieses Code:

```
SET BLOB SIZE(vxBlob;100)
C_REAL(vrValue)
vrValue:=...
vlOffset:=50
REAL TO BLOB(vrValue;vxBlob;PC double real format;vlOffset)
  ` oder Macintosh doppeltes Zahlenformat
```

- Ist die Größe von *vxBlob* auf allen Plattformen 100 Bytes
- Wird der Wert vom Typ Zahl auf allen Plattformen bei den Bytes #50 bis #57 gespeichert
- Bleiben alle anderen Bytes des BLOB unverändert
- Wurde die Variable *vlOffset* um 8 erhöht (und ist nun gleich 58)

SET BLOB SIZE

SET BLOB SIZE (BLOB ; NeueGröße {; Füller})

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	BLOB Feld oder Variable
NeueGröße	Lange Ganzzahl	➡	Neue Größe des BLOB
Füller	Lange Ganzzahl	➡	ASCII Code des Füllzeichens

Beschreibung

Der Befehl **SET BLOB SIZE** passt *BLOB* gemäß dem in *NeueGröße* übergebenen Wert an.

Wollen Sie einem BLOB neue Bytes zuweisen und diese Bytes auf einen bestimmten Wert initialisieren, übergeben Sie im optionalen Parameter *Füller* den Wert (0..255).

Beispiel 1

Nach dem Durchlaufen eines umfangreichen Prozess- oder Interprozess-BLOBs empfiehlt es sich, den dafür verwendeten Speicher wieder freizumachen. Schreiben Sie dafür:

```
SET BLOB SIZE(EinProzessBLOB;0)
SET BLOB SIZE(◇EinInterprozessBLOB;0)
```

Beispiel 2

Folgendes Beispiel erstellt ein BLOB mit 16 K gefüllt mit 0xFF:

```
C_BLOB(vxData)
SET BLOB SIZE(vxData;16*1024;0xFF)
```

Fehler verwalten

Können Sie die Größe eines BLOB wegen zu wenig Speicher nicht anpassen, wird der Fehler -108 generiert. Sie können diesen Fehler mit einer Unterbrechungsmethode **ON ERR CALL** ausfindig machen.

TEXT TO BLOB

TEXT TO BLOB (Text ; BLOB {; TextFormat {; Offset I *}})

Parameter	Typ		Beschreibung
Text	String	➡	Text zum Schreiben ins BLOB
BLOB	BLOB	➡	BLOB zum Empfangen des Textes
TextFormat	Lange Ganzzahl	➡	Format und Zeichensatz des Textes
Offset I *	Variable, Operator	➡	Versatz im BLOB (in Bytes) oder * zum Anfügen des Wertes
		⬅	Ohne *: Neuer Versatz nach Schreiben in BLOB

Beschreibung

Der Befehl **TEXT TO BLOB** schreibt den Wert *Text* in *BLOB*.

Der Parameter *TextFormat* legt das interne Format und den Zeichensatz des zu schreibenden Textwertes fest. Übergeben Sie dazu eine der folgenden vordefinierten Konstanten unter dem Thema **BLOB**:

Konstante	Typ	Wert
Mac C string	Lange Ganzzahl	0
Mac Pascal string	Lange Ganzzahl	1
Mac text with length	Lange Ganzzahl	2
Mac text without length	Lange Ganzzahl	3
UTF8 C string	Lange Ganzzahl	4
UTF8 text with length	Lange Ganzzahl	5
UTF8 text without length	Lange Ganzzahl	6

Geben Sie den Parameter *TextFormat* nicht an, verwendet 4D standardmäßig das Format *Mac C string*. In Anwendungen, die mit 4D v11 erstellt wurden, arbeitet 4D standardmäßig mit dem Unicode Zeichensatz (UTF8) zur Textverwaltung, deshalb wird empfohlen, diesen Zeichensatz zu verwenden.

Hinweise:

- Die "UTF8" Konstanten sind nur verwendbar, wenn die Anwendung im Unicode Modus arbeitet.
- Die "Mac" Konstanten können nicht mit Text größer als 32 KB arbeiten.
- Wollen Sie mit einem anderen Zeichensatz als UTF8 arbeiten, verwenden Sie den Befehl **CONVERT FROM TEXT**.

Die verschiedenen Formate werden im folgenden beschrieben:

Textformat	Beschreibung und Beispiele
C String	Der Text endet mit dem Zeichen NULL (ASCII Code $00)
UTF8	"" --> $00
	"Café" --> $43 61 66 C3 A9 00
Mac	"Café" --> $43 61 66 8E 00
Pascal String	Dem Text ist eine 1-Byte Länge vorangestellt
UTF8	-
Mac	"" --> $00
	"Café" --> $04 43 61 66 8E
Text mit Länge	Dem Text ist eine 4-Byte (UTF8) oder 2-Byte Länge (Mac) vorangestellt
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 ohne Länge	Der Text besteht nur aus seinen Zeichen
UTF8	"" --> Kein Wert
	"Café" --> $43 61 66 C3 A9
Mac	"" --> Kein Wert
	"Café" --> $43 61 66 8E

Geben Sie den optionalen Parameter * an, wird der Wert vom Typ Zahl an das BLOB angefügt und die Größe des BLOB entsprechend erweitert. Mit dem optionalen Parameter * können Sie jede Zahl vom Typ *Ganzzahl*, *Lange Ganzzahl*, *Zahl* oder *Text* (siehe andere BLOB Befehle) sequentiell in einem BLOB speichern, solange der Speicher für das BLOB ausreicht.

Geben Sie den optionalen Parameter * oder den Parameter *Offset* nicht an, wird der Textwert am Anfang des BLOB gespeichert, der bisherige Inhalt wird überschrieben. Die Größe des BLOB wird entsprechend angepasst.

Übergeben Sie den Parameter *Offset*, wird der Textwert im BLOB am Byte-Versatz geschrieben (von Null ausgehend). Egal, wo Sie den Wert schreiben, die Größe des BLOB wird gemäß der übergebenen Position erhöht (bei Bedarf bis zur Größe des Textes). Neu zugewiesene Bytes, die Sie nicht schreiben, werden auf Null initialisiert.

Nach dem Aufruf wird die Variable *Offset* zurückgegeben, erhöht um die Anzahl der geschriebenen Bytes. Von daher können Sie dieselbe Variable mit einem anderen BLOB Befehl zum Schreiben eines anderen Wertes verwenden.

Beispiel

Nach Ausführen dieses Code:

```
SET BLOB SIZE(vxBlob;0)
C_TEXT(vtValue)
vtValue:="Café" ` Länge von vtValue ist 4 Bytes
TEXT TO BLOB(vtValue;vxBlob;Mac C string) ` BLOB Größe wird 5 Bytes
TEXT TO BLOB(vtValue;vxBlob;Mac Pascal string) ` BLOB Größe wird 5 Bytes
TEXT TO BLOB(vtValue;vxBlob;Mac text with length) ` BLOB Größe wird 6 Bytes
TEXT TO BLOB(vtValue;vxBlob;Mac text without length) ` BLOB Größe wird 4 Bytes
TEXT TO BLOB(vtValue;vxBlob;UTF8 C string) ` BLOB Größe wird 6 Bytes
TEXT TO BLOB(vtValue;vxBlob;UTF8 text with length) ` BLOB Größe wird 9 Bytes
TEXT TO BLOB(vtValue;vxBlob;UTF8 text without length) ` BLOB Größe wird 5 Bytes
```

VARIABLE TO BLOB

VARIABLE TO BLOB (Variable ; BLOB {; Offset | *})

Parameter	Typ		Beschreibung
Variable	Variable	➡	Variable, die im BLOB gespeichert werden soll
BLOB	BLOB	➡	BLOB, das die Variable empfängt
Offset | *	Variable, Operator	➡	Versatz der Variablen im BLOB (in Bytes) oder *, um Variable am Ende anzufügen
		⬅	Neuer Versatz nach Schreiben ohne *

Beschreibung

Der Befehl **VARIABLE TO BLOB** speichert die Variable *Variable* in *BLOB*.

Mit dem optionalen Parameter * wird die Variable im BLOB angefügt und dessen Größe entsprechend erweitert. Mit dem optionalen Parameter * können Sie beliebig viele Variablen oder Listen sequentiell in einem BLOB speichern (siehe andere BLOB Befehle), solange der Speicher dafür ausreicht.

Geben Sie den optionalen Parameter * bzw. den Variablenparameter *Offset* nicht an, wird die Variable am Anfang des BLOB gespeichert. Sie überschreibt seinen bisherigen Inhalt; die Größe des BLOB wird entsprechend angepasst.

Geben Sie den Variablenparameter *Offset* an, wird die Variable am Versatzpunkt (beginnend mit Null) geschrieben. Unabhängig, wo Sie die Variable schreiben, wird die Größe des BLOB gemäß der übergebenen Positionierung erhöht, falls notwendig plus der Größe der Variablen. Neu zugewiesene Bytes werden im Gegensatz zu den geschriebenen Bytes auf Null initialisiert.

Nach dem Aufruf wird der Variablenparameter *Offset* zurückgegeben, erhöht um die Anzahl der geschriebenen Bytes. Von daher können Sie dieselbe Variable mit einem anderen BLOB schreibenden Befehl wiederverwenden, um eine andere Variable oder Liste zu schreiben.

VARIABLE TO BLOB akzeptiert jeden Variablentyp, einschließlich anderer BLOBs. Davon ausgenommen sind:

- Zeiger
- Array mit Zeigern

Bitte beachten Sie folgendes:

- Speichern Sie eine Variable vom Typ Lange Ganzzahl als Referenz auf eine hierarchische Liste (ListRef), speichert **VARIABLE TO BLOB** die Variable Lange Ganzzahl und nicht die Liste. Zum Speichern und Wiederfinden hierarchischer Listen in und aus einem BLOB verwenden Sie die Befehle **LIST TO BLOB** und **BLOB to list**.
- Übergeben Sie ein Objekt **C_OBJECT** oder eine Collection **C_COLLECTION** im Parameter *Variable*, setzt der Befehl eine Kopie (keine Referenz) als JSON in UTF-8 in das BLOB. Enthält das Objekt oder die Collection Zeiger, werden ihre dereferenzierten Werte im BLOB gespeichert und nicht die Zeiger selbst.

Speichern Sie jedoch eine Variable *Lange Ganzzahl*, die eine Referenz auf eine hierarchische Liste ist (*ListRef*), speichert **VARIABLE TO BLOB** die Variable *Lange Ganzzahl* und nicht die Liste. Um hierarchische Listen in und aus einem BLOB zu speichern und zu finden, verwenden Sie die Befehle **LIST TO BLOB** und **BLOB to list**.

Warnung: Speichern Sie Variablen in einem BLOB, müssen Sie später mit dem Befehl **BLOB TO VARIABLE** den Inhalt des BLOB auslesen, da Variablen in BLOBs mit einem internen 4D Format gespeichert werden.

Hinweis zur Plattformunabhängigkeit: VARIABLE TO BLOB und **BLOB TO VARIABLE** verwalten die in BLOBs gespeicherten Variablen in einem internen 4D Format. Beim Einsetzen dieser beiden Befehle müssen Sie sich deshalb nicht um den Austausch von Bytes zwischen den Plattformen kümmern. Mit anderen Worten, Sie können ein unter Windows erstelltes BLOB auf macOS wiederverwenden und umgekehrt.

Beispiel 1

Mit den beiden nachfolgenden Projektmethoden können Sie Arrays in und aus Dokumenten rasch speichern bzw. wiederfinden:

```
  ` Projektmethode SAVE ARRAY
  ` SAVE ARRAY ( String ; Zeiger )
  ` SAVE ARRAY ( Dokument ; -> Array )
C_STRING(255;$1)
C_POINTER($2)
C_BLOB($vxArrayData)
VARIABLE TO BLOB($2->;$vxArrayData)  ` Speichere Array im BLOB
COMPRESS BLOB($vxArrayData)  ` Komprimiere BLOB
BLOB TO DOCUMENT($1;$vxArrayData)  ` Sichere BLOB auf Festplatte

  ` Projektmethode LOAD ARRAY
  ` LOAD ARRAY ( String ; Zeiger )
  ` LOAD ARRAY ( Dokument ; -> Array )
C_STRING(255;$1)
C_POINTER($2)
C_BLOB($vxArrayData)
DOCUMENT TO BLOB($1;$vxArrayData)  ` Lade BLOB von Festplatte
EXPAND BLOB($vxArrayData)  ` Erweitere BLOB
BLOB TO VARIABLE($vxArrayData;$2->)  ` Finde Array aus dem BLOB wieder
```

Haben Sie diese Methoden hinzugefügt, können Sie schreiben:

```
ARRAY STRING(...;asAnyArray;...)
  ` ...
SAVE ARRAY($vsDocName;->asAnyArray)
  ` ...
LOAD ARRAY($vsDocName;->asAnyArray)
```

Beispiel 2

Mit den beiden nachfolgenden Projektmethoden können Sie jede Variablenmenge in und aus einem BLOB speichern bzw. wiederfinden:

```
  ` Projektmethode STORE VARIABLES INTO BLOB
  ` STORE VARIABLES INTO BLOB ( Zeiger { ; Zeiger ... { ; Zeiger } } )
  ` STORE VARIABLES INTO 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

  ` Projektmethode RETRIEVE VARIABLES FROM BLOB
  ` RETRIEVE VARIABLES FROM BLOB ( ´ Zeiger { ; Zeiger ... { ; Zeiger } } )
  ` RETRIEVE VARIABLES FROM 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
```

Haben Sie diese Methoden hinzugefügt, können Sie schreiben:

```
STORE VARIABLES INTO BLOB(->vxBLOB;->vgPicture;->asAnArray;->alAnotherArray)
  ` ...
RETRIEVE VARIABLES FROM BLOB(->vxBLOB;->vgPicture;->asAnArray;->alAnotherArray)
```

Systemvariablen und Mengen

Wurde die Variable erfolgreich gespeichert, wird die OK Variable auf 1 gesetzt, andernfalls auf 0. Das tritt zum Beispiel ein, wenn nicht genügend Speicher vorhanden ist.

Boolean

- Boolean Befehle
- Bool
- False
- Not
- True

Boolean Befehle

4D enthält folgende Funktionen für Boolean Berechnungen:

```
True
False
Not
```

Beispiel

Dieses Beispiel setzt die Variable *myBoolean* je nach Wert der Schaltfläche. Sie gibt Wahr zurück, wenn die Schaltfläche *myButton* angeklickt wurde und Falsch, wenn sie nicht angeklickt wurde. Ist die Schaltfläche angeklickt, hat die Variable für die Schaltfläche den Wert 1.

```
If(myButton=1) ` Wurde die Schaltfläche angeklickt,
   myBoolean:=True ` Wird myBoolean auf Wahr gesetzt
Else ` Wurde die Schaltfläche nicht angeklickt,
   myBoolean:=False ` Wird myBoolean auf Falsch gesetzt
End if
```

Das obige Beispiel lässt sich auch in einer Zeile zusammenfassen:

```
myBoolean:=(myButton=1)
```

Bool

Bool (Ausdruck) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Ausdruck	Ausdruck	⇒	Ausdruck für zurückzugebende boolean Form
Funktionsergebnis	Boolean	⇐	Boolean Form des Ausdrucks

Beschreibung

Die Funktion **Bool** gbit die boolean Form des Ausdrucks zurück, definiert im Parameter *Ausdruck*.
Je nach Ergebnistyp von *Ausdruck* gibt es folgende Werte:

Ergebnistyp Ausdruck	Zurückgegebene Boolean Form
Undefiniert	Falsch
Null	Falsch
Boolean	Falsch wenn falsch, sonst Wahr
Zahl	Falsch wenn 0, andere Wahr
Andere Typen	Falsch

Diese Funktion ist hilfreich, wenn der Code einen boolean Wert erwartet und die Bewertung des Ausdrucks einen anderen Typ ergeben kann, z.B. **Null** oder **undefiniert**.

Beispiel

Sie wählen einen Wert abhängig vom Inhalt des Attributs eines Objektfeldes, und nehmen den Fall vorweg, wo das Attribut fehlt:

```
C_TEXT($married)
$married:=Choose(Bool([Person]data.married);"Married";"Single")  //"Single", wenn Attribut "Married" nicht gesetzt ist
ALERT("This person is "+$married)
```

False

False -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Falsch

Beschreibung

Die Funktion **False** gibt den Boolean Wert *Falsch* zurück.

Beispiel

Folgendes Beispiel setzt die Variable *vbOptions* auf Falsch:

```
vbOptions:=False
```

Not

Not (Boolean) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Boolean	Boolean	⇒	Zu verneinender Boolean Wert
Funktionsergebnis	Boolean	⮌	Gegenteil von Boolean

Beschreibung

Die Funktion **Not** gibt die Verneinung des Parameters *Boolean* zurück, Wahr wird zu Falsch und Falsch wird zu Wahr.

Beispiel

Dieses Beispiel weist der Variablen zuerst Wahr zu, ändert den Wert dann in Falsch, und anschließend wieder in Wahr.

```
vResult:=True  ` vResult wird auf Wahr gesetzt
vResult:=Not(vResult)  ` vResult wird auf Falsch gesetzt
vResult:=Not(vResult)  ` vResult wird auf Wahr gesetzt
```

True

True -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Wahr

Beschreibung

Die Funktion **True** gibt den Boolean Wert Wahr zurück.

Beispiel

Folgendes Beispiel setzt die Variable *vbOptions* auf Wahr:

```
vbOptions:=True
```

Cache Verwaltung

4D enthält ein Datencache-Schema zum Beschleunigen der E/A Operationen. Die Tatsache, dass Änderungen an den Daten für eine gewisse Zeit im Datencache und nicht auf der Festplatte liegen, ist transparent für Ihre Programmierung. Ruft zum Beispiel ein Task den Befehl **QUERY** auf, integriert die 4D Datenbank Engine den Datencache im Suchvorgang.

Der Cache Manager der Datenbank wurde in 4D v16 komplett neu geschrieben, um Vorteile von Umgebungen in 64-bit besser nutzen können. Er wird automatisch aktiviert und optimiert. Mit den Befehlen dieses Kapitels lässt er sich aber auch dynamisch konfigurieren oder analysieren. Darüberhinaus gibt es ein individuell anpassbares Prioritätensystem. Weitere Informationen dazu finden Sie im Abschnitt **Prioritäten im Datenbank Cache verwalten**.

Mit den Befehlen **SET DATABASE PARAMETER** und **Get database parameter** können Sie auch den Selector Cache flush periodicity einsetzen.

- Prioritäten im Datenbank Cache verwalten
- 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

Prioritäten im Datenbank Cache verwalten

In 64-bit Versionen von 4D enthält der Datenbank Cache einen automatischen Mechanismus zum Verwalten der Prioritäten. Das sorgt für hohe Effizienz und Performance beim Zugriff auf Daten: Wird Platz zum Laden neuer Daten in den Cache benötigt, werden Daten im Cache mit niedriger Priorität zuerst entfernt, während Daten mit höherer Priorität im Cache erhalten bleiben.

Dieser Vorgang läuft vollautomatisch und in der Regel müssen Sie sich um nichts kümmern. Für besondere Fälle sind aber auch individuelle Vorgaben möglich. Über spezifische Befehle können Sie die Priorität von Objekten für die gesamte Laufzeit der Anwendung oder temporär für den aktuellen Prozess verändern. Beachten Sie, dass diese Befehle mit Bedacht verwendet werden müssen, da sie die Performance der Anwendung beeinflussen.

Überblick zum Verwalten der Priorität

Der Cache Manager wählt Daten, die bei Bedarf aus dem Cache entfernt werden müssen, nach Alter und Häufigkeit der Nutzung aus und zusätzlich nach ein Prioritätensystem. Es gibt drei Arten von Objekten mit unterschiedlicher Priorität:

- **Tabellen**: Standardmäßige Daten im Datenfeld (Zahl, Datum...), außer Blobs (siehe nächster Punkt). Die standardmäßige Priorität ist mittel
- **Blobs**: Binäre Daten im Datenfeld (Text, Bild, Objekt und Blob), die in der Datendatei gespeichert werden. Die standardmäßige Priorität ist am niedrigsten
- **Indizes**: Alle einfachen Indizes auf Felder, inkl. Volltext-Indizes und zusammengesetzte Indizes. Da häufig auf Indizes zugegriffen wird, haben sie einen Sonderstatus im Cache. Die standardmäßige Priorität ist am höchsten.

In der Regel bieten die standardmäßig definierten Prioritäten die beste Performance. Für besondere Fälle gibt es zwei Befehlssätze, um die Priorität im Cache individuell anpassen:

- Befehle, welche die Priorität im Cache für die gesamte Sitzung und für alle Prozesse ändern: **SET TABLE CACHE PRIORITY**, **SET INDEX CACHE PRIORITY** und **SET BLOBS CACHE PRIORITY**. Verwenden Sie diese Befehle in Startup Datenbankmethoden.

- Befehle, welche die Priorität im Cache nur für den aktuellen Prozess ändern: **ADJUST TABLE CACHE PRIORITY**, **ADJUST INDEX CACHE PRIORITY** und **ADJUST BLOBS CACHE PRIORITY**. Verwenden Sie diese Befehle, um die Performance einer temporären Operation in Ihrer Anwendung zu verbessern und anschließend wieder zu den ursprünglichen Prioritäten zurückzukehren.

Diese Befehle sind nur in folgenden Kontexten verfügbar:

- 4D 64-bit Versionen
- 4D Server oder 4D im lokalen Modus

Hinweis: Bitte beachten Sie, dass die Funktion **Cache info** nur Prioritätsänderungen durch die **SET**-Befehle berücksichtigt, die für die gesamte Datenbanksitzung gelten, **jedoch nicht** Prioritätsänderungen durch die **ADJUST**-Befehle, die nur für den aktuellen Prozess gelten. Verwenden Sie hierzu die entsprechenden **GET-**Befehle.

ADJUST BLOBS CACHE PRIORITY

ADJUST BLOBS CACHE PRIORITY (Tabellenname ; Priorität)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Anpassen der Priorität für Daten vom Typ Blob
Priorität	Lange Ganzzahl	➡	Wert der Priorität im Cache für BLOBs in der Tabelle

Nur für Experten

Dieser Befehl ist für Sonderfälle reserviert und muss mit Bedacht verwendet werden, da er die Performance der Datenbank beeinträchtigen kann.

Beschreibung

Der Befehl **ADJUST BLOBS CACHE PRIORITY** ändert die *Priorität* im Cache für Daten vom Typ BLOB in *Tabellenname* für den aktuellen Prozess. Ein Aufruf dieses Befehls ersetzt jeden zuvor gesetzten Wert für Priorität durch denselben Befehl im selben Prozess. Dieser Befehl passt die Priorität für temporäre Operationen an, wie z.B. eine Suche oder Datenimport.

Hinweis: ADJUST BLOBS CACHE PRIORITY funktioniert nur im lokalen Modus (4D Server und 4D); sie ist nicht im 4D remote Modus verwendbar.

Datenfelder mit Blobs sind vom Typ BLOB, Text, Bild und Objekt. Dieser Befehl verwaltet die Priorität nur, wenn diese Daten innerhalb der Datendatei gespeichert sind.
Die Priorität für skalare Datenfelder (vom Typ Datum, Zahl oder String) definieren Sie über den Befehl **ADJUST TABLE CACHE PRIORITY**.

In *Priorität* übergeben Sie eine der folgenden Konstanten unter dem Thema **Cache Verwaltung**:

Konstante	Typ	Wert	Kommentar
Cache priority high	Lange Ganzzahl	1000	
Cache priority low	Lange Ganzzahl	-900	
Cache priority normal	Lange Ganzzahl	0	Setzt den Wert für Priorität im Cache auf den Standardwert
Cache priority very high	Lange Ganzzahl	30000	
Cache priority very low	Lange Ganzzahl	-1	

Beispiel

Beim Ausführen einer sequentiellen Suche die Priorität im Cache temporär verändern für Datenfelder vom Typ Text in der Tabelle [Docs], gespeichert in der Datendatei:

```
ADJUST BLOBS CACHE PRIORITY([Docs];Cache priority very high)
QUERY([Docs];[Docs]Author#"A@")  // sequentielle Suche in einem nicht-indizierten Feld
 //... einige andere Such- oder Sortierläufe in der gleichen Tabelle durchführen
 // wenn abgeschlossen, Cache Priorität auf Normal zurücksetzen
ADJUST BLOBS CACHE PRIORITY([Docs];Cache priority normal)
```

ADJUST INDEX CACHE PRIORITY

ADJUST INDEX CACHE PRIORITY (Feldname ; Priorität)

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Feld zum Anpassen der Priorität für Index/Indizes
Priorität	Lange Ganzzahl	➡	Wert der Priorität im Cache für Index/Indizes des Datenfeldes

Nur für Experten

Dieser Befehl ist für Sonderfälle reserviert und muss mit Bedacht verwendet werden, da er die Performance der Datenbank beeinträchtigen kann.

Beschreibung

Der Befehl **ADJUST INDEX CACHE PRIORITY** setzt einen spezifischen Wert *Priorität* im Cache für den Index bzw. Indizes von *Feldname* für den aktuellen Prozess. Ein Aufruf dieses Befehls ersetzt jeden zuvor gesetzten Wert für Priorität durch denselben Befehl im selben Prozess. Dieser Befehl passt die Priorität für temporäre Operationen an, wie z.B. eine Suche oder Datenimport.

Hinweis: ADJUST INDEX CACHE PRIORITY funktioniert nur im lokalen Modus (4D Server und 4D); er ist nicht im 4D remote Modus verwendbar.

ADJUST INDEX CACHE PRIORITY verwaltet die Priorität aller Indizes von *Feldname*. Er unterstützt Volltext-Indizes, jedoch nicht zusammengesetzte Indizes.

In *Priorität* übergeben Sie eine der folgenden Konstanten unter dem Thema **Cache Verwaltung**:

Beispiel

Die Priorität im Cache für Index/ Indizes im Datenfeld [Docs]Comments temporär verändern:

```
ADJUST INDEX CACHE PRIORITY([Docs]Comments;Cache priority very high)
QUERY([Docs];[Docs]Comments%"Extra")  // Suche in einem indizierten Feld
  //... einige andere Such- oder Sortierläufe in der gleichen Tabelle ausführen
  // wenn abgeschlossen, Cache Priorität auf Normal zurücksetzen
ADJUST INDEX CACHE PRIORITY([Docs]Comments;Cache priority normal)
```

ADJUST TABLE CACHE PRIORITY

ADJUST TABLE CACHE PRIORITY (Tabellenname ; Priorität)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Anpassen der Priorität für skalare Werte
Priorität	Lange Ganzzahl	➡	Wert der Priorität im Cache für die Tabelle

Nur für Experten

Dieser Befehl ist für Sonderfälle reserviert und muss mit Bedacht verwendet werden, da er die Performance der Datenbank beeinträchtigen kann.

Beschreibung

Der Befehl **ADJUST TABLE CACHE PRIORITY** setzt einen spezifischen Wert *Priorität* im Cache für Daten in *Tabellenname* für den aktuellen Prozess. Ein Aufruf dieses Befehls ersetzt jeden zuvor gesetzten Wert für Priorität durch denselben Befehl im selben Prozess. Dieser Befehl passt die Priorität für temporäre Operationen an, wie z.B. eine Suche oder Datenimport.

Hinweis: ADJUST TABLE CACHE PRIORITY funktioniert nur im lokalen Modus (4D Server und 4D); er ist nicht im 4D remote Modus verwendbar.

Dieser Befehl verwaltet nur die Priorität für Daten in skalaren Feldern (vom Typ Datum, Zahl oder String). Die Priorität für binäre Datenfelder (vom Typ Blobs, Text, Bilder und Objekte) definieren Sie über den Befehl **ADJUST BLOBS CACHE PRIORITY**.

In *Priorität* übergeben Sie eine der folgenden Konstanten unter dem Thema **Cache Verwaltung**:

Konstante	Typ	Wert	Kommentar
Cache priority high	Lange Ganzzahl	1000	
Cache priority low	Lange Ganzzahl	-900	
Cache priority normal	Lange Ganzzahl	0	Setzt den Wert für Priorität im Cache auf den Standardwert
Cache priority very high	Lange Ganzzahl	30000	
Cache priority very low	Lange Ganzzahl	-1	

Beispiel

Die Priorität im Cache temporär für skalare Datenfelder in der Tabelle [Docs] verändern:

```
ADJUST TABLE CACHE PRIORITY([Docs];Cache priority low)
 // ... eine bestimmte Operation ausführen
ADJUST TABLE CACHE PRIORITY([Docs];Cache priority normal)
```

Cache info

Cache info {(dbFilter)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
dbFilter	Objekt	➡	Definiert die Liste der Attribute zum Zurückgeben (gefiltert nach DB)
Funktionsergebnis	Objekt	↩	Information über Cache

Nur 64-bit

Diese Funktion arbeitet nur in 4D Versionen für 64-bit.

Beschreibung

Die Funktion **Cache info** gibt ein Objekt mit detaillierten Informationen über den Inhalt des aktuellen Cache zurück (benutzter Speicher, geladene Tabellen und Indizes, etc.).

Hinweis: Sie arbeitet nur im lokalen Modus (4D Server und 4D); sie darf **nicht** in 4D im remote Modus verwendet werden.

Standardmäßig gilt die zurückgegebene Information nur für die aktuell laufende Datenbank. Mit dem optionalen Parameter *dbFilter* können Sie die Reichweite der Funktion angeben:

- "dbFilter" mit dem Wert "All" gibt Cache Information zu allen laufenden Datenbanken zurück, inkl. Komponenten
- "dbFilter" mit "" (leerer String) gibt nur Information über die aktuelle Datenbank zurück (entspricht Weglassen des Parameters *dbFilter*)

Cache info gibt ein einzelnes Objekt mit allen relevanten Informationen über den Cache zurück. Es ist folgendermaßen aufgebaut:

```
{
    "maxMem": Maximum cache size (real),
    "usedMem": Current cache size (real),
    "objects": [...] Array mit derzeit im Cache geladenen Objekten
}
```

Elemente des Arrays *objects* sind Ursprungsobjekte (Tabellen, Indizes, Blobs, etc.) die derzeit im Cache geladen sind. Jedes Element enthält spezifische Attribute, die seinen aktuellen Status beschreiben. Weiterführende Informationen zur Interpretation dieser Daten erfahren Sie über Ihren technischen Support vor Ort.

Beispiel

Cache Information für die aktuelle Datenbank erhalten:

```
C_OBJECT($cache)
$cache:=Cache info
```

Cache Information für die Datenbank und alle geöffneten Komponenten erhalten:

```
C_OBJECT($dbFilter)
OB SET($dbFilter;"dbFilter";"All")
$cache:=Cache info($dbFilter)
```

FLUSH CACHE

FLUSH CACHE {(Größe | *)}

Parameter	Typ		Beschreibung
Größe | *	Zahl, Operator	⇒	* um Cache-Speicher komplett zu leeren, oder Anzahl freizugebender Bytes

Beschreibung

Der Befehl **FLUSH CACHE** schreibt die im Arbeitsspeicher gehaltenen Daten sofort auf die Festplatte. Alle in der Datenbank durchgeführten Änderungen werden auf der Festplatte gespeichert.

Standardmäßig bleibt der aktuelle Cache-Speicher selbst unverändert, d.h. die Daten stehen für zukünftige Lesevorgänge weiterhin bereit. Sie können optional einen Parameter übergeben, um seinen Inhalt zu entleeren:

- * sichert den Cache und gibt den gesamten Cache-Speicher frei
- Ein Wert in *Größe* sichert den Cache und gibt nur die angegebene Anzahl Bytes im Cache frei

Hinweis: Die Übergabe eines Parameters dient nur für Testzwecke. Aus Performance-Gründen wird nicht empfohlen, den Cache in der Produktionsumgebung ganz oder teilweise freizumachen.

Im Normalfall sollten Sie diesen Befehl nicht aufrufen, da 4D Datenänderungen auf regulärer Basis sichert. Das Speichern des Cache wird über die Option **Daten-Cache sichern alle X Sekunden (Minuten)** auf der **Seite Datenbank/ Speicher** der Datenbank Eigenschaften bestimmt. Die Option gibt an, in welchen Zeitabständen gesichert wird. Wir empfehlen, den Standardwert von 20 Sekunden zu verwenden. Diese Zeit lässt sich auch in den Befehlen **SET DATABASE PARAMETER** und **Get database parameter** über die Konstante Cache flush periodicity setzen und lesen.

Get adjusted blobs cache priority

Get adjusted blobs cache priority (Tabellenname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle zum Erhalten des Werts der Priorität für "Blobs"
Funktionsergebnis	Lange Ganzzahl	⇐	Aktueller Wert der Priorität für Felder vom Typ Blob

Beschreibung

Die Funktion **Get adjusted blobs cache priority** gibt den aktuellen angepassten Wert der Priorität im Cache für Daten vom Typ Blob in *Tabellenname* zurück. Sie wird nur für Debugging Zwecke benötigt.

Hinweis: Get adjusted blobs cache priority funktioniert nur im lokalen Modus (4D Server und 4D); sie ist nicht im 4D remote Modus verwendbar.

Get adjusted index cache priority

Get adjusted index cache priority (Feldname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Feld zum Erhalten der Priorität für den Index
Funktionsergebnis	Lange Ganzzahl	⮌	Aktueller Wert der Priorität für Index/Indizes

Beschreibung

Die Funktion **Get adjusted index cache priority** gibt den aktuellen angepassten Wert der Priorität im Cache für den Index von *Feldname* zurück. Sie wird nur für Debugging Zwecke benötigt.

Hinweis: Get adjusted index cache priority funktioniert nur im lokalen Modus (4D Server und 4D); sie ist nicht im 4D remote Modus verwendbar.

Get adjusted table cache priority

Get adjusted table cache priority (Tabellenname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Erhalten der Priorität für skalare Werte
Funktionsergebnis	Lange Ganzzahl	⮌	Aktueller Wert der Priorität für skalare Felder

Beschreibung

Die Funktion **Get adjusted table cache priority** gibt den aktuellen angepassten Wert der Priorität im Cache für skalare Daten von *Tabellenname* zurück. Sie wird nur für Debugging Zwecke benötigt.

Hinweis: Get adjusted table cache priority funktioniert nur im lokalen Modus (4D Server und 4D); sie ist nicht im 4D remote Modus verwendbar.

Datenfelder mit skalaren Daten sind vom Typ Datum/Uhrzeit, Zahl oder String.

Get cache size

Get cache size -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Zahl	Größe des Datenbank-Cache in Bytes

Beschreibung

Die Funktion **Get cache size** gibt die Größe des aktuellen Datenbank-Cache in Bytes zurück.
Hinweis: Sie arbeitet nur im lokalen Modus (4D Server und 4D); sie darf **nicht** in 4D im remote Modus verwendet werden.

Beispiel

Siehe Beispiel im Befehl **SET CACHE SIZE**.

GET MEMORY STATISTICS

GET MEMORY STATISTICS (infoTyp ; arrNamen ; arrWerte ; arrZähler)

Parameter	Typ		Beschreibung
infoTyp	Lange Ganzzahl	➡	Selector für die zu erhaltende Information
arrNamen	Array Text	⬅	Information Titel
arrWerte	Array Zahl	⬅	Information Werte
arrZähler	Array Zahl	⬅	Anzahl der betroffenen Objekte (sofern vorhanden)

Beschreibung

Der Befehl **GET MEMORY STATISTICS** liefert Information zur Verwendung des Daten-Cache durch 4D. Damit können Sie die Arbeitsweise der Anwendung analysieren.
Im Parameter *infoTyp* übergeben Sie einen Wert, der den gewünschten Informationstyp angibt:

- 1 = Allgemeine Speicherinformation. Diese Information ist auch über den Runtime Explorer verfügbar: Größe des tatsächlichen, virtuellen, freien sowie des benutzten Speichers, Stapelspeicher und freiem Stapelspeicher, etc.
- 2 = Zusammenfassung der Belegungsstatistik des Cache der Datenbank (nur 32-bit Versionen)

Hinweis zur Kompatibilität: Wert 2 ist für 4D 64-bit Versionen überholt. Für diese Versionen können Sie die Funktion **Cache info** für die Belegungsstatistik des Cache verwenden.

Nach Ausführung des Befehls erscheinen die angeforderten statistischen Daten in den Arrays *arrNamen*, *arrWerte* und *arrZähler*. Weiterführende Informationen zur Interpretation dieser Daten erfahren Sie über Ihren Technischen Support vor Ort.

SET BLOBS CACHE PRIORITY

SET BLOBS CACHE PRIORITY (Tabellenname ; Priorität)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Setzen der Priorität für Daten vom Typ Blob für die Dauer der Sitzung
Priorität	Lange Ganzzahl	➡	Wert der Priorität für BLOBS in der Tabelle

Nur für Experten

Dieser Befehl ist für Sonderfälle reserviert und muss mit Bedacht verwendet werden, da er die Performance der Datenbank beeinträchtigen kann.

Beschreibung

Der Befehl **SET BLOBS CACHE PRIORITY** setzt einen spezifischen Wert *Priorität* im Cache für Daten vom Typ Blob in *Tabellenname* für alle Prozesse in der aktuellen Sitzung. Rufen Sie diesen Befehl in der Datenbankmethode **On Startup** oder **On Server Startup** auf.

Hinweis: SET BLOBS CACHE PRIORITY funktioniert nur im lokalen Modus (4D Server und 4D); er ist nicht im 4D remote Modus verwendbar.

Datenfelder mit Blobs sind vom Typ BLOB, Text, Bild und Objekt. Dieser Befehl verwaltet die Priorität nur, wenn diese Daten innerhalb der Datendatei gespeichert sind.

In *Priorität* übergeben Sie eine der folgenden Konstanten unter dem Thema **Cache Verwaltung**:

Konstante	Kommentar
Cache priority low	
Cache priority very low	
Cache priority normal	Setzt den Wert für Priorität im Cache auf den Standardwert
Cache priority high	
Cache priority very high	

Beispiel

In der **Datenbankmethode On Startup** eine sehr hohe Priorität für die Tabelle [Customer] setzen:

```
SET BLOBS CACHE PRIORITY([Customer];Cache priority very high)
```

SET CACHE SIZE

SET CACHE SIZE (Größe {; MinFreierPlatz})

Parameter	Typ		Beschreibung
Größe	Zahl	➡	Größe des Datenbank-Cache in Bytes
MinFreierPlatz	Zahl	➡	Speicherplatz freimachen ab der definierten Anzahl Bytes

Nur 64-bit

Diese Funktion arbeitet nur in 4D Versionen für 64-bit.

Beschreibung

Der Befehl **SET CACHE SIZE** setzt die Größe des Datenbank-Cache dynamisch und optional die Mindestgröße in Bytes, ab der Freimachen von Speicherplatz starten soll.

Hinweis: Er arbeitet nur im lokalen Modus (4D Server und 4D); er darf **nicht** in 4D im remote Modus verwendet werden.

In *Größe* übergeben Sie die neue Größe für den Datenbank-Cache in Bytes. Sie wird dynamisch beim Ausführen des Befehls gesetzt.

In *MinFreierPlatz* übergeben Sie die Mindestgröße für den Cache, wenn die Engine Platz schaffen muss, um ein Objekt in den Cache zu legen (Wert in Bytes). Über diese Option können Sie die beanspruchte Zeit beim Entfernen von Daten aus dem Cache verringern, und so eine bessere Performance erhalten.
Standardmäßig, d.h. ohne diesen Parameter, entlädt 4D mindestens 10% des Cache, wenn Platz benötigt wird. Arbeitet Ihre Anwendung mit einem umfangreichen Cache, kann es vorteilhaft sein, eine feste Größe zu verwenden, die nicht von der Größe des Cache abhängt. Sie können diese Einstellung an die Größe der Datenblöcke anpassen, die in Ihrer Anwendung verwaltet werden.

Beispiel

100 MB im Cache der aktuellen Anwendung hinzufügen:

```
C_REAL($currentCache)
$currentCache:=Get cache size
 // aktuelle Cache Größe ist z.B. 419430400
SET CACHE SIZE($currentCache+100000000)
 // aktuelle Cache Größe ist jetzt 519430400
```

SET INDEX CACHE PRIORITY

SET INDEX CACHE PRIORITY (Feldname ; Priorität)

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Feld zum Setzen der Priorität für den Index für die Dauer der Sitzung
Priorität	Lange Ganzzahl	➡	Aktueller Wert der Priorität für Index/Indizes

Nur für Experten

Dieser Befehl ist für Sonderfälle reserviert und muss mit Bedacht verwendet werden, da er die Performance der Datenbank beeinträchtigen kann.

Beschreibung

Der Befehl **SET INDEX CACHE PRIORITY** setzt einen spezifischen Wert *Priorität* im Cache für den Index/die Indizes *von Feldname* für alle Prozesse in der aktuellen Sitzung. Er sollte in der Datenbankmethode **On Startup** oder **On Server Startup** aufgerufen werden.

Hinweis: SET INDEX CACHE PRIORITY funktioniert nur im lokalen Modus (4D Server und 4D); er ist nicht im 4D remote Modus verwendbar.

Dieser Befehl verwaltet die Priorität aller Indizes von *Feldname*. Er unterstützt Volltext-Indizes, jedoch nicht zusammengesetzte Indizes.

In *Priorität* übergeben Sie eine der folgenden Konstanten unter dem Thema **Cache Verwaltung**:

Konstante	Typ	Wert	Kommentar
Cache priority high	Lange Ganzzahl	1000	
Cache priority low	Lange Ganzzahl	-900	
Cache priority normal	Lange Ganzzahl	0	Setzt den Wert für Priorität im Cache auf den Standardwert
Cache priority very high	Lange Ganzzahl	30000	
Cache priority very low	Lange Ganzzahl	-1	

Beispiel

In der **Datenbankmethode On Startup** eine sehr hohe Priorität für den Index im Feld [Customer]LastName setzen:

```
SET INDEX CACHE PRIORITY([Customer]LastName;Cache priority very high)
```

SET TABLE CACHE PRIORITY

SET TABLE CACHE PRIORITY (Tabellenname ; Priorität)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Setzen der Priorität für skalare Daten für die Dauer der Sitzung
Priorität	Lange Ganzzahl	➡	Aktueller Wert der Priorität für skalare Werte in der Tabelle

Nur für Experten

Dieser Befehl ist für Sonderfälle reserviert und muss mit Bedacht verwendet werden, da er die Performance der Datenbank beeinträchtigen kann.

Beschreibung

Der Befehl **SET TABLE CACHE PRIORITY** setzt einen spezifischen Wert *Priorität* im Cache für Daten in *Tabellenname* für alle Prozesse in der aktuellen Sitzung. Ein Aufruf dieses Befehls ersetzt jeden zuvor gesetzten Wert für Priorität durch denselben Befehl in der Sitzung. Er sollte in der Datenbankmethode **On Startup** oder **On Server Startup** aufgerufen werden.

Hinweis: SET TABLE CACHE PRIORITY funktioniert nur im lokalen Modus (4D Server und 4D); er ist nicht im 4D remote Modus verwendbar.

Dieser Befehl verwaltet nur die Priorität für Daten in skalaren Feldern (vom Typ Datum, Zahl oder String). Die Priorität für binäre Datenfelder (vom Typ Blobs, Text, Bilder und Objekte) definieren Sie über den Befehl **SET BLOBS CACHE PRIORITY**.

In *Priorität* übergeben Sie eine der folgenden Konstanten unter dem Thema **Cache Verwaltung**:

Konstante	Typ	Wert	Kommentar
Cache priority high	Lange Ganzzahl	1000	
Cache priority low	Lange Ganzzahl	-900	
Cache priority normal	Lange Ganzzahl	0	Setzt den Wert für Priorität im Cache auf den Standardwert
Cache priority very high	Lange Ganzzahl	30000	
Cache priority very low	Lange Ganzzahl	-1	

Beispiel

In der **Datenbankmethode On Startup** eine hohe Priorität für skalare Daten in der Tabelle [Customer] setzen:

```
SET TABLE CACHE PRIORITY([Customer];Cache priority very high)
```

Collections

Über Collections

Überblick

Befehle im Kapitel **Collections** erstellen und arbeiten mit Collections.

Eine Collection ist eine Sammlung von Werten unterschiedlicher Typen (Text, Zahl, Objekt, Boolean, Collection oder Null). Zum Verwalten von Collection Variablen wird Objektnotation benötigt (siehe **Objektnotation verwenden**). Weitere Informationen zu 4D Collections finden Sie auf der Seite **Datentypen** unter **Collection**.

Um auf ein Element der Collection zuzugreifen, müssen Sie die Elementnummer in eckigen Klammern übergeben:

```
collectionName[expression]
```

Sie können jeden gültigen 4D Ausdruck übergeben, der in *Ausdruck* eine positive Ganzzahl zurückgibt. Beispiele:

```
myCollection[5]  //Zugriff auf das 6. Element der Collection
myCollection[$var]
```

Hinweis: Beachten Sie, dass Elemente in Collections ab 0 gezählt werden.

Über Objektnotation können Sie einem Element der Collection einen Wert zuweisen:

```
myCol[10]:="My new element"
```

Ist die Elementnummer höher als das letzte vorhandene Element der Collection, wird die Collection automatisch angepasst und alle dazwischenliegenden neue Elemente erhalten den Wert **null**:

```
C_COLLECTION(myCol)
myCol:=New collection("A";"B")
myCol[5]:="Z"
 //myCol[2]=null
 //myCol[3]=null
 //myCol[4]=null
```

Regular oder shared Collection

Sie können zwei Arten von Collections erstellen:

- **regular** (non-shared) Collections mit dem Befehl **New collection**.
  Diese Collections unterstützen zahlreiche Datentypen, inkl. Bilder und Zeiger. Sie lassen sich ohne eine spezifische Zugriffskontrolle bearbeiten.
- **shared** Collections mit dem Befehl **New shared collection**.
  Diese Collections lassen sich zwischen Prozessen teilen, inkl. preemptive Threads. Der Zugriff auf diese Collections wird über **Use...End use** Strukturen gesteuert. Weitere Informationen dazu finden Sie auf der Seite **Shared Objects und Shared Collections**.

Collection Methoden

Referenzen auf 4D Collection können *Member Funktionen* nutzen. Sie lassen sich über Objektnotation (siehe unter **Objektnotation verwenden**) mit folgender Syntax verwenden:

```
{$result:=}myCollection.memberFunction( {params} )
```

Beachten Sie, dass eine Member Funktion auch ohne definierte Parameter in runden Klammern stehen muss, sonst wird ein Syntaxfehler erzeugt.

Zum Beispiel:

```
$newCol:=$col.copy()  //tiefe Kopie von $col in $newCol
$col.push(10;100)  //in der Collection 10 und 100 hinzufügen
```

Einige Member Funktionen geben nach Änderung die ursprüngliche Collection zurück, so dass die Aufrufe in einer Sequenz ablaufen können:

```
$col:=New collection(5;20)
$col2:=$col.push(10;100).sort()  //$col2=[5,10,20,100]
```

Warnung: Beim Einsatz von Member Funktionen müssen Sie für Eigenschaften ECMA Script konforme Pfade verwenden. Sie können also nicht ".", "[ ]", oder Leerzeichen verwenden. Gemäß der Beschreibung im Abschnitt **Identifier für**

Objekteigenschaft werden Eigenschaftennamen wie *$o["My.special.property"]* zwar unterstützt, sie sind jedoch nicht mit Member Funktionen verwendbar:

```
$vmin:=$col.min("My.special.property") //undefiniert
$vmin:=$col.min(["My.special.property"]) //Fehler
```

Parameter EigenschaftsPfad

Einige Member Funktionen akzeptieren als Parameter einen *EigenschaftsPfad*. Dieser Parameter steht für:

- Name der *Objekteigenschaft*, z.B. "lastName"
- oder Pfad der Objekteigenschaft, z.B. eine Sequenz von Untereigenschaften, durch Punkte getrennt, z.B. "employee.children.firstName".

Deshalb werden, wenn ein Parameter *EigenschaftsPfad* erwartet wird, Eigenschaftennamen mit einem oder mehreren Punkten **nicht unterstützt**, da 4D den Pfad nicht korrekt analysieren kann.

Hinweis: Weitere Informationen dazu finden Sie im Abschnitt **Identifier für Objekteigenschaft**.

Typkonvertierung zwischen Collections und 4D Arrays

Beim Übertragen von Werten zwischen typisierten Arrays und nicht-typisierten Collections führt 4D je nach Wert und zugewiesenem Arraytyp automatische Konvertierungen durch. Nachfolgend sehen Sie die jeweilige Konvertierung bei Collections in Arrays und umgekehrt.

Collections in Arrays konvertieren

Die folgenden Konvertierungen gelten für Werte der Befehle:

- **OB GET ARRAY**
- **COLLECTION TO ARRAY**

Collection Elementtyp	null	boolean	Infinity	real	string	date	picture
ARRAY TEXT	""	"false" oder "true"	"Infinity"	Zahl mit Punkttrenner (falls erforderlich)	Text	Konvertierung von Datum in Text gemäß der Einstellung für Kompatibilität oder entsprechende Selektoren	"[object Object]"
ARRAY LONGINT	0	0 oder 1	Nicht definiertes Verhalten	gerundet gemäß Standard Rundungsregeln	0 wenn String nicht mit [0-9,+,-,e,.,x] beginnt, sonst Standardkonvertierung. Unterstützt hexa Notation Prefix 0x	0	0
ARRAY REAL	0	0 oder 1	INF	Zahl	dasselbe wie **ARRAY LONGINT**	0	0
ARRAY INTEGER	0	0 oder 1	0	gerundet gemäß Standard Rundungsregeln	dasselbe wie **ARRAY LONGINT**	0	0
ARRAY BOOLEAN	False	false oder true	true	true wenn #0	true wenn string#""	true wenn date#"00/00/00"	True
ARRAY OBJECT	Undefiniert	Undefiniert	Undefiniert	Undefiniert	Undefiniert	Undefiniert	Objekt Bild
ARRAY PICTURE	0 bytes	0 bytes	0 bytes	0 bytes	0 bytes	0 bytes	Bild
ARRAY DATE	00/00/00	00/00/00	00/00/00	00/00/00	00/00/00 oder Datum wenn Format konform mit ISO8601	Datum	00/00/00
ARRAY TIME	00:00:00	00:00:00	Nicht definiertes Verhalten	Anzahl Sekunden	Anzahl Sekunden	00:00:00	00:00:00

Arrays in Collections konvertieren

Folgende Konvertierungen gelten für Werte der Befehle:

- **OB SET ARRAY**
- **ARRAY TO COLLECTION**

	ARRAY TEXT	ARRAY LONGINT	ARRAY REAL	ARRAY INTEGER	ARRAY BOOLEAN	ARRAY OBJECT	ARRAY PICTURE	ARRAY DATE	ARRAY TIME
Collection Elementtypen	String	Zahl	Zahl	Zahl	Boolean	Objekt oder Null	Bild	String oder Datum gemäß der Einstellung für Kompatibilität oder entsprechende Selektoren	Anzahl Sekunden

collection.length

Parameter	Typ		Beschreibung
collection.length	Lange Ganzzahl	⮌	Anzahl Elemente in der Collection

Beschreibung

Die Eigenschaft **collection.length** gibt die Anzahl Elemente in der Collection zurück.

collection.length wird beim Erstellen der Collection initialisiert. Hinzufügen oder Entfernen von Elementen aktualisiert die Größe bei Bedarf. Diese Eigenschaft ist nur lesbar. Sie lässt sich nicht zum Setzen der Größe der Collection verwenden.

Beispiel

```
C_COLLECTION($col)  //$col.length initialisiert auf 0
$col:=New collection("one";"two";"three")  //$col.length aktualisiert auf 3
$col[4]:="five"  //$col.length aktualisiert auf 5
$vSize:=$col.remove(0;3).length  //$vSize=2
```

ARRAY TO COLLECTION

ARRAY TO COLLECTION (Collection ; Array {; EigenschaftenName}{; Array2 ; EigenschaftenName2 ; ... ; ArrayN ; EigenschaftenNameN})

Parameter	Typ		Beschreibung
Collection	Collection	⇐	Collection zum Empfangen der Daten des Array
Array	Array	⇒	Array zum Kopieren in die Collection; ist EigenschaftenName übergeben, Array zum Kopieren in die Werte von EigenschaftenName in der Collection
EigenschaftenName	Text	⇒	Name der Objekteigenschaft, deren Wert mit Array Elementen gefüllt werden soll.

Beschreibung

Der Befehl **ARRAY TO COLLECTION** kopiert ein oder mehrere *Array*(s) in die Elemente oder die Werte *EigenschaftenName* von *Collection*.

ARRAY TO COLLECTION arbeitet mit einer *Collection* mit Werten, oder mit einer *Collection* mit Objekten. Im zweiten Fall sind der/die Parameter *EigenschaftenName* zwingend.

- Ohne den Parameter *EigenschaftenName* kopiert der Befehl alle Elemente von *Array* in *Collection*. Ist *Collection* nicht leer, werden vorhandene Elemente ersetzt und neue Elemente hinzugefügt, wenn *Array* größer als die Länge von *Collection* ist. Anschließend ist die Länge von *Collection* identisch mit der Größe von *Array*.
- Übergeben Sie einen oder mehrere Parameter *EigenschaftenName*, erstellt oder ersetzt der Befehl Objekte als Elemente von *Collection*. Jedes Objekt wird mit einer Eigenschaft mit dem Namen, übergeben im Parameter *EigenschaftenName*, und dem Wert des entsprechenden Array Elements gefüllt. Ist *Collection* nicht leer, werden vorhandene Elemente ersetzt und neue Elemente hinzugefügt, wenn *Array* größer als die Collection ist. Anschließend entspricht die Länge von *Collection* dem größten *Array*.

Beispiel 1

Ein Text Array in eine Collection kopieren:

```
C_COLLECTION($colFruits)
$colFruits:=New collection
ARRAY TEXT($artFruits;4)
$artFruits{1}:="Orange"
$artFruits{2}:="Banane"
$artFruits{3}:="Apfel"
$artFruits{4}:="Traube"
ARRAY TO COLLECTION($colFruits;$artFruits)
 //$colFruits[0]="Orange"
 //$colFruits[1]="Banane"
 //...
```

Beispiel 2

Feldwerte in einer Collection von Objekten über Arrays kopieren:

```
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}
 //...
```

Beispiel 3

Ein Text Array in eine shared Collection kopieren:

```
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 (collection ; Array {; EigenschaftenName}{; Array2 ; EigenschaftenName2 ; ... ; ArrayN ; EigenschaftenNameN})

Parameter	Typ		Beschreibung
collection	Collection	➡	Collection zum Kopieren in Array(s)
Array	Array	⬅	Array zum Empfangen der Elemente der Collection; ist EigenschaftenName definiert, zum Empfangen der Werte von EigenschaftenName in der Collection
EigenschaftenName	Text	➡	Name der Objekteigenschaft, dessen Werte in Array kopiert werden sollen ("" für alle Elemente)

Beschreibung

Der Befehl **COLLECTION TO ARRAY** füllt ein oder mehrere *Array*(s) mit Elementen oder Werten von *EigenschaftenName* von *Collection* in *Array*(s).

COLLECTION TO ARRAY kann mit einer *Collection* mit Werten oder einer *Collection* mit Objekten arbeiten. Im zweiten Fall sind der/die Parameter *EigenschaftenName* zwingend.

- Ohne den Parameter *EigenschaftenName* kopiert der Befehl alle Elemente von *Collection* in *Array*. Anschließend ist die Größe von *Array* identisch mit der Länge von *Collection*.
- Übergeben Sie einen oder mehrere Parameter *EigenschaftenName*, muss *Collection* eine Collection von Objekten sein (andere Elemente werden ignoriert). In diesem Fall gibt jeder Parameter *EigenschaftenName* den Namen einer Eigenschaft innerhalb jedes Objekts der Collection an, deren Wert(e) in das entsprechende *Array* kopiert werden soll(en). Sie können beliebige Paare *EigenschaftenName* / *Array* und unterschiedliche Arraytypen übergeben. Anschließend ist jede Größe von *Array* identisch mit der Länge von *Collection*.

In allen Fällen konvertiert 4D die Elemente oder Werte der Collection gemäß dem Typ von *Array* (falls erforderlich). Weitere Informationen zu den Regeln beim Konvertieren finden Sie auf der Seite **Typkonvertierung zwischen Collections und 4D Arrays**.

Beispiel 1

Eine Collection von Strings in ein Text Array kopieren:

```
C_COLLECTION($fruits)
$fruits:=New collection("Orange";"Banane";"Apfel";"Traube")
ARRAY TEXT($artFruits;0)
COLLECTION TO ARRAY($fruits;$artFruits)
 //$artFruits{1}="Orange"
 //$artFruits{2}="Banane"
 //...
```

Beispiel 2

Unterschiedliche Eigenschaftenwerte aus einer Collection von Objekten in verschiedene Arrays kopieren:

```
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 ({EigenschaftsPfad}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
EigenschaftsPfad	Text	➡	Pfad Objekteigenschaft für die Berechnung
Funktionsergebnis	Undefined, Zahl	⮌	Arithmetisches Mittel (Durchschnitt) der Collection Werte

Beschreibung

Die Funktion **collection.average( )** gibt das arithmetische Mittel (Durchschnitt) von definierten Werten in der Collection Instanz zurück.

Bei der Berechnung werden nur numerische Elemente berücksichtigt (andere Elementtypen werden ignoriert).

Enthält die Collection Objekte, übergeben Sie den Parameter *EigenschaftsPfad*, damit die Objekteigenschaft berücksichtigt wird.

collection.average( ) gibt *Undefiniert* zurück, wenn

- die Collection leer ist
- die Collection keine numerischen Elemente enthält
- *EigenschaftsPfad* in der Collection nicht gefunden wird

Beispiel 1

```
C_COLLECTION($col)
$col:=New collection(10;20;"Monday";True;6)
$vAvg:=$col.average() //12
```

Beispiel 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 () -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Collection	Ursprüngliche Collection mit allen entfernten Elementen

Beschreibung

Die Funktion **collection.clear( )** entfernt alle Elemente aus der Collection Instanz und gibt eine leere Collection zurück.
Hinweis: Diese Funktion verändert die ursprüngliche Collection.

Beispiel

```
C_COLLECTION($col)
$col:=New collection(1;2;5)
$col.clear()
$vSize:=$col.length  //$vSize=0
```

collection.combine( )

collection.combine (col2 {; Index}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
col2	Collection	➡	Collection zum Kombinieren
Index	Lange Ganzzahl	➡	Position zum Einfügen von Elementen zum Kombinieren in Collection (Standard=Länge+1)
Funktionsergebnis	Collection	⮌	Ursprüngliche Collection mit kombiniertem Element(en)

Beschreibung

Die Funktion **collection.combine( )** fügt *col2* am Ende oder an der angegebenen Position *Index* in der Collection Instanz ein und gibt die bearbeitete Collection zurück. Im Gegensatz zur Funktion **collection.insert( )** fügt **collection.combine( )** jeden Wert von *col2* in die ursprüngliche Collection ein und nicht als einzelnes Collection Element.

Hinweis: Diese Funktion ändert die ursprüngliche Collection.

Standardmäßig werden *col2* Elemente am Ende der ursprünglichen Collection eingefügt. In *Index* können Sie die Position angeben, an der die *col2* Elemente in die Collection eingefügt werden sollen.
Warnung: Beachten Sie, dass Elemente der Collection ab 0 gezählt werden.

- Ist *index* > Länge der Collection, wird der Startindex auf die Länge der Collection gesetzt.
- Ist *index* < 0, wird er neu berechnet als *Index:=index+length* (wird als Versatz vom Ende der Collection gewertet).
- Ist der berechnete Wert negativ, wird *Index* auf 0 gesetzt

Beispiel

```
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 (Wert {; Wert2 ; ... ; WertN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl, Text, Objekt, Collection, Datum, Boolean	➡	Wert(e) zum Zusammenfügen. Ist Wert eine Collection, werden alle Elemente der Collection in der ursprünglichen Collection hinzugefügt.
Funktionsergebnis	Collection	↩	Ursprüngliche Collection mit hinzugefügtem Wert oder Werten

Beschreibung

Die Funktion **collection.concat( )** gibt eine neue Collection mit dem Inhalt des Parameters *Wert* zurück, der am Ende der ursprünglichen Collection angefügt wird.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

Ist *Wert* eine Collection, werden all seine Elemente als neue Elemente am Ende der ursprünglichen Collection hinzugefügt. Ist *Wert* keine Collection, wird sein Inhalt als neues Element hinzugefügt.

Beispiel

```
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,"Orange","Banana","Apple","Grape",{"Intruder":"Tomato"},6,7,8]
```

collection.copy( )

collection.copy ({ZeigerAuflösen}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ZeigerAuflösen	Lange Ganzzahl	➡	Wahr = Zeiger auflösen, Falsch oder nicht vergeben = Zeiger nicht auflösen
Funktionsergebnis	Collection	⮌	Tiefe Kopie der ursprünglichen Collection

Beschreibung

Die Funktion **collection.copy( )** gibt eine tiefe/vollständige Kopie der Collection Instanz zurück. *Tiefe Kopie (deep copy)* bedeutet, dass Objekte oder Collections innerhalb der ursprünglichen Collection dupliziert werden und keine Referenz mit der zurückgegebenen Collection teilen.

Hinweise:

- Diese Funktion ändert nicht die ursprüngliche Collection
- Bei Anwendung auf eine *shared Collection*, gibt **copy( )** eine reguläre (not shared) Collection zurück.

Enthält die ursprüngliche Collection Werte vom Typ Zeiger, enthält die Kopie standardmäßig ebenfalls Zeiger. Sie können jedoch beim Kopieren Zeiger auflösen, wenn Sie im Parameter *ZeigerAuflösen* die Konstante ck resolve pointers übergeben. In diesem Fall wird jeder vorhandene Zeiger in der Collection beim Kopieren bewertet und sein aufgelöster Wert verwendet.

Beispiel

```
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)  //zeigt "Hello World!"

$what:="You!"
$col3:=$col2.copy(ck resolve pointers)
ALERT($col3[0].alpha+" "+$col3[1].what)  //zeigt "Hello You!"
```

collection.count( )

collection.count ({EigenschaftsPfad}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
EigenschaftsPfad	Text	➡	Pfad Objekteigenschaft für die Berechnung
Funktionsergebnis	Zahl	⮌	Anzahl Elemente in der Collection

Beschreibung

Die Funktion **collection.count( )** gibt die Anzahl der nicht-null Elemente in der Collection zurück.
Enthält die Collection Objekte, können Sie den Parameter *EigenschaftsPfad* übergeben. Dann werden nur die Elemente mit dem *EigenschaftsPtad* berücksichtigt.

Beispiel

```
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 (Wert {; EigenschaftsPfad}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Text, Zahl, Boolean, Datum, Objekt, Collection	➡	Wert zum Zählen
EigenschaftsPfad	Text	➡	Objekt EigenschaftsPfad für Werte zum Zählen
Funktionsergebnis	Lange Ganzzahl	↩	Anzahl Vorkommen des Wertes

Beschreibung

Die Funktion **collection.countValues( )** gibt an, wie oft *Wert* in der Collection vorkommt.
Sie können in *Wert* folgendes übergeben:

- Skalaren Wert (Text, Zahl, Boolean, Datum)
- Referenz auf Objekt oder Collection

Damit ein Element gefunden wird, müssen *Wert* und Element vom gleichen Typ sein; die Funktion verwendet den Vergleichsoperator.
Mit dem optionalen Parameter *EigenschaftsPfad* können Sie Werte in einer Collection mit Objekten zählen. Dazu übergeben Sie in *EigenschaftsPfad* den Pfad der Eigenschaft, deren Werte sie zählen wollen.
Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

Beispiel 1

```
C_COLLECTION($col)
C_LONGINT($vCount)
$col:=New collection(1;2;5;5;5;3;6;4)
$vCount:=$col.countValues(5) // $vCount=3
```

Beispiel 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
```

Beispiel 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 ({EigenschaftsPfad}{;}{Option}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
EigenschaftsPfad	Text	➡	Pfad des Attributs zum Erhalten der abweichenden Werte
Option	Lange Ganzzahl	➡	ck diacritical: diakritische Bewertung, z.B. A # a
Funktionsergebnis	Collection	⮌	Neue Collection nur mit abweichenden Werten

Beschreibung

Die Funktion **collection.distinct( )** gibt eine Collection zurück, die nur die abweichenden (unterschiedlichen) Werte aus der ursprünglichen Collection enthält.

Hinweis: Diese Funktion verändert nicht die ursprüngliche Collection.

Die zurückgegebene Collection wird automatisch sortiert. Es werden keine **Null** Werte zurückgegeben.

Enthält die Collection Objekte, übergeben Sie den Parameter *EigenschaftsPfad*, damit die Objekteigenschaft zum Erhalten der abweichenden Werte berücksichtigt wird.

Standardmäßig wird eine nicht-diakritische Bewertung ausgeführt, d.h. es wird nicht zwischen Klein- und Großschreibung unterschieden. Soll die Bewertung dies berücksichtigen, übergeben Sie im Parameter *Option* die Konstante ck diacritical.

Beispiel

```
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 (Collection2 {; Option}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Collection2	Collection	➡	Collection zum Vergleichen
Option	Lange Ganzzahl	➡	ck diacritical: Diakritische Bewertung, z.B. A # a
Funktionsergebnis	Boolean	⮌	Wahr wenn die Collections indentisch sind, sonst falsch

Beschreibung

Die Funktion **collection.equal( )** vergleicht die Collection mit *Collection2* und gibt **wahr** zurück, wenn sie identisch sind (tiefer Vergleich).

Standardmäßig wird eine nicht-diakritische Bewertung ausgeführt, d.h. es wird nicht zwischen Klein- und Großschreibung unterschieden. Soll die Bewertung dies berücksichtigen, übergeben Sie im Parameter *Option* die Konstante ck diacritical.

Hinweis: Elemente mit **Null** Werten sind nicht dasselbe wie *undefinierte* Elemente.

Beispiel

```
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)  // falsch

$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)  // falsch

$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)  // wahr

$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)  //falsch
```

collection.every( )

collection.every ({StartAb ;} MethodenName {; param {; param2 ; ... ; paramN}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
StartAb	Lange Ganzzahl	➡	Index zum Starten des Tests
MethodenName	Text	➡	Name der Methode zum Aufrufen des Tests
param	Ausdruck	➡	Parameter zum Übergeben an MethodenName
Funktionsergebnis	Boolean	⮌	Wahr, wenn alle Elemente den Test erfolgreich durchlaufen haben

Beschreibung

Die Funktion **collection.every( )** gibt **wahr** zurück, wenn alle Elemente in der Collection den Test, angegeben in *MethodenName*, erfolgreich durchlaufen haben.

Standardmäßig testet **collection.every( )** die gesamte Collection. Optional können Sie in *StartAb* den Index des Elements übergeben, ab dem der Test starten soll.

- Ist *StartAb* >= Länge der Collection, wird ***falsch*** zurückgegeben, d.h. die Collection wird nicht getestet.
- Ist *StartAb* < 0, wird es als Versatz vom Ende der Collection gewertet (*StartAb*:=*StartAb*+Länge).
- Ist *StartAb* = 0, wird die gesamte Collection getestet (Standard).

In *MethodenName* übergeben Sie den Namen der Methode zum Bewerten der Collection Elemente, zusammen mit den Parametern in *param* (optional). *MethodenName* kann jeden Test mit oder ohne die Parameter durchführen. Diese Methode empfängt einen Parameter *Object* in $1 und muss *$1.result* für jedes Element, das den Test erfüllt, auf **wahr** setzen.

MethodenName empfängt folgende Parameter:

- in *$1.value*: Elementwert zum Bewerten

- in *$2*: *param*
- in *$N*...: param2...paramN

MethodenName setzt folgende Parameter:

- *$1.result* (boolean): **wahr**, bei erfolgreicher Bewertung des Elementwerts, sonst **falsch**.
- *$1.stop* (boolean, optional): **wahr**, um Aufruf der Methode zu stoppen. Der zurückgegebene Wert ist der letzte bewertete Wert.

In allen Fällen gilt: An der Stelle, wo die Funktion **collection.every( )** das erste Collection Element findet, das in $1.result **falsch** zurückgibt, stoppt sie das Aufrufen von *MethodenName* und gibt **falsch** zurück.

Beispiel 1

```
C_COLLECTION($c)
$c:=New collection
$c.push(5;3;1;4;6;2)
$b:=$c.every("NumberGreaterThan0")  //gibt wahr zurück
$c.push(-1)
$b:=$c.every("NumberGreaterThan0")  //gibt falsch zurück
```

Der Code der Methode ***NumberGreaterThan0*** lautet:

```
$1.result:=$1.value>0
```

Beispiel 2

Dieses Beispiel testet, ob alle Elemente einer Collection vom Typ Zahl sind:

```
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
```

Der Code der Methode ***TypeLookUp*** lautet:

```
C_OBJECT($1)
C_LONGINT($2)
If(Value type($1.value)=$2)
```

```
    $1.result:=True
End if
```

collection.extract( )

collection.extract (EigenschaftsPfad {; ZielPfad}{; EigenschaftsPfad2 ; ZielPfad2 ; ... ; EigenschaftsPfadN ; ZielPfadN}{; Option})
-> Funktionsergebnis

Parameter	Typ		Beschreibung
EigenschaftsPfad	Text	➡	Eigenschaftspfad Objekt, dessen Werte für eine neue Collection entnommen werden
ZielPfad	Text	➡	Ziel Eigenschaftspfad oder Eigenschaftsname
Option	Lange Ganzzahl	➡	ck keep null: enthält Null oder undefinierte Eigenschaften in der zurückgegebenen Collection (wird standandmäßig ignoriert). Ist ZielPfad übergeben, wird dieser Parameter ignoriert.
Funktionsergebnis	Collection	⮌	Neue Collection mit den entnommenen Werten

Beschreibung

Die Funktion **collection.extract( )** erstellt und gibt eine neue Collection mit den Werten *EigenschaftsPfad* aus der ursprünglichen Collection mit Objekten zurück.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

Der Inhalt der zurückgegebenen Collection richtet sich nach dem Parameter *ZielPfad*:

- Ohne den Parameter *ZielPfad* füllt **collection.extract( )** die neue Collection mit den Werten *EigenschaftsPfad* aus der ursprünglichen Collection.
  Standardmäßig werden Elemente, für die *EigenschaftsPfad* *Null* oder *Undefiniert* ist, in der resultierenden Collection ignoriert. Übergeben Sie die Konstante ck keep null im Parameter *Option*, werden diese Werte als **Null** Elemente in die zurückgegebene Collection übernommen.
- Ist ein oder mehrere Parameter *ZielPfad* übergeben, füllt **collection.extract( )** die neue Collection mit den Eigenschaften *EigenschaftsPfad* und jedes Element der neuen Collection ist ein Objekt mit *ZielPfad* Eigenschaften, gefüllt mit den entsprechenden Eigenschaften *ZielPfad*. *Null* Werte werden beibehalten (mit dieser Syntax wird der Parameter *Option* ignoriert).

Beispiel 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]
```

Beispiel 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 (Wert {; StartAb {; Ende}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl, Text, Collection, Objekt, Datum, Boolean	➡	Wert zum Füllen
StartAb	Lange Ganzzahl	➡	Startposition (enthalten)
Ende	Lange Ganzzahl	➡	Endposition (nicht enthalten)
Funktionsergebnis	Collection	➲	Ursprüngliche Collection mit eingefüllten Werten

Beschreibung

Die Funktion **collection.fill( )** füllt die Collection mit dem angegebenen *Wert*, optional von der Position *StartAb* bis *Ende*, und gibt die resultierende Collection zurück.

Hinweis: Diese Funktion ändert die ursprüngliche Collection.

- Ohne den Parameter *StartAb* wird *Wert* für alle Elemente der Collection gesetzt (*StartAb*=0). Ohne den Parameter *Ende* wird *Wert* bis zum letzten Element der Collection gesetzt (*Ende*=length).
- Ist *StartAb* < 0, wird es neu berechnet als *StartAb:=StartAb+length* (wird als Versatz vom Ende der Collection gewertet). Ist der berechnete Wert negativ, wird *StartAb* auf 0 gesetzt.
- Ist *Ende* < 0, wird es neu berechnet als *Ende*:=*Ende*+*length*.
- Ist *Ende* < *StartAb* (übergebende oder berechnete Werte), führt die Funktion nichts aus.

Beispiel

```
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 (MethodenName {; param}{; param2 ; ... ; paramN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
MethodenName	Text	➡	Name der Methode zum Filtern der Collection
param	Ausdruck	➡	Parameter für die Methode
Funktionsergebnis	Collection	↩	Neue Collection mit gefilterten Elementen (flache Kopie)

Beschreibung

Die Funktion **collection.filter( )** gibt eine neue Collection mit den Elementen der ursprünglichen Collection zurück, für die das Ergebnis von *MethodenName* **wahr** ist. Diese Funktion gibt eine flache Kopie zurück, d.h. Objekte oder Collections in beiden Collections nutzen dieselbe Referenz. Ist die ursprüngliche Collection eine *shared collection*, ist die zurückgegebene Collection ebenfalls eine *shared collection*.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

In *MethodenName* übergeben Sie den Namen der Methode zum Bewerten der Collection Elemente, zusammen mit den Parametern in *param* (optional). *MethodenName* kann das Filtern mit oder ohne Parameter durchführen und muss in *$1.result* **wahr** für jedes zutreffende Element zurückgeben und so in die neue Collection setzen.

MethodenName empfängt folgende Parameter:

- in *$1.value*: Elementwert zum Filtern
- in *$2*: *param*
- in *$N...*: param2...paramN

MethodenName setzt folgende Parameter:

- *$1.result* (boolean): **wahr**, wenn der Elementwert zur Filterbedingung passt und behalten werden soll.
- *$1.stop* (boolean, optional): **wahr**, um Aufruf der Methode zu stoppen. Der zurückgegebene Wert ist der letzte bewertete Wert.

Beispiel 1

Die Collection von Textelementen mit einer Länge kleiner als 6 erhalten:

```
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"]
```

Der Code für die Methode ***LengthLessThan*** lautet:

```
C_OBJECT($1)
C_LONGINT($2)
If(Value type($1.value)=Is text)
   $1.result:=(Length($1.value))<$2
End if
```

Beispiel 2

Elemente nach ihrem Wertetyp filtern:

```
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}]
```

Der Code für die Methode ***TypeLookUp*** lautet:

```
C_OBJECT($1)
C_LONGINT($2)
If(OB Get type($1;"value")=$2)
   $1.result:=True
End if
```

collection.find( )

collection.find ({StartAb ;} MethodenName {; param {; param2 ; ... ; paramN}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
StartAb	Lange Ganzzahl	➡	Index zum Starten der Suche
MethodenName	Text	➡	Name der Methode zum Aufrufen des Suchergebnis
param	Ausdruck	➡	Parameter zum Übergeben an MethodenName
Funktionsergebnis		↩	Erster gefundener Wert, oder -1 wenn nicht gefunden

Beschreibung

Die Funktion **collection.find( )** gibt den ersten Wert in der Collection zurück, für den der auf jedes Element angewandte *MethodenName* **wahr** zurückgibt.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

collection.find( ) sucht standardmäßig in der gesamten Collection. Optional können Sie in *StartAb* den Index des Elements übergeben, bei dem die Suche starten soll.

- Ist *StartAb* >= Länge der Collection, wird -1 zurückgegeben, d.h. die Collection wird nicht durchsucht.
- Ist *StartAb* < 0, wird es als Versatz vom Ende der Collection gewertet (*StartAb*:=*StartAb*+Länge).
  Hinweis: Auch wenn *StartAb* negativ ist, wird die Collection weiterhin von links nach rechts durchsucht.
- Ist *StartAb* = 0, wird die gesamte Collection durchsucht (Standard).

In *MethodenName* übergeben Sie den Namen der Methode zum Bewerten der Collection Elemente, zusammen mit den Parametern in *param* (optional). *MethodenName* kann den Suchlauf mit oder ohne die Parameter durchführen. Diese Methode empfängt einen Parameter *Object* in $1 und muss *$1.result* für das erste zutreffende Element auf **wahr** setzen.

MethodenName empfängt folgende Parameter:

- in *$1.value*: Elementwert zum Bewerten
- in *$2*: *param*
- in *$N*...: param2...paramN

MethodenName setzt folgende Parameter:

- *$1.result* (boolean): **wahr**, wenn der Elementwert zur Suchbedingung passt.
- *$1.stop* (boolean, optional): **wahr**, um Aufruf der Methode zu stoppen. Der zurückgegebene Wert ist der letzte bewertete Wert.

Beispiel 1

Das erste Element mit einer Länge kleiner als 5 erhalten:

```
C_COLLECTION($col)
$col:=New collection("hello";"world";4;"red horse";"tim";"san jose")
$value:=$col.find("LengthLessThan";5)  //$value="tim"
```

Der Code der Methode ***LengthLessThan*** lautet:

```
C_OBJECT($1)
C_LONGINT($2)
If(Value type($1.value)=Is text)
   $1.result:=(Length($1.value))<$2
End if
```

Beispiel 2

Einen Stadtnamen in der Collection finden:

```
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}
```

Der Code der Methode ***FindCity*** lautet:

```
C_OBJECT($1)
C_TEXT($2)
$1.result:=$1.value.name=$2
```

collection.findIndex( )

collection.findIndex ({StartAb ;} MethodenName {; param {; param2 ; ... ; paramN}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
StartAb	Lange Ganzzahl	➡	Index zum Starten der Suche
MethodenName	Text	➡	Name der Methode zum Aufrufen des Suchergebnis
param	Ausdruck	➡	Parameter zum Übergeben an MethodenName
Funktionsergebnis	Lange Ganzzahl	↩	Index des ersten gefundenen Werts, oder -1, wenn nicht gefunden

Beschreibung

Die Methode **collection.findIndex( )** gibt den Index des ersten Werts in der Collection zurück, für den der auf jedes Element angewandte *MethodenName* **wahr** zurückgibt. Wurde kein Element der Collection mit wahr bewertet, gibt die Methode -1 zurück.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

collection.findIndex( ) sucht standardmäßig in der gesamten Collection. Optional können Sie in *StartAb* den Index des Elements übergeben, bei dem die Suche starten soll.

- Ist *StartAb* >= Länge der Collection, wird -1 zurückgegeben, d.h. die Collection wird nicht durchsucht.
- Ist *StartAb* < 0, wird es als Versatz vom Ende der Collection gewertet (*StartAb*:=*StartAb*+Länge).
  Hinweis: Auch wenn *StartAb* negativ ist, wird die Collection weiterhin von links nach rechts durchsucht.
- Ist *StartAb* = 0, wird die gesamte Collection durchsucht (Standard).

In *MethodenName* übergeben Sie den Namen der Methode zum Bewerten der Collection Elemente, zusammen mit den Parametern in *param* (optional). *MethodenName* kann den Suchlauf mit oder ohne die Parameter durchführen. Diese Methode empfängt einen Parameter *Object* in $1 und muss *$1.result* für das erste zutreffende Element auf **wahr** setzen.

MethodenName empfängt folgende Parameter:

- in *$1.value*: Elementwert zum Bewerten
- in *$2*: *param*
- in *$N*...: param2...paramN

MethodenName setzt folgende Parameter:

- *$1.result* (boolean): **wahr**, wenn der Elementwert zur Suchbedingung passt.
- *$1.stop* (boolean, optional): **wahr**, um Aufruf der Methode zu stoppen. Der zurückgegebene Wert ist der letzte bewertete Wert.

Beispiel

Die Position des ersten Stadtnamens in der Collection finden:

```
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
```

Der Code der Methode ***FindCity*** lautet:

```
C_OBJECT($1)
C_TEXT($2)
$1.result:=$1.value.name=$2
```

collection.indexOf( )

collection.indexOf (zuSuchen {; StartAb}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
zuSuchen	Ausdruck	➡	Element, nach dem in der Collection gesucht wird
StartAb	Lange Ganzzahl	➡	Index, bei dem die Suche startet
Funktionsergebnis	Lange Ganzzahl	⮌	Index des ersten Auftretens von ZuSuchen in der Collection, -1 wenn nicht gefunden

Beschreibung

Die Funktion **collection.indexOf( )** sucht in den Collection Elementen nach dem Ausdruck *zuSuchen* und gibt den Index des ersten Auftretens zurück, oder -1, wenn dieser nicht gefunden wurde.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

In *zuSuchen* übergeben Sie den gesuchten Ausdruck in der Collection. Sie können folgendes übergeben:

- Skalaren Wert (Text, Zahl, Boolean, Datum)
- Wert Null
- Referenz auf ein Objekt oder eine Collection

zuSuchen muss exakt zum gesuchten Element passen. Es gelten dieselben Regeln wie für den Gleichheitsoperator (siehe **Vergleichsoperatoren**).

Optional können Sie in *StartAb* den Collection Index übergeben, bei dem die Suche starten soll.

- Ist *StartAb* >= Länge der Collection, wird -1 zurückgegeben, d.h. die Collection wird nicht durchsucht.
- Ist *StartAb* < 0, wird es als Versatz vom Ende der Collection gewertet (*StartAb*:=*StartAb*+Länge).
  Hinweis: Auch wenn *StartAb* negativ ist, wird die Collection weiterhin von links nach rechts durchsucht.
- Ist *StartAb* = 0, wird die gesamte Collection durchsucht (Standard).

Beispiel

```
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 (SuchString {; Wert}{; Wert2 ; ... ; WertN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
SuchString	Text	⇒	Suchargument
Wert	Mixed	⇒	Wert(e) zum Vergleichen bei Platzhalter(n)
Funktionsergebnis	Collection	⇐	Element Index/Indices, die zu SuchArgument in der Collection passen

Beschreibung

Die Methode **collection.indices( )** arbeitet genauso wie die Methode **collection.query( )**, gibt jedoch in der ursprünglichen Collection **Indices** der gefundenen Elemente zurück, die zu den Suchbedingungen *SuchString* passen, und nicht die Elemente selbst. Indices werden in aufsteigender Reihenfolge zurückgegeben.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

Der Parameter SuchString verwendet folgende Syntax:

```
propertyPath comparator value {logicalOperator propertyPath comparator value}
```

Weitere Informationen zu den Parametern *queryString* und *value* finden Sie unter der Methode **dataClass.query( )**.

Beispiel

```
C_COLLECTION($c)
$c:=New collection
$c.push(New object("name";"Cleveland";"zc";35049))
$c.pushNew 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 (Index ; Element) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Index	Lange Ganzzahl	➡	Stelle zum Einfügen des Elements
Element	Ausdruck	➡	Element zum Einfügen in die Collection
Funktionsergebnis	Collection	⮌	Ursprüngliche Collection mit eingefügtem Element

Beschreibung

Die Funktion **collection.insert( )** fügt *Element* an der angegebenen Position *Index* in der Collection Instanz ein und gibt die bearbeitete Collection zurück.

Hinweis: Diese Funktion ändert die ursprüngliche Collection.

In *Index* übergeben Sie die Position, an der das Element in die Collection eingefügt werden soll.

Warnung: Beachten Sie, dass Elemente der Collection ab 0 gezählt werden.

- Ist *index* > Länge der Collection, wird der Startindex auf die Länge der Collection gesetzt.
- Ist *index* < 0, wird er neu berechnet als *Index:=index+length* (wird als Versatz vom Ende der Collection gewertet).
- Ist der berechnete Wert negativ, wird *Index* auf 0 gesetzt.

Die Collection akzeptiert zum Einfügen jede Art von *Element*, selbst eine andere Collection ist möglich.

Beispiel

```
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 (Trenner {; Option}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Trenner	Text	➡	Trenner zwischen den einzelnen Elementen
Option	Lange Ganzzahl	➡	ck ignore or empty: Null oder leere Strings im Ergebnis ignorieren
Funktionsergebnis	Text	↩	String mit allen Elementen der Collection, getrennt durch definierten Trenner

Beschreibung

Die Funktion **collection.join( )** konvertiert alle Elemente der Collection in Strings und fügt sie aneinander, getrennt durch den in *Trenner* angegebenen String. Die Funktion gibt den Ergebnisstring zurück.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

Standardmäßig werden Null oder leere Elemente der Collection im Ergebnis zurückgegeben. Übergeben Sie im Parameter *Option* die Konstante ck ignore null or empty, um sie aus dem Ergebnisstring zu entfernen.

Beispiel

```
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 (zuSuchen {; StartAb}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
zuSuchen	Ausdruck	➡	Element, nach dem in der Collection gesucht wird
StartAb	Lange Ganzzahl	➡	Index, bei dem die Suche startet
Funktionsergebnis	Lange Ganzzahl	⮌	Index des letzten Auftretens von zuSuchen in der Collection, -1 wenn nicht gefunden

Beschreibung

Die Funktion **collection.lastIndexOf( )** sucht in den Collection Elementen nach dem Ausdruck *zuSuchen* und gibt den Index des letzten Auftretens zurück, oder -1, wenn dieser nicht gefunden wurde.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

In *zuSuchen* übergeben Sie den gesuchten Ausdruck in der Collection. Sie können folgendes übergeben:

- Skalaren Wert (Text, Zahl, Boolean, Datum)
- Wert Null
- Referenz auf ein Objekt oder eine Collection

zuSuchen muss exakt zum gesuchten Element passen. Es gelten dieselben Regeln wie für den Gleichheitsoperator **Vergleichsoperatoren**.

Optional können Sie in *StartAb* den Collection Index übergeben, bei dem die Rückwärtssuche starten soll.

- Ist *StartAb* >= Länge der Collection minus eins (coll.length-1), wird die gesamte Collection durchsucht (Standard).
- Ist *StartAb* < 0, wird es neu berechnet als *StartAb*:=*StartAb*+Länge (es wird als Versatz vom Ende der Collection gewertet). Ist der berechnete Wert negativ, wird -1 zurückgegeben, d.h. die Collection wird nicht durchsucht.
  Hinweis: Auch wenn *StartAb* negativ ist, wird die Collection weiterhin von links nach rechts durchsucht.
- Ist *StartAb* = 0, wird die Collection nicht durchsucht (Standard).

Beispiel

```
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") // gibt 12 zurück
$pos2:=$col.lastIndexOf("e";6) //gibt 4 zurück
$pos3:=$col.lastIndexOf("e";15) // gibt 12 zurück
$pos4:=$col.lastIndexOf("e";-2) // gibt 10 zurück
$pos5:=$col.lastIndexOf("x") //gibt -1 zurück
```

collection.map( )

collection.map (MethodenName ; param {; param2 ; ... ; paramN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
MethodenName	Text	➡	Name der Methode zum Umwandeln der Collection Elemente
param	Ausdruck	➡	Parameter für die Methode
Funktionsergebnis	Collection	⮌	Collection der umgewandelten Werte

Beschreibung

Die Funktion **collection.map( )** erstellt eine neue Collection, basierend auf dem Ergebnis der aufgerufenen Funktion *MethodenName* für jedes Element der ursprünglichen Collection. Optional können Sie über *param* Parameter an *MethodenName* übergeben. **collection.map( )** gibt immer eine Collection mit derselben Größe wie die ursprüngliche Collection zurück.
MethodenName empfängt folgende Parameter:

- In *$1.value* (beliebiger Typ): Abzubildender Elementwert
- In *$2* (beliebiger Typ): *param*
- In *$N...* (beliebiger Typ): *param2...paramN*

MethodenName setzt folgende Parameter:

- *$1.result* (beliebiger Typ): Neuer umgewandelter Wert zum Hinzufügen in der resultierenden Collection
- *$1.stop* (Boolean): **wahr**, um Aufruf der Methode zu stoppen. Der zurückgegebene Wert ist der letzte berechnete Wert.

Beispiel

```
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]
```

Hier ist die Methode ***Percentage***:

```
C_OBJECT($1)
C_REAL($2)
$1.result:=Round(($1.value/$2)*100;2)
```

collection.max( )

collection.max ({EigenschaftsPfad}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
EigenschaftsPfad	Text	→	Pfad Objekteigenschaft für die Bewertung
Funktionsergebnis	Boolean, Collection, Datum, Objekt, Text, Zahl	←	Maximumwert in der Collection

Beschreibung

Die Funktion **collection.max( )** gibt das Element mit dem höchsten Wert in der Collection zurück (das letzte Element der Collection, als ob es mit der Funktion **collection.sort( )** in absteigender Reihenfolge sortiert wäre.)

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

Enthält die Collection verschiedene Wertetypen, gibt die Funktion **max( )** den Maximumwert innerhalb des letzten Elementtyps in der Reihenfolge der Typenliste zurück. Weitere Informationen dazu finden Sie unter **collection.sort( )**.

Enthält die Collection Objekte, übergeben Sie den Parameter *EigenschaftsPfad* für die Objekteigenschaft, deren Maximumwert Sie erhalten wollen.

Ist die Collection leer, gibt **collection.max( )** *Undefiniert* zurück.

Beispiel

```
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 ({EigenschaftsPfad}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
EigenschaftsPfad	Text	➡	Pfad Objekteigenschaft für die Bewertung
Funktionsergebnis	Boolean, Collection, Datum, Objekt, Text, Zahl	➲	Minimumwert in der Collection

Beschreibung

Die Funktion **collection.min( )** gibt das Element mit dem kleinsten Wert in der Collection zurück (das erste Element der Collection, als ob es mit der Funktion **collection.sort( )** in aufsteigender Reihenfolge sortiert wäre.)

HInweis: Diese Funktion ändert nicht die ursprüngliche Collection.

Enthält die Collection verschiedene Wertetypen, gibt die Funktion **min( )** den Minimumwert innerhalb des ersten Elementtyps in der Reihenfolge der Typenliste zurück. Weitere Informationen dazu finden Sie unter **collection.sort( )**.

Enthält die Collection Objekte, übergeben Sie den Parameter *EigenschaftsPfad* für die Objekteigenschaft, deren Minimumwert Sie erhalten wollen.

Ist die Collection leer, gibt **collection.min( )** *Undefiniert* zurück.

Beispiel

```
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 ({Kriterium}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Kriterium	Text, Collection, Lange Ganzzahl	➡	Text: Eigenschaftspfad(e) zum Sortieren der Collection Collection: Collection der Kriterium Objekte Lange Ganzzahl: ck ascending oder ck descending (skalare Werte)
Funktionsergebnis	Collection	➲	Sortierte Kopie der Collection (flache Kopie)

Beschreibung

Die Funktion **collection.orderBy( )** sortiert die Elemente der Collection und gibt eine neue sortierte Collection zurück.

Diese Funktion gibt eine flache Kopie (shallow copy) zurück, d.h. Objekte oder Collections der beiden Collections nutzen dieselbe Referenz. Ist die ursprüngliche Collection eine *shared collection*, ist die zurückgegebene Collection ebenfalls eine *shared collection*.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

Ohne den Parameter *Kriterium* sortiert die Funktion nur skalare Werte der Collection in aufsteigender Reihenfolge, d.h. andere Elementtypen wie Objekte oder Collections werden unsortiert zurückgegeben. Für eine andere Sortierung können Sie im Parameter *Kriterium* entweder die Konstante ck ascending oder ck descending übergeben (siehe unten).

Über den Parameter *Kriterium* können Sie auch definieren, wie Collection Elemente sortiert werden. Es gibt drei Syntaxarten:

- *Kriterium* ist vom **Typ Text** (Formel): "propertyPath1 {desc or asc}, propertyPath2 {desc or asc},..." (standardmäßig: *asc*)
  In diesem Fall enthält *Kriterium* eine Formel mit 1 bis x Eigenschaftspfaden und (optional) Sortierreihenfolgen, getrennt durch Kommas. Die Reihenfolge, in der die Eigenschaften übergeben sind, bestimmt die Priorität beim Sortieren der Collection Elemente.
  Eigenschaften werden standardmäßig in aufsteigender Reihenfolge sortiert. Sie können die Sortierreihenfolge im Eigenschaftspfad, getrennt durch ein Leerzeichen setzen: "asc" für aufsteigende Reihenfolge oder "desc" für absteigende Reihenfolge.

- *Kriterium* ist vom **Typ Collection:** Hier enthält jedes Element der Collection ein Objekt mit folgender Struktur:
  {"propertyPath": string,
  "descending": boolean}
  Eigenschaften werden standardmäßig in aufsteigender Reihenfolge sortiert. ("absteigend" ist falsch).
  In *Kriterium* können Sie soviel Objekte, wie erforderlich sind, hinzufügen.

- *Kriterium* ist vom **Typ Lange Ganzzahl**: Hier können Sie eine Konstante unter dem Thema **Objekte und Collections** übergeben:

Konstante	Typ	Wert	Kommentar
ck ascending	Lange Ganzzahl	0	Elemente in aufsteigender Reihenfolge sortieren (Standard)
ck descending	Lange Ganzzahl	1	Elemente in absteigender Reihenfolge sortieren

  Diese Syntax sortiert nur skalare Werte in der Collection, d.h. andere Elementtypen wie Objekte oder Collections werden unsortiert zurückgegeben.

Enthält die Collection Elemente verschiedener Typen, werden sie zuerst nach Typ gruppiert und anschließend sortiert. Für die Typen gilt folgende Reihenfolge:

1. Null
2. Boolean
3. Strings
4. Zahl
5. Objekt
6. Collection
7. Datum

Beispiel 1

Eine Collection mit Zahlen in aufsteigender und absteigender Reihenfolge sortieren:

```
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)
```

Beispiel 2

Eine Collection mit Objekten nach einer Textformel mit Eigenschaftennamen sortieren:

```
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")
```

Eine Collection mit Objekten über einen Eigenschaftspfad sortieren:

```
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")
```

Beispiel 3

Eine Collection mit Objekten über eine Collection mit Kriterium Objekten sortieren:

```
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)
```

Mit einem Eigenschaftspfad sortieren:

```
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 (MethodenName {; extraParam}{; extraParam2 ; ... ; extraParamN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
MethodenName	Text	➡	Name der Methode für die Sortierreihenfolge
extraParam	Ausdruck	➡	Parameter für die Methode
Funktionsergebnis	Collection	⮌	Sortierte Kopie der Collection (flache Kopie)

Beschreibung

Die Funktion **collection.orderByMethod( )** sortiert die Elemente der Collection nach der Methode *MethodenName* und gibt eine neue sortierte Collection zurück.

Diese Funktion gibt eine flache Kopie *(shallow copy)* zurück, d.h. Objekte oder Collections der beiden Collections nutzen dieselbe Referenz. Ist die ursprüngliche Collection eine *shared collection*, ist die zurückgegebene Collection ebenfalls eine *shared collection*.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

In *MethodenName* übergeben Sie eine Vergleichsmethode, die zwei Werte vergleicht und **wahr** in *$1.result* zurückgibt, wenn der erste Wert kleiner als der zweite ist. Sie können bei Bedarf weitere Parameter übergeben:

- *MethodenName* empfängt folgende Parameter:
  - $1 (object), es gilt:
    - *$1.value* (beliebiger Typ): erster Elementwert zum Vergleichen
    - *$1.value2* (beliebiger Typ): zweiter Elementwert zum Vergleichen
  - $2...$N (beliebiger Typ): extra Parameter
- *MethodenName* setzt folgende Parameter:
  - *$1.result* (boolean): **wahr** wenn *$1.value* < *$1.value2*, sonst false

Beispiel 1

Eine Collection mit Strings nicht in alphabetischer, sondern in numerischer Reihenfolge sortieren:

```
C_COLLECTION($c)
$c:=New collection
$c.push("33";"4";"1111";"222")
$c2:=$c.orderBy() //$c2=["1111","222","33","4"], alphabetische Reihenfolge
$c3:=$c.orderByMethod("NumAscending") // $c3=["4","33","222","1111"]
```

Der Code für ***NumAscending*** lautet:

```
$1.result:=Num($1.value)<Num($1.value2)
```

Beispiel 2

Eine Collection mit Strings nach Wortlänge sortieren:

```
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]
```

Der Code für ***WordLength*** lautet:

```
$1.result:=Length(String($1.value))>Length(String($1.value2))
```

collection.pop( )

collection.pop () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Ausdruck		Letztes Element der Collection

Beschreibung

Die Funktion **collection.pop( )** entfernt das letzte Element aus der Collection und gibt es als Funktionsergebnis zurück.
Hinweis: Diese Funktion ändert die ursprüngliche Collection.
Ist die Collection leer, gibt **collection.pop( )** ***Undefiniert*** zurück.

Beispiel

collection.pop( ) lässt sich zusammen mit **collection.push( )** für ein Feature "first-in, last-out stack" verwenden:

```
C_COLLECTION($stack)
$stack:=New collection  //$stack=[]
$stack.push(1;2)  //$stack=[1,2]
$stack.pop()  //$stack=[1] gibt 2 zurück
$stack.push(New collection(4;5))  //$stack=[[1,[4,5]]
$stack.pop()  //$stack=[1] gibt [4,5] zurück
$stack.pop()  //$stack=[] gibt 1 zurück
```

collection.push( )

collection.push (Element {; Element2 ; ... ; ElementN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Element	Ausdruck	➡	Element(e) zum Hinzufügen in der Collection
Funktionsergebnis	Collection	⮌	Ursprüngliche Collection mit hinzugefügtem Element(en)

Beschreibung

Die Funktion **collection.push( )** fügt ein oder mehrere *Element*(e) am Ende der Collection Instanz an und gibt die bearbeitete Collection zurück.

Hinweis: Diese Funktion ändert die ursprüngliche Collection.

Beispiel 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}
```

Beispiel 2

Die resultierende Collection sortieren:

```
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 (SuchString {; Wert}{; Wert2 ; ... ; WertN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
SuchString	Text	➡	Suchkriterium
Wert	Mixed	➡	Wert(e) zum Vergleichen bei Platzhalter(n)
Funktionsergebnis	Collection	↩	Zum Suchargument passende Element(e) in der Collection

Beschreibung

Die Funktion **collection.query( )** gibt alle Elemente einer Collection zurück, die zu den Suchbedingungen in *SuchString* passen. Ist die ursprüngliche Collection eine *shared collection*, ist die zurückgegebene Collection ebenfalls eine *shared collection*.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

Der Parameter *SuchString* verwendet folgende Syntax:

```
propertyPath comparator value {logicalOperator propertyPath comparator value}
```

Weitere Informationen zu den Parametern *SuchString* und *Wert* finden Sie unter der Methode **dataClass.query( )**.

Beispiel 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}]
```

Beispiel 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!))
```

Dieses Beispiel gibt Personen zurück, deren Name "in" enthält:

```
$col:=$c.query("name = :1";"@in@")
 //$col=[{name:Winch...},{name:Sterling...}]
```

Dieses Beispiel gibt Personen zurück, deren Name nicht mit dem String aus einer Variablen beginnt (beispielsweise vom Benutzer eingegeben):

```
$col:=$c.query("name # :1";$aString+"@")
 //if $astring="W"
 //$col=[{name:Smith...},{name:Sterling...},{name:Mark...}]
```

Dieses Beispiel gibt Personen zurück, deren Alter unbekannt ist (Eigenschaft ist auf null oder undefiniert gesetzt):

```
$col:=$c.query("age=null") //Platzhalter mit "null" nicht erlaubt
 //$col=[{name:Wesson...},{name:Sterling...},{name:Mark...}]
```

Dieses Beispiel gibt Personen zurück, die vor mehr als 90 Tagen eingestellt wurden:

```
$col:=$c.query("dateHired < :1";(Current date-90))
 //$col=[{name:Smith...},{name:Sterling...},{name:Mark...}] wenn heute 10/01/2018 ist
```

Hinweis: Für das letzte Beispiel muss in den Datenbankeigenschaften die Option **Verwende Datumstyp statt ISO Datumsformat in Objekten** markiert sein (siehe **Seite Kompatibilität**).

collection.reduce( )

collection.reduce (MethodenName {; InitWert}{; param}{; param2 ; ... ; paramN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
MethodenName	Text	➡	Name der Methode zum Bearbeiten von Collection Elementen
InitWert	Text, Zahl, Objekt, Collection, Datum, Boolean	➡	Wert zum Verwenden als erstes Argument für den ersten Aufruf von MethodenName
param	Ausdruck	➡	Parameter für MethodenName
Funktionsergebnis	Boolean, Collection, Datum, Objekt, Text, Zahl	↩	Ergebnis des Akkumulatorwerts

Beschreibung

Die Funktion **collection.reduce( )** wendet die Callback Funktion *MethodenName* auf einen Akkumulator und jedes Element in der Collection an (von links nach rechts), um ihn auf einen einzigen Wert zu reduzieren.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

In *MethodenName* übergeben Sie den Namen der Methode zum Bewerten der Collection Elemente, zusammen mit den Parametern in *param* (optional). *MethodenName* nimmt jedes Collection Element und führt jede gewünschte Operation durch, um das Ergebnis in *$1.accumulator* zu akkumulieren, das in *$1.value* zurückgegeben wird.

In *initWert* können Sie den Wert zum Initialisieren des Akkumulators übergeben. Ohne diesen Parameter startet *$1.accumulator* mit *Undefiniert*.

MethodenName empfängt folgende Parameter:

- in *$1.value*: Elementwert zum Bearbeiten
- in *$2*: *param*
- in *$N*...: param2...paramN

MethodenName setzt folgende Parameter:

- *$1.accumulator*: Wert, der durch die Funktion geändert werden soll und durch *initWert* initialisiert wird.
- *$1.stop* (boolean, optional): **wahr**, um Aufruf der Methode zu stoppen. Der zurückgegebene Wert ist der letzte bewertete Wert.

Beispiel 1

```
C_COLLECTION($c)
$c:=New collection(5;3;5;1;3;4;4;6;2;2)
$r:=$c.reduce("Multiply";1) //gibt 86400 zurück
```

Der Code für die Methode ***Multiply*** lautet:

```
If(Value type($1.value)=Is real)
   $1.accumulator:=$1.accumulator*$1.value
End if
```

Beispiel 2

Mehrere Collection Elemente auf ein einziges Element reduzieren:

```
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]
```

Der Code für die Methode ***Flatten*** lautet:

```
If($1.accumulator=Null)
   $1.accumulator:=New collection
End if
$1.accumulator.combine($1.value)
```

collection.remove( )

collection.remove (Index {; Wieviele}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Index	Lange Ganzzahl	➡	Element, bei dem das Entfernen startet
Wieviele	Lange Ganzzahl	➡	Anzahl Elemente zum Entfernen, ohne Angabe 1 Element
Funktionsergebnis	Collection	⮌	Collection ohne entfernte Elemente

Beschreibung

Die Funktion **collection.remove( )** entfernt ein oder mehrere Elemente aus der angegebenen Position *Index* in der Collection und gibt die bearbeitete
Collection zurück.

Hinweis: Diese Funktion verändert die ursprüngliche Collection.

In *Index* übergeben Sie die Position, an der Elemente aus der Collection entfernt werden sollen. **Warnung:** Beachten Sie, dass Elemente der Collection ab 0 gezählt werden. Ist *Index* größer als die Länge der Collection, wird der aktuelle Startindex auf die Länge der Collection gesetzt.

- Ist *Index* < 0, wird dies neu berechnet als *Index:=index+length* (es wird als Versatz vom Ende der Collection gewertet).
- Ist der berechnete Wert < 0, wird *Index* auf 0 gesetzt.
- Ist der berechnete Wert > Länge der Collection, wird *Index* auf die Länge gesetzt.

In *Wieviele* übergeben Sie die Anzahl der zu entfernenden Elemente aus *Index*. Ist *Wieviele* nicht angegeben, wird ein Element entfernt.

Versuchen Sie, aus einer leeren Collection Elemente zu entfernen, führt die Funktion nichts aus. Es wird kein Fehler generiert.

Beispiel

```
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 (Größe {; StandardWert}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Größe	Lange Ganzzahl	➡	Neue Größe der Collection
StandardWert	Zahl, Text, Objekt, Collection, Datum, Boolean	➡	Standardwert zum Füllen der neuen Elemente
Funktionsergebnis	Collection	↩	Angepasste Collection

Beschreibung

Die Funktion **collection.resize( )** setzt Collection length auf die angegebene neue *Größe* und gibt die angepasste Collection zurück.

Hinweis: Diese Funktion ändert die ursprüngliche Collection.

- Ist *Größe* < Collection length, werden überzählige Elemente aus der Collection entfernt.
- Ist *Größe* > Collection length, wird Collection length auf *Größe* erweitert.
  Neue Elemente werden standardmäßig mit **Null** Werten gefüllt. Im Parameter *StandardWert* können Sie den Wert zum Füllen des hinzugefügten Elements angeben.

Beispiel

```
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 () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Collection		Umgekehrte Kopie der Collection

Beschreibung

Die Funktion **collection.reverse( )** gibt eine tiefe (vollständige) Kopie der Collection mit all ihren Elemente in umgekehrter Reihenfolge zurück. Ist die ursprüngliche Collection eine shared collection, ist die zurückgegebene Collection ebenfalls eine shared collection.

Hinweis: Diese Funktion ändert nicht die ursprüngliche Collection.

Beispiel

```
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 () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Collection, Datum, Objekt, Text, Zahl		Erstes Element der Collection

Beschreibung

Die Funktion **collection.shift( )** entfernt das erste Element der Collection und gibt dieses als Funktionsergebnis zurück.
Hinweis: Diese Funktion ändert die ursprüngliche Collection.
Ist die Collection leer, führt diese Funktion nichts aus.

Beispiel

```
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 (StartAb {; Ende}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
StartAb	Lange Ganzzahl	➡	Start Index (enthalten)
Ende	Lange Ganzzahl	➡	Ende Index (nicht enthalten)
Funktionsergebnis	Collection	⮌	Neue Collection mit ausgeschnittenen Elementen (flache Kopie)

Beschreibung

Die Funktion **collection.slice( )** gibt einen Ausschnitt einer Collection in einer neuen Collection zurück, definiert von *StartAb* bis *Ende* (*Ende* ist nicht enthalten). Diese Funktion gibt eine flache Kopie der Collection zurück. Ist die ursprüngliche Collection eine *shared collection*, ist die zurückgegebene Collection auch eine *shared collection*.

Hinweis: Dieser Befehl ändert nicht die ursprüngliche Collection.

Die zurückgegebene Collection enthält das in *StartAb* angegebene Element und alle darauffolgenden Element bis zum in *Ende* angegebenen Element (nicht enthalten) Ist nur der Parameter *StartAb* angegeben, enthält die zurückgegebene Collection alle Elemente ab *StartAb* bis zum letzten Element der ursprünglichen Collection.

- Ist *StartAb* < 0, wird es als *StartAb:=StartAb+Länge* neu berechnet (es wird als Versatz vom Ende der Collection gewertet).
- Ist der berechnete Wert < 0, wird *StartAb* auf 0 gesetzt.
- Ist *Ende* < 0, wird es als *Ende:=Ende+Länge*.
- Ist *Ende* < *StartAb* (übergebener oder berechneter Wert), führt die Funktion nichts aus.

Beispiel

```
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 ({StartAb ;} MethodenName {; param {; param2 ; ... ; paramN}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
StartAb	Lange Ganzzahl	➡	Index zum Starten des Tests
MethodenName	Text	➡	Name der Methode zum Aufrufen des Tests
param	Ausdruck	➡	Parameter zum Übergeben an MethodenName
Funktionsergebnis	Boolean	⮌	Wahr, wenn mindestens ein Element den Test erfolgreich durchlaufen hat

Beschreibung

Die Funktion **collection.some( )** gibt **wahr** zurück, wenn mindestens ein Element in der Collection den Test, angegeben in *MethodenName*, erfolgreich durchlaufen hat.

Standardmäßig testet **collection.some( )** die gesamte Collection. Optional können Sie in *StartAb* den Index des Elements übergeben, ab dem der Test starten soll.

- Ist *StartAb* >= Länge der Collection, wird -1 zurückgegeben, d.h. die Collection wird nicht getestet.
- Ist *StartAb* < 0, wird es als Versatz vom Ende der Collection gewertet
- Ist *StartAb* = 0, wird die gesamte Collection getestet (Standard).

In *MethodenName* übergeben Sie den Namen der Methode zum Bewerten der Collection Elemente, zusammen mit den Parametern in *param* (optional). *MethodenName* kann jeden Test mit oder ohne die Parameter durchführen. Diese Methode empfängt einen Parameter *Object* in $1 und muss *$1.result* für jedes Element, das den Test erfüllt, auf **wahr** setzen.

MethodenName empfängt folgende Parameter:

- in *$1.value*: Elementwert zum Bewerten
- in *$2*: *param*
- in *$N...*: param2...paramN

MethodenName setzt folgende Parameter:

- *$1.result* (boolean): **wahr**, bei erfolgreicher Bewertung des Elementwerts, sonst **falsch**.
- *$1.stop* (boolean, optional): **wahr**, um Aufruf der Methode zu stoppen. Der zurückgegebene Wert ist der letzte bewertete Wert.

In allen Fällen gilt: An der Stelle, wo die Funktion **collection.some( )** das erste Collection Element findet, das in $1.result **wahr** zurückgibt, stoppt sie das Aufrufen von *MethodenName* und gibt **wahr** zurück.

Beispiel

```
C_COLLECTION($c)
C_BOOLEAN($b)
$c:=New collection
$c.push(-5;-3;-1;-4;-6;-2)
$b:=$c.some("NumberGreaterThan0")  // gibt falsch zurück
$c.push(1)
$b:=$c.some("NumberGreaterThan0")  // gibt wahr zurück

$c:=New collection
$c.push(1;-5;-3;-1;-4;-6;-2)
$b:=$c.some("NumberGreaterThan0")  //$b=true
$b:=$c.some(2;"NumberGreaterThan0")  //$b=false
```

Der Code der Methode ***NumberGreaterThan0*** lautet:

```
$1.result:=$1.value>0
```

collection.sort( )

collection.sort ({MethodenName {; extraParam}{; extraParam2 ; ... ; extraParamN}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
MethodenName	Text	➡	Name der Methode zur Angabe der Sortierreihenfolge
extraParam	Ausdruck	➡	Parameter für die Methode
Funktionsergebnis	Collection	⮌	Sortierte Kopie der Collection (flache Kopie)

Beschreibung

Die Funktion **collection.sort( )** sortiert die Elemente der Collection und gibt eine neue sortierte Collection zurück. Diese Funktion gibt eine flache Kopie *(shallow copy)* zurück, d.h. Objekte oder Collections der beiden Collections nutzen dieselbe Referenz.

Hinweis: Diese Funktion verändert die ursprüngliche Collection.

Wird **collection.sort( )** ohne Parameter aufgerufen, werden nur skalare Werte (Zahl, Text, Datum, Boolean) sortiert. Elemente werden standardmäßig nach Typ und in autsteigender Reihenfolge sortiert.

Wollen Sie Collection Elemente in anderer Reihenfolge oder jeden Elementtyp sortieren, müssen Sie in *MethodenName* eine Vergleichsmethode einrichten, die zwei Werte vergleicht und **wahr** in *$1.result* zurückgibt, wenn der erste Wert niedriger als der zweite ist. Sie können bei Bedarf weitere Parameter übergeben:

- *MethodenName* empfängt folgende Parameter:
  - $1 (object), where:
    - *$1.value* (beliebiger Typ): erster Elementwert zum Vergleichen
    - *$1.value2* (beliebiger Typ): zweiter Elementwert zum Vergleichen
  - $2...$N (beliebiger Typ): extra Parameter
- *MethodenName* setzt folgende Parameter:
  - *$1.result* (boolean): **wahr** wenn *$1.value* < *$1.value2*, sonst falsch

Enthält die Collection Elemente mit unterschiedlichen Typen, werden sie erst nach Typ gruppiert und anschließend sortiert. Typen werden in folgender Reihenfolge zurückgegeben:

1. Null
2. Boolean
3. String
4. Zahl
5. Objekt
6. Collection
7. Datum

Beispiel 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]
```

Beispiel 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]
```

Beispiel 3

```
C_COLLECTION($col)
$col:=New collection(33;4;66;1111;222)
$col2:=$col.sort() //Sortierung nach Zahlen: [4,33,66,222,1111]
$col3:=$col.sort("numberOrder") //Sortierung nach Alphabet: [1111,222,33,4,66]
```

```
//Projektmethode numberOrder
C_OBJECT($1)
$1.result:=String($1.value)<String($1.value2)
```

collection.sum( )

collection.sum ({EigenschaftsPfad}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
EigenschaftsPfad	Text	⇒	Pfad Objekteigenschaft für die Berechnung
Funktionsergebnis	Zahl	⇐	Summe der Collection Werte

Beschreibung

Die Funktion **collection.sum( )** gibt die Summe aller Werte in der Collection Instanz zurück.
Bei der Berechnung werden nur numerische Elemente berücksichtigt (andere Elementtypen werden ignoriert).
Enthält die Collection Objekte, übergeben Sie den Parameter *EigenschaftsPfad*, damit die Objekteigenschaft berücksichtigt wird.
collection.sum( ) gibt 0 zurück, wenn:

- die Collection leer ist
- die Collection keine numerischen Elemente enthält
- *EigenschaftsPfad* in der Collection nicht gefunden wird

Beispiel 1

```
C_COLLECTION($col)
$col:=New collection(10;20;"Monday";True;2)
$vsum:=$col.sum() //32
```

Beispiel 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 (Wert {; Wert2 ; ... ; WertN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Ausdruck	➡	Wert(e) zum Einfügen am Anfang der Collection
Funktionsergebnis	Collection	⮌	Collection mit den hinzugefügten Elementen

Beschreibung

Die Funktion **collection.unshift( )** fügt den gegebenen *Wert* bzw. die Werte am Anfang der Collection ein und gibt die geänderte Collection zurück.

Hinweis: Diese Funktion ändert die ursprüngliche Collection.

Werden mehrere Werte übergeben, werden alle auf einmal eingefügt, d.h. sie erscheinen in der resultierenden Collection in derselben Reihenfolge wie in der Argumentliste.

Beispiel

```
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 {(Wert {; Wert2 ; ... ; WertN})} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl, Text, Datum, Objekt, Collection, Zeiger	➡	Wert der Collection
Funktionsergebnis	Collection	➲	Neue Collection

Beschreibung

Die Funktion **New collection** erstellt eine neue leere oder vorgefüllte Collection und gibt ihre Referenz zurück.
Übergeben Sie keine Parameter, erstellt **New collection** eine leere Collection und gibt ihre Referenz zurück.
Sie müssen die zurückgegebene Referenz einer 4D Variablen übergeben, die mit **C_COLLECTION** deklariert wurde.
Hinweis: Beachten Sie, dass **C_COLLECTION** eine Variable vom Typ *Collection* deklariert, jedoch keine Collection anlegt.
Optional können Sie die neue Collection vorab füllen, indem Sie als Parameter einen oder mehrere *Wert(e)* übergeben.
Andernfalls können Sie durch Zuweisung der Objektnotation Elemente in Folge hinzufügen oder ändern, wie zum Beispiel:

```
myCol[10]:="My new element"
```

Geht der neue Element Index über das letzte vorhandene Element der Collection hinaus, wird die Collection automatisch angepasst und alle neuen Zwischenelemente erhalten den Wert **Null**.
Hinweis: Weitere Informationen dazu finden Sie im Abschnitt **Objektnotation verwenden**.
Sie können beliebig viele Werte der unterstützten Typen übergeben, wie Zahl, Text, Datum, Zeiger, Objekt, Collection, etc. Im Gegensatz zu Arrays können Collections unterschiedliche Typen von Daten miteinander mischen.
Beim Konvertieren müssen Sie auf folgendes achten:

- Übergeben Sie einen Zeiger, bleibt er unverändert; er wird mit dem Befehl **JSON Stringify** bewertet.
- Datumsangaben werden als Datumsformat "yyyy-mm-dd" oder als Zeichenkette "YYYY-MM-DDTHH:mm:ss.SSSZ" gespeichert, gemäß der aktuellen Datenbank-Eigenschaft für Datum innerhalb von Objekten (siehe **Seite Kompatibilität**). Wird ein 4D Datum vor dem Speichern in der Collection in Text konvertiert, berücksichtigt das Programm standardmäßig die lokale Zeit. Sie können dieses Verhalten mit dem Selektor JSON use local time des Befehls **SET DATABASE PARAMETER** verändern.
- Übergeben Sie eine Zeit, wird sie als eine Zahl in Millisekunden gespeichert (Real).

Beispiel 1

Eine neue leere Collection erstellen und diese einer 4D Collection Variable zuweisen:

```
C_COLLECTION($myCol)
$myCol:=New collection
 //$myCol=[]
```

Beispiel 2

Eine vorgefüllte Collection erstellen:

```
C_COLLECTION($filledColl)
$filledColl:=New collection(33;"mike";"november";->myPtr;Current date)
 //$filledColl=[33,"mike","november",->myPtr,2017-03-28T22:00:00.000Z]
```

Beispiel 3

Eine neue Collection anlegen und dann ein neues Element hinzufügen:

```
C_COLLECTION($coll)
$coll:=New collection("a";"b";"c")
 //$coll=["a","b","c"]
$coll[9]:="z" //10. Element mit Wert "z" hinzufügen
$vcolSize:=$coll.length //10
 //$coll=["a","b","c",null,null,null,null,null,null,"z"]
```

Hinweis: Für dieses Beispiel muss die Objekt Notation in den Struktureinstellungen aktiviert sein. (siehe Abschnitt **Objektnotation**).

New shared collection

New shared collection {(Wert {; Wert2 ; ... ; WertN})} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert		➡	Wert(e) von shared collection
Funktionsergebnis	Collection	➲	Neue shared collection

Beschreibung

Die Funktion **New shared collection** erstellt eine neue leere oder vorab gefüllte *shared collection* und gibt ihre Referenz zurück. Fügen Sie dieser Collection ein Element hinzu, muss es in die Struktur **Use...End use** eingebettet werden, sonst wird ein Fehler erzeugt. Ein Element lesen ist dagegen ohne die Struktur **Use...End use** möglich.

Hinweis: Weitere Informationen dazu finden Sie auf der Seite **Shared Objects und Shared Collections**.

Übergeben Sie keine Parameter, erstellt **New shared collection** eine leere *shared collection* und gibt ihre Referenz zurück. Sie müssen diese Referenz einer 4D Variablen zuweisen, die mit dem Befehl **C_COLLECTION** deklariert wurde.

Hinweis: Beachten Sie, dass **C_COLLECTION** eine Variable vom Typ *Collection* deklariert, aber keine Collection erstellt.

Optional können Sie die neue *shared collection* vorab füllen, indem Sie ein oder mehrere *Wert(e)* als Parameter übergeben. Andernfalls können Sie Elemente danach über Zuweisen von Objektnotation hinzufügen oder ändern (siehe Beispiel).

Geht der neue Elementindex über das letzte Element in der *shared collection* hinaus, wird die Collection automatisch vergrößert und alle neu dazugekommenen Elemente erhalten den Wert **Null**.

Shared collection unterstützt Werte mit folgendem Typ:

- Numerisch (Zahl, Lange Ganzzahl...) Zahlenwerte werden immer als Zahl gespeichert
- Text
- Boolean
- Datum
- Zeit (gespeichert als Anzahl Millisekunden - Zahl)
- Null
- shared object(*)
- shared collection(*)

Hinweis: Im Gegensatz zu standardmäßigen Collections unterstützen *shared collections* weder Bilder oder Zeiger, noch *not-shared objects/collections*.

(*) Wird ein *shared object/collection* zu einer anderen *shared collection* hinzugefügt, teilen sie sich denselben Sperrschlüssel. Weitere informationen dazu finden Sie im Abschnitt **Sperrschlüssel (Locking Identifier)**.

Beispiel

```
$mySharedCol:=New shared collection("alpha";"omega")
Use($mySharedCol)
    $mySharedCol[1]:="beta"
End use
```

Compiler

- Compilerbefehle
- Compiler Direktiven
- Richtlinien zur Typisierung
- Einzelheiten zur Syntax
- Tipps zur Optimierung
- Fehlermeldungen
- 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*

Compilerbefehle

Der in 4D integrierte Compiler übersetzt Ihre Datenbankanwendungen in Maschinensprache. Die Vorteile des Compilers sind:

- **Geschwindigkeit:** Ihre Datenbank läuft um den Faktor von 3 bis 1000 mal schneller.
- **Überprüfung des Code:** Ihre Datenbank wird auf Übereinstimmung des Code geprüft. 4D Compiler findet sowohl logische als auch Syntaxfehler.
- **Schutz:** Ist Ihre Datenbank kompiliert, können Sie den interpretierten Code löschen. Denn die kompilierte Datenbanken hat dieselben Funktionalitäten wie ihr Original. Der Unterschied ist, dass Sie nicht auf die Struktur und Prozeduren zugreifen bzw. diese verändern können, weder gewollt noch versehentlich.
- **Eigenständige, doppelklickbare Anwendungen:** Kompilierte Datenbanken lassen sich in eigenständige Anwendungen (.EXE Dateien) umwandeln, die im Finder durch ihr eigenes Symbol dargestellt werden.
- **Ausführung im preemptive Modus**: Nur kompilierter Code lässt sich in einem preemptive Prozess ausführen.

Die Befehle dieses Kapitels werden mit dem Compiler eingesetzt. Damit können Sie innerhalb Ihrer Datenbank Datentypen normieren. Der Befehl **IDLE** wird speziell in kompilierten Datenbanken verwendet.

C_BLOB	**C_REAL**	**C_TEXT**
C_BOOLEAN	**C_LONGINT**	**C_DATE**
C_POINTER	**C_PICTURE**	**C_TIME**
C_OBJECT	**IDLE**	

Hinweis zur Kompatibilität: Die überholten Befehle **_o_C_GRAPH**,**_o_C_INTEGER** und **_o_C_STRING** dürfen nicht mehr verwendet werden.

Diese Befehle, außer **IDLE**, deklarieren Variablen und weisen einen spezifischen Datentyp zu. So vermeiden Sie zweideutige Zuweisungen. Ist eine Variable nicht mit einem dieser Compilerbefehle deklariert, versucht der Compiler den Datentyp einer Variablen zu bestimmen. Oft lässt sich der Datentyp von Variablen in Formularen nur schwer bestimmen. Von daher ist es besonders wichtig, Variablen in Formularen mit Compilerbefehlen zu deklarieren.

Hinweis: Um Zeit zu gewinnen, können Sie im Compiler-Fenster die Option zum Generieren und Aktualisieren von typisierten Methoden verwenden. Diese Option definiert die Anzahl der Durchgänge zum Typisieren der Variablen in der Datenbank.

Arrays sind Variablen, die zur Kompilierung dieselben Regeln wie Standardvariablen befolgen müssen. Die Befehle zum Deklarieren von Arrays werden im Kapitel **Arrays** beschrieben.

Allgemeine Regeln beim Schreiben von Code, der kompiliert werden soll

- Die Variablen-Indirektion ist nicht erlaubt. Sie können keine Alpha Indirektion mit dem Symbol (§) für Absatz verwenden, um indirekt auf Variablen zu verweisen. Ebensowenig können Sie dafür die numerische Indirektion mit den geschweiften Klammern ({...}) verwenden. Geschweifte Klammern gelten nur für den Zugriff auf spezifische Elemente eines deklarierten Array. Sie können nur die Parameter-Indirektion verwenden.
- Sie dürfen den Datentyp von Variablen oder Arrays nicht ändern.
- Sie dürfen ein eindimensionales Array nicht in ein zweidimensionales Array umwandeln und umgekehrt.
- Sie dürfen die Länge von String Variablen oder Elementen in String Arrays nicht ändern.
- Auch wenn der Compiler den Typ der Variablen für Sie festlegt, sollten Sie bei zweideutigen Datentypen, wie z.B. in einem Formular, den Datentyp der Variablen mit Compiler-Befehlen deklarieren.
- Durch explizites Zuweisen von Datentypen für Variablen optimieren Sie außerdem Ihren Code. Das gilt besonders bei Variablen, die als Zähler eingesetzt werden. Mit Variablen vom Typ Lange Ganzzahl erreichen Sie maximale Performance.
- Wollen Sie eine Variable löschen, d.h. auf Null initialisieren, verwenden Sie den Befehl **CLEAR VARIABLE** mit dem Namen der Variablen. Verwenden Sie bei diesem Befehl keinen String für den Namen der Variablen.
- Die Funktion **Undefined** gibt immer den Wert *Falsch* zurück. Variablen sind immer definiert.
- Zahlenoperationen auf Variablen vom Typ Lange Ganzzahl und Ganzzahl sind in der Regel viel schneller als Operationen auf den standardmäßigen Typ (Zahl).
- Ist für die Methode die Eigenschaft "Als preemptive Prozess starten" markiert, darf der Code weder thread-unsafe Befehle oder andere thread-unsafe Methoden aufrufen.

Die jeweilige Vorgehensweise wird in den folgenden Abschnitten erläutert:

- **Compiler Direktiven** erklärt, wann und wo Compiler Direktiven geschrieben werden.
- **Richtlinien zur Typisierung** beschreibt die verschiedenen Konfliktarten, die beim Kompilieren von 4D Anwendungen auftreten können.
- **Einzelheiten zur Syntax** gibt zusätzliche Informationen zu verschiedenen 4D Befehlen.
- **Tipps zur Optimierung** bietet Tipps, die das Laufen von Anwendungen im kompilierten Modus beschleunigen.
- **Eine thread-safe Methode schreiben**.

Beispiel 1

Nachfolgend sehen Sie ein paar grundlegende Variablendeklarationen für den Compiler:

```
C_BLOB(vxMyBlob)  // Die Prozessvariable vxMyBlob wird als Variable vom Typ BLOB deklariert
C_BOOLEAN(<>OnWindows)  // Die Interprozessvariable <>OnWindows wird als Variable vom Typ Boolean deklariert
C_DATE($vdCurDate)  // Die lokale Variable $vdCurDate wird als Variable vom Typ Datum deklariert
C_LONGINT(vg1;vg2;vg3)  // Die 3 Prozessvariablen vg1, vg2 und vg3 werden als Variablen vom Typ Lange Ganzzahl deklariert
```

Beispiel 2

In folgendem Beispiel deklariert die Projektmethode ***OneMethodAmongOthers*** 3 Parameter:

```
// Projektmethode OneMethodAmongOthers
// OneMethodAmongOthers ( Zahl ; Datum { ; Lang } )
// OneMethodAmongOthers ( Betrag ; Prozent { ; Ratio } )

C_REAL($1) // 1. Parameter ist vom Typ Zahl
C_DATE($2) // 2. Parameter ist vom Typ Datum
C_LONGINT($3) // 3. Parameter ist vom Typ Lange Ganzzahl

// ...
```

Beispiel 3

In folgendem Beispiel akzeptiert die Projektmethode ***Capitalize*** einen Parameter vom Typ Textund gibt einen Text zurück:

```
// Projektmethode Capitalize
// Capitalize ( Text ) -> Text
// Capitalize ( Ursprungsstring ) -> String in Großbuchstaben

C_TEXT(255;$0;$1)
$0:=Uppercase(Substring($1;1;1))+Lowercase(Substring($1;2))
```

Beispiel 4

In folgendem Beispiel akzeptiert die Projektmethode ***SEND PACKETS*** einen Parameter vom Typ Zeit gefolgt von einer variablen Anzahl von Textparametern:

```
// Projektmethode SEND PACKETS
// SEND PACKETS ( Zeit ; Text { ; Text2... ; TextN } )
// SEND PACKETS ( DokRef ; Daten { ; Daten2... ; DatenN } )

C_TIME($1)
C_TEXT(${2})
C_LONGINT($vlPacket)

For($vlPacket;2;Count parameters)
   SEND PACKET($1;${$vlPacket})
End for
```

Beispiel 5

In folgendem Beispiel deklariert die Projektmethode ***COMPILER_Param_Predeclare28*** die Syntax von anderen Projektmethoden für 4D Compiler vor:

```
// Projektmethode COMPILER_Param_Predeclare28

C_REAL(OneMethodAmongOthers;$1) // OneMethodAmongOthers (Zahl ; Ganzzahl { ; Lang })
C_INTEGER(OneMethodAmongOthers;$2) // ...
C_LONGINT(OneMethodAmongOthers;$3) // ...
C_TEXT(Capitalize;255;$0;$1) // In Großbuchstaben ( Text) -> Text
C_TIME(SEND PACKETS;$1) // SEND PACKETS ( Zeit ; Text { ; Text2... ; TextN } )
C_TEXT(SEND PACKETS;${2}) // ...
```

Datentypen der Variablen

4D kennt drei Variablentypen:

- Lokale Variablen,
- Prozessvariablen,
- Interprozessvariablen.

Weitere Informationen dazu finden Sie im Abschnitt **Variablen**. Prozess- und Interprozessvariablen sind von der Struktur her dasselbe für den Compiler.

Da der Compiler nicht den Prozess bestimmen kann, in welchem die Variable verwendet wird, sollten Sie Prozessvariablen mit mehr Bedacht als Interprozessvariablen verwenden. Alle Prozessvariablen werden systematisch dupliziert, wenn ein neuer Prozess beginnt. Eine Prozessvariable kann in jedem Prozess einen anderen Wert haben, sie ist dagegen für die gesamte Datenbank vom gleichen Typ.

Variablentypen

Alle Variablen haben einen Typ. Wie im Abschnitt **Datentypen** beschrieben, gibt es folgende Typen:

Boolean
Datum
Lange Ganzzahl
Diagramm
Zeit
Bild
Numerisch (oder Zahl)
Zeiger
Text
BLOB
Objekt
Collection

Es gibt verschiedene Typen von Arrays:

Array Boolean
Array Datum
Array Ganzzahl
Array Lange Ganzzahl
Array Bild
Array Zahl
Array Zeit
Array Objekt
Array Zeiger
Array BLOB
Array Text

Hinweise:
- Der Typ Objekt ist seit 4D v14 verfügbar.
- Der Typ Collection ist ab 4D v16 R4 verfügbar.
- Der frühere Typ Alpha (fester String - vor Version 11) wird für Variablen nicht mehr verwendet. In vorhandenem Code wird er automatisch umgeleitet zum Typ Text. Die früheren Typen Ganzzahl und Diagramm (vor Version 11) werden für Variablen nicht mehr verwendet. In vorhandenem Code werden sie automatisch umgeleitet zu den Typen Lange Ganzzahl.

Symboldatei erstellen

Im interpretierten Modus kann sich der Datentyp einer Variablen ändern. Das ist möglich, da der Code interpretiert und nicht kompiliert wird. 4D interpretiert jede Anweisung separat und versteht ihren Kontext.
Im kompiliertem Modus ist die Situation anders. Während die Interpretation Zeile für Zeile abläuft, betrachtet der Kompilierungsprozess die Datenbank in ihrer Gesamtheit.

Der Compiler arbeitet folgendermaßen:

- Er analysiert systematisch die Objekte in der Datenbank. Das sind Datenbank-, Projekt-, Formular-, Objektmethoden und Trigger.
- Er durchläuft die Objekte, um den Datentyp jeder in der Datenbank verwendeten Variablen zu bestimmen und generiert eine Tabelle mit den Variablen und Methoden. Das ist die Symboldatei.
- Sind die Datentypen von allen Variablen festgelegt, übersetzt bzw. kompiliert der Compiler die Datenbank. Das ist jedoch nur möglich, wenn er den Datentyp für jede der Variablen bestimmen kann.

Trifft der Compiler auf denselben Variablennamen für zwei unterschiedliche Datentypen, kann er keine Entscheidung treffen. Er muss nämlich, um ein Objekt zu typisieren und eine Speicheradresse zu vergeben, die exakte Kennung dieses Objekts wissen, d.h. Name und Datentyp. Der Compiler erstellt für jede kompilierte Datenbank eine Übersicht, die für jede Variable Name (oder Identifier) und Platzierung (oder Speicheradresse) sowie den gemäß dem Datentyp belegten Platz angibt. Diese Übersicht heißt Symboldatei. In den Einstellungen der Datenbank können Sie festlegen, ob diese Übersicht während dem Kompilieren in Form einer Textdatei angelegt werden soll.
Diese Übersicht wird auch für die automatische Erstellung von Compiler-Methoden verwendet.

Vom Compiler typisierte Variablen

Soll der Compiler die Typisierung Ihrer Variablen prüfen oder sie selbst typisieren, ist die Platzierung der Direktiven einfach. Sie können je nach Ihrer Arbeitsweise zwischen zwei Möglichkeiten wählen:

- Sie verwenden die Direktive in der Methode, wo die Variable zuerst erscheint, das hängt auch davon ob, ob es eine lokale, Prozess- oder Interprozessvariable ist. Stellen Sie sicher, dass die Direktive beim ersten Auftreten der Variablen in der zuerst auszuführenden Methode verwendet wird. Bedenken Sie, dass der Compiler beim Kompilieren die Methoden in der Reihenfolge hernimmt, wie sie in 4D erstellt wurden und nicht in der Reihenfolge, wie sie im Explorer angezeigt werden.
- Bei systematischer Arbeitsweise gruppieren Sie alle Prozess- und Interprozessvariablen mit den verschiedenen Compiler Direktiven in der **Datenbankmethode On Startup** oder in einer Methode, die von der **Datenbankmethode On Startup** aufgerufen wird.
  Für lokale Variablen gruppieren Sie die Direktiven an den Anfang der Methode, in welcher sie erscheinen.

Standardwerte

Werden Variablen über eine Compiler Direktive typisiert, erhalten sie einen Standardwert, den sie während der Sitzung beibehalten, solange sie nicht zugewiesen sind.

Der Standardwert richtet sich nach Variablentyp und -kategorie, Ausführungskontext (interpretiert oder kompiliert), und im kompilierten Modus nach den definierten Optionen auf der **Seite Compiler** der Datenbank-Eigenschaften:

- Prozess- und Interprozessvariablen werden immer auf "auf Null" gesetzt, d.h. je nach Kontext, "0", Leerstring, leeres Blob, Nil pointer, leeres Datum (0000-00-00), etc.
- Lokale Variablen werden wie folgt gesetzt:
  - im interpretierten Modus: auf Null
  - im kompilierten Modus, je nach gewählter Option für **Lokale Variablen initialisieren** auf der Seite Compiler der Datenbank-Eigenschaften:
    - auf Null, wenn "auf Null" gewählt ist,
    - auf einen zufälligen Wert, wenn "auf zufälligen Wert" gewählt ist (0x72677267 für Zahlen und Zeiten, immer True für Booleans, sonst genauso wie bei "auf Null",
    - auf einen zufälligen Wert (für Zahlen) wenn "NIcht" gewählt ist.

Nachfolgende Tabelle zeigt diese Standardwerte:

Typ	Interprozess	Prozess	Lokal interpretiert	Lokal kompiliert "auf Null"	Lokal kompiliert "zufällig"	Lokal kompiliert "nicht"
Boolean	False	False	False	False	True	True (variiert)
Datum	00-00-00	00-00-00	00-00-00	00-00-00	00-00-00	00-00-00
Lange Ganzzahl	0	0	0	0	1919382119	909540880 (variiert)
Diagramm	0	0	0	0	1919382119	775946656 (variiert)
Zeit	00:00:00	00:00:00	00:00:00	00:00:00	533161:41:59	249345:34:24 (variiert)
Bild	Bildgröße=0	Bildgröße=0	Bildgröße=0	Bildgröße=0	Bildgröße=0	Bildgröße=0
Zahl	0	0	0	0	1.250753659382e+243	1.972748538022e-217 (variiert)
Zeiger	Nil=true	Nil=true	Nil=true	Nil=true	Nil=true	Nil=true
Text	""	""	""	""	""	""
Blob	Blobgröße=0	Blobgröße=0	Blobgröße=0	Blobgröße=0	Blobgröße=0	Blobgröße=0
Objekt	null	null	null	null	null	null
Collection	null	null	null	null	null	null

Compiler Direktiven verwenden

Compiler Direktiven sind in folgenden Fällen sinnvoll:

- Der Compiler kann den Datentyp einer Variablen nicht aus ihrem Kontext bestimmen,
- Sie möchten nicht, dass der Compiler den Typ einer Variablen über ihre Verwendung bestimmt.

Außerdem sorgen Compiler Direktiven für eine Verringerung der Kompilierungszeit.

Zweideutige Fälle

Manchmal kann der Compiler den Datentyp einer Variablen nicht bestimmen. Ist dies der Fall, generiert er eine entsprechende Fehlermeldung.
Es gibt drei Hauptgründe, die den Compiler den Datentyp nicht bestimmen lassen: Mehrfache Datentypen, Zweideutigkeit aufgrund einer gefolgerten Ableitung und keine Möglichkeit zur Bestimmung des Datentyps.

Mehrfache Datentypen

Hat eine Variable in den einzelnen Anweisungen unterschiedliche Datentypen, generiert der Compiler einen Fehler, der leicht zu beheben ist. Der Compiler wählt die zuerst gefundene Variable und weist deren Datentyp wilkürlich dem nächsten Auftreten der gleichnamigen Variablen mit einem anderen Datentyp zu. Hier ein einfaches Beispiel:

In Methode A

```
Variable:=True
```

In Methode B

```
Variable:="Der Mond ist rund"
```

Wird Methode A vor Methode B kompiliert, bewertet der Compiler die Anweisung Variable:="Der Mond ist rund" als Änderung des Datentyps gegenüber der zuvor gefundenen Variablen. Der Compiler meldet, dass eine Retypisierung stattgefunden hat und erstellt eine Fehlermeldung. In den meisten Fällen können Sie das Problem beheben, indem Sie das zweite Auftreten der Variablen umbenennen.

Zweideutigkeit aufgrund einer gefolgerten Ableitung

In manchen Fällen kann der Compiler anhand des Ablaufs ableiten, dass der Typ des Objekts nicht geeignet ist. In diesem Fall müssen Sie die Variable explizit mit einer Compiler Direktiven typisieren. Es folgt ein Beispiel, das mit Standardwerten für ein aktives Objekt arbeitet:

Sie können in einem Formular für folgende Objekte Standardwerte zuweisen: Combo Boxen, PopUp Menüs, Registerkarten, DropDown Listen, Menü/DropDown Listen und rollbare Bereiche unter Verwendung der Schaltfläche Bearbeiten für die Liste **Werte** (in der Eigenschaftenliste unter dem Thema Eingabekontrollen). Weitere Informationen dazu finden Sie im Abschnitt **Standardwerte** des Handbuchs *4D Designmodus*. Die Standardwerte werden automatisch in ein Array geladen, das denselben Namen wie das Objekt hat.
Wird das Objekt nicht in einer Methode verwendet, kann der Compiler den Typ eindeutig als Array vom Typ Text ableiten. Muss jedoch eine Initialisierung der Anzeige durchgeführt werden, könnte die Sequenz folgendermaßen lauten:

```
Case of
   :(Form event=On Load)
      MyPopUp:=2
      ...
End case
```

In diesem Fall tritt die Zweideutigkeit ein--beim Analysieren der Methoden leitet der Compiler für das Objekt MyPopUp als Datentyp Zahl ab. In diesem Fall ist es notwendig, das Array in der Formular- oder einer Compiler-Methode explizit zu deklarieren:

```
Case of
   :(Form event=On Load)
      ARRAY TEXT(MyPopUp;2)
      MyPopUp:=2
      ...
End case
```

Keine Möglichkeit zur Bestimmung des Datentyps

Dieser Fall tritt ein, wenn eine Variable ohne Typisierung in einem Kontext verwendet wird, der keine Angaben zum Datentyp liefert. Hier kann nur eine Compiler Direktive dem Compiler helfen. Dieses Phänomen tritt hauptsächlich in vier Kontexten auf:

- bei Verwendung von Zeigern (Pointer),
- wenn Sie einen Befehl mit mehr als einer Syntax verwenden,
- wenn Sie einen Befehl mit optionalen Parametern mit unterschiedlichen Datentypen verwenden,
- bei Verwendung einer 4D Methode, die über eine URL aufgerufen wird.

Zeiger

Von einem Zeiger wird nicht erwartet, dass er etwas anderes als seinen eigenen Datentyp zurückgibt.

Nehmen wir folgende Sequenz:

```
Var1:=5.2(1)
Zeiger:=->Var1(2)
Var2:=Zeiger->(3)
```

Obwohl (2) den Variablentyp definiert, auf den der Zeiger zeigt, ist der Typ von Var2 nicht festgelegt. Der Compiler erkennt während der Kompilation zwar einen Zeiger, hat jedoch keine Möglichkeit, zu erkennen, auf welchen Variablentyp er zeigt. Deshalb kann er den Datentyp von Var2 nicht ableiten. Hier ist eine Compiler Direktive erforderlich, zum Beispiel *C_REAL(Var2)*.

Befehle mit mehrfacher Syntax

Verwenden Sie eine Variable, der die Funktion **Year of** zugewiesen wurde, kann die Variable naturgemäß nur vom Typ Datum sein. Die Dinge liegen jedoch nicht immer so einfach. Nehmen wir folgendes Beispiel:

Der Befehl **GET FIELD PROPERTIES** lässt zwei Syntax-Arten zu:
GET FIELD PROPERTIES*(TabelleNum;FeldNum;Typ;Länge;Indiziert)*
GET FIELD PROPERTIES*(FeldPtr;Typ;Länge;Indiziert)*

Bei einem Befehl mit mehrfacher Syntax kann der Compiler nicht erraten, welche Syntax und Parameter Sie gewählt haben. Hier müssen Sie Compiler Direktiven einsetzen, um die im Befehl übergebenen Variablen zu typisieren, wenn diese nicht bereits an anderer Stelle in der Datenbank aufgrund ihrer Verwendung typisiert wurden.

Befehle mit optionalen Parametern mit unterschiedlichen Datentypen

Bei einem Befehl mit mehreren optionalen Parametern mit unterschiedlichen Datentypen kann der Compiler nicht bestimmen, welcher optionale Parameter verwendet wurde. Nehmen wir folgendes Beispiel:
Der Befehl **GET LIST ITEM** lässt zwei optionale Parameter zu; den ersten als Lange Ganzzahl, den zweiten als Boolean.

Der Befehl kann verwendet werden in Form von:
GET LIST ITEM*(Liste;EintragPos;EintragRef;EintragText;Unterliste;Erweitert)*
oder in Form von
GET LIST ITEM*(Liste;EintragPos;EintragRef;EintragText;Erweitert)*
Hier müssen Sie Compiler Direktiven einsetzen, um die im Befehl übergebenen optionalen Variablen zu typisieren, wenn diese nicht bereits an anderer Stelle in der Datenbank aufgrund ihrer Verwendung typisiert wurden.

Über URL aufgerufene Methoden
Schreiben Sie 4D Methoden, die über eine URL aufgerufen werden müssen und verwenden Sie nicht $1 in der Methode, müssen Sie die Textvariable $1 explizit mit folgender Sequenz deklarieren:
C_TEXT($1)
In der Tat kann der Compiler nicht feststellen, ob die 4D Methode über eine URL aufgerufen wird.

Kompilierungszeit einsparen

Sind alle in der Datenbank vorkommenden Variablen explizit deklariert, muss der Compiler die Typisierung nicht überprüfen. In diesem Fall können Sie in den Einstellungen der Datenbank definieren, dass der Compiler nur die Übersetzung der Methode vornimmt. Das spart mindestens 50% an Zeit für die Kompilierung.

Code optimieren

Über Compiler Direktiven können Sie Ihre Methoden beschleunigen. Weitere Informationen dazu finden Sie im Abschnitt **Tipps zur Optimierung**. Nehmen wir an, Sie müssen einen Zähler über eine lokale Variable erhöhen. Deklarieren Sie die Variable nicht, geht der Compiler vom Typ Zahl aus. Deklarieren Sie die Variable als Lange Ganzzahl, arbeitet die kompilierte Datenbank effizienter. Auf einem Rechner beansprucht der Typ Zahl 8 Bytes, deklarieren Sie den Zähler als Lange Ganzzahl, benötigt er nur 4 Bytes. Das Hochzählen eines 8-Byte Zählers dauert also länger als das eines 4-Byte Zählers.

Compiler Direktiven anwenden

Compiler Direktiven lassen sich auf zwei Arten verwalten, je nachdem ob der Compiler Ihre Variablen typisieren soll oder nicht.

Variablen typisieren

Der Compiler muss die Kriterien zum Erkennen von Variablen beachten.
Es gibt zwei Möglichkeiten:

1) Wurden die Variablen nicht typisiert, kann der Compiler das automatisch erledigen. Er bestimmt nach Möglichkeit, d.h. solange keine Zweideutigkeit vorliegt, den Typ der Variablen, nach der Art ihrer Verwendung. Schreiben Sie z.B.

```
V1:=True
```

bestimmt der Compiler für die Variable V1 als Datentyp Boolean.
Schreiben Sie:

```
V2:="Dies ist ein einfacher Satz"
```

bestimmt der Compiler für die Variable V2 als Datentyp Text.
Der Compiler kann den Datentyp einer Variablen auch in nicht so klaren Situationen bestimmen:

```
V3:=V1  `V3 ist vom selben Typ wie V1
V4:=2*V2  `V4 ist vom selben Typ wie V2
```

Der Compiler bestimmt den Datentyp Ihrer Variablen auch über Aufrufe von 4D Befehlen und über Ihre Methoden. Übergeben Sie z.B. in einer Methode einen Parameter vom Typ Boolean und einen vom Typ Datum, weist der Compiler in der aufgerufenen Methode den lokalen Variablen $1 und $2 die Typen Boolean und Datum zu.
Bestimmt der Compiler den Datentyp durch Schlussfolgerung, weist er nie die eingrenzenden Typen Ganzzahl, Lange Ganzzahl oder Text zu, sondern standardmäßig immer den größtmöglichen Typ. Schreiben Sie z.B.:

```
Nummer:=4
```

weist der Compiler als Datentyp Zahl zu, obwohl 4 eine Ganzzahl ist. Dadurch kann der Wert in einem anderen Zusammenhang auch den Wert 4,5 haben.
Ist es sinnvoll, eine Variable als Ganzzahl, Lange Ganzzahl oder Text zu typisieren, können Sie eine Compiler Directive verwenden. Der Vorteil dabei ist, dass diese Datentypen weniger Speicher belegen und Operationen damit rascher ablaufen.
Haben Sie Ihre Variablen typisiert und sind Sie sicher, dass Ihre Typisierung kohärent und vollständig ist, können Sie in den Einstellungen zum Kompilieren explizit festlegen, dass der Compiler diese Arbeit nicht mehr ausführen muss. Sollte Ihre Typisierung nicht hundertprozentig richtig sein, meldet der Compiler beim Kompilieren Fehler und nennt die erforderlichen Änderungen.
2) Über Befehle der Compiler Direktiven können Sie explizit die in Ihrer Datenbank verwendeten Variablen deklarieren.
Sie werden folgendermaßen verwendet:

```
C_BOOLEAN(Var)
```

Solche Direktiven sagen dem Compiler, dass er eine Variable *Var* mit dem Typ Boolean anlegen soll. Eine Anwendung mit Compiler Direktiven erleichtert die Arbeit des Compilers und vermeidet Vermutungen. Eine Compiler Direktive hat Vorrang vor Ableitungen, die auf Anweisungen oder der Verwendungsart beruhen. Variablen, die mit der Compiler Direktive **_o_C_INTEGER** definiert wurden, sind in der Tat dasselbe wie Variablen mit der Compiler Direktive **C_LONGINT**. Das sind in beiden Fällen Lange Ganzzahlen zwischen –2147483648 und +2147483647.

Vom Entwickler typisierte Variablen

Soll der Compiler Ihre Typisierung nicht prüfen, müssen Sie ihm einen Code vorgeben, um die Compiler Direktiven zu identifizieren.
Die Konvention ist folgende:
Compiler Direktiven für Prozess- und Interprozessvariablen und die Parameter sollten in einer oder mehreren Methoden gesetzt werden. Ihr Name muss mit dem Schlüsselwort **Compiler** beginnen.
Sie können mit dem Compiler standardmäßig fünf Arten von Compilermethoden generieren. Das sind die Direktiven für

Variablen, Arrays und Methodenparameter. Weitere Informationen dazu finden Sie im Abschnitt **Compilermethoden für ...** des Handbuchs *4D Designmodus*.

Hinweis: Die Syntax zum Deklarieren dieser Parameter lautet:
Direktive (MethodenName;Parameter). Sie lässt sich nicht im interpretierten Modus ausführen.

Besondere Parameter

- **Über Datenbankmethoden empfangene Parameter:** Wurden diese Parameter nicht ausdrücklich deklariert, macht dies der Compiler. Wollen Sie diese deklarieren, müssen Sie das innerhalb der Datenbankmethoden ausführen. Sie können diese Parameter Deklaration nicht in einer Compiler Methode schreiben. Beispiel: Die **Datenbankmethode On Web Connection** empfängt sechs Parameter, $1 bis $6 vom Typ Text. Sie schreiben zu Beginn der Datenbankmethode:
C_TEXT($1;$2;$3;$4;$5;$6)
- **Trigger:** Der Parameter $0 (Lange Ganzzahl), welcher das Ergebnis eines Triggers ist, wird vom Compiler typisiert, wenn der Parameter nicht ausdrücklich deklariert wurde. Wollen Sie ihn deklarieren, müssen Sie das innerhalb des Triggers ausführen. Sie können diese Parameter Deklaration nicht in einer Compiler Methode schreiben.
- **Objekte mit dem Formularereignis "On Drag Over":** Der Parameter $0 (Lange Ganzzahl), welcher das Ergebnis des Formularereignisses "On Drag Over"ist, wird vom Compiler typisiert, wenn der Parameter nicht ausdrücklich deklariert wurde. Wollen Sie ihn deklarieren, müssen Sie das innerhalb der Objektmethode tun. Sie können diese Parameter Deklaration nicht in einer Compiler Methode schreiben.

Hinweis: Der Compiler initialisiert nicht den Parameter $0. Sie müssen also, sobald Sie das Formularereignis "On Drag Over" verwenden, $0 initialisieren.

Beispiel

```
C_LONGINT($0)
If(Form event=On Drag Over)
   $0:=0
   ...
   If($DataType=ls picture)
      $0:=-1
   End if
   ...
End if
```

Vom Compiler zugelassener Spielraum

Compiler Direktiven entfernen jede Zweideutigkeit bei Datentypen. Auch wenn eine gewisse Striktheit notwendig ist, muss das nicht heißen, dass der Compiler überhaupt keine Inkonsistenz toleriert.
Weisen Sie zum Beispiel einer Variablen, die als Ganzzahl deklariert wurde, einen Wert vom Typ Zahl zu, sieht der Compiler keinen Typkonflikt und weist die Werte gemäß Ihren Direktiven zu. Wenn Sie also schreiben:

```
C_LONGINT(vInteger)
vInteger:=2.6
```

betrachtet der Compiler das nicht als Konflikt des Datentyps, der eine Kompilierung verhindert; vielmehr rundet er einfach auf den nächstgelegenen Wert auf (3 anstatt 2.6).

Richtlinien zur Typisierung

Dieser Abschnitt beschreibt die wichtigsten Gründe für Konflikte bei typisierten Variablen und zeigt Wege, sie zu vermeiden.

Konflikte bei einfachen Variablen

Es gibt folgende Varianten:

- Konflikt zwischen zwei Verwendungen
- Konflikt zwischen Verwendung und Compiler Direktive
- Konflikt durch implizierte Retypisierung
- Konflikt zwischen zwei Compiler Direktiven

Konflikt zwischen zwei Verwendungen

Der einfachste Konflikt bei Datentypen ist ein Variablenname, der zwei unterschiedliche Objekte angibt. Sie schreiben zum Beispiel:

```
Variable:=5
```

und in derselben Anwendung an anderer Stelle:

```
Variable:=Wahr
```

Das ergibt einen Datentypkonflikt. Sie beheben das Problem, wenn Sie eine der Variablen umbenennen.

Konflikt zwischen Verwendung und Compiler Direktive

Nehmen wir an, Sie schreiben:

```
Variable:=5
```

und in derselben Anwendung an anderer Stelle:

```
C_BOOLEAN(Variable)
```

Da der Compiler zuerst die Direktiven durchläuft, typisiert er Variable als Boolean, stößt er dann auf die Anweisung:

```
Variable:=5
```

ergibt das einen Datentypkonflikt. Sie beheben das Problem, wenn Sie eine der Variablen umbenennen oder die Compiler Direktive ändern.

Verwenden Sie in derselben Anweisung Variablen mit unterschiedlichen Datentypen, führt das zu Inkonsistenz. Der Compiler meldet Unstimmigkeiten. Hierzu ein Beispiel:

```
vBool:=True  `Der Compiler folgert, dass vBoolean vom Typ Boolean ist
C_LONGINT(<>vInteger)  `Deklaration einer Langen Ganzzahl durch Compiler Direktive
<>vInteger:=3  `Befehl ist kompatibel zur Compiler Direktive
Var:=<>vInteger+vBool  `Operation enthält Variablen mit inkompatiblen Datentypen
```

Konflikt durch implizierte Retypisierung

Einige Funktionen geben Variablen mit einem ganz bestimmten Datentyp zurück. Weisen Sie das Ergebnis einer solchen Variablen einer Variablen zu, die bereits anders typisiert wurde, ergibt sich u.U. ein Datentypkonflikt.
Sie schreiben zum Beispiel in einer interpretierten Anwendung:

```
IdentNo:=Request("KennNummer")IdentNo ist vom Typ Text
If(Ok=1)
   IdentNo:=Num(IdentNo)  `IdentNo ist vom Typ Zahl
   QUERY([Kunden]No=IdentNo)
End if
```

In diesem Beispiel erzeugen Sie in der dritten Zeile einen Datentypkonflikt. Hier müssen Sie das Verhalten der Variablen steuern. In einigen Fällen müssen Sie eine Variable mit einem anderen Namen dazwischenlegen. In anderen Fällen müssen Sie Ihre Methode anders strukturieren:

```
IdentNo:=Num(Request("KennNummer"))  `IdentNo ist vom Typ Zahl
If(Ok=1)
   QUERY([Kunden]No=IdentNo)
End if
```

Konflikt zwischen zwei Compiler Direktiven

Deklarieren Sie dieselbe Variable über zwei unterschiedliche Compiler Direktiven, bedeutet das eine Retypisierung. Schreiben Sie in der gleichen Datenbank:

```
C_BOOLEAN(Variable)
C_TEXT(Variable)
```

entdeckt der Compiler den Konflikt und meldet einen Fehler. Sie beheben das Problem, wenn Sie eine der Variablen umbenennen.

Anmerkung zu lokalen Variablen

Lokale Variablen haben dieselben Datentypkonflikte wie Prozess- oder Interprozessvariablen. Der einzige Unterschied ist, dass die Konsistenz nur innerhalb der angegebenen Methode vorhanden sein muss.
Bei Prozess- oder Interprozessvariablen treten Konflikte auf der allgemeinen Ebene der Datenbank auf, bei lokalen Variablen auf Methodenebene. Sie können also in derselben Methode nicht schreiben:

```
$Temp:="Guten Tag"
```

und dann

```
$Temp:=5
```

Sie können jedoch in Methode M1 schreiben:

```
$Temp:="Guten Tag"
```

und in Methode M2:

```
$Temp:=5
```

da sich die Reichweite von lokalen Variablen nur auf die Methode selbst erstreckt und nicht auf die gesamte Datenbank.

Konflikte in Arrays

Konflikte in einem Array betreffen nie die Größe, da Arrays, wie in unkompilierten Datenbanken, dynamisch verwaltet werden. Die Größe eines Arrays kann je nach Methode unterschiedlich sein. Sie müssen für ein Array auch keine max. Größe deklarieren. Sie können also ein Array auf Null setzen, Elemente hinzufügen bzw. entfernen oder den Inhalt löschen.

Beachten Sie beim Erstellen einer Datenbank, die kompiliert werden soll, folgende Regeln:

- Verändern Sie nicht die Datentypen von Elementen des Array,
- Verändern Sie nicht die Anzahl der Dimensionen in einem Array,
- Verändern Sie bei einem Array vom Typ String nicht die definierte Länge der Zeichenkette.

Datentypen bei Elementen im Array verändern

Deklarieren Sie ein Array als Ganzzahl, muss dieses Array vom Typ Ganzzahl für die ganze Datenbank erhalten bleiben. Es kann also nicht Elemente vom Typ Boolean enthalten.
Schreiben Sie:

```
ARRAY INTEGER(MyArray;5)
ARRAY BOOLEAN(MyArray;5)
```

kann der Compiler MyArray nicht typisieren. In diesem Fall müssen Sie ein Array umbenennen.

Dimensionen eines Array verändern

In einer unkompilierten Datenbank können Sie die Anzahl der Dimensionen in einem Array verändern. Der Compiler verwaltet jedoch beim Erstellen der Symboldatei eindimensionale Arrays anders als zweidimensionale Arrays.
Folglich können Sie ein eindimensionales Array nicht umändern in ein zweidimensionales, oder umgekehrt.
Die gleiche Datenbank kann als nicht enthalten:

```
ARRAY INTEGER(MyArray1;10)
ARRAY INTEGER(MyArray1;10;10)
```

Sie können jedoch in derselben Anwendung schreiben:

```
ARRAY INTEGER(MyArray1;10)
ARRAY INTEGER(MyArray2;10;10)
```

Sie können in einer Datenbank nicht die Anzahl der Dimensionen, jedoch die Größe eines Array ändern, also z.B. ein Array in einem zweidimensionalen Array verändern:

```
ARRAY BOOLEAN(MyArray;5)
ARRAY BOOLEAN(MyArray;10)
```

Hinweis: Ein zweidimensionales Array setzt sich aus mehreren eindimensionalen Arrays zusammen. Weitere Informationen dazu finden Sie im Abschnitt **Zweidimensionale Arrays**.

Implizierte erneute Typisierung

Bei Befehlen wie **COPY ARRAY**, **LIST TO ARRAY**, **ARRAY TO LIST**, **SELECTION TO ARRAY**, **SELECTION RANGE TO ARRAY**, **ARRAY TO SELECTION** oder **DISTINCT VALUES** können Sie wissentlich oder unwissentlich die Datentypen von Elementen oder die Anzahl der Dimensionen verändern. Sie befinden sich also in einer dieser Situationen.
Der Compiler generiert eine Fehlermeldung; die erforderliche Korrektur ist meist ganz offensichtlich. Beispiel für implizierte Retypisierung finden Sie im Abschnitt **Einzelheiten zur Syntax**.

Lokale Arrays

Wollen Sie eine Datenbank mit lokalen Arrays kompilieren, d.h. sie sind nur in den Methoden sichtbar, die sie erstellt haben, müssen Sie diese vor der Verwendung explizit in 4D deklarieren.
Ein Array explizit deklarieren heißt, einen Befehl vom Typ **ARRAY REAL**, **ARRAY INTEGER**, etc. verwenden. Erstellt eine Methode zum Beispiel ein lokales Array vom Typ Ganzzahl mit 10 Elementen, sollte Ihre Methode folgende Zeile enthalten:

```
ARRAY INTEGER($MyArray;10)
```

In Formularen erstellte Variablen typisieren

In Formularen erstellte Variablen sind immer Prozess- oder Interprozessvariablen. Das können Schaltflächen, DropDown-Listen, etc. sein.
In einer interpretierten Datenbank ist der Datentyp solcher Variablen unwichtig. In kompilierten Anwendungen müssen Sie dagegen darauf achten. Die Regeln dafür sind denkbar einfach:

- Sie können diese Variablen mit Compiler Direktiven typisieren, oder
- Der Compiler weist ihr einen Standardtyp zu, den Sie in den Kompilierungseinstellungen definieren können. Weitere Informationen dazu finden Sie im Abschnitt **Compilermethoden für ...** des Handbuchs *4D Designmodus*.

Variablen, die im Formular standardmäßig als Zahl gelten

Kontrollkästchen
3D Kontrollkästchen
Schaltfläche
Invertierte Schaltfläche
Unsichtbare Schaltfläche
3D Schaltfläche
Bildschaltfläche
Schaltflächengitter
Optionsfeld
3D Optionsfeld
Bildoptionsfeld
Bildmenü
Hierarchisches PopUp-Menü
Hierarchische Liste
Lineal
Halbkreisskala
Thermometer
Listbox (Typ Auswahl)

Hinweis: Die Formularvariablen Lineal, Halbkreisskala und Thermometer werden immer als Zahl typisiert, selbst wenn Sie in den Einstellungen zur Kompilierung als Schaltflächentyp Lange Ganzzahl definieren.

Für diese Variablen kann ein Datentypkonflikt nur auftreten, wenn der Name einer Variablen mit einer an anderer Stelle gesetzten Variablen identisch ist. Geben Sie in diesem Fall der zweiten Variablen einen anderen Namen.

Variable Plug-In Bereich

Ein Plug-In Bereich ist immer vom Typ Lange Ganzzahl. Diese Variable verursucht keinen Datentypkonflikt, außer der Name einer Variablen dieses Typs ist mit einer an anderer Stelle gesetzten Variablen identisch. Geben Sie in diesem Fall der zweiten Variablen einen anderen Namen.

Variable 4D Write Pro area

4D Write Pro Bereiche sind immer Variablen vom Typ Objekt. Hier kann es keinen Typisierungskonflikt geben, außer der gleiche Variablenname wird noch in einem anderen Teil der Anwendung verwendet.

Variablen, die im Formular standardmäßig als Text gelten

Nicht eingebbare Variable
Eingebbare Variable
DropDown Liste
Menü/DropDown Liste
Rollbarer Bereich
Combo Boxen
PopUp Menü
Registerkarte
Web Bereich
Spalte einer Listbox (Typ Array)

Diese Variablen sind in zwei Kategorien unterteilt:

- **Einfache Variablen** (eingebbare und nicht eingebbare Variablen):
  Sie sind standardmäßig vom Typ Text. Bei Verwendung in Methoden oder Objektmethoden haben sie den von Ihnen zugewiesenen Typ. Ein Datentypkonflikt ergibt sich nur, wenn der Name einer Variablen dieses Typs mit einer an anderer Stelle gesetzten Variablen identisch ist.
- **Anzeigevariablen** (DropDown Listen, Menüs/Dropdown Listen, rollbare Bereiche, PopUp-Menüs, Combo Boxen, Registerkarten und Spalten von Listboxen):
  Einige dienen zur Anzeige von Arrays in Formularen. Wurden im Formulareditor Standardwerte eingetragen, müssen Sie die dazugehörigen Variablen mit den Befehlen zum Deklarieren der Arrays explizit deklarieren, also z.B. **ARRAY TEXT**, **ARRAY BOOLEAN**, etc.

Listboxen

Jede Listbox fügt mehrere Variablen in den Formularen hinzu. Der Standardtyp dieser Variablen richtet sich nach dem Typ der Listbox:

	Listboxen vom Typ Array	**Listboxen vom Typ Auswahl**
Listbox	Array Boolean	Zahl (nicht verwendet)
Spalte der Listbox	Array Text	Dynamisches Typisieren
Kopfteil	Zahl	Zahl
Fußteil	Dynamisches Typisieren	Dynamisches Typisieren
Array Zeilenkontrolle	Array Boolean (Array Lange Ganzzahl akzeptiert)	-
Stilarten	Array Lange Ganzzahl	Lange Ganzzahl
Schriftfarbe	Array Lange Ganzzahl	Lange Ganzzahl
Hintergrundfarbe	Array Lange Ganzzahl	Lange Ganzzahl

Sorgen Sie für die korrekte Identifizierung und Typisierung dieser Variablen und Arrays, um während dem Kompilieren Konflikte beim Generieren zu vermeiden.

Zeiger

Enthält Ihre Datenbank Zeiger, nutzen Sie die Vorteile eines leistungsstarken und ausgeklügelten Werkzeugs von 4D. Der Compiler bewahrt alle Vorzüge von Zeigern.
Ein Zeiger kann auf Variablen mit verschiedenen Datentypen verweisen. Sie vermeiden Konflikte, wenn Sie einer Variablen nicht verschiedene Datentypen zuweisen. Achten Sie darauf, dass der Datentyp einer Variablen, auf die sich ein Zeiger bezieht, nicht verändert wird.
Hierzu ein Beispiel:

```
Variable:=5.3
Zeiger:=->Variable
Zeiger->:=6.4
Zeiger->:=Falsch
```

In diesem Fall ist der dereferenzierte Zeiger eine Variable vom Typ Zahl. Weisen Sie diesem einen Wert vom Typ Boolean zu, erzeugen Sie einen Datentypkonflikt.
Benötigen Sie Zeiger in derselben Methode für verschiedene Zwecke, achten Sie auf die korrekte Definition:

```
Variable:=5.3
Zeiger:=->Variable
Zeiger->:=6.4
Bool:=Wahr
Zeiger:=->Bool
Zeiger->:=Falsch
```

Ein Zeiger wird immer in Bezug auf das Objekt definiert, auf welches er zeigt. Deshalb kann der Compiler vom Zeiger verursachte Datenkonflikte nicht finden. Bei einem Konflikt erhalten Sie keine Fehlermeldung während der Typisierungs- oder Kompilierungsphase.
Das bedeutet jedoch nicht, dass der Compiler Konflikte im Zusammenhang mit Zeigern nicht finden kann. Er kann die Verwendung von Zeigern prüfen, wenn Sie in den Einstellungen zur Kompilierung die Option **Bereichsprüfung** markiert haben. Weitere Informationen dazu finden Sie im Abschnitt **Bereichsprüfung** des Handbuchs *4D Designmodus*.

Plug-In Befehle

Allgemein

Der Compiler analysiert während dem Kompilieren die Definitionen der in der Datenbank verwendeten Plug-In Befehle, z.B. Anzahl und Typ ihrer Parameter. Beim Typisieren treten keine Verwechslungen auf, wenn Ihre Aufrufe mit der Deklaration der Methode übereinstimmen.

Stellen Sie sicher, dass Ihre Plug-Ins im Ordner *PlugIns* installiert sind und zwar an die von 4D zugelassene Stelle: Neben die Strukturdatei der Datenbank, neben das ausführbare Programm (Windows) / in das Software Paket (Mac OS). Zur Wahrung der Kompatibilität kann auch noch der Ordner *Win4DX* bzw. *Mac4DX* neben der Strukturdatei verwendet werden. Weitere Informationen dazu finden Sie im *4D Installationshandbuch*.
Der Compiler dupliziert diese Dateien nicht, prüft jedoch, ob die dazugehörigen Routinen korrekt deklariert sind.
Liegen Ihre Plug-Ins an anderer Stelle, fordert der Compiler Sie auf, diese während der Typisierung über den Öffnen-Dialog zu setzen.

Plug-In Befehle mit spezifischen Parametern

Bestimmte Plug-Ins, zum Beispiel 4D Write enthalten Befehle, die indirekt Befehle von 4D aufrufen.
Die Syntax für den 4D Write Befehl lautet:

***WR ON EVENT**(area;event;eventMethod)*

Der letzte Parameter ist der Name der Methode, die Sie in 4D erstellt haben. Diese Methode ruft 4D Write immer auf, wenn das Ereignis empfangen wird. Es empfängt automatisch die nachfolgenden Parameter:

Parameter	Typ	Beschreibung
$0	Lange Ganzzahl	Zurückgegebene Funktion
$1	Lange Ganzzahl	4D Write Bereich
$2	Lange Ganzzahl	Umschalttaste
$3	Lange Ganzzahl	Alt-Taste (Windows); Wahltaste (Mac OS)
$4	Lange Ganzzahl	Strg-Taste (Windows), Befehlstaste (Mac OS)
$5	Lange Ganzzahl	Ereignistyp
$6	Lange Ganzzahl	Wert richtet sich nach Parameter des Ereignisses

Der Compiler kann diese Parameter nur berücksichtigen, wenn sie entweder über eine Compiler Direktive oder durch ihre Verwendung in der Methode typisiert sind. Kommen sie in einem Prozess vor, muss ihre Verwendung so klar sein, dass sich der Typ eindeutig ableiten lässt.

4D Komponenten

Mit 4D können Sie 4D Komponenten erstellen und verwalten. Eine 4D Komponente besteht aus einer Reihe von 4D Objekten, mit einer oder mehreren Funktionalitäten, die sich in verschiedenen Datenbanken, genannt Host Datenbank installieren lässt.
Eine Host Datenbank kann im interpretierten Modus sowohl interpretierte als kompilierte Komponenten verwenden. Sie können in derselben Host Datenbank interpretierte und kompilierte Komponenten installieren.
Dagegen kann eine Host Datenbank im kompilierten Modus keine interpretierten Komponenten verwenden. Hier sind nur kompilierte Komponenten möglich.
Eine interpretierte Datenbank mit interpretierten Komponenten ist kompilierbar, wenn sie keine Methoden von interpretierten Komponenten aufruft. Sonst erscheint beim Wählen des Menübefehls Compiler eine Warnung und Kompilieren ist nicht möglich.
Ein Namenskonflikt kann auftreten, wenn eine gemeinsam genutzte Projektmethode in einer Komponente denselben Namen wie eine Projektmethode der Host Datenbank hat. In diesem Fall wird beim Ausführen des Code in der Host Datenbank die Methode der Host Datenbank aufgerufen. Es ist also möglich, die Methode einer Komponente mit einer eigenen Methode zu verdecken, z.B. um eine andere Funktionalität zu erhalten
Wird der Code in der Komponente ausgeführt, wird die Methode der Komponente aufgerufen. Diese Überdeckung wird in einer Warnung im Kompilierungsereignis der Host Datenbank angezeigt.

Nutzen zwei Komponenten gemeinsam Methoden mit demselben Namen, wird beim Kompilieren der Host Datenbank ein Fehler erzeugt.

Weitere Informationen dazu finden Sie im Kapitel **4D Komponenten entwickeln und installieren** des Handbuchs *4D Designmodus*.

Lokale Variablen $0...$N und Parameterübergabe

Für lokale Variablen gelten alle bereits genannten Regeln. Wie für andere Variablen lässt sich der Datentyp nicht abändern, während eine Methode ausgeführt wird. Im folgenden sehen Sie zwei Vorgänge, die zu Datentypkonflikten führen können:

- Sie müssen aus bestimmten Gründen retypisieren. Hier helfen Zeiger, Datentypkonflikte zu vermeiden.
- Sie müssen Parameter über Indirektion ansprechen.

Zeiger zur Vermeidung von Retypisieren verwenden

Eine Variable lässt sich nicht retypisieren. Sie können sich jedoch über Zeiger auf Variablen mit unterschiedlichen Datentypen beziehen.
Nehmen wir als Beispiel eine Funktion, die die Speichergröße eines eindimensionalen Array zurückgibt. Das Ergebnis ist ein Wert vom Typ Zahl; davon ausgenommen sind Arrays vom Typ Text und vom Typ Bild. Hier richtet sich die Speichergröße nach Werten, die sich nicht als Zahl ausdrücken lassen. Weitere Informationen dazu finden Sie im Abschnitt **Arrays und Speicher**.
Für Arrays vom Typ Text bzw. Bild ist das zurückgegebene Ergebnis eine Zeichenkette. Diese Funktion benötigt als Parameter einen Zeiger auf das Array, dessen Speichergröße wir wissen wollen.
Hierfür gibt es zwei Möglichkeiten:

- Sie arbeiten mit lokalen Variablen, ohne sich um die Datentypen zu kümmern; in diesem Fall läuft die Methode nur im interpretierten Modus.
- Sie verwenden Zeiger und arbeiten im interpretierten oder kompilierten Modus.

Funktion *MemSize* nur im interpretierten Modus (Beispiel für Mac OS)

```
$Size:=Size of array($1->)
$Type:=Type($1->)

Case of
   :($Type=Real array)
      $0:=8+($Size*10)  ` $0 ist eine Zahl
      VarNum:=8+($Size*4)
   :($Type=Integer array)
      $0:=8+($Size*2)
   :($Type=LongInt array)
      $0:=8+($Size*4)
   :($Type=Date array)
      $0:=8+($Size*6)
   :($Type=Text array)
```

```
        $0:=String(8+($Size*4))+("+Sum of text lengths")  ` $0 ist ein Text
    :($Type=Picture array)
        $0:=String(8+($Size*4))+("+Sum of picture sizes")  ` $0 ist ein Text
    :($Type=Pointer array)
        $0:=8+($Size*16)
    :($Type=Boolean array)
        $0:=8+($Size/8)
End case
```

In der obigen Methode ändert sich der Datentyp von $0 entsprechend auf den Wert von $1; von daher ist er nicht mit dem Compiler kompatibel.

Funktion *MemSize* im interpretierten und kompilierten Modus (Beispiel für Mac OS)
Hier wird die Methode mit Zeigern geschrieben:

```
$Size:=Size of array($1->)
$Type:=Type($1->)
VarNum:=0
Case of
    :($Type=Real array)
        VarNum:=8+($Size*10)  ` VarNum ist eine Zahl
    :($Type=Integer array)
        VarNum:=8+($Size*2)
    :($Type=LongInt array)
        VarNum:=8+($Size*4)
    :($Type=Date array)
        VarNum:=8+($Size*6)
    :($Type=Text array)
        VarText:=String(8+($Size*4))+("+Sum of text lengths")
    :($Type=Picture array)
        VarText:=String(8+($Size*4))+("+Sum of picture sizes")
    :($Type=Pointer array)
        VarNum:=8+($Size*16)
    :($Type=Boolean array)
        VarNum:=8+($Size/8)
End case
If(VarNum#0)
    $0:=->VarNum
Else
    $0:=->VarText
End if
```

Die wichtigsten Unterschiede zwischen beiden Funktionen sind:

- Im ersten Fall ist das Funktionsergebnis die erwartete Variable,
- Im zweiten Fall ist das Funktionsergebnis ein Zeiger auf diese Variable. Sie dereferenzieren einfach Ihr Ergebnis.

Parameter Indirektion

Der Compiler verwaltet die Leistungsstärke und Flexibilität der Parameter Indirektion. Im interpretierten Modus können Sie frei wählen bei der Anzahl und den Datentypen der Parameter. Das gilt auch für den kompilierten Modus, vorausgesetzt, Sie verursachen keine Datentypkonflikte und verwenden nicht mehr Parameter, als Sie in der aufrufenden Methode übergeben haben.
Um Konflikte zu vermeiden, sollten Parameter, die über Indirektion angesprochen werden, alle vom selben Datentyp sein.
Die Indirektion funktioniert am besten, wenn Sie folgende Regel beachten: Werden nur einige der Parameter über Indirektion angesprochen, sollten Sie nach den fest vorgegebenen Parametern stehen
Innerhalb der Methode hat eine Indirektion das Format ${$i}, wobei $i eine numerische Variable ist, ${$i} der Parameter für die Indirektion.

Nehmen wir als Beispiel eine Funktion, die Werte hinzufügt und deren Summe in einem Format zurückgibt, das als Parameter übergeben wurde. Immer wenn diese Methode aufgerufen wird, kann die Anzahl der hinzugefügten Werte anders sein. Sie müssen die Werte als Parameter in der Methode übergeben, das Format als Zeichenkette. Die Anzahl der Werte kann von Aufruf zu Aufruf variieren. Die Funktion wird folgendermaßen aufgerufen:

```
Result:=MySum("##0.00";125,2;33,5;24)
```

In diesem Fall erhält die aufrufende Methode den String "182.70", das ist die Summe der Zahlen im angegebenen Format. Die Parameter der Funktion müssen in der richtigen Reihenfolge übergeben wrden, d.h. zuerst das Format, dann die Werte.
Die Funktion mit Namen *MySum* lautet:

```
$Sum:=0
For($i;2;Count parameters)
    $Sum:=$Sum+${$i}
End for
$0:=String($Sum;$1)
```

Diese Funktion können Sie nun auf verschiedene Arten aufrufen:

```
Result:=MySum("##0.00";125,2;33,5;24)
Result:=MySum("000";1;18;4;23;17)
```

Analog zu anderen lokalen Variablen ist es nicht notwendig, generische Parameter über eine Compiler Direktive zu deklarieren. Bei Bedarf, d.h. bei Zweideutigkeit oder zur Optimierung erfolgt dies mit folgender Syntax:

```
C_LONGINT($4)
```

Dieser Befehl bedeutet, dass alle Parameter ab dem vierten (einschließlich) über Indirektion angesprochen werden und vom Datentyp Lange Ganzzahl sind. $1, $2 und $3 können einen beliebigen Datentyp haben. Verwenden Sie dagegen $2 über Indirektion, wird der generische Datentyp verwendet, d.h. Lange Ganzzahl, auch wenn Sie dafür als Datentyp Zahl festgelegt hatten.

Hinweis: Der Compiler verwendet diesen Befehl während der Typisierung. Die Zahl in der Deklaration muss eine Konstante und keine Variable sein.

Reservierte Variablen und Konstanten

Einige Variablen und Konstanten in 4D erhalten Datentyp und Kennung über den Compiler. Von daher können Sie mit diesen Variablen- bzw. Konstantennamen weder neue Variablen, Methoden noch Funktionen oder Plug-In Befehle erstellen. Sie können sie jedoch wie im interpretierten Modus verwenden und ihre Werte testen.

Systemvariablen

4D enthält folgende **Systemvariablen**:

Variable	Typ
OK	Lange Ganzzahl
Document	String (255)
FldDelimit	Lange Ganzzahl
RecDelimit	Lange Ganzzahl
Error	Lange Ganzzahl
Error method	Text
Error line	Lange Ganzzahl
Error formula	Text
MouseDown	Lange Ganzzahl
KeyCode	Lange Ganzzahl
Modifiers	Lange Ganzzahl
MouseX	Lange Ganzzahl
MouseY	Lange Ganzzahl
MouseProc	Lange Ganzzahl

Variablen im Schnellbericht

Erstellen Sie in einem Bericht berechnete Spalten, legt 4D automatisch eine Variable C1 für die erste, C2 für die zweite, usw. an. Verwenden Sie diese Variablen in Methoden, bedenken Sie, dass C1, C2…Cn sich wie andere Variablen nicht retypisieren lassen.

Vordefinierte Konstanten in 4D

Die vollständige Liste vordefinierter Konstanten in 4D finden Sie unter **Konstantenthemen**. 4D Konstanten erscheinen auch im Designmodus im Explorer (siehe **Seite Konstanten** im Handbuch *4D Designmodus*).

Einzelheiten zur Syntax

Der Compiler geht davon aus, dass die üblichen Syntaxregeln für 4D Befehle angewandt werden. Für Datenbanken, die kompiliert werden sollen, sind keine besonderen Änderungen erforderlich.

Dieser Abschnitt weist auf bestimmte Eigenheiten und spezifische Details hin:

- Einige Befehle, die den Datentyp einer Variablen beeinflussen, können bei Unachtsamkeit zu Datenkonflikten führen.
- Da manche Befehle verschiedene Arten von Syntax oder Parameter zulassen, ist es von Vorteil, zu wissen, welche Variante sich am besten eignet.

Strings

Ascii (Zeichen)

Für Befehle, die mit Strings arbeiten, benötigt nur die Funktion **Character code** besondere Aufmerksamkeit. Im interpretierten Modus können Strings Zeichen beinhalten oder leer sein.
Im kompilierten Modus können Sie keinen leeren String übergeben.
Übergeben Sie einen leeren String und ist das in **Character code** übergebene Argument eine Variable, kann der Compiler beim Kompilieren keinen Fehler entdecken.

Kommunikation

SEND Variable(Variable)
RECEIVE VARIABLE(Variable)
Diese beiden Befehle dienen zum Schreiben und Empfangen von Variablen, die an die Festplatte gesendet wurden. Variablen werden in diesen Befehlen als Parameter übergeben.
Der übergebene Parameter muss immer vom selben Datentyp sein. Nehmen wir an, Sie wollen eine Liste mit Variablen an eine Datei senden. Um auszuschließen, dass der Datentyp versehentlich geändert wird, empfehlen wir, den Datentyp der Variablen in der Titelzeile der Liste anzugeben. So erhalten Sie beim Empfangen der Variablen zu Beginn immer einen Indikator. Rufen Sie dann den Befehl **RECEIVE VARIABLE** auf, wird der Transfert über eine *Case of* Anweisung ausgeführt.

Beispiel:

```
SET CHANNEL(12;"Datei")
If(OK=1)
   $Type:=Type([Client]Total_TO)
   SEND VARIABLE($Type)
   For($i;1;Records in selection)
      $Send_TO:=[Client]Total_TO
      SEND VARIABLE($Send_TO)
   End for
End if
SET CHANNEL(11)
SET CHANNEL(13;"MeineDatei")
If(OK=1)
   RECEIVE VARIABLE($Type)
   Case of
      :($Type=ls string var)
         RECEIVE VARIABLE($String)
  `Empfangene Variable bearbeiten
      :($Type=ls real)
         RECEIVE VARIABLE($Real)
  `Empfangene Variable bearbeiten
      :($Type=ls text)
         RECEIVE VARIABLE($Text)
  `Empfangene Variable bearbeiten
   End case
End if
SET CHANNEL(11)
```

Strukturzugriff

Field (field pointer) oder **(table number;field Numerisch)**
Table(table pointer) oder **(table Numerisch)** oder **(field pointer)**
Diese beiden Funktionen geben Werte mit verschiedenen Datentypen zurück, abhängig von den übergebenen Parametern:

- Übergeben Sie der Funktion **Table** einen Zeiger, ist das Ergebnis vom Typ Zahl

- Übergeben Sie der Funktion **Table** eine Zahl, wird als Ergebnis ein Zeiger zurückgegeben.
  Beide Funktionen reichen für den Compiler nicht aus, um den Datentyp des Ergebnisses zu bestimmen. Verwenden Sie in solchen Fällen eine Compiler Direktive, um jegliche Zweideutigkeit auszuschließen.

Dokumente

Beachten Sie, dass Referenzen auf Dokumente, die von den Funktionen **Open document**, **Append document** und **Create document** zurückgegeben werden, vom Typ Zeit sind.

Math

Mod (value;divider)
Der Ausdruck "25 modulo 3" lässt sich auf zwei Arten in 4D darstellen:

```
Variable:=Mod(25;3)
```

oder

```
Variable:=25%3
```

Der Compiler erkennt den Unterschied zwischen ihnen: *Mod* gilt für alle Zahlen, der Operator % dagegen nur für Ganzzahlen und Lange Ganzzahlen. Übersteigt der Operand die Reichweite des Typs Lange Ganzzahl, ist das zurückgegebene Ergebnis falsch.

Ausnahmen

IDLE
ON EVENT CALL (Methode{; ProzessName})
ABORT

ON EVENT CALL

Der Befehl **IDLE** wurde in die Programmiersprache von 4D integriert, um Ausnahmen zu verwalten. Verwenden Sie diesen Befehl in Verbindung mit dem Befehl **ON EVENT CALL**.

Sie können ihn als Direktive zur Ereignisverwaltung verwenden.
Nur der Kernel von 4D kann ein Systemereignis ausfindig machen. (Mausklick, Aktivität auf Tastatur, usw.). In den meisten Fällen werden Aufrufe des Kernel vom kompilierten Code selbst initiiert, in einer für den Benutzer transparenten Weise.

Wartet 4D dagegen passiv auf ein Ereignis –– zum Beispiel in einer Warteschleife –– ist klar, dass es keinen Aufruf geben wird.

Beispiel unter Windows

```
  `Methode Mausklick
If(MouseDown=1)
   <>vTest:=True
   ALERT("Jemand hat die Maustaste betätigt")
End if

  `Methode Warten
<>vTest:=False
ON EVENT CALL("Mausklick")
While(<>vTest=False)
  `Warteschleife für Ereignis
End while
ON EVENT CALL("")
```

In diesem Fall fügen Sie den Befehl **IDLE** folgendermaßen ein:

```
  `Methode Warten
<>vTest:=False
ON EVENT CALL("Mausklick")
While(<>vTest=False)
   IDLE
  `Kernel Aufruf, um ein Ereignis aufzuspüren
End while
ON EVENT CALL("")
```

ABORT

Verwenden Sie diesen Befehl nur in Projektmethoden zur Fehlerverwaltung. Er arbeitet genauso wie in 4D, außer in einer Methode, die von folgenden Befehlen aufgerufen wurde: **EXECUTE FORMULA**, **APPLY TO SELECTION** oder **_o_APPLY TO SUBSELECTION**. Versuchen Sie, derartige Situationen zu vermeiden.

Arrays

Der Compiler bestimmt den Datentyp eines Array über folgende 4D Befehle:

COPY ARRAY(Quelle;Ziel)
SELECTION TO ARRAY(Feld;Array)
ARRAY TO SELECTION(Array;Feld)
SELECTION RANGE TO ARRAY(Start;Ende;Feld;Array)
LIST TO ARRAY(Auswahlliste;Array{; itemRefs})
ARRAY TO LIST(Array;Auswahlliste{; itemRefs})
DISTINCT VALUES(Feld;Array)

COPY ARRAY

Dieser Befehl akzeptiert zwei Arten von Array-Parametern. Ist einer der Array-Parameter nicht an anderer Stelle deklariert, bestimmt der Compiler den Datentyp des nicht deklarierten Array anhand des deklarierten Typs.
Die Ableitung erfolgt auf zwei Arten:

- Das typisierte Array ist der erste Parameter. Der Compiler weist den Datentyp des ersten Array in dem zweiten Array zu.
- Das deklarierte Array ist der zweite Parameter. Der Compiler weist hier den Datentyp des zweiten Array dem ersten zu.

Da der Compiler bei Datentypen strikt ist, lässt sich **COPY ARRAY** nur von einem Array eines bestimmten Datentyps zu einem Array desselben Typs ausführen.
Wollen Sie also ein Array mit Elementen kopieren, die einen ähnlichen Datentyp haben, z.B. Ganzzahl, Lange Ganzzahl und Zahl oder Text und String oder Strings mit unterschiedlichen Längen, müssen Sie die Elemente einzeln kopieren.

Nehmen wir an, Sie wollen Elemente von einem Array Typ Ganzzahl zu einem Array Typ Zahl kopieren. Gehen Sie folgendermaßen vor:

```
$Size:=Size of array(ArrInt)
ARRAY REAL(ArrReal;$Size)
  `Setzen Sie für das Array Typ Zahl dieselbe Größe wie für das Array Typ Ganzzahl
For($i;1;$Size)
   ArrReal{$i}:=ArrInt{$i}
  `Kopieren Sie jedes Element
End for
```

Bedenken Sie, dass Sie während dem Prozess die Anzahl der Dimensionen eines Array nicht verändern können. Kopieren Sie ein eindimensionales Array in ein zweidimensionales Array, erzeugt der Compiler eine Fehlermeldung.

SELECTION TO ARRAY, ARRAY TO SELECTION, DISTINCT VALUES, SELECTION RANGE TO ARRAY

Diese Befehle müssen im interpretierten Modus nicht deklariert werden. Ein nicht typisiertes Array erhält den Datentyp des Feldes, das im Befehl angegeben ist.
Schreiben Sie:

```
SELECTION TO ARRAY([MyTable]IntField;MyArray)
```

ist *MyArray* eindimensional mit dem Typ Lange Ganzzahl (in der Annahme, dass *IntField* vom Typ Ganzzahl ist).

Wurde das Array typisiert, stellen Sie sicher, dass das Datenfeld vom selben Datentyp ist. Ganzzahl, Lange Ganzzahl und Zahl sind zwar ähnliche Typen, jedoch nicht vollkommen gleich. Bei den Datentypen Text und String gibt es dagegen mehr Spielraum. Wurde ein Array nicht vorab typisiert und wenden Sie einen Befehl an, der als Parameter ein Feld vom Typ String enthält, wird dem Array standardmäßig als Typ Text zugewiesen. Wurde das Array zuvor als String bzw. Text typisiert, befolgen diese Befehle Ihre Direktiven.

Dasselbe gilt für Felder vom Typ Text –– Ihre Direktiven haben Vorrang.
Bedenken Sie, dass Sie die Befehle **SELECTION TO ARRAY**, **SELECTION RANGE TO ARRAY**, **ARRAY TO SELECTION** und **DISTINCT VALUES** nur in eindimensionalen Arrays verwenden können.

Der Befehl **SELECTION TO ARRAY** hat auch eine zweite Syntax, nämlich:
SELECTION TO ARRAY(Tabelle;Array).
In diesem Fall ist die Variable *MyArray* ein Array vom Typ Lange Ganzzahl. Dasselbe gilt auch für den Befehl **SELECTION RANGE TO ARRAY**.

LIST TO ARRAY, ARRAY TO LIST

Für die Befehle **LIST TO ARRAY** und **ARRAY TO LIST** treffen nur zwei Arten von Arrays zu:

- Eindimensionales Array vom Typ String und
- Eindimensionales Array vom Typ Text.
  Für diese Befehle muss das als Parameter übergebene Array nicht typisiert sein. Es wird standardmäßig als Array vom Typ Text typisiert. Wurde es zuvor als String bzw. Text typisiert, befolgen diese Befehle Ihre Direktiven.

Zeiger in Befehlen mit Arrays

Der Compiler kann keinen Datentyp aufspüren, wenn Sie in einem Befehl mit Array Deklaration einen dereferenzierten Zeiger als Parameter übergeben. Schreiben Sie:

```
SELECTION TO ARRAY([Table]Field;Pointer->)
```

wobei Pointer-> für ein Array steht, kann der Compiler nicht prüfen, ob Feldtyp und Array-Typ identisch sind. Solche Konflikte müssen Sie selbst ausschliessen, indem Sie das Array typisieren, auf das sich der Zeiger bezieht.

Der Compiler zeigt eine Warnung, wenn er eine Anweisung mit einer Array-Deklaration findet, in der ein Parameter ein Zeiger ist. Diese Meldungen sind hilfreich beim Suchen solcher Datentypkonflikte.

Lokale Arrays

Verwendet Ihre Datenbank lokale Arrays, d.h. sie werden nur in der Methode erkannt, in welcher sie erstellt wurden, müssen diese vor der Verwendung ausdrücklich in 4D typisiert werden.
Dazu verwenden Sie die Befehle für Arrays, z.B. **ARRAY REAL**, **ARRAY INTEGER**, etc.

Erstellt eine Methode z.B. ein lokales Array vom Typ Ganzzahl mit 10 Elementen, müssen Sie zuvor das Array wie folgt typisieren:

```
ARRAY INTEGER($MyArray;10)
```

Programmiersprache

Get pointer(varName)
Typ (Objekt)
EXECUTE(statement)
TRACE
NO TRACE

Get pointer

Die Funktion **Get pointer** gibt einen Zeiger auf den ihm übergebenen Parameter zurück. Nehmen wir an, Sie möchten ein Array mit Zeigern initialisieren. Jedes Element im Array zeigt auf eine bestimmte Variable. Unser Beispiel enthält 12 Variablen mit Namen V1, V2, …V12. Sie können schreiben:

```
ARRAY POINTER(Arr;12)
Arr{1}:=->V1
Arr{2}:=->V2

Arr{12}:=->V12
```

Sie können auch schreiben:

```
ARRAY POINTER(Arr;12)
For($i;1;12)
   Arr{$i}:=Get pointer("V"+String($i))
End for
```

Am Ende dieser Operation enthalten Sie ein Array mit Zeigern, in dem jedes Element auf eine Variable Vi zeigt.

Sie können beide Sequenzen kompilieren. Werden die Variablen V1 bis V12 jedoch nicht in der Datenbank an anderer Stelle ausdrücklich verwendet, kann der Compiler sie nicht typisieren. Sie müssen sie also explizit an anderer Stelle verwenden bzw. typisieren. Es gibt zwei Möglichkeiten:

- Sie typisieren V1, V2, …V12 über eine Compiler Direktive:

```
C_LONGINT(V1;V2;V3;V4;V5;V6;V7;V8;V9;V10;V11;V12)
```

- Sie weisen diese Variablen in einer Methode zu:

```
V1:=0
V2:=0

V12:=0
```

Typ (Objekt)

Da jede Variable in einer kompilierten Datenbank nur einen Datentyp hat, scheint diese Funktion überflüssig. Der Einsatz von Zeigern ist jedoch hilfreich, z.B. wenn Sie den Datentyp der Variablen benötigen, auf die sich ein Zeiger bezieht. Da Zeiger flexibel sind, können Sie nicht immer sicher sein, auf welche Objekte sie jeweils zeigen.

EXECUTE FORMULA

Dieser Befehl bietet im interpretierten Modus Vorteile, die nicht in den kompilierten Modus übernommen werden.
Hier wird ein Methodenname, der diesem Befehl als Parameter übergeben wird, interpretiert. Deshalb vermissen Sie einige der Vorteile, die der Compiler bietet, die Syntax Ihrer Parameter lässt sich nicht überprüfen.
Außerdem können Sie keine lokalen Variablen als Parameter übergeben.
Sie können diesen Befehl durch eine Reihe von Anweisungen ersetzen. Hierzu zwei Beispiele:

Wir gehen von folgender Sequenz aus:

```
i:=FormFunc
EXECUTE FORMULA("INPUT FORM (Form"+String(i)+")")
```

Sie lässt sich ersetzen durch:

```
i:=FormFunc
VarForm:="Form"+String(i)
INPUT FORM(VarForm)
```

Anderes Beispiel:

```
$Num:=SelPrinter
EXECUTE FORMULA("Print"+$Num)
```

Hier lässt sich **EXECUTE FORMULA** ersetzen durch *Case of*:

```
Case of
   :($Num=1)
      Print1
   :($Num=2)
      Print2
   :($Num=3)
      Print3
End case
```

Der Befehl **EXECUTE FORMULA** lässt sich immer ersetzen. Da die auszuführende Methode aus der Liste der Projektmethoden in der Datenbank oder den 4D Befehlen ausgewählt wird, gibt es eine feste Anzahl an Methoden. Folglich lässt sich **EXECUTE FORMULA** immer durch eine Anweisung *Case of* oder einen anderen Befehl ersetzen. Dadurch wird Ihr Code auch rascher ausgeführt.

TRACE, NO TRACE

Diese beiden Befehle werden beim Debuggen verwendet. Sie haben in einer kompilierten Datenbank keine Funktion. Sie können sie jedoch in Ihren Methoden beibehalten, der Compiler ignoriert sie einfach.

Variablen

Undefined(Variable)
SAVE VARIABLES(document;variable1{; variable2...})
LOAD VARIABLES(document;variable1{; variable2...})
CLEAR Variable(Variable)

Undefined

Da der Compiler die Typisierung ausführt, kann eine Variable im kompilierten Modus nie undefiniert sein. Tatsächlich sind alle Variablen definiert, wenn die Kompilierung abgeschlossen ist. Von daher gibt die Funktion **Undefined** immer **Falsch** zurück, egal, welcher Parameter übergeben wurde.

Hinweis: Über die Funktion **Is compiled mode** können Sie feststellen, ob Ihre Anwendung im kompilierten Modus läuft.

SAVE VARIABLES, LOAD VARIABLES

Im interpretierten Modus können Sie prüfen, ob das Dokument vorhanden ist, indem Sie testen, ob eine der Variablen nach Ausführen des Befehls **LOAD VARIABLES** undefiniert ist. Das ist in einer kompilierten Datenbank nicht möglich, da die Funktion **Undefined** immer **Falsch** zurückgibt.

Diesen Test können Sie im interpretierten oder kompilierten Modus folgendermaßen durchführen:

1. Sie initialisieren die Variablen, die Sie von einem Wert empfangen, der kein legaler Wert für irgendeine der Variablen ist.
2. Sie vergleichen eine der empfangenen Variablen mit dem Wert der Initialisierung nach Ausführen von **LOAD VARIABLES**.

Sie schreiben folgende Methode:

```
Var1:="xxxxxx"
  `"xxxxxx" ist ein Wert, der nicht über LOAD VARIABLES zurückgegeben werden kann
Var2:="xxxxxx"
Var3:="xxxxxx"
Var4:="xxxxxx"
LOAD VARIABLES("Dokument";Var1;Var2;Var3;Var4)
If(Var1="xxxxxx")
  `Dokument nicht gefunden

Else
  `Dokument gefunden

End if
```

Variable CLEAR

Diese Routine verwendet im interpretierten Modus zwei unterschiedliche Syntaxarten:
CLEAR VARIABLE(variable)
CLEAR VARIABLE("a")
Im kompilierten Modus initialisiert die erste Syntax von **CLEAR VARIABLE**(variable) erneut die Variable (für Zahl Setzen auf Null; leerer String für Zeichenkette oder Text, etc.), da im kompilierten Modus keine Variable undefiniert sein kann.
Aus diesem Grund setzt **CLEAR VARIABLE** im kompilierten Modus keinen Speicher frei, mit Ausnahme von vier Fällen: Variablen vom Typ Text, Bild, BLOB und Array.
CLEAR VARIABLE bewirkt für ein Array dasselbe wie eine neue Deklaration des Array, dessen Größe auf Null gesetzt wird.

Für ein Array *MyArray*, dessen Elemente vom Typ *Ganzzahl* sind, bewirkt **CLEAR VARIABLE**(MyArray) dasselbe wie einer der folgenden Ausdrücke:

```
ARRAY INTEGER(MyArray;0)
  `Wenn es ein eindimensionales Array ist
ARRAY INTEGER(MyArray;0;0)
  `Wenn es ein zweidimensionales Array ist
```

Die zweite Syntax **CLEAR VARIABLE**("a") ist mit dem Compiler nicht kompatibel, da er auf Variablen über deren Adresse und nicht über deren Namen zugreift.

Zeiger mit bestimmten Befehlen

Zeiger mit bestimmten Befehlen

Nachfolgende Befehle haben folgendes gemeinsam: Sie erlauben einen optionalen ersten Parameter [Tabelle], der zweite Parameter kann ein Zeiger sein.

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	

Im kompilierten Modus ist es leicht, den optionalen Parameter [Tabelle] zurückzugeben. Wird dagegen in einem dieser Befehle als erster Parameter ein Zeiger übergeben, weiss der Compiler nicht, auf was sich der Zeiger bezieht; er behandelt ihn als Zeiger auf eine Tabelle.

Nehmen wir den Befehl **QUERY** mit folgender Syntax:
QUERY({table{;formula{;*}})
Das erste Element des Parameters *formula* muss ein Feld sein.
Schreiben Sie:

```
QUERY(PtrField->=True)
```

sucht der Compiler nach einem Symbol, das im zweiten Element ein Feld darstellt. Findet er das Zeichen "=", generiert er eine Fehlermeldung, da er den Befehl nicht mit einem Ausdruck identifizieren kann, der weiß, wie die Operation abläuft.

Schreiben Sie dagegen:

```
QUERY(PtrTable->;PtrField->=True)
```

oder

```
QUERY([Table];PtrField->=True)
```

vermeiden Sie jede Art von Zweideutigkeit.

Befehle mit Zeigern direkt verwenden

Beim Arbeiten mit Zeigern gibt es eine Besonderheit bei Befehlen, in denen der erste Parameter [Tabelle] und der zweite Parameter optional sind. Aus internen Gründen erlaubt der Compiler hier nicht, dass ein Befehl, der einen Zeiger zurückgibt, wie z.B. **Current form table**, direkt als Parameter übergeben wird. Es wird ein Fehler generiert.

Das ist zum Beispiel beim Befehl **FORM SCREENSHOT** der Fall. Nachfolgender Code funktioniert im interpretierten Modus, wird aber während dem Kompilieren zurückgewiesen:

```
 //löst Kompilierungsfehler aus
FORM SCREENSHOT(Current form table->;$formName;$myPict)
```

In diesem Fall können Sie eine Variable dazwischen setzen, damit der Compiler diesen Code korrekt bewertet:

```
 //Entsprechung kompilierbarer Code
C_POINTER($ptr)
$ptr:=Current form table
FORM SCREENSHOT($ptr->;$formName;$myPict)
```

Konstanten

Erstellen Sie Ihre eigenen 4DK# Ressourcen (Konstanten), stellen Sie sicher, dass Zahlen als Lange Ganzzahl (L) oder Zahl (R) und Zeichenketten als Strings (S) angelegt werden. Jeder andere Typ generiert eine Warnung.

Tipps zur Optimierung

Es gibt kein Patentrezept für eine gute Programmiermethode, wir können jedoch die Vorteile gut strukturierter Programme verdeutlichen. Eine gut strukturierte Datenbank erzielt beim Kompilieren viel bessere Ergebnisse als eine wenig durchdachte Datenbank. Schreiben Sie z.B. zum Verwalten von n Objektmethoden eine generische Methode, erhalten Sie sowohl im interpretierten als im kompilierten Modus ein besseres Ergebnis, als wenn Sie in Objektmethoden n-Mal dieselbe Anweisung ausführen.
Mit anderen Worten, die Qualität der Programmierung wirkt sich auf die Qualität des kompilierten Code aus.
Mit der Zeit können Sie Ihren 4D Code schrittweise verbessern. Je häufiger Sie den Compiler einsetzen, desto mehr Fehler können Sie korrigieren und so instinktiv zur effizientesten Lösung gelangen.
Im folgenden geben wir Ratschläge und nennen einige Tricks, die helfen, Zeit beim Ausführen einfacher wiederkehrender Tasks einzusparen.

Code ausführlich kommentieren

Bestimmte Programmiertechniken machen Ihren Code für Sie selbst und später für jemand anderen schwerer lesbar. Deshalb empfehlen wir, Ihre Methoden mit ausführlichen Kommentaren zu versehen. Umfangreiche Kommentare können interpretierte Datenbanken tendentiell verlangsamen, sie beeinträchtigen jedoch nicht die Ausführungszeit in einer kompilierten Datenbank.

Code über Compiler Direktiven optimieren

Über Compiler Direktiven können Sie Ihren Code beträchtlich beschleunigen. Typisieren Sie Variablen nicht gemäß ihrer Verwendung, wählt der Compiler den Datentyp mit der größtmöglichen Reichweite, um die Fehlerquote zu verringern. Typisieren Sie zum Beispiel nicht die Variable in der Anweisung: *Var:= 5*, verwendet der Compiler als Typ Zahl, auch wenn die Typisierung als Ganzzahl möglich wäre.

Numerische Variablen

Der Compiler gibt numerischen Variablen, die nicht über Compiler Direktiven kompiliert wurden, standardmäßig den Typ Zahl, wenn in den Datenbank-Eigenschaften nichts anderes festgelegt wurde. Berechnungen auf eine Zahl laufen jedoch langsamer als auf eine Lange Ganzzahl. Wenn Sie wissen, dass eine numerische Variable immer eine Ganzzahl ist, sollten Sie diese über die Compiler Direktive **C_LONGINT** deklarieren.
Eine gute Vorgehensweise ist zum Beispiel, Zähler in Schleifen als Ganzzahl zu deklarieren.

Einige Funktionen in 4D geben Ganzzahlen zurück (z.B. **Character code**, **Int**...). Weisen Sie das jeweilige Ergebnis einer nicht typisierten Variablen zu, typisiert der Compiler diese als Zahl und nicht als Ganzzahl. Achten Sie darauf, dass solche Variablen immer mit Compiler Direktiven deklariert werden, wenn Sie sicher sind, dass sie nur in einem Kontext verwendet werden.

Hier ein einfaches Beispiel für eine Funktion, die einen Random Wert mit einer bestimmten Reichweite zurückgibt:

```
$0:=Mod(Random;($2-$1+1))+$1
```

Das Ergebnis ist immer eine Ganzzahl. Bei der obigen Anweisung typisiert der Compiler $0 als Zahl und nicht als Ganzzahl oder Lange Ganzzahl. Von daher ist es besser, in die Methode eine Compiler Direktive aufzunehmen:

```
C_LONGINT($0)
$0:=Mod(Random;($2-$1+1))+$1
```

Dann wird der Parameter rascher zurückgegeben und benötigt weniger Speicherplatz.

Im folgenden Beispiel wurden zwei Variablen als Lange Ganzzahl deklariert:

```
C_LONGINT($var1;$var2)
```

eine dritte nicht typisierte Variable erhält die Summe aus den beiden ersten Variablen:

```
$var3:=$var1+$var2.
```

Der Compiler typisiert die dritte Variable $var3 als Zahl. Sie müssen sie ausdrücklich als Lange Ganzzahl deklarieren, damit das Ergebnis vom Typ Lange Ganzzahl ist.

Hinweis: Beachten Sie, wie 4D Ausdrücke berechnet. Erst erfolgt die Berechnung un dann wird das Ergebnis in der Zielvariablen zugewiesen. Im kompilierten Modus bestimmt nicht der Datentyp der Zielvariablen das berechnete Ergebnis, sondern die Datentypen der Operanden.

- Im folgenden Beispiel beruht die Berechnung auf Langen Ganzzahlen:

```
C_REAL($var3)
C_LONGINT($var1;$var2)
$var1:=2147483647
$var2:=1
$var3:=$var1+$var2
```

$var3 ist gleich –2147483648 im kompilierten und im interpretierten Modus. Das Minuszeichen beruht auf dem Überlauf, da der Bereich für Lange Ganzzahlen bei 2147483647 endet.

- In diesem Beispiel:

```
C_REAL($var3)
C_LONGINT($var1)
$var1:=2147483647
$var3:=$var1+1
```

betrachtet der Compiler den Wert 1 als Ganzzahl. Im kompilierten Modus ist $var3 gleich –2147483648, da die Berechnung auf Langen Ganzzahlen basiert. Im interpretierten Modus ist $var3 gleich 2147483648, da die Berechnung auf dem Typ Zahl beruht.

Schaltflächen sind spezifische Fälle für Zahl, die als Lange Ganzzahl deklariert werden können.

Strings

Wollen Sie den Wert eines Zeichens prüfen, vergleichen Sie ihn mit seinem Wert in **Character code** und nicht mit dem Zeichen selbst. Der reguläre Zeichenvergleich berücksichtigt nämlich alle Möglichkeiten, d.h. auch diakritische Zeichen, wie Akzente.

Besonderheiten bei Arrays, Zeigern, Variablen

Zweidimensionale Arrays

Die Abarbeitung zweidimensionaler Arrays läuft besser, wenn die zweite Dimension größer als die erste ist.

Beispiel: Das Array mit der Deklaration:

```
ARRAY INTEGER(Array;5;1000)
```

läuft besser ab, als das Array mit der Deklaration:

```
ARRAY INTEGER(Array;1000;5)
```

Felder

Müssen Sie für ein Feld mehrere Berechnungen ausführen, können Sie die Performance verbessern, wenn Sie den Wert dieses Feldes in einer Variablen speichern und die Berechnungen mit der Variablen und nicht mit dem Feld ausführen. Nehmen wir folgende Methode:

```
Case of
   :([Client]Dest="New York City")
      Transport:="Messenger"
   :([Client]Dest="Puerto Rico")
      Transport:="Air mail"
   :([Client]Dest="Overseas")
      Transport:="Express mail service"
   Else
      Transport:="Regular mail service"
End case
```

Diese Methode läuft schneller, wenn Sie schreiben:

```
$Dest:=[Client]Dest
Case of
   :($Dest="New York City")
      Transport:="Messenger"
   :($Dest="Puerto Rico")
      Transport:="Air mail"
   :($Dest="Overseas")
      Transport:="Express mail service"
   Else
      Transport:="Regular mail service"
End case
```

Zeiger

Analog zu Feldern laufen Operationen schneller, wenn Sie mit Variablen anstatt mit dereferenzierten Zeigern arbeiten. Müssen Sie mehrere Berechnungen auf eine Variable ausführen auf die sich ein Zeiger bezieht, sparen Sie Zeit, wenn Sie den dereferenzierten Zeiger in einer Variablen speichern.

Nehmen wir zum Beispiel den Zeiger *MyPtr*, der auf ein Feld oder eine Variable verweist. Sie wollen auf den Wert, auf den *MyPtr* verweist, verschiedene Tests durchführen. Sie können schreiben:

```
Case of
   :(MyPtr->=1)
      Sequenz 1
   :(MyPtr->=2)
```

```
        Sequenz 2

    End case
```

Die Tests laufen schneller ab, wenn Sie schreiben:

```
Temp:=MyPtr->
Case of
   :(Temp=1)
      Sequenz 1
   :(Temp=2)
      Sequenz 2

   End case
```

Lokale Variablen

Verwenden Sie beim Aufbau Ihres Code möglichst häufig lokale Variablen. Die Vorteile liegen auf der Hand:

- Lokale Variablen benötigen weniger Speicherplatz beim Einsatz in einer Datenbank. Sie werden beim Aufrufen der Methode erstellt und wieder zerstört, wenn die Ausführung der Methode beendet ist.
- Der generierte Code ist für lokale Variablen optimiert, insbesondere beim Typ Lange Ganzzahl. Das ist hilfreich für Zähler in Schleifen.

Fehlermeldungen

Dieser Abschnitt beschreibt die vom Compiler generierten Meldungen. Es gibt folgende Arten:

- Warnungen, die helfen, gängige Stolperquellen zu vermeiden
- Fehler, die Sie selbst korrigieren müssen
- Meldungen bei der Bereichsprüfung, die innerhalb 4D erstellt werden

Warnungen

Diese Meldungen werden während der Kompilierung erstellt. Die zugrunde liegenden Probleme werden im folgenden anhand von Beispielen erklärt.

Zeiger in COPY ARRAY

```
COPY ARRAY(Pointer->;Array)
```

Zeiger in SELECTION TO ARRAY

```
SELECTION TO ARRAY(Pointer->;MyArray)
SELECTION TO ARRAY([MyTable]MyField;Pointer->)
```

Zeiger in ARRAY TO SELECTION

```
ARRAY TO SELECTION(Pointer->;[MyTable]MyField)
```

Zeiger in LIST TO ARRAY

```
LIST TO ARRAY(List;Pointer->)
```

Zeiger in ARRAY TO LIST

```
ARRAY TO LIST(Pointer->;List)
```

Zeiger in Array-Deklaration

```
ARRAY REAL(Pointer->;5)
```

Die Anweisung **ARRAY REAL**(Array;Pointer->) generiert nicht diese Warnung. Der Wert einer Dimension eines Array hat keinen Einfluss auf seinen Typ. In diesem Beispiel muss das Array, auf welches sich der Zeiger bezieht, an anderer Stelle definert worden sein.

Zeiger in DISTINCT VALUES

```
DISTINCT VALUES(Pointer->;Array)
```

Die Verwendung der Funktion **Undefined** wird nicht empfohlen.

```
If(Undefined(Variable))
```

Die Funktion **Undefined** gibt in einer kompilierten Datenbank immer FALSCH zurück.

Diese Methode ist mit einem Kennwort geschützt.

Eine Schaltfläche mit automatischer Aktion hat keinen Namen im Formular *MyForm* auf Seite X. Um Konflikte zu vermeiden, sollten alle Schaltflächen Namen haben.

Nimmt an, dass der Zeiger auf einen alphanumerischen Ausdruck zeigt.

```
Pointer->≤2≥:="a"
```

Nimmt an, dass der Stringindex numerisch ist.

```
String≤Pointer->≥:="a"
```

Nimmt an, dass der Arrayindex vom Typ Zahl ist.

```
ALERT(MyArray{Pointer->})
```

Fehlender Parameter im Prozessaufruf des Plug-In.

```
WR SET FONT(Area)
```

Hinweis: Mit nachfolgende Tags können Sie Warnungen individuell aktivieren und deaktivieren:
//%W-warning_number, um eine Warnung zu deaktivieren.
//%W+warning_number, um eine Warnung zu aktivieren.
Warnungen auf diese Weise zu aktivieren bzw. deaktivieren, wirkt sich auf Code aus, der im Kompilierungsplan sukzessive durchlaufen wird. Wollen Sie Warnungen generell aktivieren bzw. deaktivieren, können Sie einfach das entsprechende Tag in eine Methode mit Namen "Compiler_xxx" einfügen. Diese Methoden werden vom Kompiler als erste durchlaufen. Um z.B. die Warnung "Zeiger in COPY ARRAY" zu deaktivieren, können Sie das Tag "//%W-518.1" an der entsprechenden Stelle einfügen.

Fehlermeldungen

Diese Meldungen werden während der Kompilierung generiert. Sie müssen die zugrundeliegenden Fehler korrigieren, damit der Compiler eine kompilierte Datenbank erstellen kann. Die Fehler werden im folgenden anhand von Beispielen erklärt.

Typisierung
Der Variablentyp ist mit dem Operator nicht kompatibel. Die Zuweisung mit diesen Typen ist nicht möglich.

```
MyReal:=12.3
MyBoolean:=True
MyReal:=MyBoolean
```

Anzahl Dimensionen eines Array ändern

```
ARRAY TEXT(MyArray;5;5)
ARRAY TEXT(MyArray;5)
```

Typisierungskonflikt der Variablen MyArray im Formular.

```
ARRAY INTEGER(MyArray)
```

Ein Array ohne Dimensionen deklarieren

```
ARRAY INTEGER(MyArray)
```

Es wurde eine Variable erwartet.

```
COPY ARRAY(MyArray;"")
```

Der Typ der Variablen ist unbekannt. Diese Variable wird in Methode M1 verwendet.
Der Variablentyp lässt sich nicht bestimmen. Eine Compiler Direktive ist notwendig.
Unbekannter Konstantentyp

```
OK:="Das Wetter ist schön"
```

Die Methode M1 ist unbekannt.
Die Zeile ruft eine Methode auf, die es nicht bzw. nicht mehr gibt.
Inkorrekte Verwendung eines Feldes

```
MyDate:=Add to date(BooleanField;1;1;1)
```

Die Länge eines Strings kann nicht mehr als 255 Zeichen betragen.

```
C_STRING(325;MyString)
```

Die Variable VARIABLE ist keine Methode.

```
Variable(1)
```

Die Variable VARIABLE ist kein Array.

```
Variable{5}:=12
```

Das Ergebnis der Funktion ist mit diesem Ausdruck nicht kompatibel.

```
Text:="Number"+Num(i)
```

Die in diesem Ausdruck verwendeten Variablentypen sind nicht kompatibel.

```
Integer:=MyDate*Text
```

Änderung des Variablentyps $i vom Typ feste Länge in Zahl.

```
$i:="3"
$($i):=5
```

Index des Arrays ist keine Zahl.

```
IntArray{"3"}:=4
```

Retypisierung der Variable Variable vom Typ Text zu Array Typ Text.

```
C_TEXT(Variable)
COPY ARRAY(TextArray;Variable)
```

Retypisierung der Variablen Variable vom Typ Text auf Typ Zahl.

```
Variable:=Num(Variable)
```

Retypisierung des Array MyBoolean von Array Typ Boolean auf Variable Typ Zahl.

```
Variable:=MyBoolean
```

Retypisierung des Array IntArray von Array Typ Ganzzahl auf Array Typ Text.

```
ARRAY TEXT(IntArray;12)
```

wenn IntArray an anderer Stelle als Array vom Typ Ganzzahl deklariert wurde.
Versuch, eine Variable zu dereferenzieren, die nicht vom Typ Zeiger ist.

```
Variable->:=5
```

Wenn die Variable nicht vom Typ Zeiger ist.
Retyisierung der Variablen Var1 vom Typ Text auf Typ Zahl.

```
Var1:=3.5
```

Fehlerhafte Verwendung eines Feldes.

```
Variable:=[MyTable]MyField
```

[MyTable]MyField ist ein Datenfeld. Die Variable ist numerisch.

Syntax
Das Ergebnis der Funktion ist kein Zeiger.

```
Variable:=Num("Das Wetter ist schön")->
```

Diese Funktion lässt sich nicht dereferenzieren.
Syntaxfehler:

```
If(Boolean)
End for
```

Zu viele linke geschweifte Klammern ({).
Die Zeile enthält mehr öffnende als schließende geschweifte Klammern.
Zu viele rechte geschweifte Klammern (}).
Die Zeile enthält mehr schließende als öffnende geschweifte Klammern.
Klammer zu) erwartet.
Die Zeile enthält mehr öffnende als schließende runde Klammern.
Klammer auf (erwartet.
Die Zeile enthält mehr schließende als öffnende runde Klammern.
Es wurde ein Feld erwartet.

```
If(Modified(Variable))
```

Eine linke runde Klammer wurde erwartet.

```
C_INTEGER($
```

Es wurde eine Variable erwartet.

```
C_INTEGER([MyTable]MyField)
```

Die Nummer einer Konstanten wurde erwartet.

```
C_INTEGER(${"3"})
```

Ein Semicolon ; wurde erwartet.

```
COPY ARRAY(Array1 Array2)
```

Mac OS
Zu viele öffnende Symbole für Zeichenreferenz.

```
MyString≤3:="a"
```

Zu viele schließende Symbole für Zeichenreferenz.

```
MyString3≥:="a"
```

Windows
Zu viele öffnende Symbole für Zeichenreferenz.

```
MyString[[3:="a"
```

Zu viele schließende Symbole für Zeichenreferenz.

```
MyString 3]]:="a"
```

Eine Untertabelle wurde nicht erwartet.

```
ARRAY TO SELECTION(Array;Subtable)
```

Das Argument einer IF-Anweisung muss vom Typ Boolean sein.

```
If(Pointer)
```

Der Ausdruck ist zu komplex.
Teilen Sie Ihre Anweisungen in mehrere kürzere Anweisungen auf.
Die Methode ist zu komplex.
Es gibt zuviele Anweisungen Case of...End case bzw. If…End if.
Unbekanntes Feld.
Ihre Methode wurde möglicherweise von einer anderen Datenbank kopiert und enthält •???• anstatt eines Feldnamens.
Unbekannte Tabelle.
Ihre Methode wurde möglicherweise von einer anderen Datenbank kopiert und enthält •???• anstatt des Tabellennamens.
Zeiger auf einen fehlerhaften Ausdruck.

```
Pointer:=->Variable+3
```

Inkorrekte Verwendung des String Index.

```
MyReal≤3≥or MyReal[[3]]
```

oder

```
MyString≤Variable≥or MyString[[Variable]]
```

wobei Variable keine numerische Variable ist.

Parameter
Das Funktionsergebnis kann nicht als Parameter an diese Methode oder Befehl übergeben werden.

```
MyMethod(Num(MyString))
```

wenn MyMethod einen Ausdruck vom Typ Boolean erwartet.
Es wurden dieser Methode zu viele Parameter übergeben.

```
DEFAULT TABLE(Table;Form)
```

Dieser Wert kann nicht als Parameter an diese Methode oder Befehl übergeben werden.

```
MyMethod(3+2)
```

wenn MyMethod einen Ausdruck vom Typ Boolean erwartet.
Konflikt beim Typ des Funktionsergebnis.

```
C_INTEGER($0)
$0:=False
```

Konflikt beim generischen Parametertyp

```
<gen9>C_INTEGER(${3})
For($i;3;5)
   ${$i}:=String($i)
End for</gen9>
```

Dieser 4D Befehl benötigt keinen Parameter.

```
SHOW TOOL BAR(MyVar)
```

Dieser 4D Befehl benötigt mindestens einen Parameter.

```
DEFAULT TABLE
```

MyString kann in dieser Methode nicht als Parameter übergeben werden.

```
<gen9>MyMethod(MyString)</gen9>
```

wenn MyMethod einen Parameter vom Typ Boolean erwartet.
Der Typ des Parameters $1 unterscheidet sich in der aufrufenden und aufgerufenen Methode.

```
Calculate("3+2")
```

mit der Direktiven **C_INTEGER**($1) in Calculate.
Einer der Parameter in COPY ARRAY ist eine Variable.

```
COPY ARRAY(Variable;Array)
```

Retypisierung der Variablen $1 von Typ Zahl auf Typ Text.

```
$1:=String($1)
```

Ein Array kann nicht als Parameter verwendet werden.
ReInit(MyArray)Wollen Sie ein Array in einer Methode übergeben, müssen Sie einen Zeiger auf das Array übergeben.

Operatoren
Der Variablentyp ist nicht kompatibel mit dem Operator.

```
Bool2:=Bool1+True
```

In Feldern vom Typ Boolean ist keine Addition möglich.

Der Operator > wurde nicht erwartet:

```
QUERY(MyTable;[MyTable]MyField=0;>)
```

Variablen von diesem Typ sind nicht miteinander vergleichbar:

```
If(Number=Picture2)
```

Dieser Variablentyp kann nicht negiert werden:

```
Boolean:=-False
```

Plug-in Befehle

Der Plug-In Befehl PExt scheint nicht richtig definiert zu sein.
Es wurden diesem Plug-In Befehl zu wenige Parameter übergeben.
Es wurden diesem Plug-In Befehl zu viele Parameter übergeben.
Der Plug-In Befehl *Variable* scheint nicht richtig definiert zu sein.

Allgemeine Fehler

Zwei Methoden haben den gleichen Namen: Name.
Um Ihre Datenbank zu kompilieren, muss jede Projektmethode einen anderen Namen haben.

Interner Fehler # xx.
Erscheint diese Meldung, rufen Sie den Technischen Support von 4D an und nennen Sie die Fehlernummer.

Die Variable VARIABLE lässt sich nicht typisieren. Sie wird in Methode M1 verwendet.
Der Variablentyp lässt sich nicht bestimmen. Eine Compiler Direktive ist notwendig.

Die Originalmethode ist beschädigt.
Die Methode in der Orginalstruktur ist beschädigt. Löschen oder ersetzen Sie diese.

Unbekannter 4D Befehl.
Die Methode ist beschädigt.

Retypisierung der Variablen VARIABLE im Formular FORM.
Diese Meldung erscheint, wenn Sie zum Beispiel den Namen OK einer Variablen vom Typ Diagramm in einem Formular angeben.

Der Name der Funktion NAME wird bereits als Name einer Variablen im Formular NAME verwendet.
Nennen Sie entweder die Funktion oder die Variable anders.

Eine Methode und eine Variable haben den gleichen Namen: Name.
Nennen Sie entweder die Methode oder die Variable anders.

Ein Plug-In Befehl und eine Variable haben den gleichen Namen: Name.
Nennen Sie entweder den Plug-In Befehl oder die Variable anders.

Kann in einer als thread-safe deklarierten Methode keinen Befehl aufrufen, der nicht thread-safe ist.
Ändern Sie die Methode, so dass alle Befehle thread-safe sind oder ändern Sie die Deklaration der Methode, um einen kooperativen Prozess zu starten.

Kann in einer als thread-safe deklarierten Methode keine nicht thread-safe Methode 'Methodenname' aufrufen.
Ändern Sie die Methode, so dass sie thread-safe wird oder ändern Sie die Deklaration der Methode, um einen kooperativen Prozess zu starten.

Meldung bei Bereichsprüfung

Diese Meldungen werden in 4D generiert, während die kompilierte Datenbank läuft. Sie erscheinen in einem spezifischen Fehler-Dialogfenster.

Das Ergebnis ist außerhalb des Array-Bereichs.
Ist MyArray ein Array mit fünf Elementen, erscheint diese Meldung, wenn Sie im Array z.B. auf Element 17 zugreifen wollen.

Division durch Null.

```
Var1:=0
Var2:=2
Var3:=Var2/Var1
```

Zugriff auf nicht vorhandenen Parameter.
Die lokale Variable $4 wird verwendet, in der aktuellen Methode wurden jedoch nur drei Parameter übergeben.

Der Zeiger ist nicht korrekt initialisiert.

```
MyPointer->:=5
```

wenn MyPointer noch nicht initialisiert wurde.

Der Zielstring ist kürzer als die Quelle.

```
C_STRING(MyString1;5)
C_STRING(MyString2;10)
MyString2:="Flowers"
MyString1:=MyString2
```

Ungültiger Zeichenzugriff.

```
i:=-30
MyString≤i≥:=MyString2 or MyString[[i]]:=MyString2
```

Der Parameter ist ein leerer String.

```
MyString≤1≥:=""
MyString[[1]]:=""
```

Modulo durch Null.

```
Var1:=0
Var2:=2
Var3:=Var2% Var1
```

Ungültiger Parameter in einem AUSFÜHREN Befehl.

```
EXECUTE FORMULA("MyMethod(MyAlpha)")
```

wenn *MyMethod* nicht einen alphanumerischen Parameter erwartet.

Zeiger auf eine unbekannte Variable.

```
MyPointer:=Get pointer("Variable")
MyPointer:="MyString"
```

wenn Variable nicht explizit in der Datenbank erscheint.
Versuch einer Typenänderung mit einem Zeiger.

```
Boolean:=Pointer->
```

wenn Pointer auf ein Feld vom Typ Ganzzahl zeigt.
Falsche Verwendung eines Zeigers oder Zeiger auf eine unbekannte Variable.

```
Character:=StringVar≤Pointer->≥
Character:=StringVar[[Pointer]]
```

wenn Pointer nicht auf eine Zahl zeigt.

C_BLOB

C_BLOB ({Methode ;} Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Optionaler Name der Methode
Variable	Variable	➡	Name der zu deklarierenden Variable(n)

Beschreibung

C_BLOB deklariert die spezifizierte *Variable* als Variable vom Typ BLOB.

Bei der ersten Form des Befehls wird der optionale Parameter *Methode* NICHT übergeben. Damit können Sie jede Prozess-, Interprozess- oder lokale Variable deklarieren.

Hinweis: Diese Form können Sie auch in interpretierten Datenbanken verwenden.

Bei der zweiten Form des Befehls wird der optionale Parameter *Methode* übergeben. Damit deklarieren Sie für den Compiler das Ergebnis und/oder die Parameter ($0, $1, $2 etc.) für eine Methode vor. So können Sie beim Kompilieren einer Datenbank die Phase der Variablentypisierung überspringen und dadurch Zeit einsparen.

Warnung: Die zweite Form können Sie nicht im interpretierten Modus verwenden. Von daher ist diese Syntax nur in einer Methode möglich, die nicht im interpretierten Modus ausgeführt wird. Der Methodenname muss mit "COMPILER" beginnen.

Tipp für Fortgeschrittene: Mit der Syntax **C_BLOB**(${...}) können Sie eine Variablennummer von Parametern desselben Typs deklarieren, vorausgesetzt, diese sind die letzten Parameter für die Methode. So teilt zum Beispiel **C_BLOB**(${5}) 4D und dem Compiler mit, dass die Methode beim Starten mit dem fünften Parameter eine Variablennummer von Parametern dieses Typs empfangen kann. Weitere Informationen dazu finden Sie unter der Funktion **Count parameters**.

Beispiele

Siehe Beispiele im Abschnitt **Compilerbefehle**.

C_BOOLEAN

C_BOOLEAN ({Methode ;} Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Optionaler Name der Methode
Variable	Variable	➡	Name der zu deklarierenden Variable(n)

Beschreibung

Der Befehl **C_BOOLEAN** deklariert die spezifizierten Variablen als Variablen vom Typ Boolean.

Bei der ersten Form des Befehls wird der optionale Parameter *Methode* NICHT übergeben. Damit können Sie jede Prozess-, Interprozess- oder lokale Variable deklarieren bzw. typisieren.

Hinweis: Diese Form können Sie auch in interpretierten Datenbanken verwenden.

Bei der zweiten Form des Befehls wird der optionale Parameter *Methode* übergeben. Damit deklarieren Sie für den Compiler das Ergebnis und/oder die Parameter ($0, $1, $2 etc.) für eine Methode vor. So können Sie beim Kompilieren einer Datenbank die Phase der Variablentypisierung überspringen und dadurch Zeit einsparen.

Warnung: Die zweite Form können Sie nicht im interpretierten Modus verwenden. Von daher ist diese Syntax nur in einer Methode möglich, die nicht im interpretierten Modus ausgeführt wird. Der Methodenname muss mit "COMPILER" beginnen.

Tipp für Fortgeschrittene: Mit der Syntax **C_BOOLEAN**(${...}) können Sie eine Variablennummer von Parametern desselben Typs deklarieren, vorausgesetzt, diese sind die letzten Parameter für die Methode. So teilt zum Beispiel **C_BOOLEAN**(${5}) 4D und dem Compiler mit, dass die Methode beim Starten mit dem fünften Parameter eine Variablennummer von Parametern dieses Typs empfangen kann. Weitere Informationen dazu finden Sie unter der Funktion **Count parameters**.

Beispiele

Siehe Beispiele im Abschnitt **Compilerbefehle**.

C_COLLECTION

C_COLLECTION ({Methode ;} Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Name der Methode
Variable	Variable	➡	Variablenname(n) oder ${...} Parameter zum Deklarieren

Beschreibung

Der Befehl **C_COLLECTION** weist allen angegebenen Variablen den Typ *Collection* zu.

4D unterstützt ab v16 R4 den Typ *Collection*. Variablen dieses Typs enthalten eine sortierte Liste von Attributwerte verschiedener Typen, gespeichert als ein JSON Array.

Mit der ersten Syntax (der Parameter *Methode* ist nicht übergeben) deklarieren und typisieren Sie eine lokale, Prozess- oder Interprozessvariable. Diese Syntax eignet sich für interpretierte Anwendungen.

Mit der zweiten Syntax (der Parameter *Methode* ist übergeben) deklarieren Sie das Ergebnis der Methode bzw. Parameter ($0, $1, $2, etc.) zum Compiler im voraus. Mit dieser Syntax können Sie die Typisierungsphase der Variablen überspringen, um Zeit beim Kompilieren der Anwendung zu gewinnen.

WARNUNG: Sie können die zweite Syntax nicht im interpretierten Modus ausführen. Denn diese Syntax erfordert das Abspeichern in einer Methode, deren Name mit "COMPILER" beginnen muss und das wird nicht im interpretierten Modus ausgeführt.

Fortgeschrittene Verwendung: Mit der Syntax **C_COLLECTION**(${...}) können Sie eine variable Anzahl von Parametern desselben Typs für eine Methode deklarieren, solange es die letzten Parameter dieser Methode sind. Zum Beispiel zeigt die Deklaration **C_COLLECTION**(${5}) dem Compiler an, dass die Methode ab dem fünften Parameter eine variable Anzahl von Parametern dieses Typs empfangen kann.

Beispiel

Eine Prozessvariable vom Typ Collection deklarieren und diese mit einer neuen Collection füllen:

```
C_COLLECTION(myCol)
 //hier ist der Wert myCol Null
myCol:=New collection("Green";100;"Orange";200;"Red";300)
 //myCol= ["Green",100,"Orange",200,"Red",300]
```

C_DATE

C_DATE ({Methode ;} Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Optionaler Name der Methode
Variable	Variable	➡	Name der zu deklarierenden Variable(n)

Beschreibung

Der Befehl **C_DATE** deklariert die spezifizierte *Variable* als Variable vom Typ Datum.

Bei der ersten Form des Befehls wird der optionale Parameter *Methode* NICHT übergeben. Damit können Sie jede Prozess-, Interprozess- oder lokale Variable deklarieren.

Hinweis: Diese Form können Sie auch in interpretierten Datenbanken verwenden.

Bei der zweiten Form des Befehls wird der optionale Parameter *Methode* übergeben. Damit deklarieren Sie für den Compiler das Ergebnis und/oder die Parameter ($0, $1, $2 etc.) für eine Methode vor. So können Sie beim Kompilieren einer Datenbank die Phase der Variablentypisierung überspringen und dadurch Zeit einsparen.

Warnung: Die zweite Form können Sie nicht im interpretierten Modus verwenden. Von daher ist diese Syntax nur in einer Methode möglich, die nicht im interpretierten Modus ausgeführt wird. Der Methodenname muss mit "COMPILER" beginnen.

Tipp für Fortgeschrittene: Mit der Syntax **C_DATE**(${...}) können Sie eine Variablennummer von Parametern desselben Typs deklarieren, vorausgesetzt, diese sind die letzten Parameter für die Methode. So teilt zum Beispiel **C_DATE**(${5}) 4D und dem Compiler mit, dass die Methode beim Starten mit dem fünften Parameter eine Variablennummer von Parametern dieses Typs empfangen kann. Weitere Informationen dazu finden Sie unter der Funktion **Count parameters**.

Beispiele

Siehe Beispiele im Abschnitt **Compilerbefehle**.

C_LONGINT

C_LONGINT ({Methode ;} Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Optionaler Name der Methode
Variable	Variable	➡	Name der zu deklarierenden Variable(n)

Beschreibung

Der Befehl **C_LONGINT** deklariert die spezifizierte *Variable* als Variable vom Typ Lange Ganzzahl.

Bei der ersten Form des Befehls wird der optionale Parameter *Methode* NICHT übergeben. Damit können Sie jede Prozess-, Interprozess- oder lokale Variable deklarieren.

Hinweis: Diese Form können Sie auch in interpretierten Datenbanken verwenden.

Bei der zweiten Form des Befehls wird der optionale Parameter *Methode* übergeben. Damit deklarieren Sie für den Compiler das Ergebnis und/oder die Parameter ($0, $1, $2 etc.) für eine Methode vor. So können Sie beim Kompilieren einer Datenbank die Phase der Variablentypisierung überspringen und dadurch Zeit einsparen.

Warnung: Die zweite Form können Sie nicht im interpretierten Modus verwenden. Von daher ist diese Syntax nur in einer Methode möglich, die nicht im interpretierten Modus ausgeführt wird. Der Methodenname muss mit "COMPILER" beginnen.

Tipp für Fortgeschrittene: Mit der Syntax **C_LONGINT**(${...}) können Sie eine Variablennummer von Parametern desselben Typs deklarieren, vorausgesetzt, diese sind die letzten Parameter für die Methode. So teilt zum Beispiel **C_LONGINT**(${5}) 4D und dem Compiler mit, dass die Methode beim Starten mit dem fünften Parameter eine Variablennummer von Parametern dieses Typs empfangen kann. Weitere Informationen dazu finden Sie unter der Funktion **Count parameters**.

Beispiele

Siehe Beispiele im Abschnitt **Compilerbefehle**.

C_OBJECT

C_OBJECT ({Methode ;} Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Name der Methode
Variable	Variable	➡	Name(n) der zu deklarierenden Variable(n) oder Parameter ${...}

Beschreibung

Der Befehl **C_OBJECT** weist allen spezifizierten Variablen den Typ *Objekt* zu.

Seit 4D v14 gibt es den neuen Datentyp *Objekt*. Er wird über Befehle im Kapitel **Objekte (Sprache)** oder über die Objektnotation verwaltet (siehe **Objektnotation verwenden**).

Über die erste Syntax des Befehls (der Parameter *Methode* ist nicht übergeben) deklarieren und typisieren Sie eine Prozess-, Interprozess- oder lokale Variable. Diese Syntax lässt sich in interpretierten Datenbanken einsetzen.

Über die zweite Syntax des Befehls (der Parameter Methode ist übergeben) deklarieren Sie das Ergebnis der Methode bzw. Parameter ($0, $1, $2, etc.) im Voraus für den Compiler. Verwenden Sie diese Syntax, wenn Sie die Typisierungsphase für Variablen überspringen wollen, um beim Kompilieren der Datenbank Zeit einzusparen.

WARNUNG: Sie können die zweite Syntax nicht im interpretierten Modus ausführen. Deshalb sollten Sie diese Syntax in einer Methode speichern, die nicht im interpretierten Modus ausgeführt wird. Ihr Name muss mit "COMPILER" beginnen.

Erweiterte Verwendung: Sie können mit der Syntax **C_OBJECT**(${...}) eine variable Anzahl Parameter desselben Typs für eine Methode deklarieren, solange es die letzten Parameter dieser Methode sind. So zeigt z.B. die Deklaration **C_OBJECT**(${5}) dem Compiler an, dass die Methode ab dem fünften Parameter eine variable Anzahl Parameter dieses Typs empfangen kann.

Wichtig: Der Befehl **C_OBJECT** erstellt kein Objekt mit Namen *Variable*. Wollen Sie über Objektnotation auf seine Eigenschaften zugreifen, müssen Sie das Objekt zuerst über die Funktion **New object** initialisieren, sonst wird ein Syntaxfehler generiert (siehe Beispiel).

Beispiel

```
C_OBJECT($obj)  //Die Variable $obj ist deklariert, aber das Objekt $obj existiert nicht
$obj:=New object  //Das Objekt $obj ist initialisiert
$obj.prop:=42  //...und seine Eigenschaften sind zugänglich
```

C_PICTURE

C_PICTURE ({Methode ;} Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Optionaler Name der Methode
Variable	Variable	➡	Name der zu deklarierenden Variable(n)

Beschreibung

Der Befehl **C_PICTURE** deklariert die spezifizierte *Variable* als Variable vom Typ Bild.

Bei der ersten Form des Befehls wird der optionale Parameter *Methode* NICHT übergeben. Damit können Sie jede Prozess-, Interprozess- oder lokale Variable deklarieren.

Hinweis: Diese Form können Sie auch in interpretierten Datenbanken verwenden.

Bei der zweiten Form des Befehls wird der optionale Parameter *Methode* übergeben. Damit deklarieren Sie für den Compiler das Ergebnis und/oder die Parameter ($0, $1, $2 etc.) für eine Methode vor. So können Sie beim Kompilieren einer Datenbank die Phase der Variablentypisierung überspringen und dadurch Zeit einsparen.

Warnung: Die zweite Form können Sie nicht im interpretierten Modus verwenden. Von daher ist diese Syntax nur in einer Methode möglich, die nicht im interpretierten Modus ausgeführt wird. Der Methodenname muss mit "COMPILER" beginnen.

Tipp für Fortgeschrittene: Mit der Syntax **C_PICTURE**(${...}) können Sie eine Variablennummer von Parametern desselben Typs deklarieren, vorausgesetzt, diese sind die letzten Parameter für die Methode. So teilt zum Beispiel **C_PICTURE**(${5}) 4D und dem Compiler mit, dass die Methode beim Starten mit dem fünften Parameter eine Variablennummer von Parametern dieses Typs empfangen kann. Weitere Informationen dazu finden Sie unter der Funktion **Count parameters**.

Beispiele

Siehe Beispiele im Abschnitt **Compilerbefehle**.

C_POINTER

C_POINTER ({Methode ;} Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Optionaler Name der Methode
Variable	Variable	➡	Name der zu deklarierenden Variable(n)

Beschreibung

Der Befehl **C_POINTER** deklariert die spezifizierte *Variable* als Variable vom Typ Zeiger.

Bei der ersten Form des Befehls wird der optionale Parameter *Methode* NICHT übergeben. Damit können Sie jede Prozess-, Interprozess- oder lokale Variable deklarieren.

Hinweis: Diese Form können Sie auch in interpretierten Datenbanken verwenden.

Bei der zweiten Form des Befehls wird der optionale Parameter *Methode* übergeben. Damit deklarieren Sie für den Compiler das Ergebnis und/oder die Parameter ($0, $1, $2 etc.) für eine Methode vor. So können Sie beim Kompilieren einer Datenbank die Phase der Variablentypisierung überspringen und dadurch Zeit einsparen.

Warnung: Die zweite Form können Sie nicht im interpretierten Modus verwenden. Von daher ist diese Syntax nur in einer Methode möglich, die nicht im interpretierten Modus ausgeführt wird. Der Methodenname muss mit "COMPILER" beginnen.

Tipp für Fortgeschrittene: Mit der Syntax **C_POINTER**(${...}) können Sie eine variable Anzahl von Parametern desselben Typs deklarieren, vorausgesetzt, diese sind die letzten Parameter für die Methode. So teilt zum Beispiel **C_POINTER**(${5}) 4D und dem Compiler mit, dass die Methode beim Starten mit dem fünften Parameter eine variable Anzahl von Parametern dieses Typs empfangen kann. Weitere Informationen dazu finden Sie unter der Funktion **Count parameters**.

Beispiele

Siehe Beispiele im Abschnitt **Compilerbefehle**.

C_REAL

C_REAL ({Methode ;} Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Optionaler Name der Methode
Variable	Variable	➡	Name der zu deklarierenden Variable(n)

Beschreibung

Der Befehl **C_REAL** deklariert die spezifizierte *Variable* als Variable vom Typ Zahl.

Bei der ersten Form des Befehls wird der optionale Parameter *Methode* NICHT übergeben. Damit können Sie jede Prozess-, Interprozess- oder lokale Variable deklarieren.

Hinweis: Diese Form können Sie auch in interpretierten Datenbanken verwenden.

Bei der zweiten Form des Befehls wird der optionale Parameter *Methode* übergeben. Damit deklarieren Sie für den Compiler das Ergebnis und/oder die Parameter ($0, $1, $2 etc.) für eine Methode vor. So können Sie beim Kompilieren einer Datenbank die Phase der Variablentypisierung überspringen und dadurch Zeit einsparen.

Warnung: Die zweite Form können Sie nicht im interpretierten Modus verwenden. Von daher ist diese Syntax nur in einer Methode möglich, die nicht im interpretierten Modus ausgeführt wird. Der Methodenname muss mit "COMPILER" beginnen.

Tipp für Fortgeschrittene: Mit der Syntax **C_REAL**(${...}) können Sie eine Variablennummer von Parametern desselben Typs deklarieren, vorausgesetzt, diese sind die letzten Parameter für die Methode. So teilt zum Beispiel **C_REAL**(${5}) 4D und dem Compiler mit, dass die Methode beim Starten mit dem fünften Parameter eine Variablennummer von Parametern dieses Typs empfangen kann. Weitere Informationen dazu finden Sie unter der Funktion **Count parameters**.

Beispiele

Siehe Beispiele im Abschnitt **Compilerbefehle**.

C_TEXT

C_TEXT ({Methode ;} Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Optionaler Name der Methode
Variable	Variable	➡	Name der zu deklarierenden Variable(n)

Beschreibung

Der Befehl **C_TEXT** deklariert die spezifizierte *Variable* als Variable vom Typ Text.

Bei der ersten Form des Befehls wird der optionale Parameter *Methode* NICHT übergeben. Damit können Sie jede Prozess-, Interprozess- oder lokale Variable deklarieren.

Hinweis: Diese Form können Sie auch in interpretierten Datenbanken verwenden.

Bei der zweiten Form des Befehls wird der optionale Parameter *Methode* übergeben. Damit deklarieren Sie für den Compiler das Ergebnis und/oder die Parameter ($0, $1, $2 etc.) für eine Methode vor. So können Sie beim Kompilieren einer Datenbank die Phase der Variablentypisierung überspringen und dadurch Zeit einsparen.

Warnung: Die zweite Form können Sie nicht im interpretierten Modus verwenden. Von daher ist diese Syntax nur in einer Methode möglich, die nicht im interpretierten Modus ausgeführt wird. Der Methodenname muss mit "COMPILER" beginnen.

Tipp für Fortgeschrittene: Mit der Syntax **C_TEXT**(${...}) können Sie eine Variablennummer von Parametern desselben Typs deklarieren, vorausgesetzt, diese sind die letzten Parameter für die Methode. So teilt zum Beispiel **C_TEXT**(${5}) 4D und dem Compiler mit, dass die Methode beim Starten mit dem fünften Parameter eine Variablennummer von Parametern dieses Typs empfangen kann. Weitere Informationen dazu finden Sie unter der Funktion **Count parameters**.

Beispiele

Siehe Beispiele im Abschnitt **Compilerbefehle**.

C_TIME

C_TIME ({Methode ;} Variablenname {; Variablenname2 ; ... ; VariablennameN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Optionaler Name der Methode
Variablenname	Variable	➡	Name der zu deklarierenden Variable(n)

Beschreibung

Der Befehl **C_TIME** deklariert die spezifizierte *Variable* als Variable vom Typ Zeit.

Bei der ersten Form des Befehls wird der optionale Parameter *Methode* NICHT übergeben. Damit können Sie jede Prozess-, Interprozess- oder lokale Variable deklarieren.

Hinweis: Diese Form können Sie auch in interpretierten Datenbanken verwenden.

Bei der zweiten Form des Befehls wird der optionale Parameter *Methode* übergeben. Damit deklarieren Sie für den Compiler das Ergebnis und/oder die Parameter ($0, $1, $2 etc.) für eine Methode vor. So können Sie beim Kompilieren einer Datenbank die Phase der Variablentypisierung überspringen und dadurch Zeit einsparen.

Warnung: Die zweite Form können Sie nicht im interpretierten Modus verwenden. Von daher ist diese Syntax nur in einer Methode möglich, die nicht im interpretierten Modus ausgeführt wird. Der Methodenname muss mit "COMPILER" beginnen.

Tipp für Fortgeschrittene: Mit der Syntax **C_TIME**(${...}) können Sie eine Variablennummer von Parametern desselben Typs deklarieren, vorausgesetzt, diese sind die letzten Parameter für die Methode. So teilt zum Beispiel **C_TIME**(${5}) 4D und dem Compiler mit, dass die Methode beim Starten mit dem fünften Parameter eine Variablennummer von Parametern dieses Typs empfangen kann. Weitere Informationen dazu finden Sie unter der Funktion **Count parameters**.

Beispiele

Siehe Beispiele im Abschnitt **Compilerbefehle**.

IDLE

IDLE
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **IDLE** gilt nur für den Compiler. Er wird nur in kompilierten Datenbanken mit benutzerdefinierten Methoden verwendet, d.h. wenn keine Aufrufe zurück an die 4D Engine gehen. Enthält eine Methode zum Beispiel eine *For* Schleife, in der keine 4D Befehle ausgeführt werden, kann die Schleife weder durch einen Prozess mit **ON EVENT CALL** unterbrochen werden, noch kann ein Benutzer in eine andere Anwendung wechseln. Fügen Sie in diesem Fall den Befehl **IDLE** ein, damit 4D die Ereignisse ggf. stoppen kann.

Beispiel

Hier würde die Schleife in einer kompilierten Datenbank ohne Aufruf von **IDLE** nie enden:

```
  ` Projektmethode Do Something
ON EVENT CALL("EVENT METHOD")
◇vbWeStop:=False
MESSAGE("Processing..."+Char(13)+"Tippe beliebige Taste zum Unterbrechen...")
Repeat  ` Führe Anweisungen aus, die keinen 4D Befehl enthalten
   IDLE
Until(◇vbWeStop)
ON EVENT CALL("")
```

```
  ` Projektmethode EVENT METHOD
If(Undefined(KeyCode))
   KeyCode:=0
End if
If(KeyCode#0)
   CONFIRM("Wollen Sie diese Operation wirklich beenden?")
   If(OK=1)
      ◇vbWeStop:=True
   End if
End if
```

_o_C_GRAPH

_o_C_GRAPH ({Methodenname ;} Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methodenname	String	➡	Optionaler Name der Methode
Variable	Variable	➡	Name der zu deklarierenden Variable(n)

Hinweis zur Kompatibilität

Variablen vom Typ Diagrammbereich sind überholt und werden seit 4D v14 nicht mehr unterstützt. Sie müssen Bildvariablen verwenden (siehe **GRAPH**).

_o_C_INTEGER

_o_C_INTEGER ({Methode ;} Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Optionaler Name der Methode
Variable		➡	Name der zu deklarierenden Variable(n)

Hinweis

Diesen Befehl gibt es nur noch zur Wahrung der Kompatibilität mit Datenbanken früherer Versionen. 4D und 4D Compiler wandeln inzwischen intern Ganzzahlen in Lange Ganzzahlen um. Zum Beispiel:

```
C_INTEGER($MyVar)
$TheType:=Type($MyVar)  `$TheType = 9 (Is Longint)
```

_o_C_STRING

_o_C_STRING ({Methode ;} NeueGröße ; Variable {; Variable2 ; ... ; VariableN})

Parameter	Typ		Beschreibung
Methode	Methode	➡	Methodenname
NeueGröße	Lange Ganzzahl	➡	Größe des String
Variable	Variable	➡	Name der zu deklarierenden Variable(n)

Compatibility note

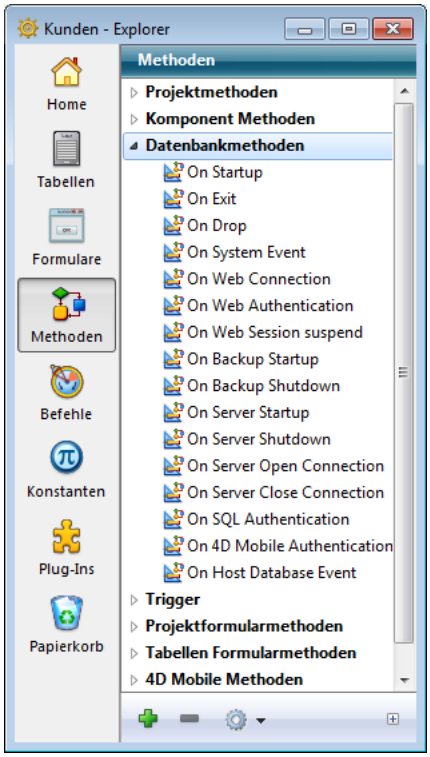
Der Befehl **_o_C_STRING** arbeitet exakt wie der Befehl **C_TEXT** (Der Parameter *Größe* wird ignoriert). Wir empfehlen jetzt, in Ihren 4D Entwicklungen nur den Befehl **C_TEXT** zu verwenden.

Datenbankmethoden

- Datenbankmethoden
- Datenbankmethode On 4D Mobile Authentication
- Datenbankmethode On Backup Shutdown
- Datenbankmethode On Backup Startup
- Datenbankmethode On Drop
- Datenbankmethode On Exit
- Datenbankmethode On Host Database Event
- Datenbankmethode On Server Close Connection
- Datenbankmethode On Server Open Connection
- Datenbankmethode On Server Shutdown
- Datenbankmethode On Server Startup
- Datenbankmethode On SQL Authentication
- Datenbankmethode On Startup
- Datenbankmethode On System Event
- Datenbankmethode On Web Authentication
- Datenbankmethode On Web Close Process
- Datenbankmethode On Web Connection

Datenbankmethoden

Datenbankmethoden laufen automatisch ab, sobald bestimmte Ereignisse bei Arbeitssitzungen auftreten.



Um eine Datenbankmethode zu erstellen bzw. zu öffnen und bearbeiten, gehen Sie folgendermaßen vor:

1. Öffnen Sie das Fenster **Explorer**.
2. Wählen Sie die Seite **Methoden**.
3. Erweitern Sie das Thema **Datenbankmethoden**.
4. Doppelklicken Sie auf die Methode.

oder:

1. Wählen Sie die Methode.
2. Drücken Sie die **Eingabetaste** bzw. **Zeilenschaltung**.

Sie erstellen eine Datenbankmethode wie jede andere Methode.

Sie können eine Datenbankmethode nicht von einer anderen Methode aus aufrufen. 4D löst Datenbankmethoden bei bestimmten Ereignissen in Arbeitssitzungen aus. Folgende Liste zeigt die Ausführung von Datenbankmethoden:

Datenbankmethode	**4D lokal**	**4D Server**	**remote 4D**
On Startup	Ja, Einmal	Nein	Ja, Einmal
On Exit	Ja, Einmal	Nein	Ja, Einmal
On Drop	Ja, Mehrmals	Nein	Ja, Mehrmals
On System Event	Ja, Mehrmals	Ja, Mehrmals	Ja, Mehrmals
On Web Connection	Ja, Mehrmals	Ja, Mehrmals	Ja, Mehrmals
On Web Authentication	Ja, Mehrmals	Ja, Mehrmals	Ja, Mehrmals
On Web Session Suspend	Ja, Mehrmals	Ja, Mehrmals	Ja, Mehrmals
On Backup Startup	Ja, Mehrmals	Ja, Mehrmals	Nein
On Backup Shutdown	Ja, Mehrmals	Ja, Mehrmals	Nein
On Server Startup	Nein	Ja, Einmal	Nein
On Server Shutdown	Nein	Ja, Einmal	Nein
On Server Open Connection	Nein	Ja, Mehrmals	Nein
On Server Close Connection	Nein	Ja, Mehrmals	Nein
On SQL Authentication	Ja, Mehrmals	Ja, Mehrmals	Ja, Mehrmals
On 4D Mobile Authentication	Ja, Mehrmals	Ja, Mehrmals	Nein
On Host Database Event	Ja, Mehrmals	Ja, Mehrmals	Ja, Mehrmals

Datenbankmethode On 4D Mobile Authentication

$1, $2, $3 -> Datenbankmethode On 4D Mobile Authentication -> Funktionsergebnis

Parameter	Typ		Beschreibung
$1	Text	⇐	Benutzername
$2	Text	⇐	Kennwort
$3	Boolean	⇐	Wahr = Digest Modus, Falsch = Basic Modus
Funktionsergebnis	Boolean	⇒	Wahr = Anfrage angenommen, Falsch = Anfrage verweigert

Beschreibung

Die **Datenbankmethode On 4D Mobile Authentication** bietet einen eigenen Weg, um das Öffnen von 4D Mobile Sessions in 4D (via REST) zu steuern. Sie dient hauptsächlich zum Filtern von Anfragen, wenn eine Verbindung zwischen einem Wakanda Server und 4D hergestellt wird.

Im allgemeinen läuft die Anfrage zum Öffnen einer 4D Mobile Session über die Methode ***mergeOutsideCatalog()***, die Identifier der Verbindung erscheinen im Header der Anfrage. Die **Datenbankmethode On 4D Mobile Authentication** wird aufgerufen, so dass Sie diese Identifier bewerten können. Sie können die Liste der Benutzer für die 4D Datenbank oder Ihre eigene Tabelle mit Identifiern verwenden.

Wichtig: Ist die **Datenbankmethode On 4D Mobile Authentication** definiert (z.B. wenn sie Code enthält), delegiert 4D die Kontrolle der 4D Mobile Anfragen komplett an diese Methode. Die Einstellung über das DropDown-Menü "Lesen/Schreiben" auf der Seite Web > 4D Mobile der Datenbank-Eigenschaften wird ignoriert (siehe Abschnitt **Zugriff Lesen/Schreiben** des Handbuchs *4D Designmodus*).

Die Datenbankmethode empfängt zwei Parameter (*$1* und *$2*) vom Typ Text und einen vom Typ Boolean (*$3*), der von 4D übergeben wird, und gibt ein Boolean *$0* zurück. Sie deklarieren diese Parameter wie folgt:

```
//Datenbankmethode On 4D Mobile Authentication
C_TEXT($1;$2)
C_BOOLEAN($0;$3)
... // Code für die Methode
```

$1 enthält den Benutzernamen, *$2* das Kennwort für die Verbindung

Das Kennwort (*$2*) wird, je nach dem von der Anfrage verwendeten Modus, in Klartext oder verschlüsselt (Hash) empfangen. Der Modus wird im Parameter *$3* angegeben und legt somit das weitere Vorgehen fest:

- Wird das Kennwort in Klartext gesendet (Basic Modus), gibt *$3* **False** zurück.
- Wird das Kennwort verschlüsselt (Hash) gesendet (Digest Modus), gibt *$3* **True** zurück.

Bei einer 4D Mobile Verbindungsanfrage von Wakanda Server wird das Kennwort immer verschlüsselt (Hash) gesendet.

Sie müssen die Identifier der 4D Mobile Verbindung in der Datenbankmethode prüfen. In der Regel prüfen Sie Name und Kennwort über eine eigene Benutzertabelle. Bei gültigen Identifiern übergeben Sie **True** in *$0*. Die Anfrage wird dann angenommen; 4D führt sie aus und gibt das Ergebnis in JSON zurück.
Andernfalls übergeben Sie **False** in *$0*. Dann wird die Verbindung abgewiesen und der Server gibt einen Authentifizierungsfehler an den Absender der Anfrage zurück.

Ist der Benutzer in der Liste der 4D Benutzer der Datenbank eingetragen, können Sie das Kennwort direkt über folgende Anweisung prüfen:

```
$0:=Validate password($1;$2;$3)
```

Die Funktion **Validate password** wurde erweitert. Der Benutzername wird jetzt als erster Parameter akzeptiert. Ein optionaler Parameter gibt an, ob das Kennwort verschlüsselt erscheint.

Wollen Sie eine eigene Liste mit Benutzern verwenden, unabhängig von der Liste der 4D Anwendung, können Sie deren Kennwörter verschlüsselt mit demselben Algoritmus sichern, den Wakanda Server beim Senden einer Verbindungsanfrage an die **Datenbankmethode On 4D Mobile Authentication** in *$2* verwendet. Um ein Kennwort mit dieser Methode zu verschlüsseln, schreiben Sie folgende Anweisung:

```
$HashedPasswd :=Generate digest($ClearPasswd ;4D digest)
```

Die Funktion **Generate digest** akzeptiert 4D digest als Hash-Algorithmus. Diese Funktion verwendet 4D auch zur internen Verwaltung von Kennwörtern.

Beispiel 1

Dieses Beispiel zur **Datenbankmethode On 4D Mobile Authentication** akzeptiert nur den Benutzer "admin" mit dem Kennwort "123", der nicht mit einem 4D Benutzer übereinstimmt:

```
// Datenbankmethode On 4D Mobile Authentication
C_TEXT($1;$2)
C_BOOLEAN($0;$3)
 //$1: user
 //$2: password
```

```
  //$3: digest mode
If($1="admin")
    If($3)
        $0:=($2=Generate digest("123";4D digest))
    Else
        $0:=($2="123")
    End if
Else
    $0:=False
End if
```

Beispiel 2

Dieses Beispiel zur **Datenbankmethode On 4D Mobile Authentication** prüft, ob die Verbindungsanfrage von einem der beiden zugelassenen Wakanda Server kommt, die unter den Benutzern der 4D Anwendung gesichert sind:

```
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
```

Datenbankmethode On Backup Shutdown

$1 -> Datenbankmethode On Backup Shutdown

Parameter	Typ		Beschreibung
$1	Lange Ganzzahl	⇦	0 = Backup korrekt ausgeführt; anderer Wert = Fehler; vom Benutzer unterbrochen oder Code von Methode On Backup Startup zurückgegeben

---

Die **Datenbankmethode On Backup Shutdown** wird immer aufgerufen, wenn ein Backup der Datenbank endet. Gründe dafür können das Ende der Kopie, Unterbrechung durch den Benutzer oder ein Fehler sein.
Das betrifft alle 4D Umgebungen: 4D im lokalen und im remote Modus, 4D Server, sowie alle 4D Anwendungen mit integrierter 4D Volume Desktop.

Mit der **Datenbankmethode On Backup Shutdown** können Sie prüfen, ob das Backup korrekt durchgeführt wurde. Sie empfängt im Parameter $1 einen Wert, der den Status des abgeschlossenen Backup anzeigt:

- Bei korrekt ausgeführtem Backup hat $1 den Wert 0 (Null).
- Wurde das Backup durch den Benutzer oder aufgrund eines Fehlers unterbrochen, enthält $1 einen anderen Wert als 0 (Null). Wurde das Backup von der **Datenbankmethode On Backup Startup** gestoppt ($0 # 0), enthält $1 den aktuell im Parameter $0 zurückgegebenen Wert. So können Sie ein eigenes Fehlerverwaltungssystem einbauen.
- Wurde das Backup aufgrund eines Fehlers gestoppt, wird in $1 die Fehlernummer zurückgegeben.

In allen Fällen erhalten Sie Informationen zum aufgetretenen Fehler über den Befehl **GET BACKUP INFORMATION**.

Hinweis: Sie müssen den Parameter $1 (Lange Ganzzahl) in der Datenbankmethode deklarieren:

```
C_LONGINT($1)
```

Bitte beachten Sie, dass bei einem Fehler während dem Backup (Festplatte voll, o.ä) die Fehlermeldung nur auf dem 4D Server Monitor oder im MSC angezeigt und in das Backup Logbuch kopiert wird. Es erscheint kein Dialogfenster mit der Meldung und die Variable *Error* wird nicht verändert. Wenn Sie den Administrator informieren wollen, dass ein Fehler aufgetreten ist, insbesondere bei einer Anwendung im Client/Server Modus, müssen Sie die **Datenbankmethode On Backup Shutdown** einsetzen.

Datenbankmethode On Backup Startup

Datenbankmethode On Backup Startup -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		0 = Backup lässt sich starten anderer Wert = Backup nicht authorisiert

---

Die **Datenbankmethode On Backup Startup** wird immer aufgerufen, wenn ein Backup der Datenbank gerade startet. Das Backup kann manuell, automatisch festgelegt oder über den Befehl **BACKUP** aufgerufen werden.
Das betrifft alle 4D Umgebungen: 4D in lokalen und im remote Modus, 4D Server und Datenbanken mit einkompilierter 4D Volume Desktop.

Mit der **Datenbankmethode On Backup Startup** können Sie prüfen, ob das Backup gestartet ist. Sie gibt im Parameter $0 einen Wert zurück, der das Backup zulässt oder verweigert.

- Bei $0 = 0 lässt sich das Backup starten.
- Bei $0 # 0 ist kein Backup zugelassen. Der Vorgang wird abgebrochen und ein Fehler zurückgegeben. Sie können den Fehler über den Befehl **GET BACKUP INFORMATION** erhalten.

Über diese Datenbankmethode können Sie die Bedingungen der Backup-Ausführung prüfen, z.B. Benutzer, Datum des letzten Backup.

Hinweis: Sie müssen den Parameter $0 (Lange Ganzzahl) in der Datenbankmethode deklarieren:

```
C_LONGINT($0)
```

Datenbankmethode On Drop

Datenbankmethode On Drop
Dieser Befehl benötigt keine Parameter

Die **Datenbankmethode On Drop** ist für 4D Anwendungen im lokalen und im remote Modus verfügbar.

Diese Datenbankmethode wird automatisch ausgeführt, wenn Objekte per Drag-and-Drop in die 4D Anwendung außerhalb eines Formulars oder Fensters gesetzt werden, z.B.:

Aktion	Plattform	Kommentar
Drop in einen leeren Bereich eines MDI Fensters	Windows	Nicht verfügbar bei Ausführung im SDI Modus, da es in diesem Kontext kein MDI Fenster gibt (siehe Abschnitt **SDI Modus unter Windows**).
Drop auf das 4D Icon im Dock	macOS	
Drop auf das 4D Icon im Desktop des Systems	Windows(*) & macOS	**Datenbankmethode On Drop** wird nur aufgerufen, wenn die Anwendung bereits gestartet ist, außer bei Anwendungen mit integrierter 4D Volume Desktop. In diesem Fall wird die Datenbankmethode auch aufgerufen, wenn die Anwendung nicht gestartet ist. Auf diese Weise lassen sich eigene Dokument-Signaturen definieren.

(*) Wird mit 4D Developer 64-bit unter Windows nicht unterstützt, da diese Aktion eine neue Instanz der Anwendung startet (System Feature).

Auf macOS müssen Sie während der Drop Aktion die **Wahl-** und **Befehlstaste** gedrückt halten, damit die Datenbankmethode aufgerufen wird.

Beispiel

Dieses Beispiel zeigt, wie Sie ein 4D Write Dokument öffnen können, das außerhalb eines Formulars gesetzt wurde:

```
  `Datenbankmethode On Drop
droppedFile:=Get file from pasteboard(1)
If(Position(".4W7";droppedFile)=Length(droppedFile)-3)
   externalArea:=Open external window(100;100;500;500;0;droppedFile;"_4D Write")
   WR OPEN DOCUMENT(externalArea;droppedFile)
End if
```

Datenbankmethode On Exit

Datenbankmethode On Exit
Dieser Befehl benötigt keine Parameter

Die **Datenbankmethode On Exit** wird einmal aufgerufen, wenn Sie eine Anwendung verlassen.
Dies geschieht in folgenden 4D Umgebungen:

- 4D im lokalen Modus
- 4D im remote Modus
- 4D Anwendung mit integriertem 4D Volume Desktop

Hinweis: Die **Datenbankmethode On Exit** wird NICHT auf 4D Server ausgelöst.
Die **Datenbankmethode On Exit** wird automatisch von 4D ausgelöst; diese Datenbankmethode können Sie im Gegensatz zu Projektmethoden nicht selbst per Programmierung aufrufen. Sie können auch Unterroutinen verwenden.
Eine Anwendung wird verlassen, wenn folgendes passiert:

- Der Benutzer wählt in der Designumgebung im Menü **Datei/Ablage** den Befehl **Beenden** oder Anwendungsumgebung die Standardaktion Beenden.
- Ein Aufruf des Befehls **QUIT 4D** wird ausgelöst
- Ein 4D Plug-In löst einen Aufruf von **QUIT 4D** aus

Beim Verlassen der Anwendung führt 4D folgende Aktionen aus:

- Ist keine **Datenbankmethode On Exit** vorhanden, beendet 4D ohne Unterscheidung der Reihe nach jeden Prozess. Der Datensatz in Bearbeitung wird abgebrochen und nicht gesichert.
- Ist eine **Datenbankmethode On Exit** vorhanden, führt 4D diese Methode in einem neu angelegten lokalen Prozess durch. Beachten Sie, dass 4D evtl. beendet wird - die **Datenbankmethode On Exit** kann zwar alle gewünschten Operationen zum Aufräumen oder Schließen durchführen, aber nicht das Beenden ansich verweigern, und endet so an einem gewissen Punkt.

Die **Datenbankmethode On Exit** ist der richtige Ort, um:

- Voreinstellungen bzw. Einstellungen (lokal auf der Festplatte) zu speichern, um sie beim Starten der nächsten Sitzung in der **Datenbankmethode On Startup** wiederzuverwenden.
- Andere Aktionen auszuführen, die bei jedem Beenden der Datenbank automatisch ablaufen sollen (ohne Datenbank-Zugriffe).

Hinweis: Beachten Sie, dass die **Datenbankmethode On Exit** ein lokaler/Client Prozess ist, d.h. er kann nicht auf die Datendatei zugreifen. Wenn also die **Datenbankmethode On Exit** eine Suche oder Sortierung ausführt, wird ein 4D Client, der gerade beendet, eingefroren und nicht abgeschlossen. Müssen Sie auf Daten zugreifen, während ein Client die Anwendung verlässt, erstellen Sie in der **Datenbankmethode On Exit** einen neuen globalen Prozess, der auf die Datendatei zugreifen kann. Stellen Sie in diesem Fall sicher, dass der neue Prozess korrekt abgeschlossen ist, bevor die **Datenbankmethode On Exit** endet, z.B. über Interprozessvariablen.
Hinweis: In einer Client/Server-Umgebung verhält sich **Datenbankmethode On Exit** unterschiedlich. Das hängt davon ab, ob der Benutzer manuell beendet (über den Menübefehl **Beenden** oder Aufrufen des Befehls **QUIT 4D**) oder ob der 4D Server abschaltet, was alle Clients zum Beenden zwingt.
Schaltet 4D Server ab und bietet eine Zeitspanne zum Beenden an (z.B. 10 Minuten), erscheint auf jedem angemeldeten Client eine entsprechende Meldung. Beendet der Benutzer innerhalb dieses Zeitrahmens, wird die **Datenbankmethode On Exit** normal ausgeführt. Dagegen wird in anderen Fällen (der Benutzer antwortet nicht innerhalb der Zeit, der Server verlangt das sofortige Beenden oder der Client wird manuell vom Administator abgemeldet) die **Datenbankmethode On Exit** zeitgleich mit dem Schließen der Verbindung zum Server ausgeführt. Dann kann der Code in der **Datenbankmethode On Exit** keinen anderen lokalen oder serverseitigen Prozess starten und nicht abwarten, dass andere Prozesse abbrechen (diese können auch nicht mehr auf den Server zugreifen). Solche Versuche generieren einen Netzwerkfehler (10001 oder 10002), da die Verbindung zum Server bereits geschlossen ist.

Um laufende Prozesse bei unerwartetem Abschalten korrekt zu stoppen, müssen Sie die Funktion **Process aborted** in jeder Schleife testen (for, while, repeat), die länger als eine Sekunde laufen kann. **Process aborted** gibt **Wahr** zurück, wenn 4D (lokal, remote oder 4D Server) gerade beendet, d.h. laufende Bearbeitungen stoppen sofort. Brechen Sie in diesem Fall alle Abläufe, wie **CANCEL TRANSACTION**, etc. ab und beenden so schnell wie möglich. Selbst wenn Sie Zeit haben, wenn der Benutzer manuell beendet, verbleibt keine Zeit, wenn die Anwendung zum Beenden gezwungen wird.

Beispiel

Folgendes Beispiel umfasst alle Methoden in der Datenbank zum Verwalten signifikanter Ereignisse während einer Arbeitssitzung und legt eine Beschreibung in ein Textdokument mit Namen "Journal."

- Die **Datenbankmethode On Startup** initialisiert die Interprozessvariable ◇*vbQuit4D*, die allen Benutzerprozessen mitteilt, ob die Datenbank verlassen wurde. Diese erstellt auch das Journal, falls es noch nicht existiert.

```
` Datenbankmethode 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
```

- Die Projektmethode *WRITE JOURNAL*, die von den anderen Methoden als Unterroutine verwendet wird, schreibt die empfangene Information in das Dokument Journal:

```
    ` Projektmethode WRITE JOURNAL
    ` WRITE JOURNAL (Text)
    ` WRITE JOURNAL (Beschreibung des Ereignisses)
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")
WRITE JOURNAL("Sitzung beginnen")
```

Beachten Sie, dass das Dokument jedes Mal geöffnet und geschlossen wird. Außerdem wird eine Semaphore als Zugriffsschutz für das Dokument eingerichtet.—Wir wollen nicht, dass zwei Prozesse gleichzeitig auf das Journal zugreifen können.

- Die Projektmethode *M_ADD_RECORDS* wird ausgeführt, wenn in der Anwendungsumgebung der Menübefehl **Neuer Datensatz** ausgewählt wird:

```
    ` Projektmethode M_ADD_RECORDS
SET MENU BAR(1)
Repeat
    ADD RECORD([Table1];*)
    If(OK=1)
        WRITE JOURNAL("Adding record #"+String(Record number([Table1]))+" in Table1")
    End if
Until((OK=0)|◇vbQuit4D)
```

Diese Methode wird durchlaufen, bis der Benutzer die Dateneingabe abbricht oder die Datenbank verlässt.

- Das Eingabeformular für *[Table 1]* beinhaltet auch die Verwaltung der Ereignisse *On Outside Call*. So kann selbst ein Prozess Dateneingabe bequem verlassen werden, wobei der Benutzer die gerade eingegebenen Daten sichern oder annullieren kann:

```
    ` Formularmethode [Table1];"Input"
Case of
    :(Form event=On Outside Call)
        If(◇vtIPMessage="QUIT")
            CONFIRM("Wollen Sie die Änderungen in diesem Datensatz sichern?")
            If(OK=1)
                ACCEPT
            Else
                CANCEL
            End if
        End if
End case
```

- Die Projektmethode *M_QUIT* wird ausgeführt, wenn in der Anwendungsumgebung im Menü **Datei/Ablage** der Befehl **Beenden** aufgerufen wird:

```
` M_QUIT
$vlProcessID:=New process("DO_QUIT";32*1024;"$DO_QUIT")
```

Die Methode arbeitet mit einem Trick. Beim Aufrufen von **QUIT 4D** wirkt sich der Befehl sofort aus. Der hervorrufende Prozess befindet sich in Stopposition, bis die Datenbank verlassen wird. Da dieser Prozess einer der Prozesse sein kann, in dem Dateneingabe stattfindet, erfolgt der Aufruf von **QUIT 4D** in einem lokalen Prozess, der nur für diesen Zweck startet:

```
` Projektmethode DO_QUIT
CONFIRM("Wollen Sie wirklich beenden?")
If(OK=1)
   WRITE JOURNAL("Datenbank verlassen")
   QUIT 4D
   ` QUIT 4D wirkt sich sofort aus, alle nachfolgenden Programmierzeilen
   ` Werden nicht mehr ausgeführt
End if
```

- Hier folgt nun die **Datenbankmethode On Exit**, die alle offenen Benutzerprozesse auffordert, nun zu beenden. Sie setzt ◇*vbQuit4D* auf Wahr und sendet Interprozessmeldungen an die Benutzerprozesse, die Dateneingabe ausführen:

```
` Datenbankmethode On Exit
◇vbQuit4D:=True
Repeat
   $vbDone:=True
   For($vlProcess;1;Count tasks)
      PROCESS PROPERTIES($vlProcess;$vsProcessName;$vlState;$vlElapsedTime;$vbVisible)
      If((($vsProcessName="ML_@")|($vsProcessName="M_@")))&($vlState>=0))
         $vbDone:=False
         ◇vtIPMessage:="QUIT"
         BRING TO FRONT($vlProcess)
         POST OUTSIDE CALL($vlProcess)
         $vhStart:=Current time
         Repeat
            DELAY PROCESS(Current process;60)
         Until((Process state($vlProcess)<0)|((Current time-$vhStart)>=?00:01:00?))
      End if
   End for
Until($vbDone)
WRITE JOURNAL("Sitzung beenden")
```

Hinweis: Prozesse, die mit *"ML_..."* oder *"M_..."* beginnen, werden von Menübefehlen mit der Eigenschaft **Starte Neuen Prozeß** gestartet. Das sind in diesem Beispiel die Prozesse, die bei Aufrufen des Menübefehls **Neuer Datensatz** starten.

Mit dem Test *(Current time-$vhStart)>=?00:01:00?* kann die Datenbankmethode die Wiederholungsschleife "Warte auf anderen Prozess" verlassen, wenn der andere Prozess nicht sofort aktiv wird.

- Hier ist ein typisches Beispiel für ein Journal, das die Datenbank anlegt:

2.6.97	15:47:25	1	Hauptprozess	Sitzung beginnt
2.6.97	15:55:43	5	ML_1	Datensatz #23 in Tab 1 hinzugefügt
2.6.97	15:55:46	5	ML_1	Datensatz #24 in Tab 1 hinzugefügt
2.6.97	15:55:54	6	$DO_QUIT	Datenbank verlassen
2.6.97	15:55:58	7	$xx	Sitzung beendet

Hinweis: *$xx* ist der Name des lokalen Prozesses, den 4D zur Ausführung von **Datenbankmethode On Exit** startet.

Datenbankmethode On Host Database Event

$1 -> Datenbankmethode On Host Database Event

Parameter	Typ		Beschreibung
$1	Lange Ganzzahl	⇐	Event code

Beschreibung

Mit der **Datenbankmethode On Host Database Event** können 4D Komponenten Code ausführen, wenn die Host Datenbank geöffnet und geschlossen wird.

Hinweis: Aus Sicherheitsgründen müssen Sie, damit Sie diese Datenbankmethode aufrufen können, ihre Ausführung explizit in der Host Datenbank zulassen. Weitere Informationen dazu finden Sie im Abschnitt **Seite Sicherheit** des Handbuchs 4D *Designmodus*.

Die **Datenbankmethode On Host Database Event** wird nur in Datenbanken, die als Komponenten von Host Datenbanken verwendet werden, automatisch ausgeführt (wenn dies in den Einstellungen der Host Datenbank authorisiert ist). Sie wird aufgerufen, wenn beim Öffnen und Schließen der Host Datenbank Ereignisse auftreten.

Um ein Ereignis zu verwalten, müssen Sie den Wert des Parameters *$1* in der Methode testen und ihn mit einer der Konstanten unter dem Thema **Datenbankereignisse** vergleichen:

Konstante	Typ	Wert	Kommentar
On after host database exit	Lange Ganzzahl	4	Die **Semaphore** der Host Datenbank wurde gerade beendet
On after host database startup	Lange Ganzzahl	2	Die **Datenbankmethode On Startup** der Host Datenbank wurde gerade beendet
On before host database exit	Lange Ganzzahl	3	Die Host Datenbank schließt. Die **Semaphore** der Host Datenbank wurde noch nicht aufgerufen. Die **Semaphore** der Host Datenbank wird nicht aufgerufen, während die **Datenbankmethode On Host Database Event** der Komponente läuft
On before host database startup	Lange Ganzzahl	1	Die Host Datenbank wurde gerade gestartet. Die **Datenbankmethode On Startup** der Host Datenbank wurde noch nicht aufgerufen. Die Datenbankmethode On Startup wird nicht aufgerufen, solange die **Datenbankmethode On Host Database Event** der Komponente läuft.

So können 4D Komponenten Voreinstellungen oder Benutzerzustände in Bezug auf die Operation der Host Datenbank laden und sichern.

Beispiel

Beispiel für die typische Struktur einer Datenbankmethode On Host Database Event:

```
// Datenbankmethode On Host Database Event
C_LONGINT($1)
Case of
   :($1=On before host database startup)
// Hier Code setzen, den Sie vor der Datenbankmethode "On Startup" der Host Datenbank ausführen wollen
   :($1=On after host database startup)
// Hier Code setzen, den Sie nach der Datenbankmethode "On Startup" der Host Datenbank ausführen wollen
// database method of the host database
   :($1=On before host database exit)
// Hier Code setzen, den Sie vor der Datenbankmethode "On Exit" der Host Datenbank ausführen wollen
   :($1=On after host database exit)
// Hier Code setzen, den Sie nach der Datenbankmethode "On Exit" der Host Datenbank ausführen wollen
End case
```

Datenbankmethode On Server Close Connection

$1, $2, $3 -> Datenbankmethode On Server Close Connection

Parameter	Typ		Beschreibung
$1	Lange Ganzzahl	⇐	ID Nummer für Benutzer, die 4D Server intern zur Identifikation des Benutzers verwendet.
$2	Lange Ganzzahl	⇐	ID Nummer für Verbindung, die 4D Server intern zur Identifikation einer Verbindung verwendet.
$3	Lange Ganzzahl	⇐	Obsolet: Gibt immer 0 zurück, muss aber deklariert werden.

Beschreibung

Die **Datenbankmethode On Server Close Connection** wird jedes Mal auf dem Server-Rechner aufgerufen, wenn ein 4D Client Prozess endet.

Analog zur **Datenbankmethode On Server Open Connection** übergibt 4D Server in der **Datenbankmethode On Server Close Connection** drei Parameter vom Typ Lange Ganzzahl, er erwartet jedoch kein Ergebnis.

Die Methode muss deshalb ausdrücklich mit drei Parametern vom Typ Lange Ganzzahl definiert werden:

```
C_LONGINT($1;$2;$3)
```

Die drei Parameter für die Datenbankmethode geben folgende Informationen zurück:

Parameter	Beschreibung
$1	ID Nummer für Benutzer, die 4D Server intern zur Identifikation des Benutzers verwendet.
$2	ID Nummer für Verbindung, die 4D Server intern zur Identifikation einer Verbindung verwendet.
$3	Obsolet: Gibt immer 0 zurück, muss aber deklariert werden.

Die **Datenbankmethode On Server Close Connection** ist das exakte Gegenstück zur **Datenbankmethode On Server Open Connection**. Weitere Informationen und eine Beschreibung der remote 4D Prozesse finden Sie in der Beschreibung dieser Datenbankmethode.

Beispiel

Siehe erstes Beispiel unter der **Datenbankmethode On Server Open Connection**.

Datenbankmethode On Server Open Connection

$1, $2, $3 -> Datenbankmethode On Server Open Connection -> $0

Parameter	Typ		Beschreibung
$1	Lange Ganzzahl	⇐	ID Nummer für Benutzer, die 4D Server intern zum Identifizieren der Anwender verwendet.
$2	Lange Ganzzahl	⇐	ID Nummer für Verbindung, die 4D Server intern zum Identifizieren einer Verbindung verwendet.
$3	Lange Ganzzahl	⇐	Obsolet: Gibt immer 0 zurück, muss aber deklariert werden.
$0	Lange Ganzzahl	⇒	0 oder weggelassen = Verbindung akzeptiert; anderer Wert = Verbindung verweigert

Wann wird diese Datenbankmethode aufgerufen?

Die **Datenbankmethode On Server Open Connection** wird einmal auf dem Server-Rechner aufgerufen, wenn sich ein remote 4D anmeldet. Diese Datenbankmethode wird ausschließlich in der 4D Server Umgebung aufgerufen.

Die **Datenbankmethode On Server Open Connection** wird immer aufgerufen, wenn:

- Ein remote 4D sich anmeldet (weil der Prozess Anwendung startet)
- Ein remote 4D die Designumgebung öffnet (weil der Prozess Design startet)
- Ein remote 4D einen globalen Prozess startet (Name beginnt nicht mit $), der das Erstellen eines kooperativen Prozesses auf dem Server erfordert (*). Dieser Prozess lässt sich über die Funktion **New process**, über einen Menübefehl oder das Dialogfenster Methode ausführen erstellen.

Bei den Aktionen mit remote 4D starten verschiedene Prozesse — Einer auf dem Client-Rechner und ein bzw. bei Bedarf zwei weitere auf dem Server-Rechner. Auf dem Client-Rechner führt der Prozess Code aus und sendet Anfragen an 4D Server. Auf dem Server-Rechner verwaltet der **4D Client-Prozess** (präemptiver Prozess) die Datenbankumgebung für den Client-Prozess (z.B. aktuelle Auswahl und Datensatz sperren für Benutzerprozesse) und beantwortet Anfragen, die vom Prozess auf dem Client-Rechner gesendet wurden. Der **4D Client Datenbankprozess** (kooperativer Prozess) ist für die Überwachung des entsprechenden 4D Client Prozesses zuständig.

(*) Ab 4D v13 werden die Server Prozesse (ein präemptiver Prozess für den Zugriff auf die Datenbank-Engine und ein kooperativer Prozess für den Zugriff auf die Programmiersprache) aus Optimierungsgründen nur erstellt, wenn sie beim Ausführen des Code auf Client-Seite notwendig sind. Hier ein Beispiel für eine 4D Code Sequenz, die in einem neuen Client Prozess läuft:

```
 // globaler Prozess beginnt ohne einen neuen Prozess auf dem Server, wie ein lokaler Prozess.
CREATE RECORD([Table_1])
[Table_1])field1_1:="Hallo Welt"
SAVE RECORD([Table_1]) // erstellt hier präemptiven Prozess auf Server
$serverTime:=Current time(*) // erstellt hier kooperativen Prozess auf Server
 //Aufruf von On Server Open Connection
```

Wichtig: Web-Verbindungen und SQL-Verbindungen lösen nicht die **Datenbankmethode On Server Open Connection** aus. Meldet sich ein Web Browser an 4D Server an, startet die **Datenbankmethode On Web Authentication** - sofern vorhanden, und/oder die **Datenbankmethode On Web Connection**. Empfängt 4D Server eine SQL Anfrage, wird die **Datenbankmethode On SQL Authentication** - sofern vorhanden, aufgerufen.

Wichtig: Startet eine Serverprozedur, wird NICHT die **Datenbankmethode On Server Open Connection** ausgelöst. **Serverprozeduren** sind Server- und keine Client-Prozesse. Sie führen Code auf dem Server-Rechner aus, beantworten jedoch keine Anfragen von remote 4D oder anderen Clients an 4D Server.

Wie wird diese Datenbankmethode aufgerufen?

Die **Datenbankmethode On Server Open Connection** wird auf dem 4D Server-Rechner innerhalb des 4D Client-Prozesses ausgeführt, der die Anweisung gibt, die Methode aufzurufen.

Beispiel: Meldet sich ein remote 4D an eine interpretierte Datenbank auf 4D Server an, starten standardmäßig die Prozesse Benutzer, Design und Registrierung für diesen Client. Die **Datenbankmethode On Server Open Connection** wird also dreimal ausgeführt — einmal innerhalb des Anwendungsprozesses, ein zweites Mal innerhalb des Client Registrierungsprozesses und ein drittes Mal innerhalb des Prozesses Design. Sind diese Prozesse jeweils der sechste, siebte und achte gestartete Prozess auf dem Server-Rechner und rufen Sie **Current process** innerhalb der **Datenbankmethode On Server Open Connection** auf, gibt die Funktion **Current process** jeweils 6, 7 und 8 zurück.

Beachten Sie, dass die **Datenbankmethode On Server Open Connection** auf dem Server-Rechner ausgeführt wird. Sie läuft im 4D Client Prozess ab, der auf dem Server läuft, unabhängig vom Prozess auf der Client-Seite. Außerdem ist der 4D Client Prozess in dem Moment, wo die Methode ausgelöst wird, noch nicht benannt. (Der Befehl **PROCESS PROPERTIES** gibt zu diesem Zeitpunkt nicht den Namen des 4D Client Prozesses zurück).

Die **Datenbankmethode On Server Open Connection** hat keinen Zugriff auf die Tabelle Prozessvariablen des Prozesses auf der Client-Seite, da diese Tabelle auf dem Client-Rechner und nicht auf dem Server-Rechner liegt.

Greift die **Datenbankmethode On Server Open Connection** auf eine Prozessvariable zu, läuft sie in einer eigenen und dynamisch erstellten Tabelle Prozessvariablen für den 4D Client Prozess ab.

4D Server übergibt der **Datenbankmethode On Server Open Connection** drei Parameter vom Typ Lange Ganzzahl und erwartet ein Ergebnis vom Typ Lange Ganzzahl. Die Methode muss deshalb ausdrücklich mit drei Parametern vom Typ Lange Ganzzahl und mit dem Funktionsergebnis vom Typ Lange Ganzzahl deklariert werden:

```
C_LONGINT($0;$1;$2;$3)
```

Geben Sie keinen Wert in *$0* zurück, d.h. die Variable ist undefiniert oder gegen Null initialisiert, nimmt 4D Server an, dass die Datenbankmethode die Verbindung akzeptiert. Akzeptieren Sie die Verbindung nicht, geben Sie in *$0* einen Wert zurück, der nicht Null ist.

Die drei Parameter für die Datenbankmethode geben folgende Informationen zurück:

Parameter	Beschreibung
$1	ID Nummer für Benutzer, die 4D Server intern zur Identifikation von Benutzern verwendet.
$2	ID Nummer für Verbindung, die 4D Server intern zur Identifikation einer Verbindung verwendet.
$3	Obsolet: Gibt immer 0 zurück, muss aber deklariert werden.

Diese ID Nummern sind nicht direkt als Informationsquellen einsetzbar, z.B. um sie in einem 4D Befehl als Parameter zu übergeben. Sie ermöglichen jedoch, einen 4D Client Prozess zwischen der **Datenbankmethode On Server Open Connection** und der **Datenbankmethode On Server Close Connection** eindeutig zu identifizieren. Die Kombination dieser Werte ist zu jedem Moment einer Server Sitzung einmalig. Ist diese Information in einer Interprozess-Array bzw. Tabelle gespeichert, können beide Datenbankmethoden Informationen austauschen. Im Beispiel am Ende dieses Abschnitts verwenden die beiden Datenbankmethoden diese Information, um Datum und Uhrzeit für Beginn und Ende einer Verbindung zum gleichen Datensatz einer Tabelle zu speichern.

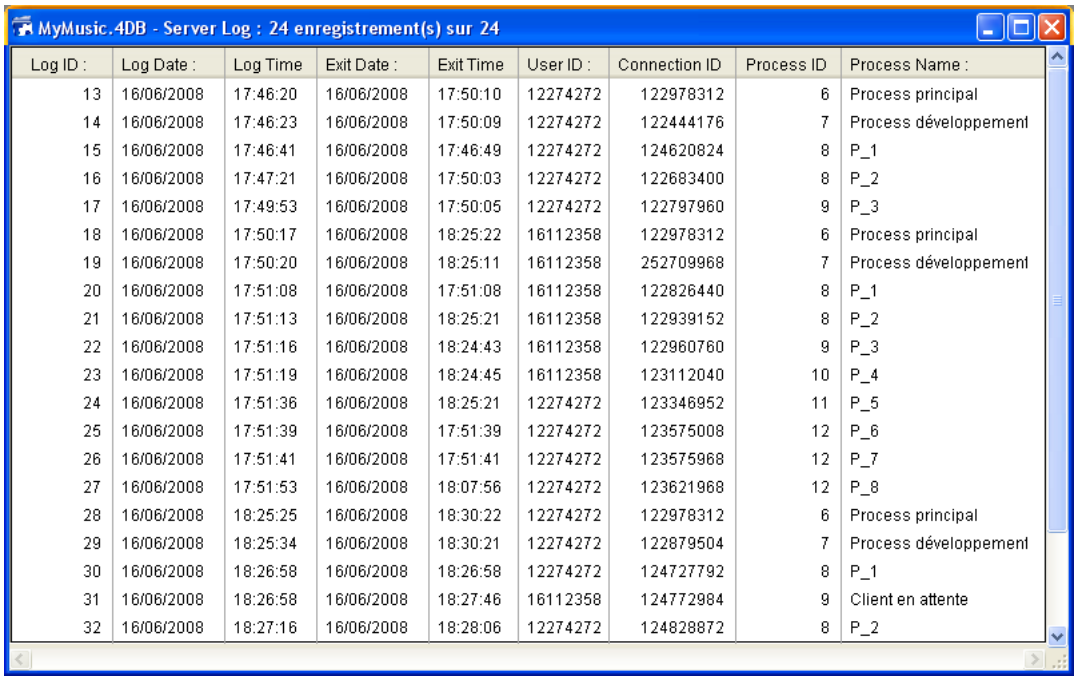
Beispiel 1

Dieses Beispiel zeigt, wie mit der **Datenbankmethode On Server Open Connection** und der **Datenbankmethode On Server Close Connection** in der Datenbank ein Logbuch der Verbindung verwaltet wird. Die Tabelle [Server Log] wird zum Auffinden der Verbindungsprozesse verwendet.

Die hier gespeicherte Information wird von der **Datenbankmethode On Server Open Connection** und der **Datenbankmethode On Server Close Connection** wie folgt verwaltet:

```
  ` Datenbankmethode On Server Open Connection
C_LONGINT($0;$1;$2;$3)
  ` Erstelle eine Datensatz [Logbuch]
CREATE RECORD([Server_Log])
[Server_Log]Log_Nr:=Sequence number([Server_Log])
  ` Sichere Anmeldedatum und Anmeldezeit
[Server_Log]Anmeldedatum:=Current date
[Server_Log]Anmeldezeit:=Current time
  ` Sichere die Verbindungsinformation
[Server_Log]Benutzer_Nr:=$1
[Server_Log]Verbindung_Nr:=$2
SAVE RECORD([Server_Log])
  ` Gibt keinen Fehler zurück, so dass die Verbindung weiterlaufen kann
$0:=0

  ` Datenbankmethode On Server Close Connection
C_LONGINT($1;$2;$3)
  ` Finde wieder Datensatz [Server_Log]
QUERY([Server_Log];[Server_Log]Benutzer_Nr=$1;*)
QUERY([Server_Log]; & ;[Server_Log]Verbindung_Nr=$2;*)
QUERY([Server_Log]; & ;[Server_Log]Prozess_Nr=0)
  ` Sichere Abmeldedatum und -zeit
[Server_Log]Abmeldedatum:=Current date
[Server_Log]Abmeldezeit:=Current time
  ` Sichere die Prozessinformation
[Server_Log]Prozess_Nr:=Current process
PROCESS PROPERTIES([Server_Log]Prozess_Nr;$vsProzName;$vlProzStatus;$vlProzZeit)
[Server_Log]Prozessname:=$vsProzName
SAVE RECORD([Server_Log])
```

Nachfolgend sehen Sie einige Einträge in [Server_Log] mit mehreren remote Anbindungen:



MyMusic.4DB - Server Log : 24 enregistrement(s) sur 24

Log ID :	Log Date :	Log Time	Exit Date :	Exit Time	User ID :	Connection ID	Process ID	Process Name :
13	16/06/2008	17:46:20	16/06/2008	17:50:10	12274272	122978312	6	Process principal
14	16/06/2008	17:46:23	16/06/2008	17:50:09	12274272	122444176	7	Process développement
15	16/06/2008	17:46:41	16/06/2008	17:46:49	12274272	124620824	8	P_1
16	16/06/2008	17:47:21	16/06/2008	17:50:03	12274272	122683400	8	P_2
17	16/06/2008	17:49:53	16/06/2008	17:50:05	12274272	122797960	9	P_3
18	16/06/2008	17:50:17	16/06/2008	18:25:22	16112358	122978312	6	Process principal
19	16/06/2008	17:50:20	16/06/2008	18:25:11	16112358	252709968	7	Process développement
20	16/06/2008	17:51:08	16/06/2008	17:51:08	16112358	122826440	8	P_1
21	16/06/2008	17:51:13	16/06/2008	18:25:21	16112358	122939152	8	P_2
22	16/06/2008	17:51:16	16/06/2008	18:24:43	16112358	122960760	9	P_3
23	16/06/2008	17:51:19	16/06/2008	18:24:45	16112358	123112040	10	P_4
24	16/06/2008	17:51:36	16/06/2008	18:25:21	12274272	123346952	11	P_5
25	16/06/2008	17:51:39	16/06/2008	17:51:39	12274272	123575008	12	P_6
26	16/06/2008	17:51:41	16/06/2008	17:51:41	12274272	123575968	12	P_7
27	16/06/2008	17:51:53	16/06/2008	18:07:56	12274272	123621968	12	P_8
28	16/06/2008	18:25:25	16/06/2008	18:30:22	12274272	122978312	6	Process principal
29	16/06/2008	18:25:34	16/06/2008	18:30:21	12274272	122879504	7	Process développement
30	16/06/2008	18:26:58	16/06/2008	18:26:58	12274272	124727792	8	P_1
31	16/06/2008	18:26:58	16/06/2008	18:27:46	16112358	124772984	9	Client en attente
32	16/06/2008	18:27:16	16/06/2008	18:28:06	12274272	124828872	8	P_2

Beispiel 2

Im folgenden Beispiel kann zwischen 2 und 4 Uhr a.m. keine neue Verbindung hergestellt werden:

```
  ` Datenbankmethode 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
```

Datenbankmethode On Server Shutdown

Datenbankmethode On Server Shutdown
Dieser Befehl benötigt keine Parameter

Die **Datenbankmethode On Server Shutdown** wird einmal auf dem Server-Rechner aufgerufen, wenn die aktuelle Datenbank auf dem Server beendet wird. Diese Datenbankmethode wird ausschließlich in der 4D Server Umgebung aufgerufen.
Um die aktuelle Datenbank auf dem Server zu schließen, können Sie auf dem Server den Menübefehl **Schließe Anwendung** oder **Beenden** wählen oder in einer Serverprozedur, die auf dem Server ausgeführt wird, den Befehls **QUIT 4D** aufrufen.
Beim Verlassen der Datenbank führt 4D folgende Aktionen durch:

- Ist keine **Datenbankmethode On Server Shutdown** vorhanden, beendet 4D Server ohne Unterscheidung der Reihe nach jeden Prozess.
- Ist eine **Datenbankmethode On Server Shutdown** vorhanden, führt 4D Server diese Methode in einem neu angelegten lokalen Prozess durch. Sie können so via Interprozesskommunikation anderen Prozessen mitteilen, die Ausführung zu beenden. Beachten Sie, dass 4D Server evtl. beendet wird – die **Datenbankmethode On Server Shutdown** kann zwar alle gewünschten Operationen zum Aufräumen oder Schließen durchführen, aber nicht das Beenden an sich verweigern, und so an einem gewissen Punkt enden.

Die **Datenbankmethode On Server Shutdown** ist der richtige Ort, um:

- Serverprozeduren zu beenden, die beim Öffnen der Datenbank automatisch starten.
- Voreinstellungen bzw. Einstellungen (lokal oder auf Festplatte) zu speichern, um sie beim Starten der nächsten Sitzung in der **Datenbankmethode On Server Startup** wiederzuverwenden.
- Andere Aktionen auszuführen.
- Andere Aktionen auszuführen, die bei jedem Beenden der Datenbank automatisch ablaufen sollen.

Wichtig: Beachten Sie beim Schließen von Serverprozeduren über die **Datenbankmethode On Server Shutdown**, dass der Server abschaltet, wenn die Ausführung dieser Datenbankmethode (und nicht der Serverprozeduren) beendet ist. Noch laufende Serverprozeduren werden abgebrochen.
Wollen Sie sicherstellen, dass die Serverprozeduren vollständig ausgeführt sind, bevor der Server abschaltet, muss diese Datenbankmethode eine Anweisung zum Beenden geben, z.B. über Testen einer Interprozessvariablen und die Zeit zum Beenden gewähren. Dazu können Sie eine Schleife von n-Sekunden oder einen Test durch eine andere Interprozessvariable einrichten.
Um Code automatisch auf einem Client-Rechner auszuführen, wenn sich ein remote 4D am Server abmeldet, verwenden Sie die **Semaphore**.

Datenbankmethode On Server Startup

Datenbankmethode On Server Startup
Dieser Befehl benötigt keine Parameter

Die **Datenbankmethode On Server Startup** wird einmal auf dem Server-Rechner aufgerufen, wenn Sie eine Datenbank mit 4D Server öffnen. Die **Datenbankmethode On Server Startup** wird ausschließlich in der 4D Server Umgebung aufgerufen.

Die **Datenbankmethode On Server Startup** ist der richtige Ort, um:

- Interprozessvariablen zu initialisieren, die Sie während der gesamten 4D Server Sitzung verwenden.
- **Serverprozeduren** automatisch beim Öffnen der Datenbank zu starten.
- Voreinstellungen bzw. Einstellungen, die während der letzten 4D Server Sitzung gesichert wurden, zu laden.
- Durch explizites Aufrufen von **QUIT 4D** das Öffnen der Datenbank nicht zuzulassen, wenn eine Bedingung nicht erfüllt ist (z.B. fehlende System-Ressourcen)
- Weitere Aktionen auszuführen, die bei jedem Öffnen der Datenbank automatisch ablaufen sollen.

Um Code automatisch auf einem Client-Rechner auszuführen, wenn sich ein remote 4D am Server anmeldet, verwenden Sie die **Datenbankmethode On Server Startup**.

Hinweis: Die **Datenbankmethode On Server Startup** wird automatisch ausgeführt, d.h. ein remote 4D kann sich erst anmelden, wenn die Ausführung der Methode abgeschlossen ist.

Datenbankmethode On SQL Authentication

$1, $2, $3 -> Datenbankmethode On SQL Authentication -> Funktionsergebnis

Parameter	Typ		Beschreibung
$1	Text	⇐	Benutzername
$2	Text	⇐	Kennwort
$3	Text	⇐	(optional) IP Adresse des Client am Ursprung der Anfrage
Funktionsergebnis	Boolean	⇒	Wahr = Anfrage angenommen, Falsch = Anfrage verweigert

---

Mit der **Datenbankmethode On SQL Authentication** können Sie Anfragen filtern, die an den integrierten SQL Server von 4D gesendet werden. Der Filter kann auf Name und Kennwort sowie optional auf der IP Adresse des Benutzers basieren. Entwickler können ihre eigene Benutzertabelle oder die der 4D Benutzer zum Prüfen der Identifizierer der Verbindung verwenden. Wurde die Anmeldung authentifiziert, muss der Befehl **CHANGE CURRENT USER** aufgerufen werden, um den Zugriff auf Anfragen innerhalb der 4D Datenbank zu steuern.

4D oder 4D Server rufen die **Datenbankmethode On SQL Authentication** - sofern vorhanden - bei jeder externen Anmeldung an den SQL Server automatisch auf. Deshalb wird das interne System zum Verwalten von 4D Benutzern nicht aktiviert. Die Verbindung wird nur angenommen, wenn die Datenbankmethode in $0 **Wahr** zurückgibt und der Befehl **CHANGE CURRENT USER** erfolgreich ausgeführt wurde. Ist eine dieser Bedingungen nicht erfüllt, wird die Anfrage verweigert.

Hinweis: Die Anweisung **SQL LOGIN**(SQL_INTERNAL;$Benutzer;$Kennwort) ruft nicht die **Datenbankmethode On SQL Authentication** auf, da es sich in diesem Fall um eine interne Anmeldung handelt.

Die Datenbankmethode empfängt von 4D bis zu drei Parameter vom Typ ($1, $2, $3) und gibt $0 als Boolean Wert zurück:

Parameter	Typ	Beschreibung
$1	Text	Benutzername
$2	Text	Kennwort
$3	Text	(optional) IP Adresse des Client am Ursprung der Anfrage (*)
$0	Boolean	Wahr = Anfrage angenommen, Falsch = Anfrage verweigert

(*) 4D gibt IPv4 Adressen in einem hybrid IPv6/IPv4 Format mit einem 96-bit Prefix zurück, z.B. ::ffff:192.168.2.34 für die IPv4 Adresse 192.168.2.34. Weitere Informationen dazu finden Sie im Abschnitt **Unterstützung von IPv6**.

Sie müssen diese Parameter folgendermaßen deklarieren:

```
  ` Datenbankmethode On Web Authentication
C_TEXT($1;$2;$3)
C_BOOLEAN($0)
  ` Code für Methode
```

Das Kennwort ($2) wird als Standardtext empfangen.

Die Identifizierer der SQL Verbindung müssen Sie über die **Datenbankmethode On SQL Authentication** prüfen. Sie können z.B. Name und Kennwort über eine eigene Benutzertabelle prüfen. Sind die Identifizierer gültig, übergeben Sie **Wahr** in $0, um die Verbindung zuzulassen.
Andernfalls übergeben Sie **Falsch** in $0; dann wird die Verbindung abgewiesen.

Hinweis: Gibt es keine **Datenbankmethode On SQL Authentication**, wird die Verbindung über das in 4D integrierte System zur Benutzerverwaltung bewertet (wenn es aktiviert ist, d.h. wenn dem Designer ein Kennwort zugewiesen ist). Ist das System nicht aktiviert, werden Benutzer mit Designer-Zugriffsrechten angemeldet, d.h. mit uneingeschränktem Zugriff.

Haben Sie in $0 Wahr übergeben, müssen Sie dann in der **Datenbankmethode On SQL Authentication** den Befehl **CHANGE CURRENT USER** erfolgreich aufrufen, damit die Anfrage angenommen wird und 4D eine SQL Sitzung für den Benutzer öffnet. Über den Befehl **CHANGE CURRENT USER** können Sie ein virtuelles Authentifizierungssystem einrichten mit doppeltem Vorteil: Es ermöglicht einerseits die Steuerung der Verbindungsaktionen und blendet andererseits die Identifizierer der Verbindung aus, so dass sie von außerhalb der 4D SQL Sitzung nicht einsehbar sind.

Das folgende Beispiel der **Datenbankmethode On SQL Authentication** prüft, ob die Anfrage der Verbindung vom internen Netzwerk stammt, bestätigt die Identifizierer und weist dann dem Benutzer "sql_user" die Zugriffsrechte für die SQL Sitzung zu.

```
C_TEXT($1;$2;$3)
C_BOOLEAN($0)
  `$1: Benutzer
  `$2: Kennwort
  `{$3: IP Adresse des Client}
ON ERR CALL("SQL_Fehler")
If(checkInternalIP($3))
  `Die Methode checkInternalIP prüft, ob die IP Adresse intern ist
   If($1="Viktor") & ($2="Hugo")
      CHANGE CURRENT USER("sql_Benutzer";"")
      If(OK=1)
         $0:=True
      Else
         $0:=False
      End if
   Else
```

```
            $0:=False
        End if
    Else
        $0:=False
    End if
```

Datenbankmethode On Startup

Datenbankmethode On Startup
Dieser Befehl benötigt keine Parameter

Die **Datenbankmethode On Startup** wird einmal aufgerufen, wenn Sie eine Datenbank öffnen.
Das geschieht in folgenden 4D Umgebungen:

- 4D im lokalen Modus
- 4D im remote Modus (auf der Client-Seite, nachdem 4D Server die Verbindung akzeptiert hat)
- 4D Anwendung mit integrierter 4D Volume Desktop

Hinweis: Die **Datenbankmethode On Startup** wird NICHT von 4D Server ausgelöst.
Die **Datenbankmethode On Startup** wird automatisch von 4D ausgelöst; diese Datenbankmethode können Sie im Gegensatz zu Projektmethoden nicht selbst per Programmierung aufrufen. Sie können auch Unterroutinen verwenden.
Die **Datenbankmethode On Startup** ist der richtige Ort, um:

- Interprozessvariablen zu initialisieren, die Sie während der gesamten Arbeitssitzung verwenden.
- Prozesse automatisch beim Öffnen der Datenbank zu starten.
- Voreinstellungen bzw. Einstellungen aus der letzten Sitzung zu laden.
- Öffnen der Datenbank über den Befehl **QUIT 4D** zu verhindern, wenn eine Bedingung nicht erfüllt ist (z.B. fehlende System-Ressourcen).
- Andere Aktionen auszuführen, die bei jedem Öffnen der Datenbank automatisch ablaufen sollen.

Wir empfehlen jedoch dringend, mit der **Datenbankmethode On Startup** KEINE Druckaufträge zu starten.

Beispiel

Siehe Beispiel unter der **Semaphore**.

Datenbankmethode On System Event

$1 -> Datenbankmethode On System Event

Parameter	Typ		Beschreibung
$1	Lange Ganzzahl	⇦	Event code

Beschreibung

Die **Datenbankmethode On System Event** wird jedes Mal aufgerufen, wenn ein Systemereignis eintritt. Das betrifft alle 4D Umgebungen: 4D (alle Modi) und 4D Server, sowie kompilierte 4D Anwendungen mit eingebundener 4D Volume Desktop.
Um ein Ereignis zu bearbeiten, müssen Sie den Wert des Parameters $1 innerhalb der Methode testen und ihn mit einer der folgenden Konstanten unter dem Thema **Datenbankereignisse** vergleichen:

Konstante	Typ	Wert	Kommentar
On application background move	Lange Ganzzahl	1	Die 4D Anwendung geht in den Hintergrund
On application foreground move	Lange Ganzzahl	2	Die 4D Anwendung kommt in den Vordergrund

Diese Ereignisse werden erzeugt, wenn eine 4D Anwendung die Ebene als Folge einer Benutzeraktion wechselt. Zum Beispiel:

- Das Fenster dieser bzw. einer anderen Anwendung wird angeklickt
- Das Fenster wird über die Tastenkombination **Alt+Tab** unter Windows, **Option+Tab** auf Mac OS ausgewählt
- Auf Mac OS wird der Befehl **Ausblenden** im Dock gewählt
- Im Dock oder in der Werkzeugleiste wird auf das Icon der Anwendung geklickt
- Unter Windows wird im Hauptfenster auf die Schaltfläche **Minimieren** geklickt

Es ist absolut notwendig, den Parameter $1 (Lange Ganzzahl) in der Datenbankmethode zu deklarieren. Der Code der Datenbankmethode ist wie folgt:

```
// Datenbankmethode On System Event
C_LONGINT($1)
Case of
   :($1=On application background move)
  // Etwas ausführen
   :($1=On application foreground move)
  // Etwas anderes ausführen
End case
```

Datenbankmethode On Web Authentication

$1, $2, $3, $4, $5, $6 -> Datenbankmethode On Web Authentication -> $0

Parameter	Typ		Beschreibung
$1	Text	⬅	URL
$2	Text	⬅	HTTP header + HTTP body
$3	Text	⬅	IP Adresse des Browser
$4	Text	⬅	IP Adresse des Server
$5	Text	⬅	Benutzername
$6	Text	⬅	Kennwort
$0	Boolean	➲	Wahr=Anfrage akzeptiert; Falsch=Anfrage abgewiesen

Beschreibung

Die **Datenbankmethode On Web Authentication** verwaltet die Zugriffe auf den Web Server. Sie wird von 4D oder 4D Server aufgerufen, wenn eine Anfrage des Web Browser die Ausführung einer 4D Methode auf dem Server erfordert (Methode, die über eine URL *4DACTION* oder *4DCGI*, ein Tag *4DSCRIPT*, etc. aufgerufen wird).

Diese Methode empfängt die Text Parameter *$1*, *$2*, *$3*, *$4*, *$5* und *$6* und gibt einen Boolean Parameter $0 zurück:

Parameter	Typ	Beschreibung
$1	Text	URL
$2	Text	HTTP Kopfteil + HTTP body (32 KB maximum)
$3	Text	IP Adresse des Web Client (Browser)
$4	Text	IP Adresse des Server
$5	Text	Benutzername
$6	Text	Kennwort
$0	Boolean	Wahr = Anfrage angenommen, Falsch = Anfrage abgewiesen

Sie müssen diese Parameter folgendermaßen deklarieren:

` Datenbankmethode On Web Authentication

```
C_TEXT($1;$2;$3;$4;$5;$6)
C_BOOLEAN($0)
```

` Code für die Methode

Hinweis: Unter Umständen sind nicht alle Parameter der **Datenbankmethode On Web Authentication** angegeben. Die von der Datenbankmethode empfangene Information richtet sich nach den Optionen, die Sie zuvor im Dialogfenster Datenbank-Eigenschaften ausgewählt haben. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

- **URL**
  Der erste Parameter ($1) ist die **URL**, welche der Benutzer im Bereich Location seines Web Browsers eingibt, aus der die Host Adresse entfernt wurde.
  Nehmen wir als Beispiel eine Intranet Verbindung. Die IP Adresse Ihres 4D Web Server Rechners ist 123.4.567.89. Nachfolgende Tabelle zeigt die Werte von $1, je nachdem, welche **URL** im Web Browser eingegeben wurde:

URL im Bereich Location des Web Browsers	Wert des Parameters $
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Kunden	/Kunden
http://123.4.567.89/Kunden	/Kunden
http://123.4.567.89/Kunden/Hinzufügen	/Kunden/Hinzufügen
123.4.567.89/Führe_dieses_aus/Wenn_OK/	/Führe_dieses_aus/Wenn_OK
Führe_jenes_aus	/Führe_jenes_aus

- **Kopfteil der HTTP Anfrage**
  Der zweite Parameter ($2) ist der Kopfteil der HTTP Anfrage, die der Web Browser sendet. Beachten Sie, dass dieser Kopfteil komplett an Ihre **Datenbankmethode On Web Authentication** übergeben wird. Sein Inhalt variiert je nach Art des Web Browsers, der versucht, die Verbindung herzustellen.
  Verwendet Ihre Anwendung diese Information, entscheiden Sie selbst, ob der Kopfteil durchlaufen werden soll.

Hinweise:

- Aus Performance-Gründen darf die Datengröße, die über den Parameter $2 läuft, nicht größer als 32 KB sein. Was darüber hinaus geht, wird vom 4D HTTP Server abgeschnitten. Weitere Informationen zu diesem Parameter finden Sie im Abschnitt **Datenbankmethode On Web Connection**.

- **IP Adresse des Web Client**
  Der Parameter $3 empfängt die IP Adresse des Browser Rechners. Mit dieser Information können Sie zwischen Intranet- und Internet-Verbindungen unterscheiden.
  Hinweis: (*) 4D gibt IPv4 Adressen in einem hybrid IPv6/IPv4 Format mit einem 96-bit Prefix zurück, z.B. ::ffff:192.168.2.34 für die IPv4 Adresse 192.168.2.34. Weitere Informationen dazu finden Sie im Abschnitt **Unterstützung von IPv6**.

- **IP Adresse des Server**
  Der Parameter $4 empfängt die IP Adresse des 4D Web Servers. Dies ermöglicht Multi-Homing, d.h., Sie können Rechner mit mehr als einer IP Adresse nutzen. Weitere Informationen dazu finden Sie im Abschnitt **Web Server, Einstellungen**.

- **Benutzername und Kennwort**
  Die Parameter $5 und $6 empfangen Benutzername und Kennwort, die der Benutzer im Dialogfenster Standard-Identifikation des Browsers eingibt. Dieser Dialog erscheint für jede Verbindung, wenn im Dialogfenster Datenbank-Eigenschaften eine Option zur Kennwortverwaltung ausgewählt wurde (siehe Abschnitt **Sicherheit der Verbindung**).

Hinweis: Gibt es den vom Browser gesendeten Benutzernamen in 4D, wird der Parameter $6 (das Kennwort des Benutzers) aus Sicherheitsgründen nicht zurückgegeben.

- **Parameter $0**
  Die **Datenbankmethode On Web Authentication** gibt in $0 einen booleschen Wert zurück:
  - Ist $0 **True**, wird die Verbindung angenommen.
  - Ist $0 **False**, wird die Verbindung zurückgewiesen.

Die **Datenbankmethode On Web Connection** wird nur ausgeführt, wenn die Verbindung von **On Web Authentication** angenommen wurde.

Warnung: Übergeben Sie keinen Wert in $0 oder ist $0 in der **Datenbankmethode On Web Authentication** nicht definiert, wird die Verbindung als akzeptiert angesehen und die **Datenbankmethode On Web Connection** wird ausgeführt.

Hinweise:

- Rufen Sie in der **Datenbankmethode On Web Authentication** keine Elemente der Oberfläche auf (**ALERT**, **DIALOG**, etc.), denn das unterbricht die Datenbankmethode und die Verbindung wird zurückgewiesen. Dasselbe gilt, wenn beim Ausführen der Datenbankmethode ein Fehler auftritt.
- Sie können die Ausführung durch 4DACTION oder 4DSCRIPT für jede Projektmethode verbieten, wenn Sie im Dialogfenster Methode-Eigenschaften die Option "Zugang per 4D HTML Tags und URLs (4DACTION...)" markieren. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

Aufrufe der Datenbankmethode On Web Authentication

Die **Datenbankmethode On Web Authentication** wird automatisch aufgerufen, unabhängig vom Modus, wenn eine Anfrage oder Bearbeitung die Ausführung einer 4D Methode erfordert. Sie wird auch aufgerufen, wenn der Web Server eine ungültige statische URL empfängt, z.B. wenn die angefragte statische Seite nicht vorhanden ist.

Die Methode wird in folgenden Fällen aufgerufen:

- Wenn 4D eine URL empfängt, die mit 4DACTION/ beginnt
- Wenn 4D eine URL empfängt, die mit 4DCGI/ beginnt
- Wenn 4D eine URL empfängt, die mit 4DSYNC/ beginnt
- Wenn 4D eine URL empfängt, die eine statische Seite aufruft, die nicht vorhanden ist
- Wenn 4D eine URL mit Root Zugriff empfängt und in den Datenbank-Eigenschaften oder über den Befehl **WEB SET HOME PAGE** keine Home Page gesetzt wurde.
- Wenn 4D ein Tag 4DSCRIPT in einer halbdynamischen Seite abarbeitet
- Wenn 4D ein Tag 4D LOOP abarbeitet, das auf einer Methode in einer halbdynamischen Seite basiert.

Hinweis zur Kompatibilität: Die Datenbankmethode wird auch aufgerufen, wenn 4D eine URL empfängt, die mit 4DMETHOD/ beginnt. Diese URL ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten.

Beachten Sie, dass die **Datenbankmethode On Web Authentication** NICHT aufgerufen wird, wenn der Server eine URL empfängt, die nach einer statischen gültigen Seite fragt.

Beispiel 1

Beispiel für die **Datenbankmethode On Web Authentication** im BASIC Modus:

```
  `Datenbankmethode On Web Authentication
C_TEXT($5;$6;$3;$4)
C_TEXT($user;$password;$BrowserIP;$ServerIP)
C_BOOLEAN($4Duser)
ARRAY TEXT($users;0)
ARRAY LONGINT($nums;0)
C_LONGINT($upos)
C_BOOLEAN($0)

$0:=False

$user:=$5
$password:=$6
$BrowserIP:=$3
$ServerIP:=$4

  `Verweigere aus Sicherheitsgründen Namen mit @
If(WithWildcard($user)|WithWildcard($password))
   $0:=False
  `Die Methode WithWildcard folgt unten
Else
  `Prüfe, ob es ein 4D Benutzer ist
```

```
    GET USER LIST($users;$nums)
    $upos:=Find in array($users;$user)
    If($upos >0)
        $4Duser:=Not(Is user deleted($nums{$upos}))
    Else
        $4Duser:=False
    End if

    If(Not($4Duser))
  `Ist es kein 4D Benutzer, suche in der Tabelle WebUser
        QUERY([WebUser];[WebUser]User=$user;*)
        QUERY([WebUser]; & [WebUser]Password=$password)
        $0:=(Records in selection([WebUser])=1)
    Else
        $0:=True
    End if
End if
  `Ist es eine Intranet Verbindung?
If(Substring($BrowserIP;1;7)#"192.100.")
    $0:=False
End if
```

Beispiel 2

Beispiel für die **Datenbankmethode On Web Authentication** im DIGEST Modus:

```
  `Datenbankmethode On Web Authentication
C_TEXT($1;$2;$5;$6;$3;$4)
C_TEXT($user)
C_BOOLEAN($0)
$0:=False
$user:=$5
  `Verweigere aus Sicherheitsgründen Namen mit @
If(WithWildcard($user)
    $0:=False
  `Die Methode WithWildcard folgt unten
Else
    QUERY([WebUser];[WebUser]User=$user)
    If(OK=1)
        $0:=Validate Digest Web Password($user;[WebUser]password)
    Else
        $0:=False  `Benutzer existiert nicht
    End if
End if
```

Die Methode ***WithWildcard*** lautet:

```
  `Methode WithWildcard
  `WithWildcard ( String ) -> Boolean
  `WithWildcard ( Name ) -> Enthält ein Jokerzeichen (@)

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
```

Datenbankmethode On Web Close Process

Datenbankmethode On Web Close Process
Dieser Befehl benötigt keine Parameter

Die **Datenbankmethode On Web Close Process** wird vom 4D Web Server immer beim Schließen einer Web Session aufgerufen. Eine Session wird in folgenden Fällen geschlossen:

- Die maximale Anzahl gleichzeitiger Sessions ist erreicht (standardmäßig 100, veränderbar über den Befehl **WEB SET OPTION**), und 4D muss neue anlegen (4D stoppt automatisch den Prozess der ältesten inaktiven Session),
- Die maximale Zeitspanne für Inaktivität der Session ist erreicht (standardmäßig 480 Minuten, veränderbar über den Befehl **WEB SET OPTION**),
- Der Befehl **WEB CLOSE SESSION** wurde aufgerufen.

Wird diese Datenbankmethode aufgerufen, bleibt der Session-Kontext (Variablen und vom Benutzer erzeugte Auswahlen) weiterhin gültig. Sie können also Daten, die zu dieser Session gehören, sichern und später wieder verwenden, insbesondere über die **Datenbankmethode On Web Connection**.

Hinweis: Im Kontext einer 4D Mobile Session (die mehrere Prozesse generieren kann) wird die **Datenbankmethode On Web Close Process** für jeden Web Prozess aufgerufen, der geschlossen wird. So können Sie alle Arten von Daten (Variablen, Auswahl, etc.) speichern, die der 4D Mobile Session Prozess generiert hat.

Ein Beispiel dazu finden Sie im Abschnitt **Web Sessions verwalten**.

Datenbankmethode On Web Connection

$1, $2, $3, $4, $5, $6 -> Datenbankmethode On Web Connection

Parameter	Typ		Beschreibung
$1	Text	⇐	URL
$2	Text	⇐	HTTP Kopfteil + HTTP Hauptteil
$3	Text	⇐	IP Adresse des Web Client (Browser)
$4	Text	⇐	IP Adresse des Server
$5	Text	⇐	Benutzername
$6	Text	⇐	Kennwort

Die **Datenbankmethode On Web Connection** wird in folgenden Fällen aufgerufen:

- Der Web Server empfängt eine Anfrage, die mit der URL 4DCGI beginnt.
- Der Web Server empfängt eine ungültige Anfrage

Weitere Informationen dazu finden Sie im letzten Abschnitt **Datenbankmethode On Web Connection aufrufen**.
Die Anfrage sollte zuvor von der **Datenbankmethode On Web Authentication** – sofern vorhanden – angenommen worden sein und der Web Server muss gestartet sein.
Die **Datenbankmethode On Web Connection** erhält sechs von 4D übergebene Textparameter ($1, $2, $3, $4, $5, $6). das sind folgende Parametertypen:

Parameter	Typ	Beschreibung
$1	Text	URL
$2	Text	HTTP Kopfteil + HTTP Hauptteil (max. 32 Kb)
$3	Text	IP Adresse des Web Client (Browser)
$4	Text	IP Adresse des Server
$5	Text	Benutzername
$6	Text	Kennwort

Sie müssen diese Parameter folgendermaßen deklarieren:

```
` Datenbankmethode On Web Connection

C_TEXT($1;$2;$3;$4;$5;$6)

` Code für die Methode
```

- **URL Extra Daten**
  Der erste Parameter ($1) ist die URL, welche der Benutzer im Adressbereich seines Web Browsers eingegeben hat, ohne die Host-Adresse.
  Nehmen wir als Beispiel eine Intranet Verbindung. Die IP Adresse Ihres 4D Web Server Rechners ist 123.4.567.89. Nachfolgende Tabelle zeigt die Werte von $1 abhängig von der im Web Browser eingegebenen URL:

URL im Bereich Location des Web Browsers	Wert des Parameters $1
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Kunden	/Kunden
http://123.4.567.89/Kunden	/Kunden
http://123.4.567.89/Kunden/Hinzufügen	/Kunden/Hinzufügen
123.4.567.89/Aktion1/Wenn_OK/Aktion2	/Aktion1/Wenn_OK/Aktion2

  Sie können diese Parameter nach Belieben nutzen. 4D ignoriert einfach den Wert, der nach dem Host-Teil der URL steht. Sie können zum Beispiel eine Konvention festlegen, nach der der Wert "/Kunden/Hinzufügen" bedeutet "gehe direkt zu der Aktion neuen Datensatz in der Tabelle [Kunden] hinzufügen". Wenn Sie den Web Benutzern Ihrer Datenbank eine Liste der möglichen Werte und/oder Standard-Bookmarks zur Verfügung stellen, können Sie auch Tastaturkürzel für die verschiedenen Teile Ihrer Anwendung einrichten. So können Web Benutzer schnell auf die Ressourcen im Web zugreifen und müssen nicht bei jedem Verbindungsaufbau erneut den gesamten Navigationspfad durchlaufen.

  Warnung: Um zu verhindern, dass ein Benutzer mit einem bei einer früheren Sitzung erstellten Lesezeichen erneut in die Datenbank gelangt, fängt 4D jede URL ab, die zu den Standard URLs von 4D gehört.

- **Kopfteil und Hauptteil der HTTP Anfrage**
  Der zweite Parameter ($2) ist Kopfteil und Hauptteil der HTTP Anfrage, die der Web Browser sendet. Beachten Sie, dass diese Information komplett in Ihre **Datenbankmethode On Web Connection** übernommen wird. Der Inhalt variiert je nach Art des Web Browsers, der versucht, die Verbindung aufzubauen.

  Mit Safari auf Mac OS könnte der Kopfteil folgendermaßen aussehen:

  ```
  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
```

Mit Microsoft Internet Explorer 8 auf Windows könnte der Kopfteil folgendermaßen aussehen

```
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
```

Enthält Ihre Anwendung diese Information, können Sie selbst wählen, ob Sie den Kopfteil und den Hauptteil durchlaufen. **Hinweis:** Aus Performance-Gründen dürfen diese Daten nicht größer als 32 KB sein. Wenn sie darüber liegen, werden sie vom 4D HTTP Server abgeschnitten.

- **IP Adresse des Web Client**
  Der Parameter $3 empfängt die IP Adresse des Browser Rechners. Mit dieser Information können Sie zwischen Intranet und Internet Verbindungen unterscheiden.
  Hinweis: 4D gibt IPv4 Adressen in einem hybrid IPv6/IPv4 Format mit einem 96-bit Prefix zurück, z.B. ::ffff:192.168.2.34 für die IPv4 Adresse 192.168.2.34. Weitere Informationen dazu finden Sie im Abschnitt **Unterstützung von IPv6**.

- **IP Adresse des Server**
  Der Parameter $4 empfängt die IP Adresse, an welche die HTTP Anfrage gesendet wurde. 4D ermöglicht 4D Multi-homing, d.h. den Einsatz von Rechnern mit mehr als einer IP Adresse. Weitere Informationen dazu finden Sie im Abschnitt **Web Server, Einstellungen**.

- **Benutzername und Kennwort**
  Die Parameter $5 und $6 empfangen Benutzernamen und Kennwort, die der Benutzer im vom Browser angezeigten Dialogfenster Standardidentifikation eingibt. Dieses Dialogfenster erscheint für jede Verbindung, wenn im Dialogfenster Datenbank-Eigenschaften die Option **Benutze Kennwort** ausgewählt wurde. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

  Hinweis: Gibt es den vom Browser gesendeten Benutzernamen in 4D, wird der Parameter $6 (Benutzerkennwort) aus Sicherheitsgründen nicht zurückgegeben.

Datenbankmethode On Web Connection aufrufen

Die **Datenbankmethode On Web Connection** lässt sich als Einstiegspunkt für den 4D Web Server verwenden, entweder über die spezielle URL /4DCGI/... oder über angepasste URL Befehle.

Warnung: Wird ein 4D Befehl aufgerufen, der ein Element der Oberfläche anzeigt (**DIALOG**, **ALERT**...), beendet das den Prozess.

Die **Datenbankmethode On Web Connection** wird in folgenden Fällen aufgerufen:

- Wenn 4D die URL /4DCGI empfängt. Die Datenbankmethode wird in $1 mit der URL /4DCGI/<action> aufgerufen.
- Wenn eine Web Seite aufgerufen wird und keine URL vom Typ <path>/<file> gefunden wird. Die Datenbankmethode wird mit der URL aufgerufen. (*)
- Wenn eine Web Seite mit einer URL vom Typ <file>/ aufgerufen wird und standardmäßig keine Home Page definiert wurde. Die Datenbankmethode wird mit der URL aufgerufen. (*)

(*) In diesen Sonderfällen startet die in $1 empfangene URL NICHT mit dem Zeichen "/".

Datensatz sperren

Überblick zu Datensatz sperren

4D und 4D Server verwalten die Datenbanken automatisch bei Mehrfachbenutzung. Zwei Benutzer bzw. zwei Prozesse können nicht gleichzeitig denselben Datensatz oder dasselbe Objekt ändern. Der zweite Benutzer bzw. Prozess kann zur selben Zeit nur im Lesemodus darauf zugreifen.

Multi-User Befehle bieten sich an, wenn Sie:

- Datensätze per Programmierung ändern.
- Eine eigene Benutzeroberfläche für Operationen im Mehrplatzbetrieb einsetzen.
- Verknüpfte Änderungen innerhalb einer Transaktion sichern.

Setzen Sie eine Datenbank in der Multiprozess-Umgebung ein, müssen Sie folgendes beachten:

- In einem Prozess ist jede Tabelle entweder im Lesemodus oder im Lese-/Schreibmodus.
- Geladene Datensätze werden gesperrt, abgelegte Datensätze werden wieder freigegeben.
- Ein gesperrter Datensatz kann nicht geändert werden.

In den folgenden Abschnitten verwenden wir die Begriffe wie folgt:
Die Person, die eine Operation in der Datenbank im Mehrplatzbetrieb ausführt, wird **lokaler Benutzer** genannt. Die anderen Benutzer der Datenbank werden **andere Benutzer** genannt. Analog dazu wird der Prozess, der in der Multiprozessumgebung eine Operation ausführt, **aktueller Prozess** genannt. Alle anderen Prozesse in Ausführung werden **andere Prozesse** genannt. Die Erläuterungen erfolgen aus der Sicht des lokalen Benutzers und des aktuellen Prozesses.

Gesperrte Datensätze

Ein gesperrter Datensatz kann weder vom lokalen Benutzer noch vom laufenden Prozess geändert werden. Er kann geladen, jedoch nicht geändert werden. Ein Datensatz ist gesperrt, sobald einer der anderen Benutzer oder Prozesse diesen im Lese-/Schreibmodus geladen hat oder der Datensatz gestapelt ist. Nur der Benutzer, der den Datensatz ändert, sieht ihn als nicht gesperrt. Für alle anderen Benutzer und Prozesse erscheint dieser Datensatz als gesperrt. Sie können keine Änderungen vornehmen. Der Datensatz hat nur dann den Status nicht gesperrt, wenn die dazugehörige Tabelle im Lese-/Schreibmodus ist.

Nur-Lesen und Lese-/Schreibmodus

Jede Tabelle in einer Anwendung ist entweder im Lese-/Schreibmodus oder Nur-Lesen für jeden Benutzer und jeden Prozess der Datenbank. **Nur-Lesen** bedeutet, dass die Datensätze einer Tabelle geladen, aber nicht geändert werden können.
Lesen/Schreiben bedeutet, dass Datensätze für die Tabelle geladen und geändert werden können, wenn kein anderer Benutzer den Datensatz zuvor gesperrt hat.
Wenn Sie den Status einer Tabelle ändern, wirkt sich die Änderung auf den nächsten geladenen Datensatz aus. Sie gilt nicht für den aktuell geladenen Datensatz

Status Nur-Lesen

Ist eine Tabelle im Modus Nur-Lesen und wird ein Datensatz geladen, ist der Datensatz immer gesperrt, d.h. er lässt sich anzeigen, drucken oder anderweitig nutzen, aber nicht verändern.

Der Status Nur-Lesen gilt nur für das Bearbeiten vorhandener Datensätze, er hat keine Auswirkung auf das Erstellen neuer Datensätze. In einer Tabelle im Modus Nur-Lesen können Sie Datensätze über die Befehle **CREATE RECORD** und **ADD RECORD** oder über die Menübefehle der Designumgebung hinzufügen. Diese neu erstellten Datensätze sind dann für alle anderen Benutzer bzw. Prozesse gesperrt. Beachten Sie, dass der Nur-Lesen Status nicht für den Befehl **ARRAY TO SELECTION** gilt, da er Datensätze erstellen und ändern kann.

4D setzt eine Tabelle für Befehle, die keinen Schreibzugriff auf Datensätze benötigen, automatisch auf Nur-Lesen. Das sind folgende Befehle: **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**.

Den Status einer Tabelle können Sie jederzeit mit der Funktion **Read only state** herausfinden.

Vor der Ausführung einer dieser Befehle sichert 4D den aktuellen Status der Tabelle (Lesemodus oder Lese-/Schreibmodus) für den laufenden Prozess . Dieser Status wird nach Ausführung des Befehls wiederhergestellt.

Status Lesen/Schreiben

Ist eine Tabelle im Lese-/Schreibmodus und wird ein Datensatz geladen, wird der Datensatz entsperrt, wenn ihn zuvor kein anderer Benutzer gesperrt hat. Ist der Datensatz von einem anderen Benutzer gesperrt, wird er als gesperrt geladen und kann vom lokalen Benutzer nicht verändert werden.

Eine Tabelle muss in den Lese-/Schreibmodus gesetzt werden, damit der geladene Datensatz entsperrt wird und änderbar ist.

Lädt ein Benutzer einen Datensatz von einer Tabelle im Lese-/Schreibmodus, kann kein anderer Benutzer diesen Datensatz zum Ändern laden. Er kann jedoch in der Tabelle Datensätze hinzufügen, entweder über die Befehle **CREATE RECORD** und **ADD RECORD** oder manuell in der Designumgebung.

Alle Tabellen sind standardmäßig im Lese-/Schreibmodus, wenn eine Datenbank geöffnet und ein neuer Prozess gestartet wird.

Status einer Tabelle ändern

Sie können den Status einer Tabelle über die Befehle **READ ONLY** und **READ WRITE** ändern. Wollen Sie den Status einer Tabelle ändern, um einen Datensatz auf Nur-Lesen oder Lesen/Schreiben zu setzen, müssen Sie den Befehl vor Laden des Datensatzes ausführen. Die Befehle **READ ONLY** und **READ WRITE** gelten nicht für den bereits geladenen Datensatz.

Jeder Prozess hat seinen eigenen Status (Nur-Lesen oder Lesen/Schreiben) für jede Tabelle in der Datenbank.

Standardmäßig, also ohne Verwenden von **READ ONLY**, sind alle Tabellen im Modus Lesen/Schreiben.

Datensätze ändern, aktualisieren, freigeben

Bevor ein lokaler Benutzer einen Datensatz ändern kann, muss die Tabelle im Status Lesen/Schreiben sein und der Datensatz geladen und entsperrt sein.

Jeder Befehl, der einen aktuellen Datensatz (sofern vorhanden) lädt, wie **NEXT RECORD**, **QUERY**, **ORDER BY**, **RELATE ONE**, etc., setzt den Datensatz auf gesperrt oder entsperrt. Der Datensatz wird gemäß dem aktuellen Status seiner Tabelle (Nur-Lesen oder Lesen/Schreiben) und seiner Verfügbarkeit geladen. Es kann auch ein Datensatz einer verknüpften Tabelle geladen werden, wenn einer der Befehle eine automatische Verknüpfung auslöst.

Ist eine Tabelle im Nur-Lesen Status für einen Prozess oder einen Benutzer, werden die Datensätze dieser Tabelle im Nur-Lesen Modus geladen, d.h. sie können von diesem Prozess oder Benutzer weder geändert, noch gelöscht werden. Dies wird zum Ansehen oder Finden von Daten empfohlen, da so andere Benutzer bzw. Prozesse weiterhin im Lese-/Schreibmodus auf die Datensätze zugreifen können.

Ist eine Tabelle im Lesen/Schreiben Status für einen Prozess oder einen Benutzer, wird jeder Datensatz dieser Tabelle im Modus Lesen/Schreiben geladen, wenn ihn nicht bereits ein anderer Benutzer bzw. Prozess gesperrt hat. Wurde ein Datensatz erfolgreich im Lesen/Schreiben Status geladen, wird er für den aktuellen Prozess bzw. Benutzer entsperrt, d.h. er lässt sich ändern und sichern. Für alle anderen Benutzer oder Prozesse ist er gesperrt.

Eine Tabelle muss in den Lesen/Schreiben Status gesetzt werden, bevor ein Datensatz zum Ändern geladen und dann gesichert wird.

- Greifen Sie auf die Tabelle mit dem Befehl **READ WRITE** zu.
- Rufen Sie den gewünschten Befehl auf, der den Datensatz zum ersten Mal lädt, beispielsweise **NEXT RECORD**, **QUERY**, **ORDER BY**, **RELATE ONE**.
- Prüfen Sie mit der Funktion **Locked** den Zustand des Datensatzes.
- Ist der Datensatz gesperrt, rufen Sie mit den Befehl **LOAD RECORD** den Datensatz so lange auf, bis **Locked** den Wert FALSE zurückgibt.
- Ändern Sie den Datensatz.
- Sichern Sie ihn und geben ihn dann frei, indem Sie den aktuellen Datensatz wechseln oder mit dem Befehl **UNLOAD RECORD** freigeben.

Hinweis: Wird **UNLOAD RECORD** in einer Transaktion verwendet, entsperrt er den aktuellen Datensatz nur für den Prozess, der diese Transaktion verwaltet. Für andere Prozesse bleibt der Datensatz gesperrt, bis die Transaktion bestätigt oder annulliert wurde.

Verwenden Sie den Befehl **LOCKED BY**, um feststellen, welcher Benutzer bzw. Prozess einen Datensatz gesperrt hat.

Hinweis: Eine gute Praxis ist, beim Starten eines Prozesses alle Tabellen in den Modus Nur-Lesen zu setzen (mit der Syntax **READ ONLY(*)**) und jede Tabelle nur bei Bedarf in den Modus Lesen/Schreiben zu setzen. Der Zugriff auf Tabellen im Nur-Lesen Modus ist schneller und effizienter für den Speicher. Außerdem ist der Status einer Tabelle im Client/Server Modus optimiert, da er keinen zusätzlichen Netzwerkverkehr verursacht: Information wird nur beim Ausführen eines Befehls an den Server gesendet, der entsprechenden Zugriff auf die Tabelle benötigt.

Schleifen zum Laden freigegebener Datensätze

Folgendes Beispiel zeigt die einfachste Schleife zum Laden eines nicht-gesperrten Datensatzes:

```
READ WRITE([Customers]) ` Setze Tabelle auf Lese-/Schreibmodus
Repeat ` Durchlaufe Schleife, bis Datensatz freigegeben ist
   LOAD RECORD([Customers]) ` Lade Datensatz und setze Status gesperrt
Until(Not(Locked([Customers])))
 ` Bearbeite diesen Datensatz
READ ONLY([Customers]) ` Setze Tabelle auf Nur Lesen
```

Die Schleife läuft solange, bis der Datensatz freigegeben ist.

Diese Schleife wird nur verwendet, wenn Sie davon ausgehen können, dass der Datensatz nicht durch jemand anderen gesperrt ist. Denn der Benutzer muss das Beenden der Schleife abwarten. Der Fall ist relativ unwahrscheinlich, da der Datensatz nur über eine Methode geändert werden kann.

Folgendes Beispiel verwendet die obige Schleife, um einen ungesperrten Datensatz zu laden und anschließend zu ändern:

```
READ WRITE([Inventory])
Repeat ` Durchlaufe Schleife, bis Datensatz freigegeben ist
   LOAD RECORD([Inventory]) ` Lade Datensatz und setze Status gesperrt
Until(Not(Locked([Inventory])))
[Inventory]Part Qty:=[Inventory]Part Qty 1 ` Ändere Datensatz
SAVE RECORD([Inventory]) ` Sichere Datensatz
UNLOAD RECORD([Inventory]) ` Lass andere Benutzer den Datensatz ändern
READ ONLY([Inventory])
```

Der Befehl **MODIFY RECORD** zeigt dem Benutzer automatisch an, ob der Datensatz gesperrt ist und unterbindet so eine Änderung. Folgendes Beispiel umgeht diese automatische Anzeige durch den Test mit der Funktion **Locked**. Ist der Datensatz gesperrt, kann der Benutzer abbrechen.

Dieses Beispiel prüft in optimaler Weise, ob der aktuelle Datensatz für die Tabelle [Commands] gesperrt ist. Wenn ja, wird der Prozess von der Methode um eine Sekunde verschoben. Diese Technik können Sie sowohl in der Multi-User- als auch in der Multiprozessumgebung einsetzen:

```
Repeat
   READ ONLY([Commands]) ` Sie benötigen Lese-/Schreibmodus gerade nicht
   QUERY([Commands])
```

```
  ` Wurde die Suche abgeschlossen und einige Datensätze zurückgegeben
    If((OK=1) & (Records in selection([Commands])>0))
       READ WRITE([Commands]) ` Setze Tabelle auf Lese-/Schreibmodus
       LOAD RECORD([Commands])
       While(Locked([Commands]) & (OK=1)) `Ist der Datensatz gesperrt, durchlaufe Schleife, bis Datensatz freigegeben ist. Wer sperrt
den Datensatz?
          LOCKED BY([Commands];$Process;$User;$SessionUser;$Name)
          If($Process=-1) ` Wurde der Datensatz gelöscht?
             ALERT("Datensatz wurde inzwischen gelöscht.")
             OK:=0
          Else
             If($User="") ` Sind Sie im Einzelplatzbetrieb
                $User:="Ihnen"
             End if
             CONFIRM("Datensatz wird bereits von "+$User+" im Prozess "+$Name+" verwendet.")
             If(OK=1) ` Wollen Sie ein paar Sekunden warten?
                DELAY PROCESS(Current process;120)
  ` Warten Sie ein paar Sekunden
                LOAD RECORD([Commands]) ` Versuchen Sie, Datensatz zu laden.
             End if
          End if
       End while
       If(OK=1) ` Datensatz ist freigegeben
          MODIFY RECORD([Commands]) ` Sie können Datensatz ändern
          UNLOAD RECORD([Commands])
       End if
       READ ONLY([Commands]) ` Wechsle wieder in Modus Nur Lesen
       OK:=1
    End if
 Until(OK=0)
```

Befehle im Mehrplatzbetrieb bzw. in der Multiprozessumgebung

Ein Reihe von Befehlen der Programmiersprache arbeiten bei nicht-gesperrten Datensätzen normal. Treffen sie auf einen gesperrten Datensatz, führen sie ganz bestimmte Aktionen aus. Es handelt sich um folgende Befehle:

- **MODIFY RECORD**: Meldet, dass der Datensatz bereits benutzt wird. Der Datensatz wird nicht angezeigt, der Benutzer kann ihn also nicht ändern. In der Benutzerumgebung erscheint der Datensatz im Modus Nur Lesen.
- **MODIFY SELECTION**: Arbeitet normal, durch Doppelklick kann der Benutzer den Datensatz ändern. Ist er gesperrt, meldet **MODIFY SELECTION**, dass der Datensatz bereits benutzt wird. Der Datensatz ist dann nur im Modus Nur Lesen verfügbar
- **APPLY TO SELECTION**: Lädt einen gesperrten Datensatz, ändert ihn jedoch nicht. Mit **APPLY TO SELECTION** können Sie so Informationen ohne Risiko aus Tabellen auslesen. Gesperrte Datensätze werden in der Menge *LockedSet* abgelegt.
- **DELETE SELECTION**: Löscht keine gesperrten Datensätze; sie werden übersprungen. Gesperrte Datensätze werden in der Menge *LockedSet* abgelegt.
- **DELETE RECORD**: Wird bei gesperrtem Datensatz nicht ausgeführt. Es erscheint keine Fehlermeldung. Sie müssen vor Ausführung dieses Befehls prüfen, ob der Datensatz freigegeben ist.
- **SAVE RECORD**: Wird bei gesperrtem Datensatz nicht ausgeführt. Es erscheint keine Fehlermeldung. Sie müssen vor Ausführung dieses Befehls prüfen, ob der Datensatz freigegeben ist.
- **ARRAY TO SELECTION**: Sichert keine gesperrten Datensätze. Gesperrte Datensätze werden in der Menge *LockedSet* abgelegt.
- **GOTO RECORD**: In einer Datenbank in der Multi-User-/Multiprozess-Umgebung können andere Benutzer Datensätze löschen bzw. hinzufügen. So können sich auch die Datensatznummern verändern. Seien Sie also vorsichtig, wenn Sie im Mehrplatzbetrieb auf einen Datensatz über seine Nummer zugreifen.
- **Mengen:** Behandeln Sie Mengen mit Vorsicht, da andere Benutzer oder Prozesse die darin enthaltenen Informationen bereits geändert haben können. Weitere Informationen dazu finden Sie im Abschnitt **Einführung in Mengen**

Get locked records info

Get locked records info (Tabellenname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren gesperrte Datensätze angezeigt werden sollen
Funktionsergebnis	Objekt	↩	Beschreibung der gesperrten Datensätze(sofern vorhanden)

Beschreibung

Der Befehl **Get locked records info** gibt ein *Objekt* zurück mit verschiedenen Informationen über die aktuell gesperrten Datensätze in *Tabellenname*.

Das zurückgegebene Objekt enthält die Eigenschaft "records" als Array mit Objekten:

```
{
    "records": [
        description object,
        (…)
    ]
}
```

Jedes Array Element "description object" beschreibt einen gesperrten Datensatz in der angegebenen Tabelle und hat folgende Eigenschaften:

Eigenschaft	Typ	Beschreibung
KontextID	UUID (String)	UUID des Datenbankkontextes für die Sperrung
KontextAttribut	Objekt	Objekt mit derselben Information wie der Befehl **LOCKED BY**, angewandt auf den Datensatz (siehe unten)
DatensatzNummer	Lange Ganzzahl	Datensatznummer des gesperrten Datensatzes

Das Objekt *KontextAttribut* hat folgende Eigenschaften:

Eigenschaft	Typ	Beschreibung
task_id	Zahl[#tab/Refererenznummer des Prozesses	
user_name	String	Benutzername aus dem 4D Kennwortsystem
user4d_id	Zahl	Benutzernummer (*)
host_name	String	Name des Host Rechners
task_name	String	Prozessname
client_version	Zahl	Version der Client Applikation
Nur wenn der Befehl auf 4D Server ausgeführt wird und die Datensatzsperrung von einem remote 4D kommt:		
is_remote_context	Boolean	Gibt an, ob die Sperrung von einem remote 4D ausgeht (*ist immer wahr*, andernfalls gibt es keine)
client_uid	UUID (String)	UUID des remote 4D, von dem die Sperrung ausgeht

(*) Sie erhalten den 4D Benutzernamen über den Wert *user4d_id* mit folgendem Code:

```
GET USER LIST($arrNames;$arrIDs)
$User4DName:=Find in array($arrIDs;user4d_id)
```

Hinweis: Der Befehl funktioniert nur mit 4D und 4D Server. Wird er über remote 4D oder eine Komponente aufgerufen, gibt er immer ein ungültiges Objekt zurück, außer die Option "Auf Server ausführen" ist aktiviert. Dann enthält das zurückgegebene Objekt jeweils Informationen über den Server bzw. die Host Datenbank.

Beispiel

Sie führen folgenden Code aus:

```
$vOlocked :=Get locked records info([Table])
```

Sind zwei Datensätze in der Tabelle [Table] gesperrt, wird in *$vOlocked* folgendes Objekt zurückgegeben:

```
{
    "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
        }
    ]
}
```


Wird der Code auf einem 4D Server ausgeführt und die Sperrung von einem remote Client Rechnern ausgelöst, wird in $vOlocked folgendes Objekt zurückgegeben:


```
{
    "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 {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Datensatz geladen werden soll Ohne Angabe Haupttabelle

Beschreibung

Der Befehl **LOAD RECORD** lädt den Datensatz der Tabelle *Tabellenname* in den Hauptspeicher. *Tabellenname* ist optional. Wird dieser Parameter nicht angegeben, bezieht sich **LOAD RECORD** auf die Haupttabelle.
Gibt es keinen aktuellen Datensatz, hat **LOAD RECORD** keine Auswirkung.

Mit der Funktion **Locked** können Sie den Zustand des aktuellen Datensatzes abfragen:

- Wurde der Datensatz im Nur Lesen-Modus geladen, gibt **Locked** den Wert TRUE zurück. Sie können den Datensatz nicht ändern.
- Wurde der Datensatz im Lese-/Schreibmodus geladen, wird aber schon von einem anderen Prozess bearbeitet, gibt **Locked** ebenfalls den Wert TRUE zurück. Sie können den Datensatz nicht ändern.
- Wurde der Datensatz im Lese-/Schreibmodus geladen und er wird von keinem anderen Prozess bearbeitet, gibt **Locked** den Wert FALSE zurück. Sie können den Datensatz im laufenden Prozess ändern.

Hinweis: Wurde dieser Befehl nach **READ ONLY** aufgerufen, wird er automatisch entladen bzw. geladen. Sie müssen hierfür nicht den Befehl **UNLOAD RECORD** aufrufen.

Normalerweise müssen Sie den Befehl **LOAD RECORD** nicht einsetzen, da Befehle wie **QUERY**, **NEXT RECORD**, **PREVIOUS RECORD**, etc., automatisch den aktuellen Datensatz laden.

Wollen Sie in der Multi-User- und Multiprozessumgebung einen vorhandenen Datensatz ändern, muss die entsprechende Tabelle im Lese-/Schreibmodus sein. Ist ein Datensatz dieser Tabelle gesperrt und wird nicht geladen, können Sie mit **LOAD RECORD** versuchen, ihn später erneut zu laden. Verwenden Sie **LOAD RECORD** in einer Schleife, können Sie warten, bis der Datensatz freigegeben wird.

Tipp: Über den Befehl **LOAD RECORD** können Sie den aktuellen Datensatz im Rahmen eines Eingabeformulars erneut laden. Alle geänderten Daten werden dann durch ihre vorigen Werte ersetzt. In diesem Fall führt **LOAD RECORD** eine allgemeine Aufhebung der Dateneingabe aus.

Locked

Locked {(Tabellenname)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle, deren Datensatz auf gesperrt geprüft wird Ohne Angabe Haupttabelle
Funktionsergebnis	Boolean	⇐	Datensatz ist gesperrt (TRUE), oder Datensatz ist freigegeben (FALSE)

Beschreibung

Die Funktion **Locked** überprüft, ob der aktuelle Datensatz der Tabelle *Tabellenname* gesichert werden kann.

Gibt **Locked** den Wert TRUE zurück, lässt sich der Datensatz nicht sichern, da er durch einen anderen Prozess bzw. Benutzer gesperrt oder im aktuellen Prozess gestapelt ist. In diesem Fall laden Sie mit **LOAD RECORD** den Datensatz erneut, bis **Locked** den Wert FALSE zurückgibt.

Gibt **Locked** den Wert FALSE zurück, ist der Datensatz freigegeben, d.h. er ist für alle anderen Benutzer gesperrt. Nur der lokale Benutzer bzw. der laufende Prozess kann den Datensatz ändern und sichern. Das geht jedoch nur, wenn auch die entsprechende Tabelle im Lese-/Schreibmodus ist.

Versuchen Sie einen Datensatz zu laden, der in der Zwischenzeit gelöscht wurde, gibt **Locked** weiter TRUE zurück. Mit dem Befehl **LOCKED BY** vermeiden Sie unnötiges Warten auf einen gelöschten Datensatz. Der Befehl **LOCKED BY** gibt in diesem Fall in *Prozessnr* -1 zurück.
LOAD RECORD und **Locked** werden oft während Transaktionen eingesetzt, um die Verfügbarkeit von Datensätzen zu prüfen. Ist ein Datensatz gesperrt, wird die Transaktion im allgemeinen abgebrochen.

LOCKED BY

LOCKED BY ({Tabellenname ;} ProzessNr ; 4DBenutzer ; Sitzungsbenutzer ; ProzessName)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle für Prüfung gesperrter Datensätze Ohne Angabe Haupttabelle
ProzessNr	Lange Ganzzahl	⬅	Referenznummer des Prozesses
4DBenutzer	String	⬅	Name des 4D Anwenders
Sitzungsbenutzer	String	⬅	Name des Benutzers, der die Sitzung geöffnet hat
ProzessName	String	⬅	Name des Prozesses

Beschreibung

Der Befehl **LOCKED BY** gibt an, welcher Anwender und welcher Prozess einen Datensatz gesperrt haben. Die Prozessnummer (*), der Benutzername in der 4D Anwendung und im System sowie der Prozessname werden in den Parametern *ProzessNr*, *4DBenutzer*, *Sitzungsbenutzer* und *ProzessName* zurückgegeben. Sie können diese Angaben in ein eigenes Dialogfenster setzen, um den Benutzer zu warnen, wenn ein Datensatz gesperrt ist.

(*) Dies ist die Nummer des Prozesses, der aktuell den Datensatz gesperrt hat. Bei einem Trigger oder einer Methode, die auf dem Server ausgeführt werden, wird die Nummer des "twin" Prozesses auf dem Serverrechner zurückgegeben. Bei einer Methode, die in einer remote Applikation ausgeführt wird, wird die Nummer des Prozesses auf dem remote Rechner zurückgegeben.

Ist der Datensatz nicht gesperrt, gibt *ProzessNr* den Wert 0 (Null) zurück, die Parameter *4DBenutzer*, *Sitzungsbenutzer* und *ProzessName* geben leere Strings zurück. Wurde ein Datensatz gelöscht, gibt *ProzessNr* den Wert -1 zurück, die Parameter *4DBenutzer*, *Sitzungsbenutzer* und *ProzessName* geben ebenfalls leere Strings zurück.

In *4DBenutzer* erhalten Sie den Benutzernamen, wie er im Kennwortsystem von 4D angegeben ist, selbst wenn kein Benutzername eingetragen ist. Gibt es kein Kennwortsystem, wird “Designer” zurückgegeben.

In *Sitzungsbenutzer* erhalten Sie den Namen des Benutzers, der die Sitzung auf dem Client-Rechner geöffnet hat. Dieser Name erscheint insbesondere im 4D Server Verwaltungsfenster für jeden geöffneten Prozess.

READ ONLY

READ ONLY {(Tabellenname | *)}

Parameter	Typ		Beschreibung
Tabellenname | *	Tabelle, Operator	➡	Tabelle, deren Zugang geändert werden soll * für alle Tabellen Ohne Angabe Haupttabelle

Beschreibung

Der Befehl **READ ONLY** ändert den Status von *Tabellenname* für den Prozess, in der die Tabelle aufgerufen wird, in den Lesemodus. Alle nachfolgend geladenen Datensätze werden gesperrt und Sie können keine Änderungen ausführen. Wird der Parameter * angegeben, wechseln alle Tabellen in den Nur-Lesen Modus.

Verwenden Sie **READ ONLY** für Datensätze, die nicht geändert werden müssen.

Hinweis: READ ONLY hat keinen Einfluss auf vorher geladene Datensätze. Ein Datensatz wird gemäß dem Status der Tabelle zum Zeitpunkt des Ladens geladen. Um einen Datensatz aus einer Tabelle im Lese-/Schreibmodus im Nur-Lesen Modus zu laden, müssen Sie zuerst den Status der Tabelle auf Nur-Lesen setzen.

Read only state

Read only state {(Tabellenname)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle, die auf den Zugriffsmodus geprüft werden soll oder Haupttabelle ohne Angabe
Funktionsergebnis	Boolean	⇐	Tabelle ist im Lesemodus (TRUE) oder Tabelle ist im Lese-/Schreibmodus (FALSE)

Beschreibung

Die Funktion **Read only state** testet, ob sich eine Tabelle im laufenden Prozess im Lese- oder Lese-/Schreibmodus befindet. Ist die Tabelle im Lesemodus, gibt die Funktion den Wert TRUE zurück, ist sie im Lese-/Schreibmodus, den Wert FALSE.

Beispiel

Folgendes Beispiel testet den Status der Tabelle [Invoices]. Ist die Tabelle im Lesemodus, wird sie in den Lese-/Schreibmodus gesetzt und der aktuelle Datensatz noch einmal geladen.

```
If(Read only state([Invoices]))
    READ WRITE([Invoices])
    LOAD RECORD([Invoices])
End if
```

Hinweis: Der Datensatz wird noch einmal geladen, damit der Benutzer ihn auch ändern kann. Wurde ein Datensatz im Lesemodus geladen, bleibt er gesperrt, bis er erneut im Lese-/Schreibmodus geladen wird.

READ WRITE

READ WRITE {(Tabellenname | *)}

Parameter	Typ		Beschreibung
Tabellenname | *	Tabelle, Operator	➡	Tabelle, deren Zugang geändert werden soll oder * für alle Tabellen oder ohne Angabe Haupttabelle

Beschreibung

Der Befehl **READ WRITE** ändert den Status von *Tabellenname* im Prozess, der die Tabelle aufruft, in den Lese-/Schreibmodus. Wird der Parameter * angegeben, gilt der Befehl für alle Tabellen.

Ein nach Aufrufen von **READ WRITE** geladener Datensatz wird entsperrt, wenn er nicht durch einen anderen Benutzer gesperrt ist. Dieser Befehl ändert nicht den Status des aktuell geladenen Datensatzes, sondern nur der nachfolgend geladenen.

Beim Öffnen einer 4D Anwendung sind alle Tabellen standardmäßig im Lese-/Schreibmodus.

Verwenden Sie **READ WRITE**, wenn Sie einen Datensatz ändern oder löschen bzw. den Zugang für andere Benutzer sperren wollen, auch wenn Sie keine Änderungen vornehmen. Eine in den Lese-/Schreibmodus gesetzte Tabelle sorgt dafür, dass andere Benutzer diese Tabelle nicht bearbeiten können. Sie können jedoch neue Datensätze anlegen.

Hinweis: READ ONLY hat keinen Einfluss auf vorher geladene Datensätze. Ein Datensatz wird gemäß dem Status der Tabelle zum Zeitpunkt des Ladens geladen. Um einen Datensatz aus einer Tabelle im Lese-/Schreibmodus im Nur-Lesen Modus zu laden, müssen Sie zuerst den Status der Tabelle auf Nur-Lesen setzen.

UNLOAD RECORD

UNLOAD RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Datensatz freigegeben werden soll Ohne Angabe Haupttabelle

Beschreibung

UNLOAD RECORD gibt den aktuellen Datensatz der Tabelle *Tabellenname* wieder für andere Prozesse frei. Wurde der Datensatz im Lese-/Schreibmodus gesperrt geladen, hebt **UNLOAD RECORD** die Sperre auf und gibt zusätzlich den benötigten Platz im Hauptspeicher wieder frei.
Wurde der Datensatz geladen, jedoch nicht gesperrt, gibt **UNLOAD RECORD** nur den benötigten Platz im Hauptspeicher wieder frei.

Tabellenname ist optional, wird dieser Parameter nicht angegeben, bezieht sich **UNLOAD RECORD** auf die Haupttabelle.

Legen Sie einen neuen Datensatz an, ist er für diesen Prozess immer im Lese-/Schreibmodus, auch wenn Sie im Lesemodus auf die Tabelle zugegriffen haben. Wollen Sie ihn für andere Prozesse freigeben, müssen Sie einen anderen aktuellen Datensatz aufrufen oder ihn mit dem Befehl **UNLOAD RECORD** freigeben.

4D gibt automatisch den aktuellen Datensatz frei, sobald Sie ihn wechseln, d.h. einen anderen aktuellen Datensatz auswählen. Mit **UNLOAD RECORD** können Sie den aktuellen Datensatz freigeben, ohne den Datensatz zu wechseln. Das spart bei Datensätzen mit großen Datenmengen, Bildfeldern oder Plug-Ins (z.B. 4D Write oder 4D Draw Dokumente) Zeit. Denn Sie können den aktuellen Datensatz beibehalten, ohne dass er Speicherplatz im Hauptspeicher belegt. Sie haben dann jedoch keinen Zugriff auf seine Feldinhalte. Dazu müssen Sie den Datensatz mit **LOAD RECORD** erneut laden. Sonst erhalten Sie leere Werte.

Datensätze

- Datensatznummern
- Stapel für Datensätze einsetzen
- 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

Datensatznummern

Jedem Datensatz sind drei Nummern zugeordnet:

- Die Nummer des Datensatzes
- Die Nummer des ausgewählten Datensatzes
- Die Sequenznummer

Nummer des Datensatzes

Die Datensatznummer ist die absolute Nummer für einen Datensatz. Jeder neue Datensatz erhält automatisch eine Datensatznummer, die konstant bleibt, bis dieser Datensatz gelöscht wird. Datensatznummern beginnen bei Null. Sie sind nicht einmalig, da die Nummern gelöschter Datensätze wieder für neue Datensätze verwendet werden. Die Nummern ändern sich auch, wenn die Datenbank komprimiert oder repariert wird.

Nummer des ausgewählten Datensatzes

Die Nummer des ausgewählten Datensatzes ist die Position des Datensatzes in der aktuellen Auswahl. Wird die aktuelle Auswahl geändert oder neu sortiert, ändert sich meist auch diese Datensatznummer. Die Numerierung für ausgewählte Datensätze beginnt bei Eins (1).

Sequenznummer

Die Sequenznummer ist eine einmalige, nicht wiederverwendbare Nummer für ein Datenfeld eines Datensatzes. Sie lässt sich über die Eigenschaft **Autoincrement** im Inspektor, das SQL Attribut AUTO_INCREMENT oder die Funktion **Sequence number** zuweisen. Sie wird nicht automatisch mit jedem Datensatz gespeichert. Sie beginnt standardmäßig bei Eins (1) und erhöht sich für jeden neuen Datensatz um Eins. Im Gegensatz zu Datensatznummern wird eine Sequenznummer nicht wieder verwendet, wenn ein Datensatz gelöscht bzw. eine Datenbank komprimiert oder repariert wird. Über Sequenznummern erhalten Sie einmalige Identifikationsnummern für Datensätze. Eine während einer Transaktion erhöhte Sequenznummer wird nach Abbrechen der Transaktion nicht mehr zurückgestellt.

Hinweis: 4D führt keine Überprüfung aus, wenn Sie den internen automatischen Zähler einer Tabelle über den Befehl **SET DATABASE PARAMETER** ändern. Setzen Sie den Zähler herab, können neu angelegte Datensätze Nummern haben, die bereits zugewiesen wurden.

Beispiele für Datensatznummern

Folgende Tabellen zeigen das Verhalten der verschiedenen Nummern für die Datensätze. Jede Zeile der Tabelle bezieht sich auf einen Datensatz. Die Anordnung der Zeilen zeigen die jeweilige Reihenfolge, in der die Datensätze in einem Ausgabeformular angezeigt würden.

- **Spalte Daten**: Daten aus einen Feld pro Datensatz. In unserem Beispiel, der Vorname.
- **Spalte Datensatznummer**: Die absolute Nummer des Datensatzes. Diese Nummer wird von der Funktion **Record number** zurückgegeben.
- **Spalte Nummer des ausgewählten Datensatzes**: Die Position des Datensatzes in der aktuellen Auswahl. Diese Nummer wird von der Funktion **Selected record number** zurückgegeben.
- **Spalte Sequenznummer**: Die einmalige Sequenznummer des Datensatzes. Diese Nummer wird von der Funktion **Record number** beim Erstellen des Datensatzes zurückgegeben. Sie wird im Datensatz abgespeichert.

Nach Eingeben der Datensätze

Die erste Tabelle zeigt die Datensätze nach der Eingabe.

- Die Datensätze sind standardmäßig nach der Datensatznummer aufgelistet.
- Die Datensatznummer beginnt bei 0.
- Die Nummer des ausgewählten Datensatzes und der Sequenznummer beginnen bei 1.

Daten	Datensatznummer	Nummer des ausgewählten Datensatzes	Sequenznummer
Ulli	0	1	1
Tom	1	2	2
Sabrina	2	3	3
Sepp	3	4	4
Lisa	4	5	5

Hinweis: Die Datensätze behalten die Standardreihenfolge bei, wenn ein Befehl die aktuelle Auswahl ohne Neusortierung ändert; zum Beispiel, nach dem Menübefehl **Alle Datensätze zeigen** in der Designumgebung oder nach Ausführen des Befehls **ALL RECORDS**.

Nach Sortieren der Datensätze

Die nächste Tabelle zeigt dieselben Datensätze nach Namen sortiert.

- Die jedem Datensatz zugewiesene Nummer bleibt erhalten.
- Die Nummer des ausgewählten Datensatzes spiegelt die neue Position der Datensätze in der sortierten Auswahl wieder.
- Die Sequenznummern bleiben konstant, wenn sie einmal zugewiesen und mit dem Datensatz abgespeichert wurden.

Daten	Datensatznummer	Nummer des ausgewählten Datensatzes	Sequenznummer
Lisa	4	1	5
Sabrina	2	2	3
Sepp	3	3	4
Tom	1	4	2
Ulli	0	5	1

Nach Löschen eines Datensatzes

Folgende Tabelle zeigt die Datensätze, nachdem Sepp gelöscht wurde.

- Nur die Nummern der ausgewählten Datensätze haben sich geändert. Sie geben die Reihenfolge wieder, in der die Datensätze angezeigt werden.

Daten	Datensatznummer	Nummer des ausgewählten Datensatzes	Sequenznummer
Lisa	4	1	5
Sabrina	2	2	3
Tom	1	3	2
Ulli	0	4	1

Nach Hinzufügen eines Datensatzes

Die nächste Tabelle zeigt die Datensätze, nachdem ein neuer Datensatz für Leo hinzugefügt wurde.

- Der neue Datensatz wird am Ende der aktuellen Auswahl angefügt.
- Der neue Datensatz erhält die Nummer des gelöschten Datensatzes für Sepp.
- Die Sequenznummer erhöht sich um Eins.

Daten	Datensatznummer	Nummer des ausgewählten Datensatzes	Sequenznummer
Ulli	0	1	1
Tom	1	2	2
Sabrina	2	3	3
Lisa	4	4	5
Leo	3	5	6

Nach Ändern und Sortieren der Auswahl

Folgende Tabelle zeigt die Datensätze, nachdem die Auswahl auf drei Datensätze reduziert und sortiert wurde.

- Nur die Nummern der ausgewählten Datensätze haben sich geändert.

Daten	Datensatznummer	Nummer des ausgewählten Datensatzes	Sequenznummer
Sabrina	2	1	3
Leo	3	2	6
Tom	1	3	2

Stapel für Datensätze einsetzen

Mit den Befehlen **PUSH RECORD** und **POP RECORD** können Sie Datensätze in den Datensatzstapel legen (“push”) bzw. aus dem Datensatzstapel entfernen (“pop”).

Jeder Prozess hat für jede Tabelle einen eigenen Datensatzstapel. Die Stapel werden nach der sogenannten LIFO-Methode (Last In First Out) verwaltet. Ihre Kapazität wird durch den verfügbaren Arbeitsspeicher begrenzt.

PUSH RECORD und **POP RECORD** sollten Sie mit Bedacht benutzen. Jeder in den Stapel gelegte Datensatz beansprucht Speicherplatz. Zu viele hineingelegte Datensätzen können zu einer Speicherüberlastung oder einem vollen Stapel führen.
Mit diesen beiden Befehlen können Sie bei der Dateneingabe überprüfen, ob dieser Datensatz bereits vorhanden ist. Geben Sie dazu den aktuellen Datensatz auf den Stapel, überprüfen Sie die Datensätze in der Datei mit Such- oder Sortierbefehlen, holen dann den Datensatz wieder zurück und fahren mit der Eingabe fort.

4D leert den Datensatzstapel jeder Tabelle automatisch für den Hauptprozess, wenn Sie nach Ausführung der Methode in das Startfenster zurückkehren. Sie sollten jedoch jeden Datensatz, den Sie auf den Stapel legen wieder vom Stapel nehmen, auch wenn Sie den Datensatz nicht benötigen.

Müssen Sie beim Eingeben eines Datensatzes einen Wert auf Eindeutigkeit prüfen, verwenden Sie den Befehl **SET QUERY DESTINATION**. Dadurch müssen Sie nicht vor und nach Aufrufen von **QUERY** die Befehle **PUSH RECORD** und **POP RECORD** aufrufen, damit die im aktuellen Datensatz eingegebenen Daten erhalten bleiben. **SET QUERY DESTINATION** führt eine Suche aus, die weder die Auswahl noch den aktuellen Datensatz ändert.

CREATE RECORD

CREATE RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, für die ein Datensatz angelegt werden soll Ohne Angabe Haupttabelle

Beschreibung

Der Befehl **CREATE RECORD** legt einen neuen leeren Datensatz der Tabelle *Tabellenname* im Arbeitsspeicher an. Er erscheint jedoch nicht auf dem Bildschirm. Verwenden Sie **ADD RECORD**, um einen neuen Datensatz anzulegen und zur Eingabe anzuzeigen. *Tabellenname* ist optional. Wird kein Parameter angegeben, bezieht sich **CREATE RECORD** auf die Haupttabelle.

Sie verwenden **CREATE RECORD** anstelle von **ADD RECORD**, wenn der Datensatz per Programmierung mit Daten gefüllt wird. Der neue Datensatz wird der aktuelle Datensatz, die aktuelle Auswahl bleibt jedoch unverändert.

Dieser Datensatz existiert nur im Arbeitsspeicher, bis der Befehl **SAVE RECORD** für die Tabelle ausgeführt wird. Wird der aktuelle Datensatz vor dem Sichern geändert, z.B. durch eine Suche, geht der aktuelle Datensatz verloren.

Hinweis: Für diesen Befehl muss *Tabellenname* nicht im Lese-/Schreibmodus sein. Er lässt sich auch verwenden, wenn die Tabelle im Nur-Lesen Modus ist (siehe **Überblick zu Datensatz sperren**).

Beispiel

Folgendes Beispiel archiviert Datensätze, die älter als 30 Tage sind. Dazu werden in einer Archiv-Tabelle neue Datensätze erzeugt. Diese Datensätze werden anschließend in der Tabelle [Accounts] gelöscht. Der Code sieht folgendermaßen aus:

```
  ` Finde Datensätze älter als 30 Tage
QUERY([Accounts];[Accounts]Entered<(Current date 30))
For($vlRecord;1;Records in selection([Accounts])) ` Durchlaufe einmal pro Datensatz
   CREATE RECORD([Archive]) ` Erstelle neuen Archiv-Datensatz
   [Archive]Number:=[Account]Number ` Kopiere Felder in Archiv-Datensatz
   [Archive]Entered:=[Account]Entered
   [Archive]Amount:=[Account]Amount
   SAVE RECORD([Archive]) ` Sichere Archiv-Datensatz
   NEXT RECORD([Accounts]) ` Gehe in Accounts zum nächsten Datensatz
End for
DELETE SELECTION([Accounts]) ` Lösche Datensätze in Accounts
```

DELETE RECORD

DELETE RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, in der ein Datensatz gelöscht werden soll Ohne Angabe Haupttabelle

Beschreibung

Der Befehl **DELETE RECORD** löscht den aktuellen Datensatz der Tabelle *Tabellenname* im Prozess. Gibt es keinen aktuellen Datensatz für *Tabellenname* im Prozess, hat **DELETE RECORD** keine Auswirkung.
In einem Formular können Sie anstelle dieses Befehls eine Schaltfläche "Datensatz löschen" einrichten.

Hinweise:

- Wird der aktuelle Datensatz vor Aufrufen des Befehls **DELETE RECORD** aus dem Speicher geladen, z.B. nach Ausführen des Befehls **UNLOAD RECORD**, ist die aktuelle Auswahl von *Tabellenname* nach dem Löschen leer.
- Der Befehl **DELETE RECORD** führt nichts aus, wenn die Tabelle im Modus **READ ONLY** ist, unabhängig davon, ob der zu löschende Datensatz gesperrt ist oder nicht.

Das Löschen von Datensätzen ist endgültig und kann nicht widerrufen werden.

Tabellenname ist optional, wird der Parameter nicht angegeben, bezieht sich **DELETE RECORD** auf die Haupttabelle.

Wird ein Datensatz gelöscht, wird die Datensatznummer beim Anlegen neuer Datensätze wiederverwendet. Verwenden Sie die Datensatznummer nicht zum Identifizieren von Datensätzen, falls Sie Datensätze aus der Datenbank löschen.

Beispiel

Folgendes Beispiel löscht einen Datensatz Mitarbeiter. Der Code fragt den Benutzer, welchen Mitarbeiter er löschen will, sucht den entsprechenden Datensatz und löscht ihn dann:

```
vFind:=Request("zu löschende Mitarbeiterkennung:") ` Erhalte Mitarbeiterkennung
If(OK=1)
   QUERY([Employee];[Employee]ID =vFind) ` Finde Mitarbeiter
   DELETE RECORD([Employee]) ` Lösche Mitarbeiter
End if
```

DISPLAY RECORD

DISPLAY RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle des anzuzeigenden Datensatzes Ohne Angabe Haupttabelle

Beschreibung

Der Befehl **DISPLAY RECORD** zeigt den aktuellen Datensatz der Tabelle *Tabellenname* im aktuellen Eingabeformular. Er wird solange angezeigt, bis ein Ereignis das Fenster neu aufbaut. Das kann z.B. das Ausführen von **ADD RECORD** sein, Zurückkehren zu einem Eingabeformular oder zur Menüleiste. *Tabellenname* ist optional. Wird der Parameter nicht angegeben, bezieht sich **DISPLAY RECORD** auf die Haupttabelle. Gibt es keinen aktuellen Datensatz, hat **DISPLAY RECORD** keine Auswirkung.

DISPLAY RECORD wird oft für eigene Ablaufanzeigen oder für selbst ablaufende Dia-Shows eingesetzt.

Gibt es eine Formularmethode, wird ein Ereignis *On Load* erzeugt.

Warnung: Rufen Sie **DISPLAY RECORD nicht** in einem Prozess für die Web Anbindung auf, da der Befehl auf dem Rechner mit 4D Web Server und nicht auf der Arbeitsstation mit dem Web Browser ausgeführt wird.

Beispiel

Folgendes Beispiel erstellt eine Reihe von Datensätzen als Dia-Show:

```
ALL RECORDS([Demo]) ` Alle Datensätze auswählen
FORM SET INPUT([Demo];"Anzeigen") ` Setze Formular für die Anzeige
For($vlRecord;1;Records in selection([Demo])) ` Durchlaufe alle Datensätze
   DISPLAY RECORD([Demo]) ` Datensatz anzeigen
   DELAY PROCESS(Current process;180) ` Pause für 3 Sekunden
   NEXT RECORD([Demo]) ` Gehe zum nächsten Datensatz
End for
```

DUPLICATE RECORD

DUPLICATE RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Datensatz dupliziert werden soll Ohne Angabe Haupttabelle

Beschreibung

Der Befehl **DUPLICATE RECORD** erstellt einen neuen Datensatz für die Tabelle *Tabellenname*, der ein Duplikat des aktuellen Datensatzes ist. Der duplizierte Datensatz wird der aktuelle Datensatz. Gibt es keinen aktuellen Datensatz, führt **DUPLICATE RECORD** nichts aus. Sie müssen **SAVE RECORD** verwenden, um den neuen Datensatz zu sichern.
Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich **DUPLICATE RECORD** auf die Haupttabelle.

DUPLICATE RECORD kann bei der Dateneingabe ausgeführt werden. So können Sie ein Duplikat des aktuell angezeigten Datensatzes erstellen. Denken Sie jedoch daran, zuerst **SAVE RECORD** auszuführen, damit alle Änderungen am Original Datensatz gesichert werden.

Hinweis zur Kompatibilität: Ab Version 11 unterstützt dieser Befehl NICHT mehr Untertabellen.

GOTO RECORD

GOTO RECORD ({Tabellenname ;} AbsoluteNr)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, in der ein Datensatz ausgewählt werden soll Ohne Angabe Haupttabelle
AbsoluteNr	Lange Ganzzahl	➡	Absolute Datensatznummer

Beschreibung

GOTO RECORD wählt den angegebenen Datensatz der Tabelle *Tabellenname* als den aktuellen Datensatz. Der zweite Parameter entspricht der Nummer, die von der Funktion **Record number** zurückgegeben wird. Nach Ausführen dieses Befehls ist dieser Datensatz der einzige Datensatz in der Auswahl.
Tabellenname ist optional. Wird dieser Parameter nicht angegeben, bezieht sich **GOTO RECORD** auf die Haupttabelle.

Ist *AbsoluteNr* kleiner als die kleinste Datensatznummer oder größer als die größte Datensatznummer in der Datenbank, erzeugt 4D eine Fehlermeldung, die angibt, dass die Datensatznummer außerhalb des Bereiches liegt. Ist *AbsoluteNr* gleich der Datensatznummer eines gelöschten Datensatzes, gibt 4D die Fehlermeldung -10503 zurück und die Auswahl ist leer.

Beispiel

Siehe Beispiel unter Funktion **Record number**.

Is new record

Is new record {(Tabellenname)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle des zu prüfenden Datensatzes Ohne Angabe Standardtabelle
Funktionsergebnis	Boolean	⮌	Wahr, wenn der Datensatz erstellt wurde, Sonst Falsch

Beschreibung

Die Funktion **Is new record** gibt *Wahr* zurück, wenn der aktuelle Datensatz von *Tabellenname* erstellt und noch nicht im aktuellen Prozess gesichert wurde.

Hinweis zur Kompatibilität: Sie erhalten dieselbe Information mit der Funktion **Record number**, wenn Sie prüfen, ob er -3 zurückgibt.
Für diesen Fall sollten Sie jedoch die Funktion **Is new record** anstatt **Record number** verwenden. **Is new record** gewährleistet eine bessere Kompatibilität für zukünftige 4D Versionen.

4D Server: Diese Funktion gibt für das Formularereignis *On Validate* ein anderes Ergebnis zurück, je nachdem ob es in 4D im lokalen oder im remote Modus ausgeführt wird. Im lokalen Modus gibt sie *Falsch* zurück (der Datensatz gilt als bereits angelegt). Im remote Modus gibt sie *Wahr* zurück, weil hier der Datensatz bereits auf dem Server erstellt wurde, die Information jedoch noch nicht an den Client gesendet wurde.

Beispiel

Die beiden folgenden Anweisungen sind identisch. Wir raten jedoch dringend, die zweite Variante zu wählen, da sie mit späteren Versionen von 4D kompatibel ist:

```
If(Record number([Table])=-3)  `Nicht empfohlen
   ` ...
End if

If(Is new record([Table]))  `Dringend empfohlen
   ` ...
End if
```

Is record loaded

Is record loaded {(Tabellenname)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle des zu prüfenden Datensatzes Ohne Angabe Standardtabelle
Funktionsergebnis	Boolean	➲	Wahr, wenn der Datensatz geladen wurde, sonst Falsch

Beschreibung

Die Funktion **Is record loaded** gibt *Wahr* zurück, wenn der aktuelle Datensatz von *Tabellenname* in den aktuellen Prozess geladen wurde.

4D Server: In der Regel werden bei Tabellen, die über automatische Verknüpfungen miteinander verbunden sind, die aktuellen Datensätze der verknüpften Tabellen automatisch geladen (siehe **Verknüpfungen**). Aus Optimierungsgründen lädt 4D Server diese Datensätze jedoch nur bei Bedarf, z.B. wenn ein Feld des verknüpften Datensatzes gelesen oder zugewiesen wird. In diesem Kontext gibt der Befehl **Is record loaded** als Ergebnis im remote Modus *Falsch* zurück (im lokalen Modus gibt er *Wahr* zurück).

Beispiel

Anstelle der automatischen Aktionen “Nächster Datensatz” oder “Voriger Datensatz” für die Schaltflächen können Sie Objektmethoden hinterlegen, die die Arbeitsleistung verbessern. Die Schaltfläche “Nächster” zeigt den Beginn der Auswahl, wenn der Benutzer am Ende der Auswahl ist. Die Schaltfläche "Voriger" zeigt das Ende der Auswahl, wenn der Benutzer am Anfang der Auswahl ist.

```
  ` Objektmethode für die Schaltfläche “Voriger” (ohne automatische Aktion)
If(Form event=On Clicked)
   PREVIOUS RECORD([Group])
   If(Not(Is record loaded([Group])))
      GOTO SELECTED RECORD([Group];Records in selection([Group]))
  `Gehe zum letzten Datensatz in der Auswahl
   End if
End if

  ` Objektmethode für die Schaltfläche “Nächster” (ohne automatische Aktion)
If(Form event=On Clicked)
   NEXT RECORD([Group])
   If(Not(Is record loaded([Group])))
      GOTO SELECTED RECORD([Groups];1)
  `Gehe zum ersten Datensatz in der Auswahl
   End if
End if
```

Modified record

Modified record {(Tabellenname)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Name der Tabelle, zu welcher der Datensatz gehört Ohne Angabe Haupttabelle
Funktionsergebnis	Boolean	⇐	Datensatz geändert (Wahr), oder Datensatz nicht geändert (Falsch)

Beschreibung

Die Funktion **Modified record** gibt *Wahr* zurück, wenn der aktuelle Datensatz von *Tabellenname* geändert, jedoch nicht gesichert wurde, sonst *Falsch*. Sie können somit beispielsweise bei Eingabeformularen rasch feststellen, ob Sie einen Datensatz sichern müssen, bevor Sie zum nächsten wechseln. Diese Funktion gibt für einen neuen Datensatz immer **Wahr** zurück.

Beachten Sie, dass diese Funktion in folgenden Kontexten immer **Wahr** zurückgibt:

- Der aktuelle Datensatz ist ein neuer Datensatz
- Nach Ausführen der Befehle **PUSH RECORD** und **POP RECORD**
- sobald einem Feld des Datensatzes ein Wert zugewiesen wurde, selbst wenn es derselbe Wert wie der vorige ist.
  Zum Beispiel: **Modified record** gibt nach Ausführen der folgenden Anweisung **Wahr** zurück:

```
[Table_1]Field_1:=[Table_1]Field_1
```

Beispiel

Folgendes Beispiel zeigt eine typische Verwendung:

```
If(Modified record([Customers]))
   SAVE RECORD([Customers])
End if
```

POP RECORD

POP RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Datensatz vom Stapel geholt werden soll Ohne Angabe Haupttabelle

Beschreibung

Der Befehl **POP RECORD** holt den letzten Datensatz der Tabelle *Tabellenname* aus den Stapel.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich **POP RECORD** auf die Haupttabelle.

PUSH RECORD nimmt den Datensatz aus der aktuellen Auswahl. Ändern Sie dann die Auswahl und holen den Datensatz wieder mit **POP RECORD**, ist er nicht der aktuelle Datensatz der aktuellen Auswahl. Mit dem Befehl **ONE RECORD SELECT** wird er zum aktuellen Datensatz. Bei allen Befehlen, die den aktuellen Datensatz vor dem Sichern ändern, geht die Kopie im Speicher verloren.

Beispiel

Folgendes Beispiel holt den Kundendatensatz aus dem Stapel:

```
POP RECORD([Customers]) ` Holt den Kundendatensatz aus dem Stapel
```

PUSH RECORD

PUSH RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Datensatz auf den Stapel gelegt werden soll Ohne Angabe Haupttabelle

Beschreibung

Der Befehl **PUSH RECORD** legt den aktuellen Datensatz - und evtl. dazugehörige Unterdatensätze - der Tabelle *Tabellenname* auf den Stapel. **PUSH RECORD** kann vor dem Sichern eines Datensatzes ausgeführt werden.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich **PUSH RECORD** auf die Haupttabelle.

PUSH RECORD nimmt den Datensatz aus der aktuellen Auswahl und sperrt ihn für alle anderen Prozesse und Benutzer, bis er wieder zurückgeholt und vom Stapel entfernt wird. Durch den Befehl **POP RECORD** können Sie diesen Datensatz zurückerhalten.

Hinweis zur Kompatibilität: Ab Version 11 unterstützt dieser Befehl NICHT mehr Untertabellen.

Beispiel

Folgendes Beispiel legt den Kundendatensatz auf den Stapel:

```
PUSH RECORD([Customer]) ` Lege Kundendatensatz auf den Stapel
```

Record number

Record number {(Tabellenname)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle des Datensatzes, dessen Nummer zurückgegeben werden soll
Funktionsergebnis	Lange Ganzzahl	➲	Aktuelle Datensatznummer

Beschreibung

Record number gibt die physikalische Nummer des aktuellen Datensatzes der Tabelle *Tabellenname* zurück. Ist kein aktueller Datensatz vorhanden, z.B. wenn der Zeiger auf einen Datensatz vor oder nach der aktuellen Auswahl verweist, gibt **Record number** den Wert -1 zurück. Ist der aktuelle Datensatz zwar erzeugt, aber noch nicht gesichert, gibt **Record number** den Wert -3 zurück.

Datensatznummern können sich ändern. Nummern von gelöschten Datensätzen werden wiederverwendet.

4D Server: Diese Funktion gibt für das Formularereignis *On Validate* ein anderes Ergebnis zurück, je nachdem, ob sie in 4D im lokalen oder im remote Modus ausgeführt wird. Im lokalen Modus gibt die Funktion eine Datensatznummer zurück (der Datensatz wird als bereits angelegt betrachtet). Im remote Modus gibt sie -3 zurück, weil in diesem Fall der Datensatz bereits auf dem Server angelegt wurde, die Information jedoch noch nicht an den Client gesendet wurde.

Hinweis: Es wird empfohlen, die Funktion **Is new record** zu verwenden, um zu prüfen, ob ein Datensatz gerade erstellt wird.

Beispiel

Folgendes Beispiel sichert die aktuelle Datensatznummer und sucht dann nach den anderen Datensätzen mit demselben Inhalt:

```
$RecNum:=Record number([People])  ` Erhalte Datensatznummer
QUERY([People];[People]Last =[People]Last)
  ` Gibt es noch einen mit diesem Nachnamen?
  ` Zeige Anzahl der Personen mit demselben Nachnamen
ALERT("Es gibt "+String(Records in selection([People])+" mit diesem Namen.")
GOTO RECORD([People];$RecNum)  ` Gehe zurück zum selben Datensatz
```

Records in table

Records in table {(Tabellenname)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Anzahl der Datensätze abgefragt wird Ohne Angabe Haupttabelle
Funktionsergebnis	Lange Ganzzahl	➲	Gesamtanzahl der Datensätze in Tabelle

Beschreibung

Die Funktion **Records in table** gibt die Anzahl der in *Tabellenname* gespeicherten Datensätze zurück. *Tabellenname* ist optional. Wird dieser Parameter nicht angegeben, bezieht sich **Records in table** auf die Haupttabelle.

Dagegen gibt die Funktion **Records in selection** nur die Anzahl der Datensätze in der aktuellen Auswahl zurück.
Benutzen Sie **Records in table** während einer Transaktion, werden die in der Transaktion erzeugten Datensätze mitgezählt.

Beispiel

Folgendes Beispiel nennt in einer Meldung die Anzahl der Datensätze in der Tabelle:

```
ALERT("Es gibt "+String(Records in table([People]))+" Datensätze in der Tabelle.")
```

SAVE RECORD

SAVE RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Datensatz gesichert werden soll Ohne Angabe Haupttabelle

Beschreibung

Der Befehl **SAVE RECORD** sichert den aktuellen Datensatz von *Tabellenname* im aktuellen Prozess. Gibt es keinen aktuellen Datensatz, wird **SAVE RECORD** nicht ausgeführt.

Sie verwenden **SAVE RECORD**, um einen Datensatz zu sichern, der über Programmierung angelegt oder geändert wurde. Ein Datensatz, den der Anwender im Formular geändert und bestätigt hat, muss nicht mit **SAVE RECORD** gesichert werden. Ein Datensatz, den der Anwender im Formular geändert und abgebrochen hat, lässt sich weiterhin mit **SAVE RECORD** sichern.

Rufen Sie **SAVE RECORD** auf, wenn kein Feld im Datensatz geändert wurde, führt der Befehl nichts aus, d.h. der Trigger wird nicht aufgerufen. **SAVE RECORD** müssen Sie anwenden, wenn Sie:

- einen neuen Datensatz sichern wollen, der mit **CREATE RECORD** oder **DUPLICATE RECORD** angelegt wurde.
- Daten aus **RECEIVE RECORD** sichern wollen.
- einen Datensatz sichern wollen, der durch eine Methode geändert wurde.
- einen Datensatz sichern wollen, der nach Aufrufen der Befehle **_o_ADD SUBRECORD**, **_o_CREATE SUBRECORD** oder **_o_MODIFY SUBRECORD** einen neuen oder geänderten Unterdatensatz enthält.
- die Änderungen sichern wollen, bevor ein Befehl zum Wechseln des aktuellen Datensatzes aufgerufen wird.
- einen Datensatz während der Dateneingabe sichern wollen.

Führen Sie **SAVE RECORD** nicht während dem Ereignis *On Validate* für ein bestätigtes Formular aus, denn dann wird der Datensatz doppelt gesichert.

Hinweis: Bei Datensätzen mit bearbeiteten Objektfeldern müssen Sie dies 4D vor Aufrufen von **SAVE RECORD** explizit mitteilen. Weitere Informationen dazu finden Sie im Abschnitt **Objektfelder sichern**.

Beispiel

Folgendes Beispiel ist Teil einer Methode, die Datensätze aus einem Dokument ausliest. Der Programmierabschnitt erhält einen Datensatz und sichert ihn bei korrektem Empfang:

```
RECEIVE RECORD([Customers]) ` Empfange Datensatz von Festplatte
If(OK=1) ` Bei korrektem Empfang···
   SAVE RECORD([Customers]) ` sichere diesen
End if
```

Sequence number

Sequence number {(Tabellenname)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, für deren Datensätze eine fortlaufende Nummer vergeben wird Ohne Angabe Haupttabelle
Funktionsergebnis	Lange Ganzzahl	➲	Sequenznummer

Beschreibung

Sequence number gibt die nächste Sequenznummer für *Tabellenname* zurück. Die Sequenznummer ist für jede Tabelle einmalig und wird für jeden neu angelegten Datensatz der Tabelle um 1 (*) erhöht.

(*) Aus Optimierungsgründen startet die Erhöhung nur beim ersten Aufruf von **Sequence number** oder über ein Feature mit Zugang zur Sequenznummer (siehe unten). Außerdem lässt sich die Nummerierung über den Befehl **SET DATABASE PARAMETER** verändern. Von daher lässt sich der zurückgegebene Wert nicht als Zählen von Datensätzen der Tabelle *Tabellenname* bewerten.

Die Nummerierung beginnt standardmäßig bei 1. Über den Befehl **SET DATABASE PARAMETER** können Sie die Nummerierung für eine Tabelle ändern.

Hinweis: Gibt es keinen aktuellen Datensatz und wurde die Nummerierung über den Befehl **SET DATABASE PARAMETER** geändert, wird diese Nummer zum Erstellen des nächsten Datensatzes reserviert, aber von der Funktion **Sequence number** nur zurückgegeben, wenn aktuell der Befehl **SAVE RECORD** aufgerufen wurde.

Die Funktion **Sequence number** ist hilfreich, wenn:

- nicht in Einer-Schritten hochgezählt werden soll.
- die Nummer Teil eines Codes sein soll, z.B. einer Artikelnummer

Um eine Sequenznummer über eine Methode zu speichern, erstellen Sie ein Datenfeld vom Typ Lange Ganzzahl in der Tabelle und weisen die Nummer diesem Feld zu.

Die von dieser Funktion für *Tabellenname* zurückgegebene Sequenznummer ist dieselbe Nummer, die über die Option **Autoincrement** in den Feldeigenschaften des Inspektors oder über das Symbol #N als Standardwert für ein Feld dieser Tabelle in einem Formular zugewiesen wurde. Weitere Informationen dazu finden Sie im Abschnitt **Standardwerte** des Handbuchs *4D Designmodus*.

Hinweis: Die automatische Erhöhung der Nummer lässt sich auch über das SQL Attribut AUTO_INCREMENT setzen.

Soll die Sequenznummer nicht mit 1 beginnen, müssen Sie lediglich die Differenz zu **Sequence number** hinzufügen. Folgende Methode lässt die Nummerierung der Datensätze mit 1000 beginnen:

```
[Table1]Seq Field :=Sequence number([Table1])+999
```

Beispiel

Folgendes Beispiel ist Teil einer Formularmethode. Sie testet, ob es ein neuer Datensatz ist; z.B. ob die Rechnungsnummer ein leerer String ist. Ist der Datensatz neu, weist die Methode eine Nummer zu. Diese Nummer setzt sich aus zwei Teilen zusammen: Der Sequenznummer und dem Benutzerkürzel, das beim Öffnen der Datenbank eingegeben wird. Die Sequenznummer wird als fünfstelliger String formatiert:

```
If([Invoices]Invoice No="")
   ` Ist Rechnungsnummer leer, erstelle neue Rechnungsnummer
   ` Rechnungsnummer ist String, der mit Benutzerkürzel endet.
   [Invoices]Invoice No:=String(Sequence number;"00000")+[Invoices]OpID
End if
```

Datum und Zeit

- 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 (Datum ; Jahre ; Monate ; Tage) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Datum	Datum	➡	Datum, zu dem Tage, Monate und Jahre hinzugefügt werden sollen
Jahre	Lange Ganzzahl	➡	Anzahl der hinzuzufügenden Jahre
Monate	Lange Ganzzahl	➡	Anzahl der hinzuzufügenden Monate
Tage	Lange Ganzzahl	➡	Anzahl der hinzuzufügenden Tage
Funktionsergebnis	Datum	⮌	Resultierendes Datum

Beschreibung

Die Funktion **Add to date** fügt dem in *Datum* übergebenen Datum *Jahre*, *Monate* und *Tage* hinzu und gibt das Ergebnis zurück.

Sie können zwar auch mit den **Datumsoperatoren** einem Datum Tage hinzufügen. Mit **Add to date** fügen Sie Monate und Jahre jedoch schneller hinzu, da Sie nicht die Anzahl Tage pro Monat bzw. Schaltjahre berücksichtigen müssen (Das ist bei Datumsoperatoren der Fall).

Beispiel

```
  ` Diese Zeile kalkuliert das Datum in einem Jahr, gleicher Tag
$vdInOneYear:=Add to date(Current date;1;0;0)

  ` Diese Zeile kalkuliert das Datum im folgenden Monat, gleicher Tag
$vdNextMonth:=Add to date(Current date;0;1;0)

  ` Diese Zeile macht dasselbe wie $vdTomorrow:=Current date+1
$vdTomorrow:=Add to date(Current date;0;0;1)
```

Current date

Current date {(*)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Gibt das aktuelle Datum vom Server zurück
Funktionsergebnis	Datum	➲	Aktuelles Datum

Beschreibung

Die Funktion **Current date** gibt das aktuelle Tagesdatum zurück. Es entspricht dem Datum vom System des Rechners.
4D Server: Verwenden Sie den optionalen Parameter (*) auf dem remote Rechner, gibt er das aktuelle Datum des Servers zurück.

Beispiel 1

Folgendes Beispiel zeigt eine Meldung mit dem aktuellen Datum an:

```
ALERT("Das Datum ist "+String(Current date)+".")
```

Beispiel 2

Erstellen Sie eine Anwendung für den internationalen Markt, müssen Sie u.U. wissen, ob Ihre 4D Version mit Datumsangaben im Format MM/DD/YYYY (US Version) oder DD.MM.YYYY (deutsche Version) läuft. Das ist hilfreich beim Einrichten von Feldern zur Eingabe von Datum.
Verwenden Sie dafür die folgende Projektmethode:

```
  ` Sys date format globale Funktion
  ` Sys date format -> String
  ` Sys date format -> Standardmäßiges 4D Datumsformat

C_STRING(31;$0;$vsDate;$vsMDY;$vsMonth;$vsDay;$vsYear)
C_LONGINT($1;$vlPos)
C_DATE($vdDate)

 `Erhalte Datumswert, wo Werte für Monat, Tag und Jahr alle unterschiedlich sind.
$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:="" `Initialisiere Funktionsergebnis
$vsDate:=String($vdDate)
$vlPos:=Position("/";$vsDate) `Suche ersten / Trenner im String ..
$vsMDY:=Substring($vsDate;1;$vlPos-1) `Extrahiere die erste Stelle vom Datum
$vsDate:=Substring($vsDate;$vlPos+1) `Entferne die ersten Stellen und ersten / Trenner
Case of
   :($vsMDY=$vsMonth) `Stellen zeigen den Monat an
      $0:="MM"
   :($vsMDY=$vsDay) `Stellen zeigen den Tag an
      $0:="DD"
   :($vsMDY=$vsYear) `Stellen zeigen das Jahr an
      $0:="YYYY"
End case
$0:=$0+"/" `Starte mit Aufbau des Funktionsergebnisses
$vlPos:=Position("/";$vsDate) `Suche zweiten / Trenner im String ../..
$vsMDY:=Substring($vsDate;1;$vlPos-1) `Extrahiere die nächsten Stellen aus dem Datum
$vsDate:=Substring($vsDate;$vlPos+1) `Reduziere den String auf die letzten Stellen im Datum
Case of
   :($vsMDY=$vsMonth) `Stellen zeigen den Monat an
```

```
        $0:=$0+"MM"
    :($vsMDY=$vsDay)  `Stellen zeigen den Tag an
        $0:=$0+"DD"
    :($vsMDY=$vsYear)  `Stellen zeigen das Jahr an
        $0:=$0+"YYYY"
End case
$0:=$0+"/"  `Fahre fort mit Aufbau des Funktionsergebnisses
Case of
    :($vsDate=$vsMonth)  `Stellen zeigen den Monat an
        $0:=$0+"MM"
    :($vsDate=$vsDay)  `Stellen zeigen den Tag an
        $0:=$0+"DD"
    :($vsDate=$vsYear)  `Stellen zeigen das Jahr an
        $0:=$0+"YYYY"
End case
  `An diesem Punkt ist $0 gleich MM/DD/YYYY oder DD.MM.YYYY oder...
```

Current time

Current time {(*)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Gibt die aktuelle Systemzeit zurück
Funktionsergebnis	Zeit	➲	Aktuelle Zeit

Beschreibung

Die Funktion **Current time** gibt die aktuelle Systemzeit des Rechners zurück. Achten Sie darauf, dass die Uhr Ihres Rechners richtig eingestellt ist, damit **Current time** richtige Werte liefert.

Die aktuelle Zeit liegt zwischen *00:00:00* und *23:59:59*. Mit **String** oder **Time string** erhalten Sie das Textformat für den von **Current time** zurückgegebenen Zeitausdruck.

4D Server: Der optionale Parameter (*) gilt nur für den 4D Server. Dadurch wird die aktuelle Zeit des Servers zurückgegeben.

Beispiel 1

Folgendes Beispiel fragt ab, wieviel Zeit eine Operation benötigt:

```
$vhStartTime:=Current time  ` Sichere die Startzeit
LongOperation  ` Führe die Operation durch
ALERT("Die Operation benötigte "+String(Current time-$vhStartTime))
  ` Zeige die Dauer an
```

Beispiel 2

Folgendes Beispiel entnimmt die Stunden, Minuten und Sekunden aus der aktuellen Zeit:

```
$vhNow:=Current time
ALERT("Aktuelle Stunde ist: "+String($vhNow\3600))
ALERT("Aktuelle Minute ist: "+String(($vhNow\60)%60))
ALERT("Aktuelle Sekunde ist: "+String($vhNow%60))
```

Date

Date (DatumString) -> Funktionsergebnis

Parameter	Typ		Beschreibung
DatumString	String	➡	String, der das zurückzugebende Datum angibt
Funktionsergebnis	Datum	↩	Datum

Beschreibung

Die Funktion **Date** bewertet *DatumString* und gibt ein Datum zurück.

Der Parameter *DatumString* muss entweder das ISO Datumsformat oder die regionalen Einstellungen, die im Betriebssystem definiert sind, befolgen.

ISO Datumsformat
Der String muss folgende Formatierung haben: JJJJ-MM-DDTHH:MM:SS", zum Beispiel "2013-11-20T10:20:00". In diesem Fall bewertet **Date** den Parameter *DatumString* korrekt, unabhängig von den aktuellen Spracheinstellungen. Es lassen sich auch dezimale Sekunden (Werte kleiner als 1 Sekunde) mit vorangestelltem Punkt hinzufügen, z.B.: "2013-11-20T10:20:00.9854". Passt das Format *DatumString* nicht exakt in dieses ISO Schema, wird das Datum als abgekürztes Datumsformat bewertet, das sich nach den regionalen Einstellungen des Systems richtet.

Hinweis: Ab 4D v14 wird empfohlen, das Format "YYYY-MM-DDTHH:MM:SS**Z**" zu verwenden, da es dem ISO Standard entspricht und die Darstellung der Zeitzone ermöglicht.

Regionale Einstellungen
Passt das Format *DatumString* nicht zum ISO Format, werden zur Bewertung die regionalen Einstellungen verwendet, die im Betriebssystem als abgekürztes Format definiert sind. In der deutschen Version von 4D muss das Datum standardmäßig die Reihenfolge TT.MM.JJ (Tag, Monat, Jahr) haben. Monat und Tag können eine oder zwei Stellen haben, das Jahr kann zwei- oder vierstellig sein. Ist das Jahr zweistellig, bewertet **Date** je nach dem eingetragenen Wert, ob das Datum zum 21. oder 20. Jahrhundert gehört. Dabei gilt 30 standardmäßig als Schlüsselwert.

- Einen Wert größer als oder gleich 30 wertet 4D als 20. Jahrhundert und fügt vor dem Wert 19 hinzu.
- Einen Wert kleiner als 30 wertet 4D als 21. Jahrhundert und fügt vor dem Wert 20 hinzu.

Diese Standardeinstellung können Sie mit dem Befehl **SET DEFAULT CENTURY** verändern. Folgende Zeichen gelten als Trennung im Datum: Punkt (.), Komma (,), Schrägstrich (/) Bindestrich (-) und Leerzeichen.

- Übergeben Sie in *DatumString* ein ungültiges Datum, z.B. "13.35.97" oder "aa.12.97", gibt **Date** ein leeres Datum zurück (*00.00.00*). Sie müssen selbst prüfen, dass *DatumString* ein gültiges Datum enthält.
- Wird *DatumString* als undefiniert gewertet, gibt **Date** ein leeres Datum zurück (00/00/00). Das ist hilfreich, wenn als Ergebnis eines Ausdrucks ein Datum erwartet wird, auch wenn es undefiniert ist.

Hinweis: Ab 4D v16 R6 lassen sich Datumsangaben in Objektattributen als Datumstyp speichern. Weitere Informationen dazu finden Sie auf der **Seite Kompatibilität** unter der Option *Verwende Datumstyp statt ISO Datumsformat in Objekten*. Über die Funktion **Value type** erfahren Sie, ob das Datum im Objektattribut als Datumstyp oder als String abgespeichert ist (siehe letztes Beispiel).

Beispiel 1

Folgendes Beispiel fordert den Benutzer auf, ein Datum einzugeben. Die eingegebene Zeichenkette wird in ein Datum konvertiert und in der Variablen *reqDate* gespeichert:

```
vdRequestedDate:=Date(Request("Gib das Datum ein:";String(Current date)))
If(OK=1)
   ` Mach etwas mit dem in vdRequestedDate gespeicherten Datum
End if
```

Beispiel 2

Hier sehen Sie verschiedene Fälle:

```
vdDate:=Date("12/25/94") //12/25/94 auf einem US System
vdDate2:=Date("40/40/94") //00/00/00
vdDate3:=Date("Es war 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
```

Beispiel 3

Das Datum wird nach einem Datum im ISO Format bewertet:

```
$vtDateISO:="2013-06-05T20:00:00"
$vDate:=Date($vtDateISO)
 //$vDate stellt 6. Juni 2013 dar, unabhängig von der Sprache des Systems.
```

Beispiel 4

Ein Datum aus einem Objektattribut erhalten, egal in welchem Format das Datum im aktuellen Attribut gespeichert wird:

```
If(Value type($myObj.myDate)=ls date)  //Ist als Datumstyp gespeichert, muss nicht konvertiert werden
    $vDate:=$myObj.myDate
Else  //Ist als String gespeichert
    $vDate:=Date($myObj.myDate)
End if
```

Day number

Day number (Datum) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Datum	Datum	➡	Datum, dessen Wochentag ermittelt werden soll
Funktionsergebnis	Lange Ganzzahl	⮌	Zahl für den Wochentag, auf den ein Datum fällt

Beschreibung

Die Funktion **Day number** gibt eine Ganzzahl zurück, die dem Wochentag von *Datum* entspricht.
Hinweis: Day number gibt *2* für Nulldatum (!00.00.00!) zurück.
4D bietet folgende vordefinierten Konstanten unter dem Thema **Tage und Monate**:

Konstante	Typ	Wert
Sunday	Lange Ganzzahl	1
Monday	Lange Ganzzahl	2
Tuesday	Lange Ganzzahl	3
Wednesday	Lange Ganzzahl	4
Thursday	Lange Ganzzahl	5
Friday	Lange Ganzzahl	6
Saturday	Lange Ganzzahl	7

Hinweis: Day number gibt einen Wert zwischen 1 und 7 zurück. Mit der Funktion **Day of** erhalten Sie in einem Datum die Tageszahl innerhalb des Monats.

Beispiel

Folgendes Beispiel ist eine Funktion, die den aktuellen Tag als Zeichenkette zurückgibt:

```
$viDay :=Day number(Current date)  ` $viDay erhält die Zahl des aktuellen Tages
Case of
   :($viDay =1)
      $0:="Sonntag"
   :($viDay =2)
      $0:="Montag"
   :($viDay =3)
      $0:="Dienstag"
   :($viDay =4)
      $0:="Mittwoch"
   :($viDay =5)
      $0:="Donnerstag"
   :($viDay =6)
      $0:="Freitag"
   :($viDay =7)
      $0:="Samstag"
End case
```

Day of

Day of (Datum) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Datum	Datum	⇒	Datum, dessen Tag ermittelt werden soll
Funktionsergebnis	Lange Ganzzahl	⇐	Tag innerhalb eines Monats

Beschreibung

Die Funktion **Day of** gibt eine Ganzzahl zurück, die dem Tag von *Datum* entspricht.

Hinweis: Day of gibt einen Wert zwischen 1 und 31 zurück. Mit dem Befehl **Day number** erhalten Sie den Wochentag für ein Datum.

Beispiel 1

Folgendes Beispiel erläutert die Verwendung von **Day of**. Die Ergebnisse werden der Variablen *vResult* zugewiesen:

```
vResult:=Day of(!25.12.97!) ` vResult erhält 25
vResult:=Day of(Current date) ` vResult erhält Tag des aktuellen Datums
```

Beispiel 2

Siehe Beispiel für die Funktion **Current date**.

Milliseconds

Milliseconds -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Anzahl Millisekunden, die seit dem Start des Rechners vergangen sind

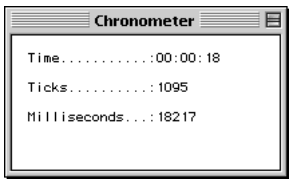
Beschreibung

Die Funktion **Milliseconds** gibt die Anzahl Millisekunden (1000stel Sekunde) seit dem Start des Rechners zurück.

Beispiel

Folgender Code zeigt für eine Minute das Fenster "Chronometer":

```
Open window(100;100;300;200;0;"Chronometer")
$vhTimeStart:=Current time
$vlTicksStart:=Tickcount
$vrMillisecondsStart:=Milliseconds
Repeat
   GOTO XY(2;1)
   MESSAGE("Time.....:"+String(Current time-$vhTimeStart))
   GOTO XY(2;3)
   MESSAGE("Ticks.....:"+String(Tickcount-$vlTicksStart))
   GOTO XY(2;5)
   MESSAGE("Milliseconds...:"+String(Milliseconds-$vrMillisecondsStart))
Until((Current time-$vhTimeStart)>=?00:01:00?)
CLOSE WINDOW
```



Month of

Month of (Datum) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Datum	Datum	➡	Datum, dessen Monat ermittelt werden soll
Funktionsergebnis	Lange Ganzzahl	⮌	Zahl, die den Monat für ein Datum zurückgibt

Beschreibung

Die Funktion **Month of** gibt eine Ganzzahl zurück, die dem Monat von *Datum* entspricht.
Hinweis: Month of gibt die Monatszahl, nicht den Monatsnamen zurück (siehe Beispiel 1).
4D bietet folgende vordefinierte Konstanten unter dem Thema **Tage und Monate**:

Konstante	Typ	Wert
January	Lange Ganzzahl	1
February	Lange Ganzzahl	2
March	Lange Ganzzahl	3
April	Lange Ganzzahl	4
May	Lange Ganzzahl	5
June	Lange Ganzzahl	6
July	Lange Ganzzahl	7
August	Lange Ganzzahl	8
September	Lange Ganzzahl	9
October	Lange Ganzzahl	10
November	Lange Ganzzahl	11
December	Lange Ganzzahl	12

Beispiel 1

Folgendes Beispiel erläutert die Verwendung von **Month of**. Die Ergebnisse werden der Variablen *vResult* zugewiesen:

```
vResult:=Month of(!25.12.97!)  ` vResult erhält 12
vResult:=Month of(Current date)  ` vResult erhält Monat des aktuellen Datums
```

Beispiel 2

Siehe Beispiel zur Funktion **Current date**.

SET DEFAULT CENTURY

SET DEFAULT CENTURY (Jahrhundert {; Schlüsseljahr})

Parameter	Typ		Beschreibung
Jahrhundert	Lange Ganzzahl	➡	Standardjahrhundert (minus eins) für Datumseingabe mit zweistelliger Jahreszahl
Schlüsseljahr	Lange Ganzzahl	➡	Schlüsseljahr für Datumseingabe mit zweistelliger Jahreszahl

Beschreibung

Der Befehl **SET DEFAULT CENTURY** legt das Jahrhundert fest, das 4D bei der Datumseingabe verwendet, wenn das Jahr nur zweistellig eingegeben wird.
Der Parameter *Schlüsseljahr* gibt an, wie 4D die Datumseingabe mit zweistelliger Jahreszahl interpretiert:

- Ist das Jahr größer oder gleich dem Schlüsseljahr, verwendet 4D das aktuelle Jahrhundert.
- Ist das Jahr kleiner als das Schlüsseljahr, verwendet 4D das nächste Jahrhundert (in Bezug auf das Standardjahrhundert).

4D wählt für Jahrhundert standardmäßig das 20. Jahrhundert und setzt 30 als Schlüsseljahr.
Beispiele:

- 25.05.97 bedeutet 25. Mai 1997
- 25.05.30 bedeutet 25. Mai 1930
- 25.05.29 bedeutet 25. Mai 2029
- 25.05.07 bedeutet 25. Mai 2007

Um diese Standardeinstellung zu ändern, geben Sie den Befehl **SET DEFAULT CENTURY** ein. Der Befehl wirkt sich sofort aus. Sie können ein neues Standardjahrhundert oder das Standardjahrhundert und ein Schlüsseljahr übergeben.
Geben Sie in *Jahrhundert* nur das neue Standardjahrhundert minus Eins an, interpretiert 4D das Datum mit zweistelliger Jahreszahl als das gegenwärtige Jahrhundert.
Beispiel:

```
SET DEFAULT CENTURY(20)  ` Wähle als Standard das 21. Jahrhundert
```

- 25.5.97 bedeutet 25. Mai 2097
- 25.5.07 bedeutet 25. Mai 2007

Sie können auch ein eigenes Schlüsseljahr definieren. In folgendem Beispiel ist das Schlüsseljahr 1995:

```
SET DEFAULT CENTURY(19;95)
  ` Wechsle zum 21. Jahrhundert, wenn das Jahr kleiner als 95 ist.
```

- 25.5.97 bedeutet 25. Mai 1997
- 25.5.07 bedeutet 25. Mai 2007

Hinweis: Dieser Befehl gilt nur für die Datumseingabe mit zweistelliger Jahreszahl.
Es gilt in jedem Fall:

- 25.5.1997 bedeutet 25. Mai 1997
- 25.5.2097 bedeutet 25. Mai 2097
- 25.5.1907 bedeutet 25. Mai 1907
- 25.5.2007 bedeutet 25. Mai 2007

Dieser Befehl betrifft nur die Dateneingabe. Er hat keine Auswirkung auf das Speichern von Daten, Berechnungen, usw.

Tickcount

Tickcount -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl	⮌	Anzahl Ticks (60stel Sekunde), die seit dem Start des Rechners vergangen sind

Beschreibung

Die Funktion **Tickcount** gibt die Anzahl Ticks (60stel Sekunde) zurück, die seit dem Start des Rechners vergangen sind.
Hinweis: Tickcount gibt einen Wert vom Typ Lange Ganzzahl zurück.

Beispiel

Siehe Beispiel zum Befehl **Milliseconds**.

Time

Time (ZeitWert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ZeitWert	String, Lange Ganzzahl	⇒	Wert, der als Zeit zurückgegeben werden soll
Funktionsergebnis	Zeit	⇐	In Zeitwert festgelegte Zeit

Beschreibung

Die Funktion **Time** gibt den Wert vom Typ Zeit aus der Umwandlung von *ZeitWert* zurück.
Der Parameter *ZeitWert* kann folgendes enthalten:

- String mit einer Zeit in einem standardmäßigen Zeitformat von 4D enthalten, entsprechend der Sprache Ihres Betriebssystems. Weitere Informationen dazu finden Sie in der Beschreibung zur Funktion **String**
- Lange Ganzzahl mit der Anzahl Sekunden, die seit 00:00:00 vergangen sind

Hinweis: Wird *ZeitWert* als undefiniert gewertet, gibt **Time** eine leere Zeit (00:00:00) zurück. Das ist hilfreich, wenn als Ergebnis eines Ausdrucks (z.B. ein Objektattribut) eine Zeit erwartet wird, auch wenn sie undefiniert ist.

Beispiel 1

Das folgende Beispiel zeigt eine Warnung mit dem Inhalt "1:00 P.M. = 13 Stunden 0 Minuten":

```
ALERT("1:00 P.M. = "+String(Time("13:00:00");Hour Min))
```

Beispiel 2

Jeder numerische Wert lässt sich als Zeit ausdrücken:

```
vTime:=Time(10000)
 //vTime ist 02:46:40
vTime2:=Time((60*60)+(20*60)+5200)
 //vTime2 ist 02:46:40
```

Time string

Time string (Sekunden) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Sekunden	Lange Ganzzahl, Zeit	→	Sekunden ab Mitternacht
Funktionsergebnis	String	↩	Zeit als Zeichenkette im 24-Stunden Format

Beschreibung

Die Funktion **Time string** gibt den String des Zeitausdrucks zurück, den Sie in *Sekunden* übergeben haben. Zeitkonstanten werden so geschrieben: ?9:30:00?.

Der String ist im Format *HH:MM:SS*.

Bei Überschreitung der Sekundenanzahl für einen Tag (86.400) fügt **Time string** weiter Stunden, Minuten und Sekunden hinzu. Beispiel: **Time string** (86401) gibt 24:00:01 zurück.

Hinweis: Benötigen Sie alphanumerische Zeitausdrücke in bestimmten Formaten, verwenden Sie **String**.

Beispiel

Folgendes Beispiel zeigt die Meldung “46800 Sekunden ist 13:00:00.”

```
ALERT("46800 Sekunden ist "+Time string(46800))
```

Timestamp

Timestamp -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	String		Gibt die aktuelle Zeit im ISO Format mit Millisekunden zurück.

Beschreibung

Die Funktion **Timestamp** gibt die aktuelle UTC Zeit (Weltzeit) im ISO Format mit Millisekunden zurück, z.B. yyyy-MM-ddTHH:mm:ss.SSSZ, wobei das Zeichen "Z" die GMT Zeitzone anzeigt.

Jede von **Timestamp** zurückgegebene Zeit wird gemäß dem Standard ISO 8601 ausgedrückt. Weitere Informationen zu diesem Standard finden Sie unter: https://de.wikipedia.org/wiki/ISO_8601

Hinweis: Diese Funktion eignet sich nicht für Zeitmessungen; dafür verwenden Sie die Funktion **Milliseconds**.

Beispiel

Sie können **Timestamp** in einem Logbuch verwenden, um zu sehen, wann genau das Ereignis geschieht. Wie hier gezeigt, können in derselben Sekunde mehrere Operationen stattfinden:

```
$vhDocRef:=Append document("TimestampProject.log")
$logWithTimestamp:=Timestamp+Char(Tab)+"Log with timestamp"+Char(Carriage return)
SEND PACKET($vhDocRef;String($logWithTimestamp))
```

Ergebnis:

```
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 (Datum) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Datum	Datum	➡	Datum, dessen Jahrzahl ermittelt werden soll
Funktionsergebnis	Lange Ganzzahl	➲	Zahl, die das Jahr für ein Datum zurückgibt

Beschreibung

Die Funktion **Year of** gibt eine Ganzzahl zurück, die der Jahreszahl von *Datum* entspricht.

Beispiel 1

Folgendes Beispiel erläutert die Verwendung von **Year of**. Die Ergebnisse werden der Variablen *vResult* zugewiesen.

```
vResult:=Year of(!25.12.97!)  ` vResult erhält 1997
vResult:=Year of(!25.12.1997!)  ` vResult erhält 1997
vResult:=Year of(!25.12.1897!)  ` vResult erhält 1897
vResult:=Year of(!25.12.2097!)  ` vResult erhält 2097
vResult:=Year of(Current date)  ` vResult erhält das Jahr des aktuellen Datums
```

Beispiel 2

Siehe Beispiel zur Funktion **Current date**.

Diagramme

GRAPH
GRAPH SETTINGS
_o_GRAPH TABLE

GRAPH

GRAPH (Diagrammbild ; Diagrammnr | Diagrammeinstellungen ; xBeschriftung {; yElemente} {; yElemente2 ; ... ; yElementeN})

Parameter	Typ		Beschreibung
Diagrammbild	Bildvariable	➡	Bildvariable
Diagrammnr | Diagrammeinstellungen	Lange Ganzzahl, Objekt	➡	Lange Ganzzahl: Nummer des Diagrammtyps Objekt (nur 64-bit): Diagrammeinstellungen
xBeschriftung	Array	➡	Beschriftung der X-Achse
yElemente	Array	➡	Elemente der Y1- bis Y8-Achsen

Beschreibung

Hinweis zur Kompatibilität: *Der Befehl GRAPH funktioniert ab Version 14 nur mit einer Bildvariablen als erster Parameter. Die Syntax mit einem grafischen Bereich (4D Chart) ist überholt und wird nicht mehr unterstützt.*

Der Befehl **GRAPH** erstellt ein Diagramm für eine Bildvariable, die auf Werten aus Arrays basieren.

Mit **GRAPH** erstellte Diagramme lassen sich mit der integrierten SVG rendering Engine erstellen. Sie haben Oberflächenfunktionen, die Bildvariablen zugewiesen sind: Ein Kontextmenü im Anwendungsmodus, um insbesondere das Anzeigeformat zu wählen, Rollbalken, etc.

Hinweis: SVG (Scalable Vector Graphics) ist ein Dateiformat für Grafiken, gekennzeichnet mit der Endung .svg. Es basiert auf XML und ist heute weitverbreitet. Es ist besonders für Web Browser geeignet. Weitere Informationen dazu finden Sie im Internet unter http://www.w3.org/Graphics/SVG/. Der Befehl **SVG EXPORT TO PICTURE** kann auch die Vorteile der integrierten SVG Rendering Engine nutzen.

In *Diagrammbild* übergeben Sie den Namen der Bildvariablen, die das Diagramm im Formular anzeigt.

Der zweite Parameter bestimmt den Diagrammtyp. Es gibt zwei Optionen:

- Sie übergeben einen Parameter *Diagrammnr* vom Typ *Lange Ganzzahl* (alle 4D Versionen): Der Wert muss zwischen 1 und 8 liegen. Die einzelnen Typen sehen Sie in Beispiel 1. Sie können ein bereits erstelltes Diagramm wieder ändern. Ändern Sie dazu den Parameter *Diagrammnr* und führen den Befehl **GRAPH** erneut aus. Über den Befehl **GRAPH SETTINGS** können Sie dann bestimmte Merkmale des Diagramms ändern. Siehe Beispiel 1.
- Sie übergeben einen Parameter *Diagrammeinstellungen* vom Typ *Objekt* (nur 64-bit Versionen von 4D, außer 64-bit Versionen von 4D Server unter Windows): So können Sie den Diagrammtyp mit seinen spezifischen Einstellungen (Legende, xMin, etc.) in einem Aufruf definieren. Dazu verwenden Sie die Konstanten unter dem Thema **Graph Parameter** (siehe unten). Mit dieser Syntax können Sie den Diagrammtyp mit seinen spezifischen Einstellungen (Legende, xMin, etc.) in einem Aufruf definieren. Jetzt können Benutzer die erstellten Diagramme als reguläre SVG Bilder speichern und haben so die Möglichkeit, sie über einen standardmäßigen Browser wie FireFox, Chrome oder Safari anzuzeigen (die generierten Diagramme haben bessere Übereinstimmung mit den SVG Standards in Browsern). Diese Syntax ermöglicht außerdem Zugriff auf verschiedene weitere Einstellungen, z.B. um den Abstand zwischen Balken, Ränder und die Farbe von Balken selbst einzustellen. Siehe Beispiele 2, 3 und 4. **Warnung**: Mit dieser Syntax dürfen Sie NICHT den Befehl **GRAPH SETTINGS** verwenden.

xBeschriftung definiert die Bezeichnungen für die X-Achse (den unteren Teil des Diagramms). Das können Elemente eines Arrays vom Typ alphanumerisch, Datum, Zeit oder Zahl sein. In *xBeschriftung* muss immer die gleiche Anzahl Array-Elemente sein wie in jeder Achse *yElemente.*

yElemente sind Felder aus Arrays mit numerischen Werten. Sie können bis zu acht Datenmengen darstellen. Diagramme vom Typ Kreis stellen nur den ersten Parameter *yElemente* dar.

Automatische IDs

Den Elementen im SVG Diagramm werden automatisch spezifische IDs zugewiesen:

IDs	Beschreibung
ID_graph_1 to ID_graph_8	Spalten, Linien, Flächen....
ID_graph_shadow_1 to ID_graph_shadow_8	Schatten für Spalten, Linien, Flächen...
ID_bullet_1 to ID_bullet_8	Punkte (nur Linien- und Streudiagramme)
ID_pie_label_1 to ID_pie_label_8	Beschriftung Tortendiagramm (nur Tortendiagramm)
ID_legend	Legende
ID_legend_1 to ID_legend_8	Legende Titel
ID_legend_border	Legende Ränder
ID_legend_border_shadow	Schatten für Legende Ränder
ID_x_values	Werte der X Achse
ID_y_values	Werte der Y Achse
ID_y0_axis	Werte der Z Achse
ID_background	Hintergrund
ID_background_shadow	Hintergrund Schatten
ID_x_grid	Gitter auf X Achse
ID_x_grid_shadow	Schatten für Gitter auf X Achse
ID_y_grid	Gitter auf Y Achse
ID_y_grid_shadow	Schatten für Gitter auf Y Achse

Attribute für Diagrammeinstellungen

Im Parameter *Diagrammeinstellungen* übergeben Sie ein Objekt mit den verschiedenen Eigenschaften für das Diagramm. Sie können eine der folgenden Konstanten unter dem Thema **Graph Parameter** verwenden:

Konstante	Typ	Wert	Kommentar
Graph background color	Zeichenkette	graphBackgroundColor	**Mögliche Werte:** Text (kein Standardwert) Wird ein als SVG Bild gespeichertes Diagramm anderweitig geöffnet, wird die Hintergrundfarbe nur berücksichtigt, wenn die SVG Rendering Engine die *SVG Norm tiny 1.2* unterstützt (auf IE und Firefox, aber nicht auf Chrome).
Graph background opacity	Zeichenkette	graphBackgroundOpacity	**Mögliche Werte:** Ganzzahl, Bereich 0-100 **Standardwert:** 100 Wird ein als SVG Bild gespeichertes Diagramm anderweitig geöffnet, wird die Hintergrunddichte nur berücksichtigt, wenn die SVG Rendering Engine die *SVG Norm tiny 1.2* unterstützt (auf IE und Firefox, aber nicht auf Chrome).
Graph background shadow color	Zeichenkette	graphBackgroundShadowColor	**Mögliche Werte**: Zu SVG passende Farbangabe (Text), z.B. "#7F8E00", "Pink", oder "#0a1414"
Graph bottom margin	Zeichenkette	bottomMargin	**Mögliche Werte**: Zahl **Standardwert**: 12
Graph colors	Zeichenkette	colors	**Mögliche Werte**: Text Array. Farben für jede Diagrammreihe. **Standardwerte**: Blaugrün (#19BAC9), Gelb (#FFC338), Purpur (#573E82), Grün (#4FA839), Orange (#D95700), Blau (#1D9DF2), Gelbgrün (#B5CF32), Rot (#D43A26)
Graph column gap	Zeichenkette	columnGap	**Mögliche Werte**: Lange Ganzzahl **Standardwert**: 12 Setzt den Abstand zwischen den Balken
Graph column width max	Zeichenkette	columnWidthMax	**Mögliche Werte**: Zahl **Standardwert**: 200
Graph column width min	Zeichenkette	columnWidthMin	**Mögliche Werte**: Zahl **Standardwert**: 10
Graph default height	Zeichenkette	defaultHeight	**Mögliche Werte**: Zahl (Standardwert: 400), bei Diagrammtyp=7 (Kreis) ist der Standardwert = 600
Graph default width	Zeichenkette	defaultWidth	**Mögliche Werte**: Zahl (Standardwert: 600). Bei Diagrammtyp=7 (Kreis) ist der Standardwert = 800
Graph display legend	Zeichenkette	displayLegend	**Mögliche Werte**: Boolean(Standardwert: wahr)
Graph document background color	Zeichenkette	documentBackgroundColor	**Mögliche Werte**: Zu SVG passende Farbangabe (Text), z.B. "#7F8E00", "Pink" oder "#0a1414". Wird ein als SVG Bild gesichertes Diagramm anderswo geöffnet, erscheint die Hintergrundfarbe des Dokuments nur, wenn die SVG Rendering Engine die *SVG Norm tiny 1.2* unterstützt (auf IE, Firefox, aber nicht auf Chrome).
Graph document background opacity	Zeichenkette	documentBackgroundOpacity	**Mögliche Werte**: Ganzzahl, Bereich 0-100 (Standardwert: 100). Wird ein als SVG Bild gesichertes Diagramm anderswo geöffnet, erscheint die Hintergrunddichte des Dokuments nur, wenn die SVG Rendering Engine die *SVG Norm tiny 1.2* unterstützt (auf IE, Firefox, aber nicht auf Chrome).
Graph font color	Zeichenkette	fontColor	**Mögliche Werte:** Zu SVG passende Farbangabe (Text), z.B. "#7F8E00", "Pink" oder "#0a1414"
Graph font size	Zeichenkette	fontSize	**Mögliche Werte**: Lange Ganzzahl **Standardwert**: 12. Ist Diagrammtyp=7 (Kreis), siehe Graph pie font size
Graph left margin	Zeichenkette	leftMargin	**Mögliche Werte**: Zahl **Standardwert**: 12
Graph legend font color	Zeichenkette	legendFontColor	**Mögliche Werte**: Zu SVG passende Farbangabe (Text), z.B. "#7F8E00", "Pink" oder "#0a1414"
Graph legend icon gap	Zeichenkette	legendIconGap	**Mögliche Werte**: Zahl **Standardwert**: Graph legend icon height/2
Graph legend icon height	Zeichenkette	legendIconHeight	**Mögliche Werte**: Zahl **Standardwert**: 20
Graph legend icon width	Zeichenkette	legendIconWidth	**Mögliche Werte**: Zahl **Standardwert**: 20
Graph legend labels	Zeichenkette	legendLabels	**Mögliche Werte**: Text Array. Ohne Angabe zeigt 4D Icons ohne Text.
Graph line width	Zeichenkette	lineWidth	**Mögliche Werte**: Zahl **Standardwert**: 2

Konstante	Typ	Wert	Kommentar
Graph pie direction	Zeichenkette	pieDirection	**Mögliche Werte**: 1 oder -1 **Standardwert**: 1 1 gbit die Richtung im Uhrzeigersinn an, -1 die Richtung gegen den Uhrzeigersinn
Graph pie font size	Zeichenkette	pieFontSize	**Mögliche Werte**: Zahl **Standardwert**: 16
Graph pie shift	Zeichenkette	pieShift	**Mögliche Werte**: Zahl **Standardwert**: 8
Graph pie start angle	Zeichenkette	pieStartAngle	**Mögliche Werte**: Zahl (positiv oder negativ) **Standardwert**: 0 ist ein Startwinkel von 0° (nach oben zeigende Position) Ein positiver Wert ist ein Winkel in Bezug auf die aktuelle Richtung des Tortendiagramms. Ein negativer Wert ist ein Winkel in Bezug auf die Gegenrichtung des Tortendiagramms.
Graph plot height	Zeichenkette	plotHeight	**Mögliche Werte**: Zahl **Standardwert**: 12
Graph plot radius	Zeichenkette	plotRadius	**Mögliche Werte**: Zahl **Standardwert**: 12
Graph plot width	Zeichenkette	plotWidth	**Mögliche Werte**: Zahl **Standardwert**: 12
Graph right margin	Zeichenkette	rightMargin	**Mögliche Werte**: Zahl **Standardwert**: 2
Graph top margin	Zeichenkette	topMargin	**Mögliche Werte**: Zahl **Standardwert**: 2
Graph type	Zeichenkette	graphType	**Mögliche Werte**: Lange Ganzzahl [1 bis 8]: 1 = Säulen, 2 = proportional, 3 = gestapelt, 4 =Linien, 5 = Flächen, 6 = Punkte, 7 = Kreis, 8 = Bilder. **Standardwert**: 1 Bei Null wird das Diagramm nicht gezeichnet und es erscheint **keine** Fehlermeldung. Ist der Diagrammtyp außerhalb des Bereichs, wird das Diagramm auch nicht gezeichnet, aber es erscheint eine Fehlermeldung. Für Diagramme vom Typ Bild (Wert= 8) müssen die verwendeten Bilder in folgendem Ordner liegen: 4D/Resources/GraphTemplates/Graph_8_Pictures/. Es gibt kein Muster für Bildnamen; 4D sortiert die im Ordner enthaltenen Dateien und weist die erste Datei dem ersten Diagramm zu. Die Dateien können vom Typ SVG oder Bild sein.
Graph xGrid	Zeichenkette	xGrid	**Mögliche Werte:** Boolean **Standardwert**: Wahr Nur mit den proportionalen Typen 4 und 6 verwendbar
Graph xMax	Zeichenkette	xMax	**Mögliche Werte**: Zahl, Datum, Zeit (gleicher Typ wie Parameter *xBeschriftung*). Nur Werte kleiner als xMax erscheinen im Diagramm. xMax wird nur für die Diagrammtypen 4, 5 oder 6 verwendet, wenn xProp=wahr und *xBeschriftung* vom Typ Zahl, Datum oder Zeit ist. Ohne Angabe oder wenn xMin>xMax berechnet 4D automatisch den Wert xMax.
Graph xMin	Zeichenkette	xMin	**Mögliche Werte**: Zahl, Datum, Zeit (gleicher Typ wie Parameter *xBeschriftung*). Nur Werte höher als xMin werden im Diagramm angezeigt. xMin wird nur für die Diagrammtypen 4, 5 oder 6 verwendet, wenn xProp=wahr und *xBeschriftung* vom Typ Zahl, Datum oder Zeit. Ohne Angabe oder wenn xMin>xMax berechnet 4D automatisch den Wert xMin.
Graph xProp	Zeichenkette	xProp	**Mögliche Werte**: Boolean **Standardwert**: Falsch Wahr für proportionale x-Achse; Falsch für normale x-Achse. xProp wird nur für die Diagrammtypen 4, 5 oder 6 verwendet.
Graph yGrid	Zeichenkette	yGrid	**Mögliche Werte**: Boolean **Standardwert**: Wahr
Graph yMax	Zeichenkette	yMax	**Mögliche Werte**: Zahlen Ohne Angabe berechnet 4D automatisch den Wert yMax.
Graph yMin	Zeichenkette	yMin	**Mögliche Werte**: Zahlen Ohne Angabe berechnet 4D automatisch den Wert yMin.

Beispiel 1

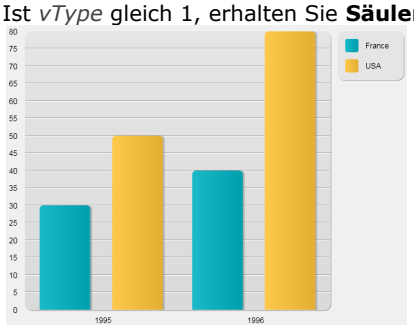
Syntax mit Diagrammnr: Folgendes Beispiel zeigt die verschiedenen Diagrammtypen, die Sie erhalten können. Der Code dafür wird in eine Formular- oder Objektmethode eingetragen. Er entspricht jedoch nicht der Realität, da die Daten hier konstant sind:

```
C_PICTURE(vGraph) //Variable des Diagramms
ARRAY TEXT(X;2) //Erstelle Array für die X-Achse
X{1}:="1995" //X Beschriftung #1
```

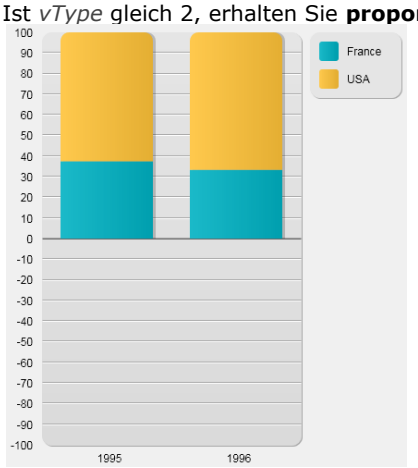
```
X{2}:="1996"  //X Beschriftung #2
ARRAY REAL(A;2)  //Erstelle Array für die Y-Achse
A{1}:=30  //Füge einige Daten ein
A{2}:=40
ARRAY REAL(B;2)  //Erstelle Array für die Y-Achse
B{1}:=50  //Füge einige Daten ein
B{2}:=80
vType:=1  //Initialisiere Diagrammtyp
GRAPH(vGraph;vType;X;A;B)  //Erstelle Diagramm
GRAPH SETTINGS(vGraph;0;0;0;0;False;False;True;"France";"USA")
 //Setze Beschriftung für das Diagramm
```

Sie erhalten als Ergebnis folgende Diagramme.

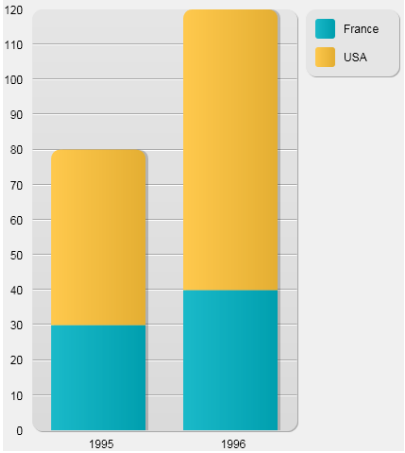
- Ist *vType* gleich 1, erhalten Sie **Säulen**



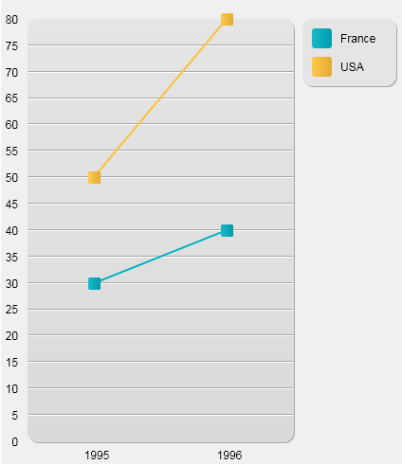
- Ist *vType* gleich 2, erhalten Sie **proportionale Säulen**



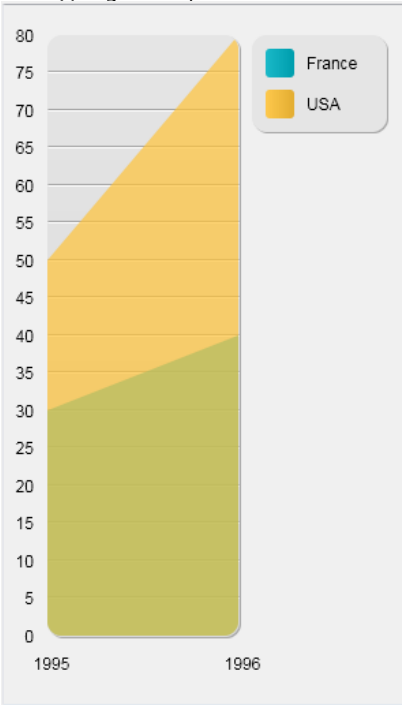
- Ist *vType* gleich 3, erhalten Sie **gestapelte Säulen**



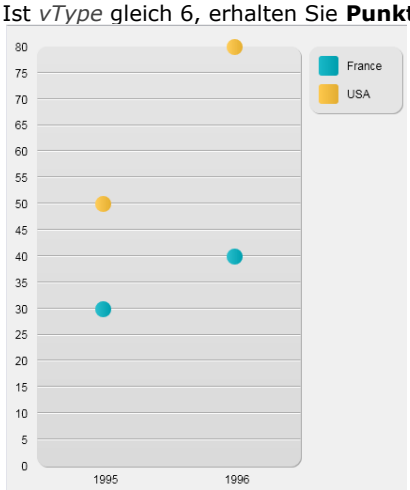
- Ist *vType* gleich 4, erhalten Sie **Linien**



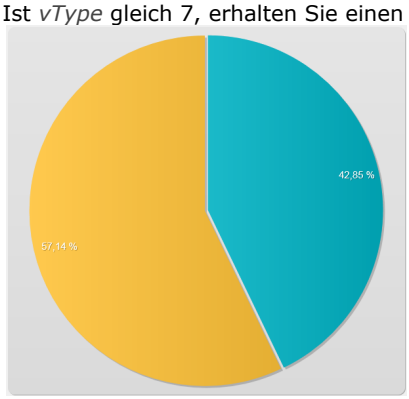
- Ist *vType* gleich 5, erhalten Sie **Flächen**



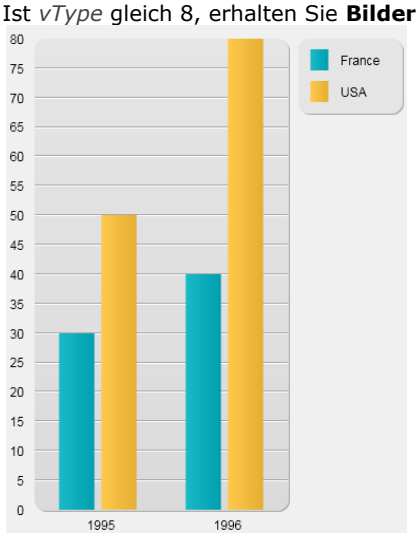
- Ist *vType* gleich 6, erhalten Sie **Punkte**



- Ist *vType* gleich 7, erhalten Sie einen **Kreis**



- Ist *vType* gleich 8, erhalten Sie **Bilder**



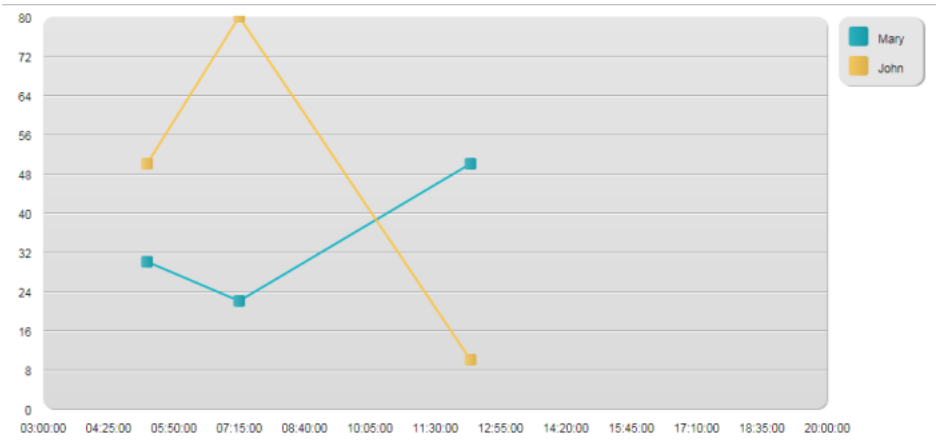
Hinweis: Bilder sind standardmäßig einfache Rechtecke

Beispiel 2

Mit den gleichen Werten können Sie eigene Einstellungen hinzufügen, um eine andere Ansicht zu erhalten:

```
C_PICTURE(vGraph) //Bildvariable
ARRAY TIME(X;3) //Array für die x-Achse erstellen
X{1}:=?05:15:10? //X Label #1
X{2}:=?07:15:10? //X Label #2
X{3}:=?12:15:55? //X Label #3

ARRAY REAL(A;3) //Array für die y-Achse erstellen
A{1}:=30 //Einige Daten eingeben
A{2}:=22
A{3}:=50

ARRAY REAL(B;3) //Ein weiteres Array für die y-Achse erstellen
B{1}:=50 //Einige Daten eingeben
B{2}:=80
B{3}:=10

C_OBJECT(vSettings) //Diagrammeinstellungen festlegen

OB SET(vSettings;Graph type;4) //Liniendiagramm

ARRAY TEXT(aLabels;2) //Diagrammbeschriftung setzen
aLabels{1}:="Mary"
aLabels{2}:="John"
OB SET ARRAY(vSettings;Graph legend labels;aLabels)

 //Optionen
OB SET(vSettings;Graph xProp;True) //proportional setzen
OB SET(vSettings;Graph xGrid;False) //Vertikales Raster entfernen
OB SET(vSettings;Graph xMin;?03:00:00?) //Begrenzungen definieren
OB SET(vSettings;Graph xMax;?20:00:00?)
```

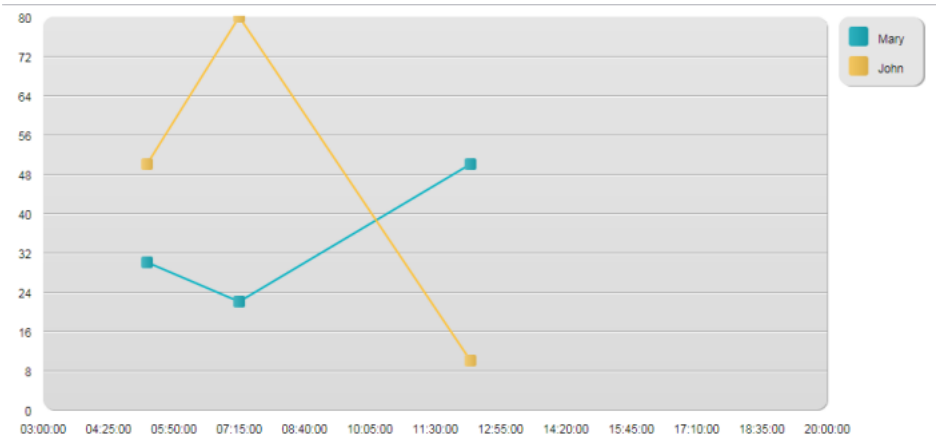
```
GRAPH(vGraph;vSettings;X;A;B) //Diagramm zeichnen
```



Beispiel 3

Mit den gleichen Werten können Sie eigene Einstellungen hinzufügen, um eine andere Ansicht zu erhalten:

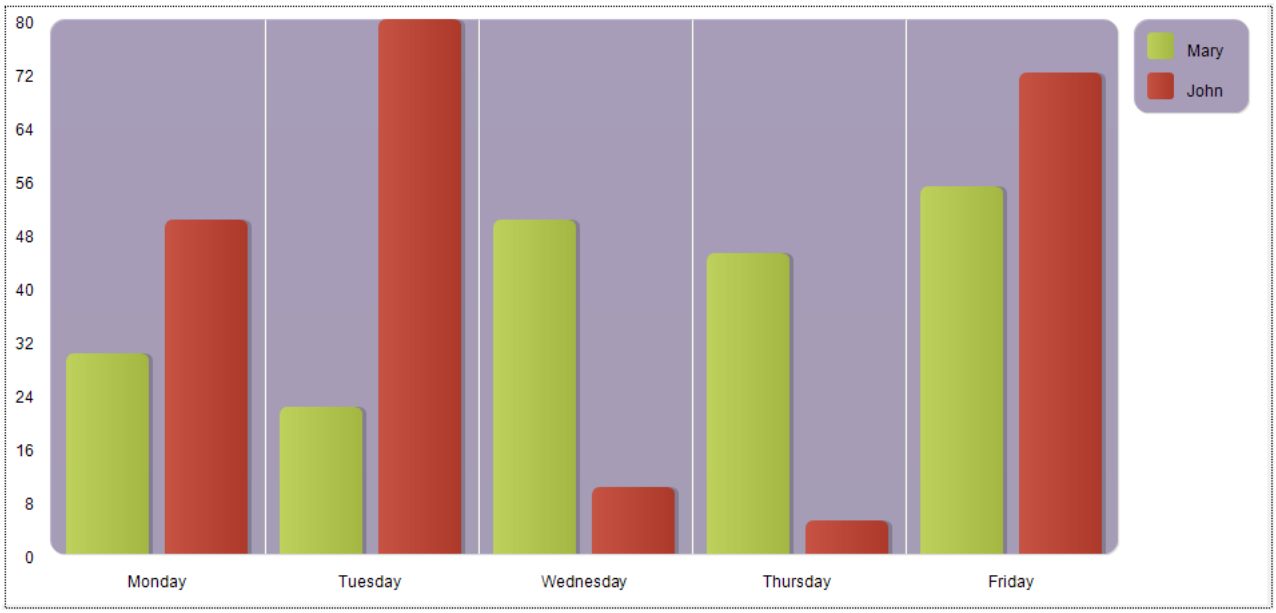
```
C_PICTURE(vGraph) //Bildvariable
ARRAY TIME(X;3) //Array für die x-Achse erstellen
X{1}:=?05:15:10? //X Label #1
X{2}:=?07:15:10? //X Label #2
X{3}:=?12:15:55? //X Label #3

ARRAY REAL(A;3) //Array für die y-Achse erstellen
A{1}:=30 //Einige Daten eingeben
A{2}:=22
A{3}:=50

ARRAY REAL(B;3) //Ein weiteres Array für die y-Achse erstellen
B{1}:=50 //Einige Daten eingeben
B{2}:=80
B{3}:=10

C_OBJECT(vSettings) //Diagrammeinstellungen festlegen

OB SET(vSettings;Graph type;4) //Liniendiagramm

ARRAY TEXT(aLabels;2) //Diagrammbeschriftung setzen
aLabels{1}:="Mary"
aLabels{2}:="John"
OB SET ARRAY(vSettings;Graph legend labels;aLabels)

 //Optionen
OB SET(vSettings;Graph xProp;True) //proportional setzen
OB SET(vSettings;Graph xGrid;False) //Vertikales Raster entfernen
OB SET(vSettings;Graph xMin;?03:00:00?) //Begrenzungen definieren
OB SET(vSettings;Graph xMax;?20:00:00?)

GRAPH(vGraph;vSettings;X;A;B) //Diagramm zeichnen
```



Beispiel 4

In diesem Beispiel passen wir ein paar Einstellungen an:

```
C_PICTURE(vGraph) //Variable des Diagramms
ARRAY TEXT(X;5) //Array für die x-Achse erstellen
X{1}:="Monday" //X Bezeichnung #1
X{2}:="Tuesday" //X Bezeichnung #2
X{3}:="Wednesday" //X Bezeichnung #3
X{4}:="Thursday" //X Bezeichnung #4
X{5}:="Friday" //X Bezeichnung #5

ARRAY LONGINT(A;5) //Array für die y-Achse erstellen
A{1}:=30 //Ein paar Daten einfügen
A{2}:=22
A{3}:=50
A{4}:=45
A{5}:=55

ARRAY LONGINT(B;5) //ein anderes Array für die y-Achse erstellen
B{1}:=50 //Ein paar Daten einfügen
B{2}:=80
B{3}:=10
B{4}:=5
B{5}:=72

C_OBJECT(vSettings) //Einstellungen des Diagramms starten

OB SET(vSettings;Graph type;1) //Balken definieren

ARRAY TEXT(aLabels;2) //Beschriftung für das Diagramm setzen
aLabels{1}:="Mary"
aLabels{2}:="John"
OB SET ARRAY(vSettings;Graph legend labels;aLabels)

 //Optionen
OB SET(vSettings;Graph yGrid;False) //vertikales Raster entfernen
OB SET(vSettings;Graph background color;"#573E82") //eine Hintergrundfarbe setzen
OB SET(vSettings;Graph background opacity;40)
ARRAY TEXT($aTcols;2) //die Farben für das Diagramm setzen
$aTcols{1}:="#B5CF32"
$aTcols{2}:="#D43A26"
OB SET ARRAY(vSettings;Graph colors;$aTcols)
GRAPH(vGraph;vSettings;X;A;B) //Das Diagramm zeichnen
```


80
72
64
56
48
40
32
24
16
8
0
Monday
Tuesday
Wednesday
Thursday
Friday
Mary
John

GRAPH SETTINGS

GRAPH SETTINGS (Diagrammbild ; XMin ; XMax ; YMin ; YMax ; XProp ; XRaster ; YRaster ; Titel {; Titel2 ; ... ; TitelN})

Parameter	Typ		Beschreibung
Diagrammbild	Bildvariable	➡	Bildvariable
XMin	Lange Ganzzahl, Datum, Zeit	➡	Minimalwert auf der X-Achse für proportionale Diagramme (nur Linien oder Punkte)
XMax	Lange Ganzzahl, Datum, Zeit	➡	Maximalwert auf der X-Achse für proportionale Diagramme (nur Linien oder Punkte)
YMin	Lange Ganzzahl	➡	Minimalwert auf der Y-Achse
YMax	Lange Ganzzahl	➡	Maximalwert auf der Y-Achse
XProp	Boolean	➡	TRUE für proportionale X-Achse; FALSE für normale X-Achse (nur Linien oder Punkte)
XRaster	Boolean	➡	TRUE für Raster auf der X-Achse; FALSE, wenn kein Raster auf der X-Achse (nur wenn XProp WAHR ist)
YRaster	Boolean	➡	TRUE für Raster auf der Y-Achse; FALSE, wenn kein Raster auf der Y-Achse;
Titel	String	➡	Titel für Diagrammbeschriftung

Beschreibung

Der Befehl **GRAPH SETTINGS** ändert die Einstellungen für Diagramme in einem Formular. Das Diagramm muss zuvor mit **GRAPH** angezeigt werden. **GRAPH SETTINGS** hat keine Auswirkung auf ein Kreisdiagramm. Dieser Befehl muss im gleichen Prozess wie das Formular aufgerufen werden.

Hinweis: Sie dürfen diesen Befehl nicht aufrufen, wenn das Diagramm mit dem Parameter *Diagrammeinstellungen* vom Typ *Objekt* des Befehls **GRAPH** erstellt wurde. Weitere Informationen dazu finden Sie in der Beschreibung zu **GRAPH**.

XMin, *XMax*, *YMin* und *YMax* stellen die Minimal- bzw. Maximal-Werte auf der X-Achse bzw. Y-Achse dar. Sind XMin,YMin und XMax, YMax gleich 0, (je nach Datentyp 0, ?00:00:00? oder !00.00.00!) wird die X-Achse bzw. Y-Achse automatisch festgelegt. Die Parameter *XMin* und *XMax* werden nur für proportionale Diagramme berücksichtigt (*XProp* ist **Wahr**).

XProp ist ein boolescher Wert. Hat er den Wert **Wahr**, werden die Werte der X-Achse proportional verteilt. Das gilt nur für Diagramme vom Typ 4 (Liniendiagramm) und 6 (Punktediagramm) und nur für Werte vom Typ Zahl, Datum oder Zeit.

Hinweis: Mit 4D Server, 64-bit Version für OS X, wird der Parameter *XProp* auch für Diagramme vom Typ 5 (Flächendiagramm) berücksichtigt.

XRaster und *YRaster* sind boolesche Werte. Haben Sie den Wert Wahr, wird ein Raster auf der X-Achse bzw. Y-Achse angelegt. Ein Raster für die X-Achse erscheint nur für Diagramme vom Typ Punkt oder Linie und nur bei proportionaler X-Achse.

Titel gibt den/die Titel für die Beschriftung an.

Beispiel

Siehe Beispiel zum Befehl **GRAPH**.

_o_GRAPH TABLE

_o_GRAPH TABLE
Dieser Befehl benötigt keine Parameter

Beschreibung

***Befehl gelöscht**: Dieser Befehl wird mit 4D v14 nicht mehr unterstützt*.

Drag and Drop

- Einführung in Drag and Drop
- Datenbankmethode On Drop
- DRAG AND DROP PROPERTIES
- Drop position
- SET DRAG ICON

Einführung in Drag and Drop

4D ermöglicht, Objekte in Ihren Formularen per Drag-and-Drop Technik zu bewegen. Sie können ein Objekt in ein anderes bewegen, egal ob es im gleichen oder einem anderen Fenster liegt. Drag-and-Drop ist also innerhalb eines Prozesses oder von einem Prozess zum nächsten möglich.

Sie können Objekte per Drag-and-Drop zwischen 4D Formularen und anderen Anwendungen bewegen und umgekehrt. Es ist z.B. möglich, ein GIF-Bild per Drag-and-Drop in ein 4D Datenfeld vom Typ Bild zu ziehen. Sie können auch Text in einem Textverarbeitungsprogramm auswählen und in eine 4D Variable vom Typ Text ziehen.
Sie können Objekte direkt in die 4D Anwendung ziehen. Dafür muss kein Formular im Vordergrund sein. In diesem Fall können Sie die **Datenbankmethode On Drop** zum Steuern der Drag-and-Drop Aktion verwenden. Sie können z.B. ein 4D Write Dokument öffnen, indem Sie es auf das 4D Programm-Icon ziehen.

Hinweis: Zu Beginn gehen wir davon aus, dass eine Drag-and-Drop Aktion Daten von einem Punkt zu einem anderen "transportiert". Später sehen wir dann, dass Drag-and-Drop auch als Metapher für jede Art von Operation dienen kann.

Objekteigenschaften Dragfähig und Dropfähig

Aktive Objekte in einem Formular können Eigenschaften für Drag-and-Drop aufweisen, d.h. sie können mit gedrückter Maustaste bewegt werden. Diese Eigenschaften legen Sie in der Eigenschaftenliste fest.

Dragfähig bedeutet, dass der Benutzer das Objekt bei gedrückter Maustaste ziehen kann. Das gezogene Objekt ist das **Quellobjekt**.

Dropfähig bedeutet, dass das Objekt ein bei gedrückter Maustaste gezogenes Objekt aufnehmen kann. Das empfangende Objekt ist das **Zielobjekt**.

Automatisches Drag und **Automatisches Drop**: Diese zusätzlichen Eigenschaften sind für Textfelder und Variablen sowie Combo Boxen und Listboxen verfügbar. **Automatisches Drop** ist auch für Felder vom Typ Bild und Variablen möglich. Damit können Sie einen automatischen Drag and Drop Modus aktivieren, der auf Kopieren des Inhalts basiert. (Die Drag and Drop Aktion wird nicht mehr über 4D Formularereignisse verwaltet). Weitere Informationen siehe unten im Abschnitt "Automatisches Drag and Drop".

Neu erstellte Objekte haben keine Eigenschaften für Drag-and-Drop. Sie entscheiden selbst, ob Sie diese Eigenschaften zuweisen.

Die Drag-and-Drop Technik ist möglich für alle Objekte in einem Eingabeformular, für eigene Elemente in einem Array, z.B. rollbarer Bereich, Einträge einer hierarchischen Liste oder Zeilen einer Listbox. Sie können umgekehrt auch ein Objekt per Drag&Drop in ein eigenes Element eines Arrays, in einen Eintrag einer hierarchischen Liste oder in Zeilen einer Listbox bewegen. Sie können dagegen nicht Objekte aus einem Detail-Bereich eines Ausgabeformulars per Drag&Drop bewegen.

Über die **Datenbankmethode On Drop** können Sie Drag-and-Drop in der Anwendung auch außerhalb von Formularen einsetzen.

Da 4D jede Art von aktivem Objekt (Datenfeld oder Variable) als Quell- oder Zielobjekt zulässt, lässt sich eine drag- und drop-fähige Benutzeroberfläche schnell und problemlos erstellen. So können Sie beispielsweise eine Schaltfläche per Drag-and-Drop bewegen.

Hinweise:

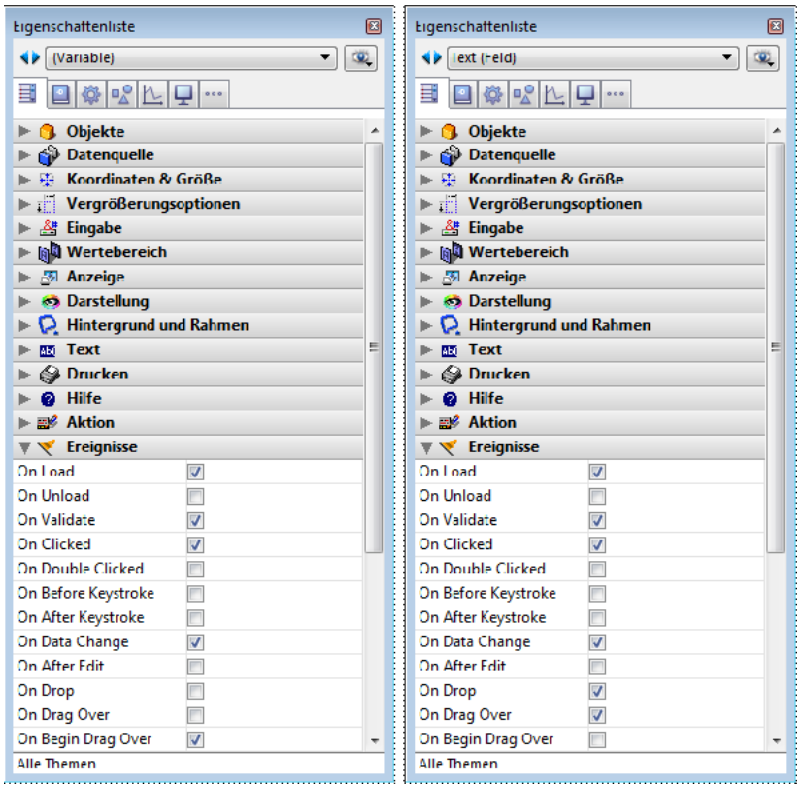
- Um einen Text oder eine Schaltfläche mit dem Attribut "dragfähig" zu ziehen, müssen Sie zuerst unter Windows die **Alt-Taste**, auf Macintosh die **Wahltaste** drücken.
- Bei Bildvariablen und -feldern werden standardmäßig das Bild und seine Referenz übertragen. Wollen Sie nur die Referenz übertragen, drücken Sie während der Operation unter Windows die **Alt-Taste**, auf Mac OS die **Wahltaste**.
- Gibt es für ein Objekt vom Typ Listbox die Eigenschaften "dragfähig" und "verschiebbare Zeilen", hat die Eigenschaft "verschiebbare Zeilen" Vorrang, wenn eine Zeile bewegt wird. In diesem Fall ist keine Drag-Aktion möglich.

Ein Objekt, das sowohl drag- als auch dropfähig ist, lässt sich auch auf sich selbst ziehen, außer Sie schließen diese Operation aus. Weitere Informationen dazu finden Sie in den folgenden Abschnitten.
Folgendes Bild zeigt die Eigenschaftenliste mit den markierten Eigenschaften dropfähig bzw. dragfähig für die ausgewählten Objekte:

Drag-and-Drop per Programmierung verwalten

4D stellt drei Formularereignisse zur Verfügung: *On Begin Drag Over*, *On Drag Over* und *On Drop*. Beachten Sie, dass *On Begin Drag Over* **im Kontext des Quellobjekts der Drag Aktion generiert wird,** während *On Drag Over* und *On Drop* **nur zum Zielobjekt gesendet werden**.

Das Programm kann diese Ereignisse nur ausführen, wenn sie in der Eigenschaftenliste entsprechend aktiviert wurden:



On Begin Drag Over

Das Formularereignis *On Begin Drag Over* lässt sich für jedes dragfähige Formularobjekt wählen und wird immer erzeugt, wenn das Objekt die Eigenschaft **dragfähig** hat.
Im Gegensatz zumFormularereignis *On Drag Over* wird *On Begin Drag Over* im Quellobjekt der Drag Aktion aufgerufen, entweder in der Methode oder in der Formularmethode des Quellobjekts
Dieses Ereignis dient zur erweiterten Verwaltung der Drag-Aktion. Sie können damit folgendes ausführen:

- Die Daten und Signaturen aus der Zwischenablage (Pasteboard) erhalten (über den Befehl **GET PASTEBOARD DATA**).
- Daten und Signaturen in die Zwischenablage (Pasteboard) legen (über den Befehl **APPEND DATA TO PASTEBOARD**).
- Während der Drag-Aktion einen eigenen Icon verwenden (über den Befehl **SET DRAG ICON**).
- Über $0 in der Methode des zu ziehenden Objekts Drag erlauben oder verweigern: Um anzuzeigen, dass Drag Aktionen erlaubt sind, muss die Methode des Quellobjekts 0 (Null) zurückgeben, Sie müssen also $0=0 ausführen. Um anzuzeigen, dass Drag Aktionen verweigert werden, muss die Methode des Quellobjekts -1 zurückgeben, Sie müssen also $0=-1 ausführen. Wird kein Ergebnis zurückgegeben, geht 4D davon aus, dass Drag Aktionen akzeptiert werden.

4D Daten werden vor Aufrufen des Ereignisses in die Zwischenablage (pasteboard) gelegt. Wird z.B. Drag ohne die Option **Automatisches Drag** ausgeführt, liegt der gezogene Text bereits in der Zwischenablage, wenn das Ereignis aufgerufen wird.

On Drag Over

Das Ereignis *On Drag Over* wird immer zum Zielobjekt gesendet, wenn der Mauszeiger über das Objekt bewegt wird. Als Antwort auf dieses Ereignis:

- Rufen Sie den Befehl **DRAG AND DROP PROPERTIES** auf, der Sie über das Quellobjekt informiert.
- **Bestätigen** oder **Annullieren** Sie Drag&Drop je nach Natur und Typ von Zielobjekt (dessen Objektmethode gerade ausgeführt wird) und Quellobjekt.

Zum Bestätigen von Drag muss die Methode des Zielobjekts *0* (Null) zurückgeben. Sie schreiben also *$0:=0*. Zum Abweisen von Drag muss die Methode des Zielobjekts *-1* (minus eins) zurückgeben. Sie schreiben also *$0:=-1*. Während einem Ereignis *On Drag Over* behandelt 4D die Objektmethode wie eine Funktion. Wird kein Ergebnis zurückgegeben, nimmt 4D an, dass Drag akzeptiert wird.

Bei Bestätigen von Drag wird das Zielobjekt markiert, bei Abweisen von Drag wird das Zielobjekt nicht markiert. Bestätigen von Drag bedeutet jedoch nicht, dass die bewegten Daten in das Zielobjekt gesetzt werden. Es bedeutet lediglich, dass bei Loslassen der Maustaste an dieser Stelle das Zielobjekt die bewegten Daten aufnehmen würde.

Legen Sie für ein dropfähiges Objekt kein Ereignis *On Drag Over* fest, wird dieses Objekt für alle "drag over" Operationen markiert, egal welcher Art die bewegten Daten sind.

Mit dem Ereignis *On Drag Over* können Sie die erste Phase einer Drag-and-Drop Operation steuern. Sie können einerseits testen, ob die bewegten Daten mit dem Zielobjekt kompatibel sind und dann Drag bestätigen oder annullieren. Sie können andererseits auch den Benutzer darüber informieren, denn – je nach Ihrer Entscheidung – wird das Zielobjekt markiert oder nicht markiert.

Das Code-handling für ein Ereignis *On Drag Over* sollte kurz sein und schnell ausgeführt werden, da dieses Ereignis bei jeder Mausbewegung erneut an das aktuelle Zielobjekt gesendet wird.

WARNUNG: Ab Version 11 wird bei einer **Drag-and-Drop Operation auf Interprozessebene**, d.h. das Quellobjekt liegt in einem anderen Prozess (Fenster) als das Zielobjekt, die Objektmethode des Zielobjekts für ein Ereignis *On Drag Over* im

Kontext des Zielprozesses ausgeführt. Um den Wert der bewegten Elemente herauszufinden, müssen Sie die Befehle zur Interprozess Kommunikation verwenden.
Wir empfehlen für diesen Fall, das Ereignis On Begin Drag Over und die Befehle im Kapitel **Pasteboard** zu verwenden.

On Drop

Das Ereignis *On Drop* wird an das Zielobjekt gesendet, wenn der Mauszeiger auf das Objekt losgelassen wird. Dieses Ereignis ist die zweite Phase der Drag&Drop Operation. Sie wird als Folge der Benutzeraktion ausgeführt.

Dieses Ereignis wird nur an das Objekt gesendet, wenn Drag während den Ereignissen *On Drag Over* bestätigt wurde. Führen Sie für ein Objekt das Ereignis *On Drag Over* aus und weisen Drag zurück, tritt das Ereignis *On Drop* nicht ein. Haben Sie also während dem Ereignis *On Drag Over* bereits die Kompatibilität der Daten zwischen Quell- und Zielobjekten getestet und ein mögliches Drag bestätigt, müssen Sie die Daten während des Ereignisses *On Drop* nicht erneut testen. Sie wissen bereits, dass die Daten für das Zielobjekt geeignet sind.

Ein interessanter Aspekt bei der Integration von Drag&Drop ist, dass 4D Ihnen freie Hand lässt. Beispiele:

- Beim Bewegen eines Items aus einer hierarchischen Liste bestimmen Sie selbst, ob er am Anfang, Ende und in der Mitte des Textfeldes eingefügt wird.
- Ihr Formular enthält eine Bildschaltfläche mit zwei unterschiedlichen Bildern, das kann z.B. ein voller oder ein leerer Papierkorb sein. Bewegen Sie ein Objekt in diese Schaltfläche, kann das aus der Sichtweise des Benutzers bedeuten "lösche das per Drag&Drop Technik in den Papierkorb gelegte Objekt." Hier transportiert Drag&Drop nicht Daten von einem Punkt zu einem anderen; sondern führt stattdessen eine Aktion aus.
- Bewegen Sie ein Array-Element von einem Palettenfenster in ein Formularobjekt, kann das bedeuten "Zeige in diesem Fenster den Kundendatensatz, dessen Namen Sie gerade per Drag&Drop Technik aus dem Palettenfenster entnommen haben, welches die in der Datenbank gespeicherten Kunden anzeigt."
- Usw.

Die Drag and Drop Oberfläche ist ein Komplex, mit dem Sie jede Metapher in der Benutzeroberfläche integrieren können.

Drag-and-Drop Befehle

Der Befehl **DRAG AND DROP PROPERTIES** gibt folgendes zurück:

- Einen Zeiger auf das bewegte Objekt (Datenfeld oder Variable)
- Die Element- oder Itemnummer, wenn das gezogene Objekt ein Array-Element oder Eintrag einer Liste ist.
- Die Prozessnummer des Quellprozesses

Die Funktion **Drop position** gibt die Elementnummer des Array oder die Position des Eintrags in der Liste zurück, wenn das Zielobjekt ein Array (z.B. rollbarer Bereich), eine hierarchische Liste, Text oder eine Combo Box ist, sowie die Spaltennummer, wenn das Objekt eine Listbox ist.

Mit **RESOLVE POINTER** und **Type** testen Sie Natur und Typ des Quellobjektes.

Soll die Drag-and-Drop Operation gezogene Daten kopieren, richtet sich die Funktionalität dieser Befehle nach der Anzahl der beteiligten Prozesse:

- Ist Drag-and-Drop auf einen Prozess beschränkt, führen Sie mit diesen Befehlen die geeigneten Aktionen aus, z.B. dem Zielobjekt das Quellobjekt zuweisen.
- Bei Drag-and-Drop auf Interprozessebene ist beim Zugreifen auf gezogene Daten Vorsicht geboten. Sie müssen vom Quellprozess aus auf die Daten zugreifen. Stammen die gezogenen Daten aus einer Variablen, verwenden Sie den Befehl **GET PROCESS VARIABLE**, um den richtigen Wert zu erhalten. Stammen die gezogenen Daten aus einem Datenfeld, beachten Sie, dass der aktuelle Datensatz für eine Tabelle für beide Prozesse unterschiedlich sein kann. Sie müssen auf den richtigen Datensatz zugreifen.

Für den letzten Fall empfehlen wir, die Befehle im Kapitel **Pasteboard** und das Ereignis On Begin Drag Over zu verwenden.

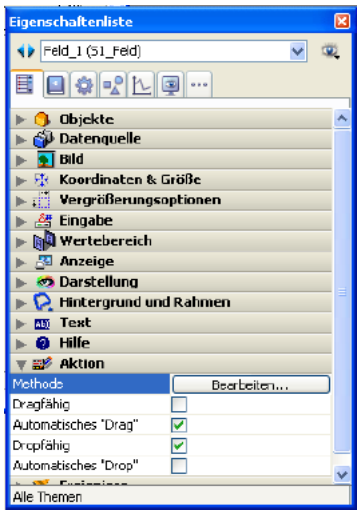
Soll die Drag-and-Drop Operation keine Daten bewegen, sondern ist sie vielmehr eine Metapher in der Benutzeroberfläche für eine spezifische Operation, können Sie ausführen, was immer Sie möchten.

"Pasteboard" Befehle verwenden

Verwenden Sie die Drag-and-Drop Technik, um heterogene Daten oder Dokumente zwischen zwei 4D Anwendungen oder einer 4D Anwendung und einer Anwendung von Drittherstellern zu übertragen, liefern die Befehle im Kapitel "Pasteboard" die erforderlichen Werkzeuge. Sie verwenden zwei Übertragungsbereiche (pasteboards) für die Daten: Einen für kopierte oder ausgeschnittene Daten (Zwischenablage); den anderen für Daten, die per Drag-and-Drop Technik bewegt werden (Pasteboard). Beide Übertragungsbereiche verwenden dieselben Befehle. Je nach Kontext greifen Sie auf die Zwischenablage oder Pasteboard zu.
Weitere Informationen dazu finden Sie im Abschnitt **Pasteboards verwalten**.

Automatisches Drag-and-Drop

Textbereiche (Felder, Variablen, Combo Boxen und Listboxen) sowie Objekte vom Typ Bild ermöglichen automatisches Drag-and-Drop, d.h. Verschieben oder Kopieren einer Text- oder Bildauswahl von einem Bereich in einen anderen durch einen Klick. Diese Operation lässt sich im gleichen 4D Bereich, zwischen zwei verschiedenen 4D Bereichen oder zwischen 4D und einer anderen Anwendung, z.B. WordPad verwenden.
Hinweis: Bei automatischem Drag-and-Drop zwischen zwei 4D Bereichen werden die Daten verschoben, d.h. sie werden aus dem Quellbereich entfernt. Wollen Sie die Daten nur kopieren, halten Sie während der Aktion unter Windows die **Strg-Taste**, auf OS X die **Wahltaste** gedrückt. Auf OS X müssen Sie erst klicken, dann die **Wahltaste** drücken und dann den Eintrag ziehen.
Automatisches Drag-and-Drop lässt sich für jedes Formularobjekt separat konfigurieren. Dazu stehen in der Eigenschaftenliste die beiden Optionen Automatisches Drag und Automatisches Drop zur Verfügung.



- **Automatisches Drag** (nur Objekte vom Typ Text): Ist diese Option markiert, wird der automatische Drag-Modus für das Objekt aktiviert. In diesem Modus wird das Formularereignis *On Begin Drag* nicht generiert.
  Wollen Sie das standardmäßige Drag erzwingen, wenn automatisches Drag aktiviert ist, drücken Sie während der Aktion unter Windows die **Alt-Taste**, auf OS X die **Wahltaste**. Auf OS X müssen Sie erst die **Wahltaste** drücken, dann klicken und dann den Eintrag ziehen. Diese Option ist für Bilder nicht verfügbar.
- **Automatisches Drop**: Diese Option aktiviert den automatischen Drop-Modus. In diesem Modus verwaltet 4D automatisch das Einfügen bewegter Daten vom Typ Text oder Bild, die auf das Objekt gezogen wurden (die Daten werden in das Objekt kopiert). In diesem Fall werden die Formularereignisse *On Drag Over* und *On Drop* nicht erzeugt. Die Ereignisse *On After Edit* (während einem Drop) und *On Data Change* (wenn ein Objekt Fokus verliert) werden dagegen erzeugt.
  Bei per Drop bewegten Daten, die nicht vom Typ Text oder Bild sind, also z.B. ein anderes 4D Objekt, eine andere Datei, oder bei komplexen Daten, richtet sich die Anwendung nach der Option Dropfähig: Ist sie markiert, werden die Formularereignisse *On Drag Over* und *On Drop* erzeugt. Ist sie nicht markiert, wird Drop verweigert. Dabei spielt auch die Option "Drag&Drop von außerhalb von 4D verweigern" eine Rolle.

Drag-and-Drop von außerhalb von 4D verweigern (Kompatibilität)

4D erlaubt ab Version 11 Drag-and-Drop für Auswahlen, Objekte bzw. externe Dateien in 4D, z.B. eine Bilddatei. Der Code der Datenbank muss diese Möglichkeit unterstützen.
Diese Möglichkeit kann jedoch die Funktionsweise für Datenbanken beeinträchtigen, die aus früheren 4D Versionen konvertiert wurden, wenn der vorhandene Code nicht entsprechend angepasst wird. Dafür gibt es in den Einstellungen der Datenbank auf der Seite Anwendung>Kompatibilität die Option **Drag-and-Drop von außerhalb von 4D verweigern**. Sie ist in konvertierten Datenbanken standardmäßig markiert.
Ist diese Option markiert, wird das Ziehen von externen Objekten per Drop-Technik in 4D Formulare verweigert. Beachten Sie, dass sich externe Objekte weiterhin in Objekte mit der Eigenschaft **Automatisches Drop** einfügen lassen, wenn die Anwendung diese Objekte interpretieren kann (Text oder Bild).

Datenbankmethode On Drop

Die **Datenbankmethode On Drop** ist für 4D Anwendungen im lokalen und im remote Modus verfügbar.

Diese Datenbankmethode wird automatisch ausgeführt, wenn Objekte per Drag-and-Drop in die 4D Anwendung außerhalb eines Formulars oder Fensters gesetzt werden, z.B.:

Aktion	Plattform	Kommentar
Drop in einen leeren Bereich eines MDI Fensters	Windows	Nicht verfügbar bei Ausführung im SDI Modus, da es in diesem Kontext kein MDI Fenster gibt (siehe Abschnitt **SDI Modus unter Windows**).
Drop auf das 4D Icon im Dock	macOS	
Drop auf das 4D Icon im Desktop des Systems	Windows(*) & macOS	**Datenbankmethode On Drop** wird nur aufgerufen, wenn die Anwendung bereits gestartet ist, außer bei Anwendungen mit integrierter 4D Volume Desktop. In diesem Fall wird die Datenbankmethode auch aufgerufen, wenn die Anwendung nicht gestartet ist. Auf diese Weise lassen sich eigene Dokument-Signaturen definieren.

(*) Wird mit 4D Developer 64-bit unter Windows nicht unterstützt, da diese Aktion eine neue Instanz der Anwendung startet (System Feature).

Auf macOS müssen Sie während der Drop Aktion die **Wahl-** und **Befehlstaste** gedrückt halten, damit die Datenbankmethode aufgerufen wird.

Beispiel

Dieses Beispiel zeigt, wie Sie ein 4D Write Dokument öffnen können, das außerhalb eines Formulars gesetzt wurde:

```
  `Datenbankmethode On Drop
droppedFile:=Get file from pasteboard(1)
If(Position(".4W7";droppedFile)=Length(droppedFile)-3)
   externalArea:=Open external window(100;100;500;500;0;droppedFile;"_4D Write")
   WR OPEN DOCUMENT(externalArea;droppedFile)
End if
```

DRAG AND DROP PROPERTIES

DRAG AND DROP PROPERTIES (Quellobjekt ; Quellelement ; Quellprozess)

Parameter	Typ		Beschreibung
Quellobjekt	Zeiger	⇐	Zeiger zum Bewegen des Quellobjekts per Drag-and-Drop
Quellelement	Lange Ganzzahl	⇐	Gezogene Elementnummer des Array oder gezogener Eintrag der hierarchischen Liste oder -1, wenn Quellobjekt weder Array, Liste noch hierarchische Liste ist
Quellprozess	Lange Ganzzahl	⇐	Nummer des Quellprozesses

Beschreibung

Hinweis zur Kompatibilität: Ab 4D Version 11 empfehlen wir, Drag-and-Drop Operationen, insbesondere auf Interprozessebene, über das Ereignis On Begin Drag Over und die Befehle im Kapitel **Pasteboard** zu verwalten.

Der Befehl **DRAG AND DROP PROPERTIES** fragt Informationen über das Quellobjekt ab, wenn für ein "komplexes" Objekt (Array, Listbox, hierarchische Liste) ein Ereignis On Drag Over oder On Drop eintritt.

Sie verwenden **DRAG AND DROP PROPERTIES** innerhalb der Objektmethode des Objekts bzw. innerhalb einer Unterroutine, die das Objekt aufruft, für die das Ereignis On Drag Over oder On Drop eintritt (das Zielobjekt).

Wichtig: Ein Formularobjekt akzeptiert bewegte Daten nur, wenn die Eigenschaft **dropfähig** zugewiesen ist. Außerdem muss die dazugehörige Objektmethode für On Drag Over und/oder On Drop aktiviert sein, damit diese Ereignisse durchgeführt werden.

Nach dem Aufruf gilt folgendes:

- Der Parameter *Quellobjekt* ist ein Zeiger auf das Quellobjekt (das per Drag&Drop bewegte Objekt). Beachten Sie, dass dieses Objekt das Zielobjekt sein kann (das Objekt, für welches das Ereignis On Drag Over oder On Drop eintritt) oder ein anderes Objekt. Werte per Drag-and-Drop aus und in dasselbe Objekt bewegen ist nützlich für Arrays und hierarchische Listen — ein einfacher Weg für den Benutzer, um ein Array oder eine Liste manuell zu sortieren.
- Ist der per Drag-and-Drop bewegte Wert ein Array-Element (das Quellobjekt ist ein Array), gibt der Parameter *Quellelement* die Nummer dieses Elements zurück. Ist der per Drag-and-Drop bewegte Wert eine Zeile einer Listbox, gibt der Parameter *Quellelement* die Nummer dieser Zeile zurück. Ist der per Drag-and-Drop bewegte Wert ein Listeneintrag (das Quellobjekt ist eine hierarchische Liste), gibt der Parameter *Quellelement* die Position dieses Eintrags zurück. Gehört das Quellelement dagegen zu keiner dieser Kategorien, hat *Quellelement* den Wert -1 (minus 1).
- Drag-and-Drop Operationen sind auch zwischen Prozessen möglich. Der Parameter *Quellprozess* ist gleich der Prozessnummer, zu der das Quellobjekt gehört. Sie müssen den Wert dieses Parameters testen. Sie können auf eine Drag-and-Drop Operation im selben Prozess durch einfaches Kopieren der Quelldaten in das Zielobjekt antworten. Bei einer Drag-and-Drop Operation auf Interprozessebene erhalten Sie die Quelldaten aus dem Objekt des Quellprozesses über den Befehl **GET PROCESS VARIABLE**. Ist das Quellobjekt ein Datenfeld, müssen Sie den Wert aus dem Quellprozess via Interprozesskommunikation holen oder in diesem speziellen Fall auf das Ereignis On Drag Over antworten (siehe nachfolgend). Im Normalfall integrieren Sie die Drag-and-Drop Benutzeroberfläche aus Quellvariablen, wie z.B. Arrays und Listen in Dateneingabebereiche (Datenfelder oder Variablen).

Rufen Sie **DRAG AND DROP PROPERTIES** auf, und es gibt kein Drag-and-Drop Ereignis, gibt *Quellobjekt* den Zeiger NIL zurück, *Quellelement* den Wert -1 und *Quellprozess* den Wert 0.

Tipp: 4D verwaltet automatisch die grafische Darstellung von Drag-and-Drop. Sie müssen dann in geeigneter Weise auf das Ereignis antworten. Wie Sie aus den folgenden Beispielen ersehen, ist eine Möglichkeit, die gezogenen Daten zu kopieren. Sie können alternativ dazu auch ausgeklügelte Benutzeroberflächen integrieren, wo z.B. ein per Drag-and-Drop bewegtes Array-Element aus einem Palettenfenster das Zielfenster (das Fenster, welches das Zielobjekt enthält) mit strukturierten Daten füllt (z.B. mehrere Datenfelder aus einem Datensatz, der nur von dem Element des Quell-Arrays erkannt wird).

Sie verwenden **DRAG AND DROP PROPERTIES** während einem Ereignis On Drag Over, um je nach Typ und Natur des Quellobjekts zu entscheiden, ob das Zielobjekt die Drag-and-Drop Operation akzeptiert. Bei Bestätigen von Drag-and-Drop gibt die Objektmethode *$0:=0* zurück. Bei Annullieren von Drag-and-Drop gibt die Objektmethode *$0:=-1* zurück. Dieser Vorgang wird auf dem Bildschirm grafisch dargestellt: Bei Bestätigen wird das potentielle Ziel der Drag-and-Drop Operation markiert.

Beispiel 1

Verschiedene Formulare Ihrer Datenbank enthalten rollbare Bereiche, die Sie per Drag-and-Drop von einem Teil des rollbaren Bereichs in einen anderen Teil darin manuell umsortieren wollen. Anstatt für jeden Fall einen spezifischen Code zu schreiben, können Sie eine generische Projektmethode integrieren, die all diese rollbaren Bereiche verwaltet:

```
` Verwalte Projektmethode für Drag-and-Drop im gleichen Array
` Verwalte Drag-and-Drop im gleichen Array ( Zeiger ) -> Boolean
` Verwalte Drag-and-Drop im gleichen Array ( -> Array ) -> War Drag-and-Drop im gleichen Array

Case of
   :(Form event=On Drag Over)
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
      If($vpSrcObj=$1)
` Bestätige Drag-and-Drop, wenn aus dem gleichen Array
         $0:=0
      Else
         $0:=-1
      End if
   :(Form event=On Drop)
```

```
   ` Hole Information über das Quellobjekt für Drag-and-Drop
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
   ` Hole Nummer des Zielelements
      $vlDstElem:=Drop position
   ` Wurde das Element nicht auf sich selbst bewegt
      If($vlDstElem#$vlSrcElem)
   ` Sichere bewegtes Element in Element 0 des Array
         $1->{0}:=$1->{$vlSrcElem}
   ` Lösche das bewegte Element
         DELETE FROM ARRAY($1->;$vlSrcElem)
   ` War das Zielelement außerhalb des bewegten Elements
         If($vlDstElem>$vlSrcElem)
   ` Setze Nummer des Zielelements zurück
            $vlDstElem:=$vlDstElem-1
         End if
   ` Geschah Drag-and-Drop außerhalb des letzten Elements
         If($vlDstElem=-1)
   ` Setze Nummer des Zielelements auf ein neues Element am Ende des Array
            $vlDstElem:=Size of array($1->)+1
         End if
   ` Füge dieses neue Element ein
         INSERT IN ARRAY($1->;$vlDstElem)
   ` Setze seinen zuvor im Element Null des Array gesicherten Wert
         $1->{$vlDstElem}:=$1->{0}
   ` Das Element wird das neu ausgewählte Element des Array
         $1->:=$vlDstElem
      End if
End case
```

Diese Projektmethode können Sie folgendermaßen einsetzen:

```
   ` Objektmethode für rollbaren Bereich in Array

Case of
   `...
   :(Form event=On Drag Over)
      $0:=Handle self array drag and drop(Self)
   :(Form event=On Drop)
      Handle self array drag and drop(Self)
   `...
End case
```

Beispiel 2

Verschiedene Formulare Ihrer Datenbank enthalten eingebbare Textbereiche, in die Sie per Drag-and-Drop Daten aus unterschiedlichen Quellen bewegen wollen. Anstatt für jeden Fall einen spezifischen Code zu schreiben, können Sie eine generische Projektmethode integrieren, die all diese eingebbaren Textbereiche verwaltet:

```
   ` Verwalte Projektmethode für Bewegen in Textbereich
   ` Verwalte Bewegen in Textbereich ( Zeiger )
   ` Verwalte Bewegen in Textbereich ( -> Text oder String Variable )

Case of
   ` Verwende dieses Ereignis, um Drag-and-Drop zu bestätigen oder abzuweisen
   :(Form event=On Drag Over)
   ` Initialisiere $0 für Abweisen
      $0:=-1
   ` Hole Information über Quellobjekt für Drag-and-Drop
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
   ` In diesem Beispiel ist Drag-and-Drop im gleichen Objekt nicht erlaubt
      If($vpSrcObj#$1)
   ` Hole Typ der gezogenen Daten
         $vlSrcType:=Type($vpSrcObj->)
         Case of
            :($vlSrcType=ls text)
   ` OK für Textvariablen
               $0:=0
            :($vlSrcType=ls string var)
   ` String Variable ist OK
               $0:=0
```

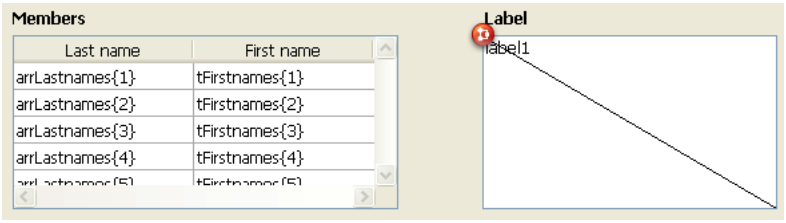
```
            :(($vlSrcType=String array)|($vlSrcType=Text array))
   ` String und Text Arrays sind OK
               $0:=0
            :(($vlSrcType=ls longint)|($vlSrcType=ls real)
               If(ls a list($vpSrcObj->))
   ` Hierarchische Liste ist OK
                  $0:=0
               End if
         End case
      End if

   ` Zeigen Sie mit diesem Ereignis die aktuelle Drag&Drop Aktion an
     :(Form event=On Drop)
      $vtDraggedData:=""
   ` Hole Information über Quellobjekt für Drag&Drop
      DRAG AND DROP PROPERTIES($vpSrcObj;$vlSrcElem;$vlPID)
   ` Hole Typ der gezogenen Variablen
      $vlSrcType:=Type($vpSrcObj->)
      Case of
   ` Ist es ein Array
         :(($vlSrcType=String array)|($vlSrcType=Text array))
            If($vlPID#Current process)
   ` Lies Element aus der Quellprozessinstanz der Variablen
               GET PROCESS VARIABLE($vlPID;$vpSrcObj->{$vlSrcElem};$vtDraggedData)
            Else
   ` Kopiere das Array-Element
               $vtDraggedData:=$vpSrcObj->{$vlSrcElem}
            End if
   ` Ist es eine Liste
         :(($vlSrcType=ls real)|($vlSrcType=ls longint)
   ` Ist es eine Liste aus einem anderen Prozess
            If($vlPID#Current process)
   ` Hole Listenreferenz aus dem anderen Prozess
               GET PROCESS VARIABLE($vlPID;$vpSrcObj->;$vlList)
            Else
               $vlList:=$vpSrcObj->
            End if
   ` Existiert die Liste
            If(ls a list($vpSrcObj->)
   ` Hole Text aus dem Eintrag der entsprechenden Position
               GET LIST ITEM($vlList;$vlSrcElem;$vlItemRef;$vsItemText)
               $vtDraggedData:=$vsItemText
            End if
         Else
   ` Es ist eine String oder Text Variable
            If($vlPID#Current process)
               GET PROCESS VARIABLE($vlPID;$vpSrcObj->;$vtDraggedData)
            Else
               $vtDraggedData:=$vpSrcObj->
            End if
      End case
   ` Gibt es gerade etwas zu Bewegen (das Quellobjekt kann leer sein)
      If($vtDraggedData#"")
         $1->:=$1->+$vtDraggedData
      End if
End case
```

Diese Projektmethode können Sie folgendermaßen einsetzen:

```
   ` Objektmethode [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
```

Beispiel 3

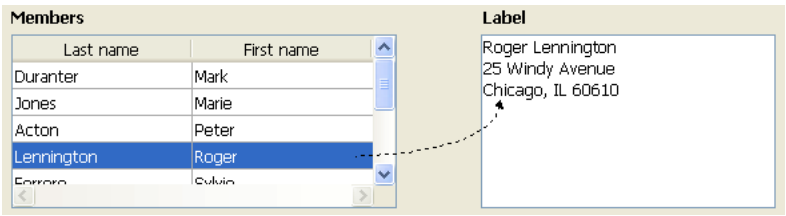
Wir wollen einen Textbereich, z.B. ein Etikett, mit Daten füllen, die per Drag-and-Drop aus einer Listbox entnommen werden.



Die Objektmethode label1 lautet:

```
Case of
   :(Form event=On Drag Over)
      DRAG AND DROP PROPERTIES($source;$arrayrow;$processnum)
      If($source=Get pointer("list box1"))
         $0:=0  ` Drop ist angenommen
      Else
         $0:=-1
      End if
   :(Form event=On Drop)
      DRAG AND DROP PROPERTIES($source;$arrayrow;$processnum)
      QUERY([Members];[Members]LastName=arrNames{$arrayrow})
      If(Records in selection([Members])#0)
         label1:=[Members]FirstName+" "+[Members]LastName+Char(Carriage return)+[Members]Address+
         Char(Carriage return)+[Members]City+","+" "+[Members]State+" "+[Members]ZipCode
      End if
End case
```

Nun lässt sich folgende Aktion ausführen



Drop position

Drop position {(SpaltenNr | BildPosY)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
SpaltenNr | BildPosY	Lange Ganzzahl	⇐	Spaltennummer der Listbox (-1 wenn Drop außerhalb der letzten Spalte erfolgt) oder Position der Y Koordinate im Bild
Funktionsergebnis	Lange Ganzzahl	⇒	Zahl (Array/Listbox) oder • Position (hierarchische Liste) oder • Position in String (Text/Combobox) des Zieleintrags oder • -1, wenn Drop außerhalb des letzten Array Elements oder Listeneintrags erfolgt oder • Position der X Koordinate im Bild

Beschreibung

Die Funktion **Drop position** findet in einem "komplexen" Zielobjekt die Stelle, an die ein Objekt per Drag-and-Drop bewegt wurde.

Sie verwenden **Drop position** bei einem Drag-and-Drop Ereignis, das in einem Array, einer Listbox, einer hierarchischen Liste oder einem Feld vom Typ Text oder Bild auftritt.

- Ist das Zielobjekt ein Array, gibt die Funktion eine Elementnummer zurück.
- Ist das Zielobjekt eine Listbox, gibt die Funktion eine Zeilennummer zurück. In diesem Fall gibt die Funktion im optionalen Parameter *SpaltenNr* die Nummer zurück, wo Drop erfolgt ist.
- Ist das Zielobjekt eine hierarchische Liste, gibt die Funktion die Position des Eintrags zurück.
- Ist das Zielobjekt ein Text vom Typ Variable oder Feld, oder eine Combobox, gibt die Funktion die Position des Zeichens innerhalb des Strings zurück.
  Die Funktion gibt in all diesen Fällen -1 zurück, wenn das Quellobjekt außerhalb des letzten Elements bzw. Eintrags des Zielobjekts bewegt wurde.
- Ist das Zielobjekt ein Bild vom Typ Variable oder Feld, gibt die Funktion die vertikale und über den optionalen Parameter *BildPosY* die horizontale Position des Klicks zurück. Die zurückgegebenen Werte werden in Pixel und in Relation zum lokalen Koordinatensystem ausgedrückt.

Rufen Sie **Drop position** bei einem Ereignis in einem Array, einer Listbox, einer Combobox, einer hierarchischen Liste oder einem Text auf, das kein Drag-and-Drop Ereignis ist, gibt die Funktion ebenfalls -1 zurück.

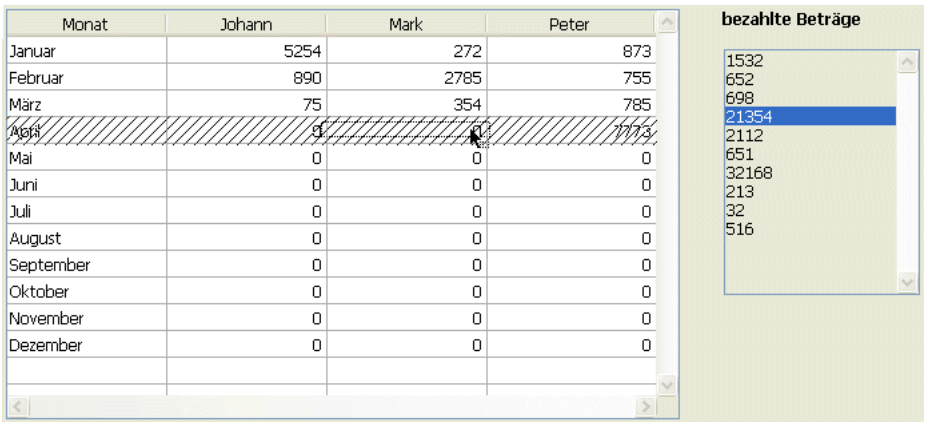
Wichtig: Ein Formularobjekt nimmt bewegte Daten nur an, wenn die Eigenschaft **dropfähig** zugewiesen ist. Außerdem muss die dazugehörige Objektmethode für On Drag Over und/oder On Drop aktiviert sein, damit diese Ereignisse durchgeführt werden.

Beispiel 1

Siehe Beispiele für den Befehl **DRAG AND DROP PROPERTIES**.

Beispiel 2

Im folgenden Beispiel soll eine Liste bezahlter Beträge nach Monat und Person aufgeteilt werden. Das wird per Drag-and-Drop aus einem rollbaren Bereich ausgeführt.



Die Objektmethode der Listbox enthält folgenden Code:

```
Case of
   :(Form event=On Drag Over)
      DRAG AND DROP PROPERTIES($source;$arrayrow;$processnum)
      If($source=Get pointer("SA1"))
`Wenn Drop vom rollbaren Bereich kommt
         $0:=0
      Else
         $0:=-1  `Drop ist abgelehnt
      End if
   :(Form event=On Drop)
```

```
        DRAG AND DROP PROPERTIES($source;$arrayrow;$processnum)
        $rownum:=Drop position($colnum)
        If($colnum=1)
            BEEP
        Else
            Case of  `Gezogene Werte hinzufügen
                    $colnum=2
                    arrJohann{$rownum}:=arrJohann{$rownum}+SA1{$arrayrow}
                    $colnum=3
                    arrMark{$rownum}:=arrMark{$rownum}+SA1{$arrayrow}
                    $colnum=4
                    arrPeter{$rownum}:=arrPeter{$rownum}+SA1{$arrayrow}
            End case
            DELETE FROM ARRAY(SA1;$arrayrow)  `Bereich aktualisieren
        End if
    End case
```

SET DRAG ICON

SET DRAG ICON (Icon {; horOffset {; vertOffset}})

Parameter	Typ		Beschreibung
Icon	Bild	➡	Icon, das während Drag verwendet wird
horOffset	Lange Ganzzahl	➡	Horizontaler Versatz vom linken Bildrand in Bezug auf den Mauszeiger (>0 = nach links, <0 = nach rechts)
vertOffset	Lange Ganzzahl	➡	Vertikaler Versatz vom oberen Bildrand in Bezug auf den Mauszeiger (>0 = nach oben, <0 = nach unten)

Beschreibung

Der Befehl **SET DRAG ICON**setzt das Bild *Icon* während Drag-and-Drop Operationen, die per Programmierung verwaltet werden, unter den Mauszeiger.

Dieser Befehl lässt sich nur für das Formularereignis On Begin Drag Over aufrufen (siehe Befehl **Form event**).

Im Parameter *Icon* übergeben Sie das entsprechende Bild. Seine maximale Größe ist 256x256 Pixel. Höhere Werte werden automatisch verkleinert.

In *horOffset* und *vertOffset* können Sie für den Versatz Werte in Pixel angeben:

- In *horOffset*, übergeben Sie den horizontalen Versatz vom linken Rand von Icon in Bezug auf den Mauszeiger. Setzen Sie einen positiven Wert für den Versatz nach links, einen negativen Wert für den Versatz nach rechts.
- In *vertOffset* übergeben Sie den vertikalen Versatz vom oberen Rand von Icon in Bezug auf den Mauszeiger. Setzen Sie einen positiven Wert für den Versatz nach oben, einen negativen Wert für den Versatz nach unten.

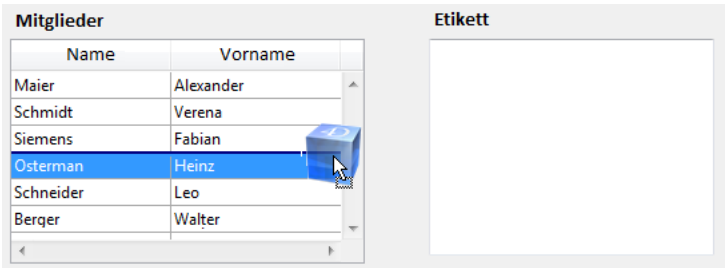
Lassen Sie diesen Parameter weg, wird das Bild Icon in die Mitte des Mauszeigers gesetzt.

Beispiel

Der Benutzer kann in einem Formular ein Etikett durch Ziehen einer Zeile per Drag-and-Drop erstellen. Dazu schreiben Sie in der Objektmethode der Listbox folgende Anweisung:

```
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
```

Ziehen Sie eine Zeile, erscheint das Bild wie folgt:



Beachten Sie, dass Sie den Mauszeiger in Bezug auf das Bild verändern können:

```
SET DRAG ICON(vpict;0;0)
```

Drucken

- Integration des Treibers PDF Creator unter 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

Integration des Treibers PDF Creator unter Windows

Die Unterstützung von Drucken als PDF ist je nach Windows Version unterschiedlich:

- für Windows 8 und frühere Versionen müssen Sie den Treiber PDFCreator verwenden.
- ab Windows 10 ist ein native Microsoft Treiber integriert.

Hinweis: Auf Mac OS wird das Drucken als PDF nativ vom System unterstützt.

Windows 8 und frühere Versionen

Das Drucken als PDF unter Windows wird über den Treiber PDFCreator unterstützt, um einfache und funktionale Druckfunktionen für PDF zu bieten. Die beiden Befehle **GET PRINT OPTION** und **SET PRINT OPTION** verwenden diesen Treiber.

PDFCreator ist ein freier Treiber (OpenSource), der AFPL (Aladdin Free Public License) lizenziert ist. Um den Treiber PDFCreator nutzen zu können, müssen Sie die passende Version laden und in Ihrer Umgebung installieren. Er wird nicht standardmäßig von 4D installiert und Sie benötigen Zugriffsrechte als Administrator. Sie können ihn laden unter http://sourceforge.net/projects/pdfcreator/files/PDFCreator/PDFCreator%200.9.9

Hinweis: Sie müssen eine mit 4D kompatible Version von PDFCreator verwenden. Weitere Informationen dazu finden Sie in der jeweiligen Zertifizierungsmatrix der 4D Produkte auf der 4D Web Site unter **Ressourcen** (**Zertifizierung**).

Beim Installieren wird ein neuer virtueller Drucker mit Standardnamen "PDFCreator" in Ihrem System installiert. Sie können diesen Namen bei Bedarf verändern.

Ab Windows 10

Windows 10 enthält einen native PDF Treiber, über den 4D direkt PDF Dokumente erstellen kann. Es ist kein third-party Treiber wie PDFCreator erforderlich.

Der Treibername ist "Microsoft Print to PDF".

Dieses Beispiel erstellt mit 4D Druckbefehlen ein PDF Dokument unter Windows 10:

```
$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 (Objekt {; Objekt2 ; ... ; ObjektN})

Parameter	Typ		Beschreibung
Objekt	Feld, Variable	➡	Zu berechnendes Datenfeld oder Variable

Beschreibung

Der Befehl **ACCUMULATE** muss aufgerufen werden, bevor Sie mit **PRINT SELECTION** eine Auswahl mit Zwischensummen ausdrucken.

Warnung: Sie **müssen** vor jedem Bericht mit Umbrüchen die Befehle **BREAK LEVEL** und **ACCUMULATE** ausführen, da diese die Umbruchberechnung aktivieren. Weitere Informationen dazu finden Sie unter der Funktion **Subtotal**.

ACCUMULATE weist 4D an, für jeden Parameter *Objekt* Zwischensummen zu speichern. Diese werden für jede mit dem Befehl **BREAK LEVEL** angegebene Umbruchebene berechnet. Verwenden Sie die Funktion **Subtotal** in der Formular- oder Objektmethode für die Zwischensumme eines Arguments *Objekt*.

Führen Sie **ACCUMULATE** zur Berechnung der Umbrüche aus, bevor der Bericht mit **PRINT SELECTION** gedruckt wird.

Verwenden Sie die Funktion **PRINT SELECTION** in der Formular- bzw. in der Objektmethode, um die Zwischensumme eines Parameters *Objekt* zurückzugeben.

Beispiel

Siehe Beispiel zum Befehl **BREAK LEVEL**.

BLOB to print settings

BLOB to print settings (Druckeinstellungen {; Parameter}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Druckeinstellungen	BLOB	➡	BLOB mit Druckeinstellungen
Parameter	Lange Ganzzahl	➡	0=Gesicherte Werte für Anzahl Kopien und Seitenbereich, 1=Auf Standardwerte zurücksetzen
Funktionsergebnis	Lange Ganzzahl	⬅	Status Code: 1=Operation erfolgreich, 0=kein aktueller Drucker, -1=nicht korrekte Parameter, 2=Geänderter Drucker

Beschreibung

Die Funktion **BLOB to print settings** ersetzt die aktuellen Druckeinstellungen in 4D durch die Parameter im BLOB *Druckeinstellungen*. Dieses BLOB muss zuvor über die Funktionen **Print settings to BLOB** oder von 4D Pack erstellt werden (siehe unten).

In *Parameter* können Sie definieren, wie die Grundeinstellungen "Anzahl Kopien" und "Seiten" behandelt werden:

- Übergeben Sie 0 oder lassen diesen Parameter weg, werden die im BLOB gespeicherten Werte wiederhergestellt
- Übergeben Sie 1, werden die Werte auf den Standard zurückgesetzt: Die Anzahl Kopien wird auf 1 gesetzt und Seitenbereich auf "alle Seiten".

Die Druckeinstellungen gelten für den aktuellen Drucker und die Dauer der Sitzung, solange sie nicht von Befehlen wie **PAGE SETUP**, **SET PRINT OPTION** oder **PRINT SELECTION** ohne den Parameter > geändert werden. Die definierten Parameter werden insbesondere von den Befehlen **PRINT SELECTION**, **PRINT LABEL**, **PRINT RECORD**, **Print form** und **QR REPORT** verwendet, sowie von den Menübefehlen in 4D - dazu gehören auch die der Designumgebung.

Die Befehle **PRINT SELECTION**, **PRINT LABEL** und **PRINT RECORD** müssen mit dem Parameter > (falls zutreffend) aufgerufen werden, um die von **BLOB to print settings** definierten Einstellungen beizubehalten.

Die Funktion gibt einen der folgenden Status Codes zurück:

- -1: das BLOB ist nicht korrekt
- 0: kein aktueller Drucker ausgewählt (in diesem Fall führt die Funktion nichts aus),
- 1: das BLOB wurde korrekt geladen
- 2: das BLOB wurde korrekt geladen, aber der Name des aktuellen Druckers hat sich geändert (*).
  Hinweis: Code (2) wird immer zurückgegeben, wenn das BLOB von der 4D Pack Funktion erstellt wurde, selbst wenn der Druckername sich aktuell nicht geändert hat, da diese Information in den 4D Pack BLOBs nicht enthalten ist.

(*) Die Einstellung richtet sich nach dem aktuell ausgewählten Drucker im Moment, wo das BLOB gesichert wird. Die Anwendung dieser Einstellungen auf einen anderen Drucker wird unterstützt, wenn beide Drucker vom gleichem Modell sind. Bei unterschiedlichen Druckern werden nur allgemeine Parameter wiederhergestellt.

Windows / OS X

Das BLOB *Druckeinstellungen* lässt sich auf beiden Plattformen sichern und einlesen. Es gibt allgemeingültige Druckeinstellungen, aber auch plattformspezifische, die sich nach dem Treiber und dem Betriebssystem richten. Nutzen beide Plattformen das gleiche BLOB *Druckeinstellungen*, können Teilinformationen verlorengehen.
Wir empfehlen, bei Verwendung in einer heterogenen Umgebung pro Plattform ein BLOB *Druckeinstellungen* einzurichten, damit für jede Plattform nicht nur die allgemeinen, sondern die maximalen Einstellungen verfügbar sind.

Kompatibilität mit 4D Pack Funktionen

BLOBs mit Druckeinstellungen, die mit der bisherigen Funktion von 4D Pack erstellt wurden, lassen sich laden und von der Funktion **BLOB to print settings** verwenden. Beachten Sie jedoch, dass sie beim Sichern mit **Print settings to BLOB** konvertiert werden und sich dann nicht mehr mit öffnen lassen. Die Funktion **BLOB to print settings** speichert weit mehr Druckerdaten als .

Beispiel

Die zuvor auf die Festplatte gesicherten Druckeinstellungen auf den aktuellen 4D Druckauftrag anwenden:

```
C_BLOB(curSettings)
DOCUMENT TO BLOB(Get 4D folder(Active 4D Folder)+"current4Dsettings.blob";curSettings)
 //current4Dsettings wurde von Druckeinstellungen zu BLOB erstellt
$err:=BLOB to print settings(curSettings;0)
Case of
   :($err=1)
      /alles ist OK
   :($err=2)
      CONFIRM("Drucker hat sich geändert! Druckeinstellungen überprüfen?")
      If(OK=1)
         PRINT SETTINGS
      End if
   :($err=0)
      ALERT("Es gibt keinen aktuellen Drucker.")
   :($err=-1)
```

```
        ALERT("ungültige Datei Einstellungen.")
    End case
```

BREAK LEVEL

BREAK LEVEL (Ebene {; Seitenumbruch})

Parameter	Typ		Beschreibung
Ebene	Lange Ganzzahl	➡	Anzahl der Umbrüche
Seitenumbruch	Lange Ganzzahl	➡	Ebene, auf der ein Seitenumbruch erfolgen soll

Beschreibung

Der Befehl **BREAK LEVEL** legt die Anzahl der Umbrüche für den Bericht fest, der mit dem Befehl **PRINT SELECTION** gedruckt wird.

Warnung: Sie müssen die Befehle **BREAK LEVEL** und **ACCUMULATE** vor jedem Bericht ausführen, der Umbrüche enthalten soll. Diese Befehle aktivieren den Seitenumbruch für einen Bericht. Weitere Informationen dazu finden Sie in der Beschreibung zur Funktion **Subtotal**.

Der Parameter *Ebene* gibt die Höchstzahl der Ebenen an, für die ein Umbruch ausgeführt werden soll. Die Anzahl der Ebenen muss kleiner oder gleich der Anzahl der Sortierebenen sein, die Sie im zuvor ausgeführten Sortierbefehl angegeben haben. Enthält *Ebene* mehr Umbruchebenen als Sortierebenen vorhanden sind, werden diese nicht berücksichtigt.

Jede erzeugte Umbruchebene druckt die entsprechenden Umbruch- und Kopfbereiche des Formulars.

Mit dem optionalen Parameter *Seitenumbruch* erzeugen Sie während dem Drucken einen Seitenumbruch, d.h., eine neue Seite wird begonnen.

Beispiel

Folgendes Beispiel druckt einen Bericht mit zwei Umbruchebenen. Die Auswahl wird auf vier Ebenen sortiert, mit dem Befehl **BREAK LEVEL** soll jedoch nur auf zwei Ebenen ein Umbruch erfolgen. Ein Datenfeld wird mit **ACCUMULATE** berechnet:

```
ORDER BY([Emp]Dept;>;[Emp]Title;>;[Emp]Last;>;[Emp]First;>)
  ` Sortiere auf vier Ebenen
BREAK LEVEL(2) ` Aktiviere Umbruch für 2 Ebenen (Dept und Title)
ACCUMULATE([Emp]Salary) ` Berechne die Gehälter
FORM SET OUTPUT([Emp];"Dept salary") ` Wähle die Berichtvorlage
PRINT SELECTION([Emp]) ` Drucke den Bericht
```

CLOSE PRINTING JOB

CLOSE PRINTING JOB
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **CLOSE PRINTING JOB** schließt den zuvor mit dem Befehl **OPEN PRINTING JOB** geöffneten Druckauftrag und sendet alle gesammelten Druckdokumente an den aktuellen Drucker.
Nach Ausführen dieses Befehls ist der Drucker wieder für andere Druckaufträge verfügbar.

Get current printer

Get current printer -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	String		Name des aktuellen Druckers

Beschreibung

Die Funktion **Get current printer** gibt den Namen des aktuellen Druckers zurück, der in der 4D Anwendung definiert ist. Das ist standardmäßig der Drucker, der beim Starten von 4D im System festgelegt ist.

Wird der aktuelle Drucker über einen Drucker Server (Spooler) verwaltet, wird unter Windows der vollständige Zugriffspfad, auf Mac OS der Name des Spoolers zurückgegeben.

Über den Befehl **PRINTERS LIST** erhalten Sie die Liste der verfügbaren Drucker sowie zusätzliche Informationen. Über den Befehl **SET CURRENT PRINTER** können Sie den aktuellen Drucker verändern.

Hinweis: Wird die Konstante Generic PDF driver mit **SET CURRENT PRINTER** verwendet, gibt **Get current printer** "_4d_pdf_printer" oder, falls zutreffend, den aktuellen Namen des PDF Treibers zurück (Option ist mit 4D 32-bit nicht verfügbar).

Systemvariablen und Mengen

Ist kein Drucker installiert, wird die Systemvariable OK auf 0 (Null) gesetzt, sonst auf 1.

Get print marker

Get print marker (MarkeNum) -> Funktionsergebnis

Parameter	Typ		Beschreibung
MarkeNum	Lange Ganzzahl	⇒	Nummer der Marke
Funktionsergebnis	Lange Ganzzahl	⇐	Position der Marke

Beschreibung

Die Funktion **Get print marker** gibt die aktuelle Position der Marke während des Druckvorgangs zurück.
Die Funktion lässt sich folgendermaßen einsetzen:

- Während des Formularereignisses *On header* im Kontext der Befehle **PRINT SELECTION** und **PRINT RECORD**
- Während des Formularereignisses *On Printing Detail* im Kontext der Funktion **Print form**.

Die Koordinaten werden in Pixel zurückgegeben (1 pixel = 1/72 Zoll, 1 Zoll = 2,54 cm).
Im Parameter *MarkeNum* setzen Sie eine Konstante aus dem Thema **Formularbereich** ein:

Konstante	Typ	Wert
Form break0	Lange Ganzzahl	300
Form break1	Lange Ganzzahl	301
Form break2	Lange Ganzzahl	302
Form break3	Lange Ganzzahl	303
Form break4	Lange Ganzzahl	304
Form break5	Lange Ganzzahl	305
Form break6	Lange Ganzzahl	306
Form break7	Lange Ganzzahl	307
Form break8	Lange Ganzzahl	308
Form break9	Lange Ganzzahl	309
Form detail	Lange Ganzzahl	0
Form footer	Lange Ganzzahl	100
Form header	Lange Ganzzahl	200
Form header1	Lange Ganzzahl	201
Form header10	Lange Ganzzahl	210
Form header2	Lange Ganzzahl	202
Form header3	Lange Ganzzahl	203
Form header4	Lange Ganzzahl	204
Form header5	Lange Ganzzahl	205
Form header6	Lange Ganzzahl	206
Form header7	Lange Ganzzahl	207
Form header8	Lange Ganzzahl	208
Form header9	Lange Ganzzahl	209

Beispiel

Siehe Beispiel unter dem Befehl **SET PRINT MARKER**.

GET PRINT OPTION

GET PRINT OPTION (Option ; Wert1 {; Wert2})

Parameter	Typ		Beschreibung
Option	Lange Ganzzahl	➡	Optionsnummer oder PDF Optionscode
Wert1	Lange Ganzzahl, Text	⬅	Wert 1 der Option
Wert2	Lange Ganzzahl, Text	⬅	Wert 2 der Option

Beschreibung

Der Befehl **GET PRINT OPTION** gibt den aktuellen Wert von *Option* zurück.

Der Parameter *Option* erlaubt, die zu erhaltende Option anzugeben. Sie können entweder eine Standardoption (Lange Ganzzahl) oder einen PDF Optionscode (String) erhalten. Der Befehl gibt in *Wert1* und optional in *Wert2* die aktuellen Werte der definierten Option zurück.

Für Standardoptionen zum Drucken können Sie eine vordefinierte Konstante unter dem Thema **Druckoptionen** übergeben:

Konstante	Typ	Wert	Kommentar
Paper option	Lange Ganzzahl	1	Verwenden Sie nur *Wert1*, enthält er den Namen des Papierformats. Verwenden Sie beide Parameter, enthält *Wert1* die Papierbreite und *Wert2* die Papierhöhe. Breite und Höhe werden in Pixel auf dem Bildschirm angegeben. Über den Befehl **PRINT OPTION VALUES** erhalten Sie Name, Höhe und Breite aller Papierformate, die der Drucker anbietet.
Orientation option	Lange Ganzzahl	2	Nur *Wert1*: 1=Hochformat, 2=Querformat. Bei einer anderen Ausrichtung gibt **GET PRINT OPTION** 0 in *Wert1* zurück. **64-bit Versionen**: Diese Option lässt sich im Druckauftrag aufrufen, d.h. Sie können im gleichen Auftrag zwischen Hoch- und Querformat wechseln.
Scale option	Lange Ganzzahl	3	nur *Wert1*: Skalierungswert in Prozent. Bedenken Sie jedoch, dass einige Drucker keine Skalierung zulassen. Übergeben Sie einen ungültigen Wert, wird die Eigenschaft beim Drucken auf 100% gesetzt.
Number of copies option	Lange Ganzzahl	4	nur *Wert1*: Anzahl der Kopien zum Drucken
Paper source option	Lange Ganzzahl	5	(nur Windows) nur *Wert1*: Nummer, die dem Index des zu verwendenden Papierschachts entspricht. Über den Befehl **PRINT OPTION VALUES** erhalten Sie das Array mit den Namen. Diese Option ist nur unter Windows verwendbar.
Color option	Lange Ganzzahl	8	(nur Windows) nur *Wert1*: Code zum Verwalten der Farbe: 1=schwarz/weiß (monochrome), 2=Farbe. **64-bit Versionen**: Diese Option wird in 4D 64-bit Versionen nicht unterstützt (überholt)
Destination option	Lange Ganzzahl	9	*Wert1*: Code für Druckausgabe: 1=Drucker, 2=(PC)/PS File (Mac), 3=PDF Datei, 5=Bildschirm (OS X Treiber). Ist *Wert1* ungleich 1 oder 5, enthält *Wert2* den Pfadnamen des Ergebnisdokuments. Dieser Pfad wird benützt, bis ein anderer Pfad angegeben wird. Gibt es bereits eine gleichnamige Datei am Zielort, wird sie ersetzt. Ist der aktuelle Wert nicht in der vordefinierten Liste, enthält *Wert1* mit **GET PRINT OPTION** -1 und die Systemvariable OK wird auf 1 gesetzt. Tritt ein Fehler auf, werden *Wert1* und die Systemvariable OK auf 0 gesetzt. **Hinweis**: Unter Windows können Sie das Druckziel auf 3 (PDF-Datei) setzen, wenn der Treiber PDF Creator installiert ist. Werden die Werte (9;3;Pfad) übergeben, startet 4D automatisch ein "stilles" PDF Drucken, das alle übergebenen Codes für Option berücksichtigt. Übergeben Sie in *Wert2* einen leeren String oder lassen diesen Parameter weg, erscheint beim Drucken ein Sichern-Dialog. Nach dem Drucken wird wieder auf die aktuellen Einstellungen zurückgesetzt. Das vereinfacht das Steuern von Drucken als PDF für 4D und ermöglicht das Schreiben von Multiplattform Code. Sind die Werte (9;3;Pfad) nicht übergeben, wird das Drucken nicht durch 4D gesteuert und alle für PDF Creator übergebenen Optionen werden ignoriert.
Double sided option	Lange Ganzzahl	11	(nur Windows) *Wert1*: 0=Einseitig oder Standard), 1=Doppelseitig. Ist *Wert1*=1, enthält *Wert2* die Bindung: 0=Bindung links (Standard), 1=Bindung oben. **Hinweis:** Diese Option ist nur unter Windows verwendbar.
Spooler document name option	Lange Ganzzahl	12	nur *Wert1*: Namen des aktuellen Druckdokuments, das in der Dokumentenliste des Druck-Servers erscheint. Der hier definierte Name wird für alle Druckdokumente der Sitzung verwendet, solange kein neuer Name oder ein leerer String übergeben wird. Um die Standardoperation wiederherzustellen (Methodenname statt Methode, Tabellenname für Datensatz, etc.), übergeben Sie in *Wert1* einen leeren String.
Mac spool file format option	Lange Ganzzahl	13	(nur Mac) nur *Wert1*: 0= Druckauftrag im PDF-Modus (Standardwert) 1=Druckauftrag im PostScript Modus **Hinweise:** - Diese Option hat keine Auswirkung unter Windows. - Auf Mac OS X wird standardmäßig im PDF-Modus gedruckt. Der PDF-Treiber unterstützt jedoch keine PICT-Bilder mit eingebundener PostScript Information – diese Bilder werden über vektororientierte Programme erstellt. Um dieses Problem zu vermeiden, können Sie über diese Option den Druckmodus für die aktuelle Sitzung auf Mac OS X verändern. Bedenken Sie, dass das Drucken im PostScript Modus unerwünschte Nebenwirkungen haben kann. **64-bit Versionen:** Diese Option wird nicht unterstützt; sie wird ersetzt durch die Option Generic PDF driver des Befehls **SET CURRENT PRINTER**.
Hide printing progress option	Lange Ganzzahl	14	(nur Mac) Nur *Wert1*: 1=Fenster mit dem Druckverlauf ausblenden, 0=Fenster mit dem Druckverlauf anzeigen (Standard). Diese Option ist besonders hilfreich beim Drucken von PDF auf OS X im Hintergrund. **Hinweis**: Es gibt bereits eine Option Druckverlauf im Dialogfenster Einstellungen (Seite **Anwendung/Optionen**). Diese gilt jedoch global für die Anwendung und blendet nicht alle Fenster unter Mac OS X aus.
Page range option	Lange Ganzzahl	15	*Wert1*=erste Seite zum Drucken (Standardwert ist 1) und optional *Wert2*=letzte Seite zum Drucken (Standardwert -1 = Dokumentende)
Legacy printing layer option	Lange Ganzzahl	16	(nur 4D 64-bit Versionen für Windows) nur *Wert1*: 1=auf GDI basierende bisherige Druckebene für nachfolgende Druckaufträge auswählen. 0=die D2D Druckerebene verwenden (Standard) **64-bit Versionen**: Selector wird nur in 4D 64-bit Versionen für Windows (Einzelplatz) unterstützt: auf anderen Plattformen wird er ignoriert. Er dient hauptsächlich dazu, damit bisherige Plug-Ins in 4D Druckaufträgen in 4D 64-bit Anwendungen drucken können.

Ein PDF Optionscode besteht aus zwei Teilen, kombiniert als *OptionTyp:OptionName.* Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **SET PRINT OPTION**.

Hinweis: Der Befehl **GET PRINT OPTION** funktioniert hauptsächlich mit PostScript Druckern. Sie können ihn auch mit anderen Druckertypen, wie PCL oder Ink verwenden, dann sind jedoch u.U. einige Optionen nicht verfügbar.

Systemvariablen und Mengen

Wird der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

Get print preview

Get print preview -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		Wahr = Druckvorschau Falsch = Keine Druckvorschau

Beschreibung

Die Funktion **Get print preview** gibt Wahr zurück, wenn die Anweisung **SET PRINT PREVIEW**(True) im aktuellen Prozess aufgerufen wurde.

Beachten Sie, dass der Benutzer diese Option vor Bestätigen des Dialogfensters verändern kann. Den endgültigen Druckmodus erhalten Sie über die Funktion **Is in print preview**.

GET PRINTABLE AREA

GET PRINTABLE AREA (Höhe {; Breite})

Parameter	Typ		Beschreibung
Höhe	Lange Ganzzahl	⇐	Höhe des druckbaren Bereichs
Breite	Lange Ganzzahl	⇐	Breite des druckbaren Bereichs

Beschreibung

Der Befehl **GET PRINTABLE AREA** gibt in den Parametern *Höhe* und *Breite* die Größe des druckbaren Bereichs in Pixel an. Sie richtet sich nach den aktuellen Druckparametern, der Papierausrichtung, etc.

Die zurückgegebenen Größen variieren nicht von einer Seite zur nächsten, z.B. nach einem Seitenumbruch.
Dieser Befehl ist, zusammen mit der Funktion **Get printed height** hilfreich, um die Anzahl der verfügbaren Pixel für den Druck zu erfahren oder um ein Objekt auf der Seite zu zentrieren.

Hinweis: Weitere Informationen zur Druckverwaltung und der Terminologie in 4D finden Sie in der Beschreibung zum Befehl **GET PRINTABLE MARGIN**.

Um die Gesamtgröße der Seite zu sehen:

- Fügen Sie entweder die Ränder, die der Befehl **GET PRINTABLE MARGIN** liefert, zu den Werten von **GET PRINTABLE AREA** hinzu
- oder verwenden Sie folgende Syntax:

```
SET PRINTABLE MARGIN(0;0;0;0) ` Setze Papierrand
GET PRINTABLE AREA(hPaper;wPaper) ` Papiergröße
```

GET PRINTABLE MARGIN

GET PRINTABLE MARGIN (Links ; Oben ; Rechts ; Unten)

Parameter	Typ		Beschreibung
Links	Lange Ganzzahl	⇦	Linker Rand
Oben	Lange Ganzzahl	⇦	Oberer Rand
Rechts	Lange Ganzzahl	⇦	Rechter Rand
Unten	Lange Ganzzahl	⇦	Unterer Rand

Beschreibung

Der Befehl **GET PRINTABLE MARGIN** gibt die aktuellen Werte der verschiedenen Ränder zurück, die mit den Befehlen **Print form**, **PRINT SELECTION** und **PRINT RECORD** definiert wurden.

Die Werte werden in Pixel angegeben und gehen vom Papierrand aus.

Über den Befehl **GET PRINTABLE AREA** erhalten Sie den Papierrand und können damit dann den druckbaren Bereich berechnen.

Druckbare Ränder verwalten

Die Druckberechnungen in 4D basieren standardmäßig auf "druckbaren Rändern". Der Vorteil dieses Systems ist, dass die Formulare sich automatisch selbst an neue Drucker anpassen, da sie im druckbaren Bereich positioniert werden. Der Nachteil ist, dass sich bei voreingestellten Formaten die zu druckenden Elemente nicht exakt drucken lassen, da ein Druckerwechsel auch die druckbaren Rändern ändern kann.

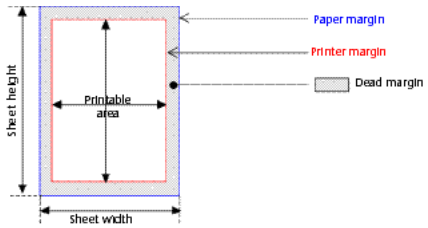
Über die Routinen **Print form**, **PRINT RECORD** und **PRINT SELECTION** lässt sich das Drucken von Formularen auf einen festen Rand einstellen, der auf allen Druckern gleich ist: den Papierrand, d.h. die tatsächliche Begrenzung eines Blattes. Dazu verwenden Sie die Befehle **GET PRINTABLE MARGIN**, **SET PRINTABLE MARGIN** und **GET PRINTABLE AREA**.

Definitionen

Papierrand: Das ist die tatsächliche Begrenzung des Blattes.

Druckerrand: Das ist der Rand, auf den der Drucker nicht drucken kann. Das ist abhängig vom Material, wie Druckerrollen, Endposition des Druckkopfs, o.ä.. Er variiert von einem Drucker zum anderen und auch von einem Format zum anderen.

Toter Rand: Das ist der Bereich zwischen Papierrand und Druckerrand.



Get printed height

Get printed height -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Position der Marke

Beschreibung

Die Funktion **Get printed height** gibt die Gesamthöhe (in Pixel) des Druckbereichs zurück, der mit der Funktion **Print form** verwendet wird.

Der zurückgegebene Wert liegt zwischen Null (dem oberen Rand der Seite) und der Gesamthöhe, die der Befehl **GET PRINTABLE AREA** zurückgibt (die maximale Größe des druckbaren Bereichs).

Drucken Sie mit der Funktion **Print form** einen neuen Bereich, wird die Höhe dieses Bereichs zu diesem Wert hinzugefügt. Reicht der druckbare Bereich dafür nicht aus, wird eine neue Seite angelegt. Der zurückgegebene Wert ist Null (0).

Rechter und linker druckbarer Rand haben keinen Einfluss auf den zurückgegebenen Wert. Der Wert berücksichtigt nur den oberen und unteren Rand, welche über den Befehl **SET PRINTABLE MARGIN** definiert werden können.

Hinweis: Weitere Informationen zur Druckverwaltung und 4D Terminologie finden Sie unter dem Befehl **GET PRINTABLE MARGIN**.

Is in print preview

Is in print preview -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		Wahr = Druckvorschau Falsch = Keine Druckvorschau

Beschreibung

Die Funktion **Is in print preview** gibt Wahr zurück, wenn im Druckdialog die Option **Auf Bildschirm** markiert ist, sonst Falsch. Diese Einstellung gilt lokal für den Prozess.

Im Gegensatz zu **Get print preview** gibt **Is in print preview** den endgültigen Wert der Option zurück, d.h. nachdem der Benutzer das Dialogfenster bestätigt hat. Diese Funktion lässt Sie mit Gewissheit bestimmen, ob das Drucken als Vorschau auf dem Bildschirm erfolgt.

Beispiel

Dieses Beispiel berücksichtigt alle Druckarten:

```
SET PRINT PREVIEW(True)  // Standardmäßig als Vorschau drucken
PRINT SETTINGS
If(OK=1)
 //Der Benutzer hat evtl. das Druckziel geändert
   If(Is in print preview)  // Wahr, wenn Vorschau
      FORM SET OUTPUT([Invoices];"aufBildschirm")
   Else
      FORM SET OUTPUT([Invoices];"anDrucker"
   End if
   OPEN PRINTING JOB
   ALL RECORDS([Invoices])
   PRINT SELECTION([Invoices];>)
   CLOSE PRINTING JOB
End if
```

Level

Level -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl	➲	Aktuelle Umbruch- oder Kopfteilebene

Beschreibung

Die Funktion **Level** gibt die aktuelle Umbruch- bzw. Kopfteilebene beim Ausdruck zurück.
Sie gibt die Nummer der Ebene während der Ereignisse *On Header* und *On Printing Break* zurück.

Level 0 ist die zuletzt zu druckende Ebene und eignet sich zum Drucken der Gesamtsumme. **Level** gibt 1 zurück, wenn 4D einen Umbruch auf das erste sortierte Feld druckt. 2 bei einem Umbruch auf das zweite sortierte Feld, usw..

Beispiel

Dieses Beispiel dient als Vorlage für eine Formularmethode. Es zeigt die Ereignisse, die eintreten können, wenn ein Bericht mit Summen ein Formular als Ausgabeformular verwendet. **Level** wird beim Drucken eines Kopfteils oder Umbruchs aufgerufen:

```
  ` Methode eines Formulars, das als Ausgabeformular für einen Summenbericht dient
$vpFormTable:=Current form table
Case of
  ` ...
   :(Form event=On Header)
  ` Ein Kopfteilbereich soll gerade ausgedruckt werden.
      Case of
         :(Before selection($vpFormTable->))
  ` Code für ersten Umbruch im Kopfteil
         :(Level=1)
  ` Code für Umbruchebene 1 im Kopfteil
         :(Level=2)
  ` Code für Umbruchebene 2 im Kopfteil
  ` ...
      End case
   :(Form event=On Printing Detail)
  ` Ein Datensatz soll gerade ausgedruckt werden.
  ` Code für jeden Datensatz
   :(Form event=On Printing Break)
  ` Ein Umbruchbereich soll gerade ausgedruckt werden.
      Case of
         :(Level=0)
  ` Code für Umbruchebene 0
         :(Level=1)
  ` Code für Umbruchebene 1
  ` ...
      End case
   :(Form event=On Printing Footer)
      If(End selection($vpFormTable->))
  ` Code für letzten Fußteil
      Else
  ` Code für einen Fußteil
      End if
End case
```

OPEN PRINTING JOB

OPEN PRINTING JOB
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **OPEN PRINTING JOB** öffnet einen Druckauftrag und stapelt alle darauffolgenden Druckaufträge, bis der Befehl **CLOSE PRINTING JOB** aufgerufen wird. Mit diesem Befehl können Sie Druckjobs steuern und insbesondere sicherstellen, dass keine anderen unerwarteten Druckjobs in eine Drucksequenz eingefügt werden können.

Sie können **OPEN PRINTING JOB** mit allen 4D Druckbefehlen, den Befehlen des Schnellberichteditors sowie den Druckbefehlen der Plug-Ins 4D Write und 4D View verwenden. Dieser Befehl ist dagegen nicht kompatibel mit dem Plug-Ins 4D Chart, sowie den meisten Plug-Ins von Drittherstellern.

Wird ein Druckjob mit diesem Befehl geöffnet, wird der Drucker in den Modus “beschäftigt” gesetzt, bis das Drucken tatsächlich startet. Startet ein nicht-kompatibles Plug-In einen Druckauftrag in diesem Kontext, wird der Fehler „Drucker beschäftigt“ zurückgegeben.

Sie müssen den Befehl **CLOSE PRINTING JOB** aufrufen, um den Druckjob zu beenden und das Druckdokument an den Drucker zu senden. Geben Sie diesen Befehl nicht an, bleibt das Druckdokument im Stapel und der Drucker ist nicht verfügbar, bis Sie die 4D Anwendung beenden.
Der Druckjob gilt lokal für den Prozess, das Drucken wird jedoch auf der globalen Ebene ausgeführt. In 4D lässt sich nur ein Druckauftrag zur gleichen Zeit öffnen, alle Prozesse zusammengenommen.

OPEN PRINTING JOB verwendet die aktuellen Druckeinstellungen. Das sind die Standardeinstellungen oder Einstellungen, die über die Befehle **PAGE SETUP** bzw. **SET PRINT OPTION** gesetzt wurden. Befehle, welche die Druckeinstellungen verändern, müssen vor Aufrufen von **OPEN PRINTING JOB** ausgeführt werden. Sonst wird ein Fehler erzeugt.

PAGE BREAK

PAGE BREAK {(* | >)}

Parameter	Typ		Beschreibung
* | >		➡	* Bricht mit Print form gestarteten Druckauftrag ab, > hält Druckauftrag offen

Beschreibung

PAGE BREAK veranlasst den Ausdruck einer Seite. Der Befehl wird mit der Funktion **Print form** im Kontext des Formularereignisses On Printing Detail verwendet, um Seitenumbrüche zu erzwingen und die zuletzt im Speicher erstellte Seite zu drucken. Verwenden Sie **PAGE BREAK** nicht mit dem Befehl **PRINT SELECTION**, sondern verwenden Sie zum Erstellen von Seitenumbrüchen stattdessen die Befehle **Subtotal** oder **BREAK LEVEL** mit dem optionalen Parameter.

Die Parameter * und > sind optional.

Der Parameter (*) annulliert den Druckvorgang, der mit dem Befehl **Print form** erzeugt wurde. Bei Ausführung dieses Befehls wird der laufende Druckvorgang unmittelbar gestoppt.

Hinweis: Unter Windows kann diese Operation durch Spooler-Eigenschaften des Druck-Servers gestört werden. Ist der Drucker so eingestellt, dass er den Druck sofort beginnt, bleibt das Abbrechen ohne Auswirkung. Damit **PAGE BREAK**(*) korrekt arbeitet, ist es besser, für den Drucker die Eigenschaft "Drucken nach dem Spoolen der letzten Seite beginnen."

Der Parameter (>) ändert den Befehl **PAGE BREAK** in zwei Punkten:

- Er hält den Druckauftrag offen, bis der Befehl **PAGE BREAK** ohne Parameter ausgeführt wird.
- Der Druckauftrag ist vorrangig. Solange gedruckt wird, kann kein anderer Druckvorgang dazwischengeschoben werden. Die zweite Option ist besonders hilfreich bei gespoolten Druckaufträgen. Der Parameter > garantiert, dass der Druckauftrag als eine Datei gespoolt wird. Das verringert die Druckzeit.

Hinweis: Klickt der Benutzer beim Drucken auf dem Bildschirm im Dialogfenster Druckvorschau auf die Schaltfläche **Abbrechen**, setzt der Befehl **PAGE BREAK** die Systemvariable OK auf 0 (Null).

Beispiel 1

Siehe Beispiel zur Funktion **Print form**.

Beispiel 2

Siehe Beispiel zum Befehl **SET PRINT MARKER**.

PAGE SETUP

PAGE SETUP ({Tabellenname ;} Formularname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, die das Formular enthält Ohne Angabe Haupttabelle
Formularname	String	➡	Name des Formulars

Beschreibung

Der Befehl **PAGE SETUP** legt ein Formular *Formularname* fest, dessen voreingestelltes Papierformat beim nächsten Druckvorgang benutzt werden soll. Der Befehl gilt nur für den nächsten Druckbefehl. Dieses Papierformat wird zusammen mit dem Formular in der Designumgebung abgespeichert.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich **PAGE SETUP** auf die Haupttabelle.

In folgenden Fällen werden die Druckdialoge nicht angezeigt. Es wird mit den Standardeinstellungen gedruckt:

- Sie rufen **PRINT SELECTION** auf und übergeben den optionalen Parameter *
- Sie rufen **PRINT RECORD** auf und übergeben den optionalen Parameter *
- Sie geben eine Reihe von Aufrufen an **Print form** aus, denen nicht der Aufruf des Befehls **PRINT SETTINGS** vorausgeht.

Mit **PAGE SETUP** können Sie die Druckdialoge umgehen UND eigene Druckereinstellungen verwenden.

Beispiel

Für die Tabelle [Design Stuff] werden mehrere (leere) Formulare erstellt. Das Formular "PS100" enthält ein Papierformat mit der Skalierung 100%, das Formular "PS90" ein Papierformat mit der Skalierung 90%, usw.. Mit folgender Projektmethode können Sie die Auswahl einer Tabelle in verschiedenen Größen drucken. Sie müssen nicht jedes Mal den Druckdialog aufrufen und die Größe vorgeben:

```
  ` Projektmethode AUTOMATIC SCALED PRINTING
  ` AUTOMATIC SCALED PRINTING ( Pointer ; String {; Long } )
  ` AUTOMATIC SCALED PRINTING ( ->[Table]; "Ausgabeformular" {; Scaling } )
If(Count parameters>=3)
   PAGE SETUP([Design Stuff];"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
```

Diese Projektmethode rufen Sie dann folgendermaßen auf:

```
  ` Suche nach aktuellen Rechnungen
QUERY([Invoices];[Invoices]Paid=False)
  ` Drucke Summenbericht in 90% aus
AUTOMATIC SCALED PRINTING(->[Invoices];"Summenbericht";90)
  ` Drucke Einzelbericht in 50% aus
AUTOMATIC SCALED PRINTING(->[Invoices];"Einzelbericht";50)
```

Print form

Print form ({Tabellenname ;} Formularname {; BereichStart {; BereichEnde}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	→	Zu druckende Tabelle, ohne Angabe Standardtabelle
Formularname	String, Objekt	→	Name (String) des Formulars oder POSIX Pfad (String) zu einer .json Datei mit Beschreibung des Formulars oder Objekt mit Beschreibung des zu druckenden Formulars
BereichStart	Lange Ganzzahl	→	Druckmarke oder Beginn des Bereichs (wenn BereichEnde angegeben)
BereichEnde	Lange Ganzzahl	→	Ende des Bereichs (wenn BereichStart angegeben)
Funktionsergebnis	Lange Ganzzahl	←	Höhe des gedruckten Bereichs

Beschreibung

Die Funktion **Print form** druckt das Formular *Formularname* mit den aktuellen Feldwerten und Variablen der Tabelle *Tabellenname*. **Print form** eignet sich zum Drucken komplexer Berichte, die eine komplette Steuerung des Druckvorgangs erforderlich machen.
Print form führt keine Berechnungen in Datensätzen bzw. Umbrüchen durch, erstellt weder Seitenumbrüche noch Kopf- oder Fußteile. Diese Operationen müssen Sie selbst ausführen. Mit **Print form** werden Datenfelder und Variablen nur in einer festen Rahmengröße gedruckt.

Im Parameter *Formularname* übergeben Sie:

- Den Namen des Formulars oder
- Den Pfad (in POSIX Syntax) zu einer gültigen .json Datei mit der Beschreibung des Formulars (siehe **Dateipfade für Formulare**) oder
- Ein Objekt mit der Beschreibung des Formulars. Weitere Informationen dazu finden Sie unter **Dynamische Formulare**

Da **Print form** nach Drucken des Formulars keinen Seitenumbruch auslöst, können Sie so verschiedene Formulare auf einer Seite kombinieren. **Print form** ist für komplexe Druckvorgänge mit verschiedenen Tabellen und Formularen gut geeignet. Mit dem Befehl **PAGE BREAK** legen Sie zwischen den Formularen einen Seitenumbruch an. Um ein Formular, dessen Höhe den verfügbaren Platz übersteigt, auf der nächsten Seite weiterzudrucken, rufen Sie vor **PAGE BREAK** den Befehl **CANCEL** auf.

Es gibt drei verschiedene Syntaxarten:

- **Detailbereich drucken**

```
height:=Print form(MeineTabelle;MeinFormular)
```

In diesem Fall druckt die Funktion nur den Detailbereich, das ist im Formular der Bereich zwischen Kopfzeile und der Marke D.

- **Formularbereich drucken**

```
height:=Print form(MeineTabelle;MeinFormular;Marke)
```

In diesem Fall druckt die Funktion den durch Marke angegebenen Bereich. Als Parameter übergeben Sie eine Konstante unter dem Thema **Formularbereich**:

Konstante	Typ	Wert
Form break0	Lange Ganzzahl	300
Form break1	Lange Ganzzahl	301
Form break2	Lange Ganzzahl	302
Form break3	Lange Ganzzahl	303
Form break4	Lange Ganzzahl	304
Form break5	Lange Ganzzahl	305
Form break6	Lange Ganzzahl	306
Form break7	Lange Ganzzahl	307
Form break8	Lange Ganzzahl	308
Form break9	Lange Ganzzahl	309
Form detail	Lange Ganzzahl	0
Form footer	Lange Ganzzahl	100
Form header	Lange Ganzzahl	200
Form header1	Lange Ganzzahl	201
Form header10	Lange Ganzzahl	210
Form header2	Lange Ganzzahl	202
Form header3	Lange Ganzzahl	203
Form header4	Lange Ganzzahl	204
Form header5	Lange Ganzzahl	205
Form header6	Lange Ganzzahl	206
Form header7	Lange Ganzzahl	207
Form header8	Lange Ganzzahl	208
Form header9	Lange Ganzzahl	209

- **Bereich drucken**

```
height:=Print form(MeineTabelle;MeinFormular;BereichStart;BereichEnd)
```

In diesem Fall druckt die Funktion den Bereich zwischen den Parametern *BereichStart* und *BereichEnde*. Die Werte müssen in Pixel angegeben werden.

Der von **Print form** zurückgegebene Wert gibt die Höhe des druckbaren Bereichs an. Diesen Wert berücksichtigt die Funktion **Get printed height** automatisch.

Mit **Print form** erscheinen keine Druckdialoge. Der Bericht verwendet nicht die Druckereinstellungen, die dem Formular in der Designumgebung zugewiesen wurden. Sie können die Druckereinstellungen vor Aufrufen von **Print form** auf zwei Arten festlegen:

- Sie rufen **PRINT SETTINGS** auf. In diesem Fall bestimmt der Benutzer die Einstellungen.
- Sie rufen **PAGE SETUP** auf. In diesem Fall werden die Einstellungen per Programmierung festgelegt.

4D erstellt die zu druckende Seite im Hauptspeicher. Die Seite wird ausgedruckt, wenn die Seite im Speicher voll ist oder wenn Sie den Befehl **PAGE BREAK** aufrufen. Sie müssen den Druckvorgang mit **PAGE BREAK** abschließen, damit auch die letzte Seite ausgedruckt wird, selbst wenn sie nicht voll ist (außer im Kontext des Befehls **OPEN PRINTING JOB**, siehe nachfolgende Warnung).

Warnung: Wird die Funktion in einem Druckauftrag aufgerufen, der mit **OPEN PRINTING JOB** geöffnet wurde, dürfen Sie für die letzte Seite NICHT **PAGE BREAK** aufrufen, da **OPEN PRINTING JOB** sie automatisch druckt. Rufen Sie in diesem Fall den Befehl **PAGE BREAK** auf, wird eine leere Seite gedruckt.

Dieser Befehl druckt externe Bereiche und Objekte, z.B. 4D Write oder 4D View Bereiche. Der Bereich wird für jede Ausführung des Befehls neu aufgebaut.

Warnung: Mit **Print form** werden keine Unterformulare ausgedruckt. Wollen Sie nur ein Formular für solch ein Objekt ausdrucken, verwenden Sie dafür den Befehl **PRINT RECORD**.

Print form erstellt nur ein Ereignis *On Printing Detail* für die Formularmethode.

4D Server: Diese Funktion lässt sich auf 4D Server im Rahmen einer Serverprozedur ausführen. In diesem Kontext müssen Sie folgendes beachten:

- Stellen Sie sicher, dass auf dem Server Rechner kein Dialogfenster erscheint, außer für spezifische Anforderungen.
- Bei einem Druckerproblem, z.B. kein Papier oder Drucker nicht verfügbar, erscheint keine Fehlermeldung.

Beispiel 1

Folgendes Beispiel arbeitet wie der Befehl **PRINT SELECTION**. Der Bericht verwendet jedoch zwei verschiedene Formulare, je nachdem, ob der Datensatz für Scheck oder Bareinzahlung ist:

```
QUERY([Register])  ` Wähle die Datensätze
If(OK=1)
   ORDER BY([Register])  ` Sortiere die Datensätze
   If(OK=1)
      PRINT SETTINGS  ` Zeige Druckdialoge
      If(OK=1)
         For($vlRecord;1;Records in selection([Register]))
            If([Register]Type ="Scheck")
               Print form([Register];"Scheckvordruck")
 ` Verwenden Sie ein Formular für Schecks
            Else
               Print form([Register];"Vordruck für Bareinzahlung")
 ` Verwenden Sie ein anderes Formular für Bareinzahlung
            End if
            NEXT RECORD([Register])
         End for
         PAGE BREAK  ` Stellen Sie sicher, dass die letzte Seite gedruckt wird
      End if
   End if
End if
```

Beispiel 2

Siehe Beispiel zum Befehl **SET PRINT MARKER**.

PRINT LABEL

PRINT LABEL ({Tabellenname }{;}{ Dokumentname {; * | >}})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Zu druckende Tabelle oder Haupttabelle ohne Angabe
Dokumentname	String	➡	Name des gespeicherten Etikettenformats
* | >		➡	* Druckdialoge unterdrücken oder > Eigene Druckparameter beibehalten

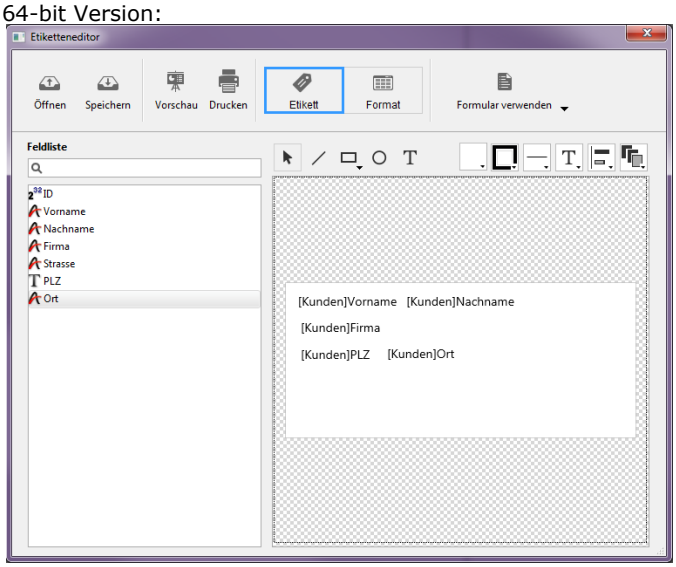
Beschreibung

PRINT LABEL druckt für den laufenden Prozess die Datensätze der aktuellen Auswahl von *Tabellenname* als Etiketten. *Tabellenname* ist optional. Wird der Parameter nicht angegeben, bezieht sich **PRINT LABEL** auf die Haupttabelle.

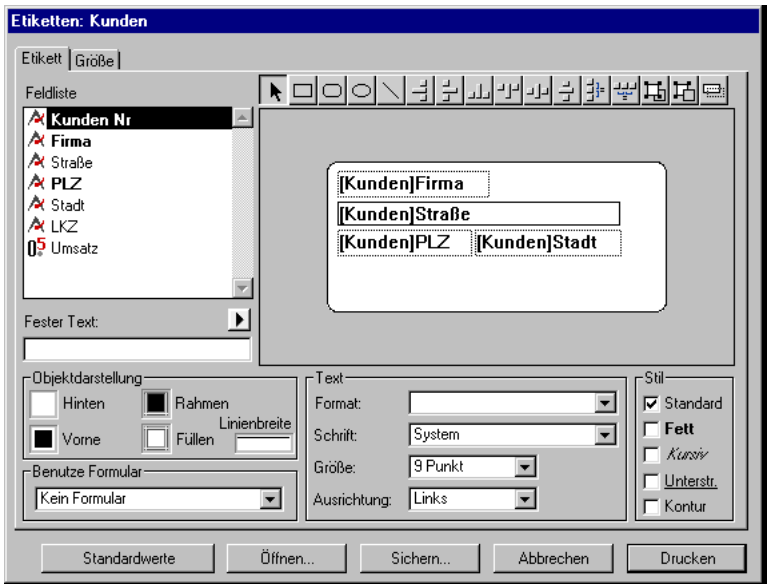
Geben Sie *Dokumentname* nicht an, benutzt **PRINT LABEL** das aktuelle Ausgabeformular von *Tabellenname*. Dieser Befehl gilt nicht für Unterformulare. Weitere Informationen dazu finden Sie im **Überblick** des Handbuchs *4D Designmodus*.

Geben Sie *Dokumentname* an, können Sie auf den Etiketteneditor oder auf ein gespeichertes Etikettenformular zugreifen.

64-bit Version:



32-bit Version:



Hinweis zur Kompatibilität: Die 32-bit Version des Etiketteneditors unterstützt nur ASCII Zeichen. Verwenden Sie einen erweiterten Zeichensatz, müssen Sie Etiketten über das aktuelle Ausgabeformular drucken.

PRINT LABEL zeigt standardmäßig vor dem Drucken die Druckdialoge (in 4D 32-bit Versionen) oder das Dialogfenster Drucken (in 4D 64-bit Versionen). Bricht der Benutzer einen der Dialoge ab, wird der Befehl abgebrochen und die Etiketten werden nicht gedruckt.

PRINT LABEL zeigt standardmäßig vor dem Drucken die Druckdialoge an. Bricht der Benutzer einen dieser Dialoge ab, wird der Befehl abgebrochen. Es werden keine Etiketten gedruckt.
Sie können diese Druckdialoge mit den optionalen Parametern Stern (*) oder größer als (>) unterdrücken:

- Der Parameter * löst einen Druckauftrag mit den aktuellen Parametern zum Drucken aus.
- Der Parameter > löst einen Druckauftrag aus, ohne die aktuellen Druckparameter zu verändern. Diese Einstellung ist hilfreich, um mehrere aufeinanderfolgende Aufrufe von **PRINT LABEL** unter Beibehaltung der individuell festgelegten

Druckparameter auszuführen (z.B. in einer Schleife). Ein Beispiel dazu finden Sie in der Beschreibung zum Befehl **PRINT RECORD**.

Diese Parameter haben jedoch keine Auswirkung, wenn der Etiketteneditor beteiligt ist.

Ist der Etiketteneditor nicht beteiligt, hat die Variable **OK** den Wert 1, wenn alle Etiketten gedruckt werden; ansonsten den Wert 0 (Null) (der Benutzer hat z.B. im Druckdialog auf die Schaltfläche **Abbrechen** geklickt).

Geben Sie den Parameter *Dokumentname* nicht an, öffnet **PRINT LABEL** das Standardfenster für die Dokumentenauswahl. Sie können nun das gewünschte Dokument auswählen.
Geben Sie den Parameter *Dokumentname* an und ist ein Dokument dieses Namens vorhanden, lädt **PRINT LABEL** die Formatierung für die Etiketten und druckt sie mit den Druckparametern, die bei der Definition der Etiketten festgelegt wurden.
Geben Sie den Parameter *Dokumentname* an und ist kein Dokument mit diesem Namen vorhanden, ruft der Befehl **PRINT LABEL** das Standardfenster für die Dokumentenauswahl auf. Sie können dann das gewünschte Dokument auswählen. Möchten Sie sicherstellen, dass nicht doch zufällig ein Dokument mit dem von Ihnen angegebenen Namen existiert, können Sie z.B. den Text, den der Aufruf von *Char (1)* ergibt, als Argument übergeben. Da dieses Zeichen nicht direkt über die Tastatur erzeugt werden kann, ist es unwahrscheinlich, dass ein Dokument mit diesem Namen existiert.

Hinweis: Ist *Tabellenname* im Designmodus auf unsichtbar gesetzt, erscheint der Etiketteneditor nicht.

4D Server: Dieser Befehl lässt sich auf 4D Server im Rahmen einer Serverprozedur ausführen. In diesem Kontext müssen Sie folgendes beachten:

- Stellen Sie sicher, dass auf dem Server Rechner kein Dialogfenster erscheint, außer für spezifische Anforderungen. Dazu müssen Sie den Befehl mit dem Parameter * oder > aufrufen.
- Die Syntax zum Aufrufen des Etiketteneditors funktioniert nicht mit 4D Server. In diesem Fall wird die Systemvariable OK auf 0 (Null) gesetzt.
- Bei einem Druckerproblem, z.B. kein Papier oder Drucker nicht verfügbar, erscheint keine Fehlermeldung.

Beispiel 1

Folgendes Beispiel druckt Etiketten mit dem Ausgabeformular einer Tabelle. Es verwendet zwei Methoden. Die erste setzt das richtige Ausgabeformular und druckt dann die Etiketten:

```
ALL RECORDS([Addresses]) ` Wähle alle Datensätze
FORM SET OUTPUT([Addresses];"Etikettenausgabe") ` Wähle das Ausgabeformular
PRINT LABEL([Addresses]) ` Drucke Etiketten
FORM SET OUTPUT([Addresses];"Ausgabe") ` Stelle Standardausgabeformular wieder her
```

Die zweite Methode ist die Formularmethode für das Formular *Etikettenausgabe*. Es enthält eine Variable mit Namen *vLabel*, die die Verkettung der Datenfelder erhält. Ist das zweite Addressenfeld (Addr2) leer, entfernt es die Methode. Beachten Sie, dass dieses Task mit dem Etiketteneditor automatisch ausgeführt wird. Die Formularmethode erstellt das Etikett für jeden Datensatz:

```
  ` [Addresses]; Formularmethode "Etikettenausgabe"
Case of
   :(Form event=On Load)
      vLabel:=[Addresses]Name1+" "+[Addresses]Name2
      +Char(13)+[Addresses]Addr1+Char(13)
      If([Addresses]Addr2 #"")
         vLabel:=vLabel+[Addresses]Addr2+Char(13)
      End if
      vLabel:=vLabel+[Addresses]City+", "+[Addresses]State+" "+[Addresses]ZipCode
End case
```

Beispiel 2

Folgendes Beispiel lässt den Benutzer die Tabelle [People] suchen und druckt dann automatisch die Etiketten “My Labels”:

```
QUERY([People])
If(OK=1)
   PRINT LABEL([People];"My Labels";*)
End if
```

Beispiel 3

Folgendes Beispiel lässt den Benutzer die Tabelle [People] suchen und dann die zu druckenden Etiketten auswählen:

```
QUERY([People])
If(OK=1)
   PRINT LABEL([People];"")
End if
```

Beispiel 4

Folgendes Beispiel lässt den Benutzer die Tabelle [People] suchen und zeigt dann den Etiketteneditor, so dass der Benutzer beliebige Etiketten anlegen, sichern, laden und drucken kann:

```
QUERY([People])
If(OK=1)
    PRINT LABEL([People];Char(1))
End if
```

Print object

Print object ({* ;} Objekt {; posX {; posY {; Breite {; Höhe}}}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Mit Stern: Objektname oder Ohne Stern: Variable
posX	Lange Ganzzahl	➡	Horizontale Position des Objekts
posY	Lange Ganzzahl	➡	Vertikale Position des Objekts
Breite	Lange Ganzzahl	➡	Breite des Objekts (Pixel)
Höhe	Lange Ganzzahl	➡	Höhe des Objekts (Pixel)
Funktionsergebnis	Boolean	⮌	Wahr = Objekt wird vollständig gedruckt; sonst Falsch

Beschreibung

Die Funktion **Print object** druckt die Formularobjekte, definiert durch die Parameter *Objekt* und * und positioniert durch die Parameter *posX* und *posY*.

Vor Aufrufen dieser Funktion müssen Sie die Tabelle oder das Projektformular, welches die zu druckenden Objekte enthält, über den Befehl **FORM LOAD** bestimmen.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname, also eine Zeichenkette ist. Übergeben Sie keinen *, geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz (nur Objekttyp) anstelle eines String.
Die Parameter *posX* und *posY* geben den Ausgangspunkt zum Drucken der Objekte an. Die Werte müssen in Pixel ausgedrückt werden. Ohne diese Parameter wird das Objekt gemäß seiner Position im Formular gedruckt.
Die Parameter *Breite* und *Höhe* definieren die Breite und Höhe des Formularobjekts. Die Funktion **Print object** unterstützt keine Objekte mit variabler Größe. Zur Größenverwaltung von Objekten müssen Sie den Befehl **OBJECT GET BEST SIZE** verwenden. Über diesen Befehl können Sie auch die am besten geeignete Größe für Objekte mit Text herausfinden. Analog zu **FORM LOAD** legt auch **Print object** nicht automatisch Seitenumbrüche an. Das muss der Entwickler gemäß seinen Anforderungen selbst verwalten.
Sie können über 4D Befehle Objekteigenschaften, wie Farbe, Größe, etc. direkt verändern.

Die Funktion gibt *Wahr* zurück, wenn das Objekt vollständig gedruckt wurde und *Falsch*, wenn das nicht der Fall ist, d.h. es konnten nicht alle in *Objekt* zugeordneten Daten innerhalb des gesetzten Rahmens gedruckt werden.
Die Funktion gibt beim Drucken von Listboxen *Falsch* zurück, wenn nicht alle Zeilen der Listbox gedruckt werden konnten. In diesem Fall können Sie einfach die Funktion **Print object** so oft aufrufen, bis sie *Wahr* zurückgibt. Ein spezifischer Mechanismus sorgt dafür, dass der Inhalt des Objekts nach jedem Aufruf weiter gescrollt wird.

Hinweise:

- In der aktuellen 4D Version gilt dieser Mechanismus nur für Objekte vom Typ Listbox. Für alle anderen Objekte gibt er immer *Wahr* zurück. Diese Funktionsweise wird in zukünftigen 4D Versionen auf andere Objekte mit variablem Inhalt ausgeweitet.
- Über den Befehl **LISTBOX GET PRINT INFORMATION** können Sie den Status während dem Druckvorgang kontrollieren.

Die Funktion **Print object** lässt sich nur im Kontext eines Druckauftrags verwenden, der zuvor über den Befehl **OPEN PRINTING JOB** geöffnet wurde. Wird er nicht in diesem Kontext aufgerufen, führt er nichts aus. Im gleichen Druckauftrag lassen sich mehrere Funktionen **Print object** aufrufen.

Hinweis: Hierarchische Listen, Unterformulare und Web Areas sind nicht druckbar.

Beispiel 1

Zehn Objekte in einem Formular drucken:

```
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
```

Beispiel 2

Eine komplette Listbox drucken:

```
Repeat
    $end:=Print object(*;"mylistbox")
Until($end)
```

PRINT OPTION VALUES

PRINT OPTION VALUES (Option ; NamenArray {; Info1Array {; Info2Array}})

Parameter	Typ		Beschreibung
Option	Lange Ganzzahl	➡	Optionsnummer
NamenArray	Array Text	⬅	Namen der Werte
Info1Array	Array Lange Ganzzahl	⬅	Werte (1) der Option
Info2Array	Array Lange Ganzzahl	⬅	Werte (2) der Option

Beschreibung

Der Befehl **PRINT OPTION VALUES** gibt in *NamenArray* eine Liste der für *Option* verfügbaren Werte zurück. Sie können optional in *Info1Array* und *Info2Array* für jeden Wert Informationen wiederfinden.
Der Parameter *Option* spezifiziert die zu erhaltende Option. Sie müssen eine vordefinierte Konstante unter dem Thema **Druckoptionen** übergeben:

Konstante	Typ	Wert	Kommentar
Paper option	Lange Ganzzahl	1	Verwenden Sie nur *Wert1*, enthält er den Namen des Papierformats. Verwenden Sie beide Parameter, enthält *Wert1* die Papierbreite und *Wert2* die Papierhöhe. Breite und Höhe werden in Pixel auf dem Bildschirm angegeben. Über den Befehl **PRINT OPTION VALUES** erhalten Sie Name, Höhe und Breite aller Papierformate, die der Drucker anbietet.
Paper source option	Lange Ganzzahl	5	(nur Windows) nur *Wert1*: Nummer, die dem Index des zu verwendenden Papierschachts entspricht. Über den Befehl **PRINT OPTION VALUES** erhalten Sie das Array mit den Namen. Diese Option ist nur unter Windows verwendbar.

Nach Ausführung des Befehls sind das Array *NamenArray* und, falls zutreffend, die Arrays *Info1Array* und mit den Namen und Informationen der verfügbaren Werte gefüllt.
Übergeben Sie im Parameter *Option* den Wert 1 (paper option), gibt der Befehl folgende Information zurück:

- In *NamenArray* die Namen der verfügbaren Papierformate;
- In *Info1Array* die Höhe der Papierformate;
- In *Info2Array* die Breite der Papierformate.

Hinweis: Um diese Information zu erhalten, muss der Druckertreiber Zugriff auf eine gültige PPD-Datei (PostScript Printer Description) des Druckers haben.

Um über den Befehl **SET PRINT OPTION** ein bestimmtes Papierformat anzuwenden, können Sie entweder einen Wert aus *NamenArray* übergeben oder die entsprechenden Werte aus *Info1Array* und *Info2Array*.
Übergeben Sie im Parameter *Option* den Wert 5 (paper source option), gibt der Befehl die Namen der verschiedenen Papierschächte aus *NamenArray* zurück und deren interne Windows ID Nummern in *Info1Array* (*Info2Array* bleibt leer).
Die Reihenfolge der Werte in den Arrays legt der Druckertreiber fest. Um einen Papierschacht über den Befehl **SET PRINT OPTION** anzugeben, müssen Sie den Index des gewünschten Elements übergeben, wie in den Arrays *NamenArray* oder *Info1Array* angegeben.

Hinweis: Diese Option ist nur unter Windows verwendbar.

Weitere Informationen zu den einzelnen Druckoptionen finden Sie in der Beschreibung zu den Befehlen **SET PRINT OPTION** und **GET PRINT OPTION**.

Alle von diesen Befehlen zurückgegebenen Informationen gelten für das Betriebssystem. Ausführliche Informationen dazu finden Sie in der Dokumentation zu Ihrem System.

Hinweis: Der Befehl **GET PRINT OPTION** funktioniert nur mit PostScript Druckern.

PRINT RECORD

PRINT RECORD ({Tabellenname}{;}{* | >})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren aktueller Datensatz gedruckt werden soll Ohne Angabe Haupttabelle
* | >	Operator	➡	* Druckdialoge unterdrücken oder > Eigene Druckparameter beibehalten

Beschreibung

Der Befehl **PRINT RECORD** druckt den aktuellen Datensatz von *Tabellenname*, ohne die aktuelle Auswahl zu verändern. Zum Drucken wird das aktuelle Ausgabeformular benutzt. Gibt es keinen aktuellen Datensatz für *Tabellenname*, hat **PRINT RECORD** keine Auswirkung.
PRINT RECORD druckt auch Unterformulare, was mit **Print form** nicht möglich ist.

Hinweis: Haben Sie am aktuellen Datensatz Änderungen vorgenommen und diese noch nicht gesichert, dann druckt der Befehl die geänderten Feldwerte und nicht die Werte, die auf der Festplatte gespeichert sind.

PRINT RECORD zeigt standardmäßig vor dem Drucken die Druckdialoge (in 4D 32-bit Versionen) oder das Dialogfenster Drucken (in 4D 64-bit Versionen). Bricht der Benutzer einen der Dialoge ab, wird der Befehl abgebrochen und der Datensatz nicht gedruckt.

Sie können die Druckdialoge mit den optionalen Parametern Stern (*) oder größer als (>) unterdrücken:

- Der Parameter * löst einen Druckauftrag mit den aktuellen Parametern zum Drucken aus oder mit den Parametern, die im Befehl **PAGE SETUP** bzw. **SET PRINT OPTION** festgelegt wurden.
- Der Parameter > löst einen Druckauftrag aus, ohne die aktuellen Druckparameter zu verändern. Diese Einstellung ist hilfreich, um mehrere aufeinanderfolgende Aufrufe von **PRINT RECORD** unter Beibehaltung der individuell festgelegten Druckparameter auszuführen (z.B. in einer Schleife).

Diese Parameter haben jedoch keine Auswirkung, wenn der Etiketteneditor beteiligt ist.

4D Server: Dieser Befehl lässt sich auf 4D Server im Rahmen einer Serverprozedur ausführen. In diesem Kontext müssen Sie folgendes beachten:

- Stellen Sie sicher, dass auf dem Server Rechner kein Dialogfenster erscheint, außer für spezifische Anforderungen. Dazu müssen Sie den Befehl mit dem Parameter * oder > aufrufen.
- Die Syntax zum Aufrufen des Etiketteneditors funktioniert nicht mit 4D Server. In diesem Fall wird die Systemvariable OK auf 0 (Null) gesetzt.
- Bei einem Druckerproblem, z.B. kein Papier oder Drucker nicht verfügbar, erscheint keine Fehlermeldung.

Warnung: Verwenden Sie mit **PRINT RECORD** nicht den Befehl **PAGE BREAK**. Dieser kann nur kombiniert mit der Funktion **Print form** verwendet werden.

Beispiel 1

Folgendes Beispiel druckt den aktuellen Datensatz von der Tabelle [Invoices]. Der Code ist in der Objektmethode für eine Schaltfläche **Drucken** im Eingabeformular enthalten. Klickt der Benutzer auf die Schaltfläche, wird der Datensatz in einem eigens dafür angelegten Ausgabeformular gedruckt.

```
FORM SET OUTPUT([Invoices];"Drucke eine von der Dateneingabe")
  ` Wähle Ausgabeformular für Druck
PRINT RECORD([Invoices];*) ` Drucke die komplette Rechnung(ohne die Druckdialoge)
FORM SET OUTPUT([Invoices];"Standardausgabe")
  ` Weise wieder voriges Ausgabeformular zu.
```

Beispiel 2

Folgendes Beispiel druckt denselben aktuellen Datensatz in zwei verschiedenen Formen. Der Code dazu befindet sich in der Objektmethode zu einer Schaltfläche **Drucken** im Eingabeformular. Dafür definieren Sie eigene Druckparameter und verwenden diese für beide Formen.

```
PRINT SETTINGS `Benutzer definiert die Druckparameter
If(OK=1)
   FORM SET OUTPUT([Angestellte];"Detailliert") `Benutze erstes Druckformular
   PRINT RECORD([Angestellte];>) `Drucke mit benutzerdefinierten Parametern
   FORM SET OUTPUT([Angestellte];"Einfach") `Benutze zweites Druckformular
   PRINT RECORD([Angestellte];>) `Drucke mit benutzerdefinierten Parametern
   FORM SET OUTPUT([Angestellte];"Ausgabe") `Stelle Standardausgabeformular wieder her
End if
```

PRINT SELECTION

PRINT SELECTION ({Tabellenname}{;}{* | >})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Auswahl zu drucken ist Ohne Angabe Haupttabelle
* | >	Operator	➡	* Druckdialoge unterdrücken oder > Eigene Druckparameter beibehalten

Beschreibung

PRINT SELECTION druckt die aktuelle Auswahl aus *Tabellenname*. Die Datensätze werden für den laufenden Prozess im aktuellen Ausgabeformular gedruckt. **PRINT SELECTION** führt dieselbe Aktion wie der Befehl Drucken in der Anwendungsumgebung aus. Ist die Auswahl leer, hat der Befehl keine Auswirkung.

PRINT SELECTION zeigt standardmäßig vor dem Drucken die Druckdialoge (in 4D 32-bit Versionen) oder das Dialogfenster Drucken (in 4D 64-bit Versionen). Bricht der Benutzer einen der Dialoge ab, wird der Befehl abgebrochen und das Dokument nicht gedruckt.

Tabellenname ist optional. Wird dieser Parameter nicht angegeben, bezieht sich **PRINT SELECTION** auf die Haupttabelle.

Sie können diese Druckdialoge mit den optionalen Parametern Stern (*) oder größer als (>) unterdrücken:

- Der Parameter * löst einen Druckauftrag mit den aktuellen Parametern zum Drucken aus oder mit den Parametern, die im Befehl **PAGE SETUP** bzw. **SET PRINT OPTION** festgelegt wurden.
- Der Parameter > löst einen Druckauftrag aus, ohne die aktuellen Druckparameter zu verändern. Diese Einstellung ist hilfreich, um mehrere aufeinanderfolgende Aufrufe von **PRINT SELECTION** unter Beibehaltung der individuell festgelegten Druckparameter auszuführen (z.B. in einer Schleife). Ein Beispiel dazu finden Sie in der Beschreibung zum Befehl **PRINT RECORD**.
  Diese Parameter haben jedoch keine Auswirkung, wenn der Etikетteneditor beteiligt ist.

Während dem Drucken werden Methoden des Ausgabeformulars und/oder die Objektmethoden von Formularen ausgeführt. Das hängt sowohl von den Ereignissen ab, die in den Dialogfenstern Formular- bzw. Objekteigenschaften der Designumgebung aktiviert wurden, als auch von den gerade ablaufenden Ereignissen:

- Ein Ereignis *On Header* wird während dem Ausdrucken des Kopfteils aufgerufen.
- Ein Ereignis *On Printing Detail* wird direkt vor dem Ausdrucken eines Datensatzes aufgerufen.
- Ein Ereignis *On Printing Break* wird direkt vor dem Ausdrucken eines Umbruchs aufgerufen.
- Ein Ereignis *On Printing Footer* wird direkt vor dem Ausdrucken eines Fußteils aufgerufen.

Mit der Funktion **Before selection** überprüfen Sie, ob 4D den ersten Kopfteil ausdruckt, mit der Funktion **End selection**, ob der letzte Fußteil ausgedruckt wird. Weitere Informationen dazu finden Sie in der Beschreibung dieser Funktionen, sowie der Funktionen **Form event** und **Level**.

Um einen Bericht mit Zwischensummen oder Umbrüchen zu drucken, muss die Auswahl vorher sortiert werden. Integrieren Sie dann in jedem Umbruchteil des Berichts eine Variable mit einer Objektmethode, die die Zwischensumme zuweist. Dazu können Sie auch die arithmetischen Funktionen, wie z.B. **Sum** und **Average** verwenden. Weitere Informationen dazu finden Sie in den Beschreibungen zu **Subtotal**, **BREAK LEVEL** und **ACCUMULATE**.

Warnung: Verwenden Sie mit **PRINT SELECTION** nicht den Befehl **PAGE BREAK**. Dieser Befehl muss mit der Funktion **Print form** verwendet werden.

Nach dem Druck können Sie durch Abfragen der OK-Variablen feststellen, ob der Druck ohne Unterbrechung erfolgt ist. Wurde in einem der beiden Druckdialoge auf Abbrechen gedrückt oder wurde der Druckvorgang selbst unterbrochen, hat OK den Wert 0. Wurde ohne Unterbrechung gedruckt, hat OK den Wert 1.

4D Server: Dieser Befehl lässt sich auf 4D Server im Rahmen einer Serverprozedur ausführen. In diesem Kontext müssen Sie folgendes beachten:

- Stellen Sie sicher, dass auf dem Server Rechner kein Dialogfenster erscheint, außer für spezifische Anforderungen. Dazu müssen Sie den Befehl mit dem Parameter * oder > aufrufen.
- Die Syntax zum Aufrufen des Etikетteneditors funktioniert nicht mit 4D Server. In diesem Fall wird die Systemvariable OK auf 0 (Null) gesetzt.
- Bei einem Druckerproblem, z.B. kein Papier oder Drucker nicht verfügbar, erscheint keine Fehlermeldung.

Beispiel

Folgendes Beispiel wählt alle Datensätze in der Tabelle [People] aus und zeigt dann mit dem Befehl **DISPLAY SELECTION** die Datensätze an. Der Benutzer kann die zu druckenden Datensätze markieren. Der Befehl **USE SET** verwendet nur diese Datensätze, sie werden mit **PRINT SELECTION** gedruckt:

```
ALL RECORDS([People])  `Wähle alle Datensätze
DISPLAY SELECTION([People];*)  `Zeige die Datensätze
USE SET("UserSet")  `Verwende nur die vom Benutzer markierten Datensätze
PRINT SELECTION([People])  `Drucke diese Datensätze
```

PRINT SETTINGS

PRINT SETTINGS {(Dialoge)}

Parameter	Typ		Beschreibung
Dialoge	Lange Ganzzahl	➡	Dialogfenster anzeigen

Beschreibung

Der Befehl **PRINT SETTINGS** ruft einen oder zwei Druckdialoge auf. Sie müssen diesen Befehl vor den Befehlen für Druckformular bzw. dem Befehl **OPEN PRINTING JOB** aufrufen.

Mit dem optionalen Parameter *Dialoge* können Sie die Anzeige der Druckdialoge steuern. Sie können eine der folgenden Konstanten unter dem Thema **Druckoptionen** verwenden. Die Anzeige der Druckdialoge richtet sich nach der eingesetzten 4D Version. Es gilt folgendes:

Wert für Dialoge (Konstante)	4D 64-bit	4D 32-bit
0 oder weggelassen	Druckdialog	Seite einrichten+Druckdialog
1 (Page setup dialog)	Seite einrichten	Seite einrichten
2 (Print dialog)	Druckdialog (= 0 oder weggelassen)	Druckdialog

Hinweis: Der Druckdialog enthält das Kontrollkästchen *Auf Bildschirm*. Damit kann der Benutzer auf dem Bildschirm drucken. Sie können diese Option aktivieren oder deaktivieren, wenn Sie vor dem Befehl **PRINT SETTINGS** den Befehl **SET PRINT PREVIEW** aufrufen.

Beispiel

Siehe Beispiel zur Funktion **Print form**.

Systemvariablen und Mengen

Klickt der Benutzer in beiden Dialogen auf die Schaltfläche OK, nimmt die Systemvariable OK den Wert 1 an. Klickt er auf die Schaltfläche **Abbrechen**, nimmt die Systemvariable OK den Wert 0 an.

Print settings to BLOB

Print settings to BLOB (Druckeinstellungen) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Druckeinstellungen	BLOB	⇐	aktuelle Druckeinstellungen
Funktionsergebnis	Lange Ganzzahl	⇒	Statuswert: 1=Operation erfolgreich, 0=Kein aktueller Drucker

Beschreibung

Die Funktion **Print settings to BLOB** sichert die aktuellen 4D Druckeinstellungen im BLOB *Druckeinstellungen*. Der Parameter *Druckeinstellungen* speichert alle Einstellungen zum Drucken:

- Layout Parameter wie Papier, Ausrichtung, Größe ...
- Druckparameter wie Anzahl Kopien, Papierquelle ...

Diese Funktion muss zusammen mit der Funktion **BLOB to print settings** verwendet werden. Damit lassen sich die aktuellen Druckeinstellungen des Benutzers sichern und später erneut laden, so dass Benutzer nicht für jeden gestarteten Druckauftrag die Parameter neu definieren müssen. Zusätzlich lassen sich auch persönliche Druckeinstellungen für spezifische Druckertreiber bewahren, die nicht standardmäßig vorgegeben sind.

Das generierte BLOB darf nicht per Programmierung verändert werden und ist nur für diese Funktion verwendbar.

Print settings to BLOB gibt 1 zurück, wenn das BLOB korrekt generiert wurde; 0, wenn kein aktueller Drucker ausgewählt ist.

Windows / OS X

Das BLOB *Druckeinstellungen* lässt sich auf beiden Plattformen sichern und einlesen. Es gibt allgemeingültige Druckeinstellungen, aber auch plattformspezifische, die sich nach dem Treiber und dem Betriebssystem richten. Nutzen beide Plattformen das gleiche BLOB *Druckeinstellungen*, können Teilinformationen verlorengehen.
Wir empfehlen, bei Verwendung in einer heterogenen Umgebung pro Plattform ein BLOB *Druckeinstellungen* einzurichten, damit für jede Plattform nicht nur die allgemeinen, sondern die maximalen Einstellungen verfügbar sind.

Beispiel

Die aktuellen Druckeinstellungen auf der Festplatte speichern:

```
C_BLOB(curSettings)
PRINT SETTINGS  //zeigt das Dialogfenster Druckeinstellungen für den Benutzer
If(OK=1)
    $err:=Print settings to BLOB(curSettings)
    If($err=1)
        BLOB TO DOCUMENT(Get 4D folder+"current4Dsettings.blob";curSettings)
    Else
        ALERT("Kein Drucker ausgewählt")
    End if
End if
```

PRINTERS LIST

PRINTERS LIST (NamenArray {; AltNamenArray {; ModelleArray}})

Parameter	Typ		Beschreibung
NamenArray	Array Text	⇐	Druckernamen
AltNamenArray	Array Text	⇐	Windows: Druckerort, Mac OS: Eigene Druckernamen
ModelleArray	Array Text	⇐	Druckermodelle

Beschreibung

Der Befehl **PRINTERS LIST** füllt die als Parameter übergebenen Array(s) mit den Namen, sowie optional Ort, eigenen Druckernamen und den für den Rechner verfügbaren Druckermodellen.

Hinweis: Werden die Drucker über einen Druckertreiber (Spooler) verwaltet, wird unter Windows der vollständige Zugriffspfad, auf Mac OS der Name des Spoolers zurückgegeben.

Im Parameter *NamenArray* übergeben Sie den Namen des Textarrays. Es enthält nach Ausführen des Befehls die Namen der verfügbaren Drucker. Auf Mac OS sind es die festgelegten "Systemnamen". Sie können ein zweites optionales Array *AltNamenArray* übergeben. Sein Inhalt richtet sich nach der Plattform:

- Unter Windows erhalten Sie für jeden Drucker seine Platzierung im Netzwerk oder das lokale Port.
- Auf Mac OS erhalten Sie für jeden Drucker seinen eigenen Namen, den der Benutzer verändern kann. Dieser Name lässt sich z.B. in Dialogfenstern verwenden.

Über den optionalen Parameter *ModelleArray* erhalten Sie das Modell jedes Druckers.
Hinweis: Dieser Parameter wird in 32-bit Versionen von 4D für Mac nicht unterstützt.

Über die Routinen **SET CURRENT PRINTER** und **Get current printer** können Sie den gewählten Drucker in 4D aufrufen bzw. ändern. Hier müssen Sie die Namen übergeben, die im ersten Array zurückgegeben werden (*NamenArray*).

Unter Windows können Sie den Druckernamen manuell auf Ebene des Betriebssystems ändern. Ort und Modell hängen auch von physikalischen Merkmalen ab. Deshalb können Sie über die optionalen Arraywerte die Merkmale des gewählten Druckers prüfen — z.B., dass alle Client-Rechner denselben Drucker verwenden.

Auf Mac OS können Sie die Überprüfung über den Druckernamen (Name des Druckservers) ausführen. Dieser ist für alle angeschlossenen Rechner gleich.

Systemvariablen und Mengen

Wird der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null), die zurückgegebenen Arrays sind dann leer.

Printing page

Printing page -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl	➲	Seitennummer der gerade druckenden Seite

Beschreibung

Die Funktion **Printing page** gibt die Seitennummer der Seite zurück, welche gerade gedruckt wird. Mit dem zurückgegebenen Wert können Sie die gedruckten Seiten automatisch nummerieren. Sie kann nur beim Drucken mit dem Befehl **PRINT SELECTION** oder dem Menü **Drucken** in der Anwendungsumgebung verwendet werden.

Beispiel

Folgendes Beispiel ändert die Position der Seitennummern in einem Bericht, damit er doppelseitig gedruckt wird. Die Vorlage für den Bericht enthält zwei Variablen zum Anzeigen der Seitennummern. Die Variable links unten (*vLeftPageNum*) druckt die geraden Seiten, die Variable rechts unten (*vRightPageNum*) die ungeraden Seiten. Das Beispiel prüft die geraden Seiten und löscht bzw. setzt dann die entsprechende Variable:

```
Case of
   :(Form event=On Printing Footer)
      If((Printing page% 2)=0) ` Modulo ist 0, es ist eine gerade Seite
         vLeftPageNum:=String(Printing page) ` Setze linke Seitennummer
         vRightPageNum:="" ` Lösche rechte Seitennummer
      Else ` Sonst ist es eine ungerade Seiten
         vLeftPageNum:="" ` Lösche linke Seitennummer
         vRightPageNum:=String(Printing page) ` Setze rechte Seitennummer
      End if
End case
```

SET CURRENT PRINTER

SET CURRENT PRINTER (DruckerName)

Parameter	Typ		Beschreibung
DruckerName	String	➡	Name des gewünschten Druckers

Beschreibung

Der Befehl **SET CURRENT PRINTER** bezeichnet den Drucker, der zum Drucken der aktuellen 4D Anwendung verwendet werden soll.

Im Parameter *DruckerName* übergeben Sie den Namen des gewünschten Druckers. Die Liste der wählbaren Drucker erhalten Sie über den Befehl **PRINTERS LIST**.

Übergeben Sie in *DruckerName* einen leeren String, wird der im System definierte Drucker verwendet.

SET CURRENT PRINTER ermöglicht zum Drucken von PDFs, den generischen PDF Drucker vom System anzugeben. Der entsprechende Wert richtet sich nach der 4D Version und der Version des Betriebssystems.

- **Windows 8 und vorherige Versionen:**
  gibt als Druckziel den vom Druckertreiber PDFCreator installierten virtuellen Drucker an. Das vereinfacht das Drucken von PDF Dokumenten unter Windows (siehe **Integration des Treibers PDF Creator unter Windows**). Um ein PDF Dokument mit dem Parameter *DruckerName* zu drucken, übergeben Sie den Namen des virtuellen Druckers, der vom Druckertreiber PDFCreator installiert wurde. Er lautet standardmäßig "PDFCreator", kann jedoch beim Installieren des Treibers u.U. verändert worden sein. Damit 4D automatisch danach sucht und den virtuellen Drucker verwendet, auch wenn er angepasst wurde, übergeben Sie in *DruckerName* folgende Konstante unter dem Thema **Druckoptionen**:

Konstante	Typ	Wert	Kommentar
PDFCreator Printer name	Zeichenkette	PDFCreator	Anzeige des Druckernamens

- **Ab Windows 10:**
  Windows 10 enthält einen native PDF Druckertreiber, über den 4D direkt PDFs erstellen kann. Es ist kein Treiber von Drittherstellern erforderlich, wie z.B. PDFCreator.
  Der Name des Treibers ist "Microsoft Print to PDF" (siehe Beispiel im Abschnitt **Integration des Treibers PDF Creator unter Windows**).

- **Auf OS X und ab Windows 10 (4D v15 R5 64-bit und höher):**
  Über eine Konstante unter dem Thema **Druckoptionen** können Sie automatisch den generischen PDF Drucker angeben, unabhängig von der Plattform. Das vereinfacht das Schreiben von generischem Code.

Konstante	Typ	Wert	Kommentar
Generic PDF driver	Zeichenkette	_4d_pdf_printer	**Hinweis:** Diese Funktionalität ist in 32 bit Versionen von 4D nicht verfügbar - Setzt auf OS X den aktuellen Drucker auf den standardmäßigen Druckertreiber. Er ist nicht sichtbar und wird nicht von **PRINTERS LIST** zurückgegeben. Beachten Sie, dass über **SET PRINT OPTION** ein PDF Dokumentpfad gesetzt sein muss, andernfalls wird Fehler 3107 zurückgegeben. - Setzt unter Windows den aktuellen Drucker auf den Windows PDF Druckertreiber (genannt "Microsoft Print to PDF"). Diese Option ist nur unter Windows 10 mit installierter PDF Option verfügbar. In anderen Windows-Versionen oder wenn kein PDF Treiber verfügbar ist, führt der Befehl nichts aus und die Systemvariable OK wird auf 0 gesetzt.

Sie müssen zuerst **SET CURRENT PRINTER** und dann **SET PRINT OPTION** aufrufen, so dass die verfügbaren Optionen zum gewählten Drucker passen. Außerdem müssen Sie **SET CURRENT PRINTER** nach **PAGE SETUP** aufrufen, sonst gehen die Druckeinstellungen verloren.

Sie können diesen Befehl zusammen mit den Befehlen **PRINT SELECTION**, **PRINT RECORD**, **Print form** und **QR REPORT** verwenden, er gilt für alle Druckvorgänge von 4D, inkl. im Designmodus. Sie müssen Druckbefehle, wo anwendbar, immer mit dem Parameter > aufrufen, damit die angegebenen Einstellungen erhalten bleiben.

Systemvariablen und Mengen

Wurde der Drucker korrekt ausgewählt, wird die Systemvariable OK auf 1 gesetzt. In anderen Fällen, z.B. wenn der angegebene Drucker nicht gefunden wird, wird die Systemvariable OK auf 0 gesetzt. Der aktuelle Drucker wird beibehalten.

Beispiel

Mit 4D Developer Edition 64-bit ein PDF Dokument under Windows 10 erstellen:

```
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 (MarkeNum ; MarkePos {; *})

Parameter	Typ		Beschreibung
MarkeNum	Lange Ganzzahl	⇒	Nummer der Marke
MarkePos	Lange Ganzzahl	⇒	Neue Position der Marke
*	Operator	⇒	Mit Angabe = Nachfolgende Marke bewegen Ohne Angabe = Nachfolgende Marke nicht bewegen

Beschreibung

Der Befehl **SET PRINT MARKER** definiert die Position der Marke während des Druckvorgangs. In Kombination mit den Befehlen **Get print marker** und **OBJECT MOVE** können Sie die Größe des Druckbereichs einstellen.

SET PRINT MARKER lässt sich in zwei Kontexten verwenden:

- In den Befehlen **PRINT SELECTION** und **PRINT RECORD** während des Formularereignisses On header.
- In der Funktion **Print form** während des Formularereignisses On Printing Detail. Diese Operation erleichtert das Drucken eigener Berichte (siehe Beispiel).
  Der Befehl wirkt sich nur auf das Drucken aus. Die Änderungen erscheinen nicht auf dem Bildschirm und werden auch nicht gesichert.

Die Auswirkung des Befehls beschränkt sich auf das Drucken; auf dem Bildschirm erscheint keine Änderung. In Formularen gemachte Änderungen werden nicht gesichert.

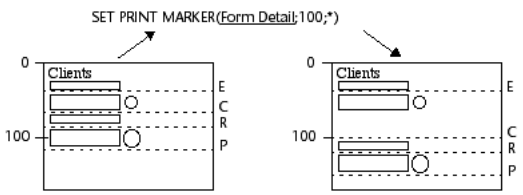
Im Parameter *MarkeNum* setzen Sie eine Konstante aus dem Thema **Formularbereich** ein:

Konstante	Typ	Wert
Form break0	Lange Ganzzahl	300
Form break1	Lange Ganzzahl	301
Form break2	Lange Ganzzahl	302
Form break3	Lange Ganzzahl	303
Form break4	Lange Ganzzahl	304
Form break5	Lange Ganzzahl	305
Form break6	Lange Ganzzahl	306
Form break7	Lange Ganzzahl	307
Form break8	Lange Ganzzahl	308
Form break9	Lange Ganzzahl	309
Form detail	Lange Ganzzahl	0
Form footer	Lange Ganzzahl	100
Form header	Lange Ganzzahl	200
Form header1	Lange Ganzzahl	201
Form header10	Lange Ganzzahl	210
Form header2	Lange Ganzzahl	202
Form header3	Lange Ganzzahl	203
Form header4	Lange Ganzzahl	204
Form header5	Lange Ganzzahl	205
Form header6	Lange Ganzzahl	206
Form header7	Lange Ganzzahl	207
Form header8	Lange Ganzzahl	208
Form header9	Lange Ganzzahl	209

In *MarkePos* geben Sie die gewünschte neue Position der Marke in Pixel an.

Ist der optionale Parameter * übergeben, werden alle Marken, die auf die in *MarkeNum* angegebene Marke folgen, dieselbe Anzahl Pixel und in dieselbe Richtung bewegt.

Achtung: In diesem Fall werden alle darunter liegenden Objekte ebenfalls bewegt.
Mit dem Parameter * können Sie die Position der Marke *MarkeNum* über die ursprüngliche Position nachfolgender Marken bewegen — nachfolgende Marken werden simultan verschoben.

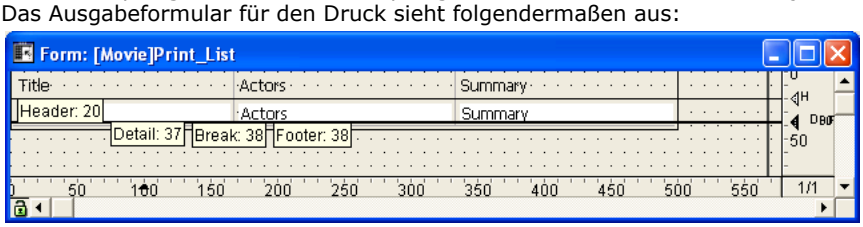


Hinweise:

- Dieser Befehl ändert nur die Position einer bereits vorhandenen Marke. Sie können damit keine neue Marke hinzufügen. Verweisen Sie auf eine Marke, die im Formular nicht existiert, führt der Befehl nichts aus.
- Die Funktionsweise der Druckmarken im Designmodus wird beibehalten, d.h. die Marke kann nicht höher als die davorliegende, oder tiefer als die darauffolgende Marke liegen (ohne Angabe des Parameters *).

Beispiel

Dieses Beispiel generiert einen dreispaltigen Bericht, die Zeilenhöhe wird je nach Feldinhalt on-the-fly berechnet.
Das Ausgabeformular für den Druck sieht folgendermaßen aus:



Für das Formular wurde das Formularereignis On Printing Detail gewählt.

Zur Erinnerung: Es spielt keine Rolle, welcher Bereich gedruckt wird. Die Funktion **Print form** generiert nur diese Art von Formularereignis.

Die Zeilenhöhe muss für jeden Datensatz angepasst werden an die Spalten "Actors" oder "Summary". Das Ergebnis sieht folgendermaßen aus:

Title	Actors	Summary
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 Captain 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 Burmester, 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 Hurt, 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.

Die Projektmethode für Drucken sieht folgendermaßen aus:

```
C_LONGINT(vLprint_height;$vLheight;vLprinted_height)
C_STRING(31;vSprint_area)
PAGE SETUP([Film];"Print_List3")
GET PRINTABLE AREA(vLprint_height)
vLprinted_height:=0
ALL RECORDS([Film])

vSprint_area:="Header"  //Kopfbereich drucken
$vLheight:=Print form([Film];"Print_List3";Form header)
$vLheight:=21  //Feste Höhe
vLprinted_height:=vLprinted_height+$vLheight

While(Not(End selection([Film])))
   vSprint_area:="Detail"  `Detailbereich drucken
   $vLheight:=Print form([Film];"Print_List3";Form detail)
 `Detailberechnung wird in der Formularmethode ausgeführt
   vLprinted_height:=vLprinted_height+$vLheight
   If(OK=0)  `CANCEL wurde in der Formularmethode ausgeführt
      PAGE BREAK
```

```
        vLprinted_height:=0
        vSprint_area:="Header"  `Kopfbereich erneut drucken
        $vLheight:=Print form([Film];"Print_List3";Form header)
        $vLheight:=21
        vLprinted_height:=vLprinted_height+$vLheight
        vSprint_area:="Detail"
        $vLheight:=Print form([Film];"Print_List3";Form detail)
        vLprinted_height:=vLprinted_height+$vLheight
    End if
    NEXT RECORD([Film])
End while
PAGE BREAK  `Sicherstellen, dass die letzte Seite gedruckt wird
```

Die Formularmethode Print_List3 lautet:

```
C_LONGINT($l;$t;$r;$b;$fixed_wdth;$exact_hght;$l1;$t1;$r1;$b1)
C_LONGINT($final_pos;$i)
C_LONGINT($detail_pos;$header_pos;$hght_to_print;$hght_remaining)

Case of
    :(vSprint_area="Detail")  `Detailbereich wird gedruckt
        OBJECT GET COORDINATES([Film]Actors;$l;$t;$r;$b)
        $fixed_wdth:=$r-$l  `Berechnet die Größe des Textfeldes Actors
        $exact_hght:=$b-$t
        OBJECT GET BEST SIZE([Film]Actors;$wdth;$hght;$fixed_wdth)
  `Größe des Feldes wird passend zum Inhalt optimiert
        $movement:=$hght-$exact_hght

        OBJECT GET COORDINATES([Film]Summary;$l1;$t1;$r1;$b1)
        $fixed_wdth1:=$r1-$l1  `Berechnet die Größe des Textfeldes Summary
        $exact_hght1:=$b1-$t1
        OBJECT GET BEST SIZE([Film]Summary;$wdth1;$hght1;$fixed_wdth1)
  `Größe des Feldes wird passend zum Inhalt optimiert
        $movement1:=$hght1-$exact_hght1
        If($movement1>$movement)
  `Das höchste Feld festlegen
            $movement:=$movement1
        End if

        If($movement>0)
            $position:=Get print marker(Form detail)
            $final_pos:=$position+$movement
  `Die Marke Detail und nachfolgende Marken bewegen
            SET PRINT MARKER(Form detail;$final_pos;*)
  `Textbereiche anpassen
            OBJECT MOVE([Film]Actors;$l;$t;$r;$hght+$t;*)
            OBJECT MOVE([Film]Summary;$l1;$t1;$r1;$hght1+$t1;*)

  `Trennlinien anpassen
            OBJECT GET COORDINATES(*;"H1Line";$l;$t;$r;$b)
            OBJECT MOVE(*;"H1Line";$l;$final_pos-1;$r;$final_pos;*)
            For($i;1;4;1)
                OBJECT GET COORDINATES(*;"VLine"+String($i);$l;$t;$r;$b)
                OBJECT MOVE(*;"VLine"+String($i);$l;$t;$r;$final_pos;*)
            End for
        End if

  `Verfügbaren Platz berechnen
        $detail_pos:=Get print marker(Form detail)
        $header_pos:=Get print marker(Form header)
        $hght_to_print:=$detail_pos-$header_pos
        $hght_remaining:=printing_height-vLprinted_height
        If($hght_remaining<$hght_to_print)  `Höhe ist zu gering
            CANCEL  `Formular auf die nächste Seite setzen
        End if
End case
```

SET PRINT OPTION

SET PRINT OPTION (Option ; Wert1 {; Wert2})

Parameter	Typ		Beschreibung
Option	Lange Ganzzahl	➡	Optionsnummer oder PDF Optionscode
Wert1	Lange Ganzzahl, Text	➡	Wert 1 der Option
Wert2	Lange Ganzzahl, Text	➡	Wert 2 der Option

Beschreibung

Der Befehl **SET PRINT OPTION** verändert per Programmierung den Wert der Druckoption. Jede über diesen Befehl definierte Option gilt für die gesamte Datenbank und die Dauer der Sitzung, solange kein anderer Befehl aufgerufen wird, der die Druckparameter verändert, z.B. **PRINT SETTINGS**, **PRINT SELECTION** ohne den Parameter >. Wurde ein Druckauftrag geöffnet, wird die Option gesetzt und kann erst nach Beenden des Druckauftrags verändert werden.

Der Parameter *Option* gibt die zu verändernde Option an. Sie können eine vordefinierte Konstante unter dem Thema **Druckoptionen** übergeben oder den Code einer PDF Option (nur mit dem Treiber PDFCreator unter Windows).

In *Wert1* und optional *Wert2* übergeben Sie den bzw. die neuen Werte der definierten *Option*. Die Anzahl und Art der zu übergebenden Werte richtet sich nach der Art der angegebenen Option.

Optionsnummer (Konstante) verwenden

Nachfolgende Liste zeigt die Optionen und ihre möglichen Werte:

Konstante	Typ	Wert	Kommentar
Paper option	Lange Ganzzahl	1	Verwenden Sie nur *Wert1*, enthält er den Namen des Papierformats. Verwenden Sie beide Parameter, enthält *Wert1* die Papierbreite und *Wert2* die Papierhöhe. Breite und Höhe werden in Pixel auf dem Bildschirm angegeben. Über den Befehl **PRINT OPTION VALUES** erhalten Sie Name, Höhe und Breite aller Papierformate, die der Drucker anbietet.
Orientation option	Lange Ganzzahl	2	Nur *Wert1*: 1=Hochformat, 2=Querformat. Bei einer anderen Ausrichtung gibt **GET PRINT OPTION** 0 in *Wert1* zurück. **64-bit Versionen**: Diese Option lässt sich im Druckauftrag aufrufen, d.h. Sie können im gleichen Auftrag zwischen Hoch- und Querformat wechseln.
Scale option	Lange Ganzzahl	3	nur *Wert1*: Skalierungswert in Prozent. Bedenken Sie jedoch, dass einige Drucker keine Skalierung zulassen. Übergeben Sie einen ungültigen Wert, wird die Eigenschaft beim Drucken auf 100% gesetzt.
Number of copies option	Lange Ganzzahl	4	nur *Wert1*: Anzahl der Kopien zum Drucken
Paper source option	Lange Ganzzahl	5	(nur Windows) nur *Wert1*: Nummer, die dem Index des zu verwendenden Papierschachts entspricht. Über den Befehl **PRINT OPTION VALUES** erhalten Sie das Array mit den Namen. Diese Option ist nur unter Windows verwendbar.
Color option	Lange Ganzzahl	8	(nur Windows) nur *Wert1*: Code zum Verwalten der Farbe: 1=schwarz/weiß (monochrome), 2=Farbe. **64-bit Versionen**: Diese Option wird in 4D 64-bit Versionen nicht unterstützt (überholt)
Destination option	Lange Ganzzahl	9	*Wert1*: Code für Druckausgabe: 1=Drucker, 2=(PC)/PS File (Mac), 3=PDF Datei, 5=Bildschirm (OS X Treiber). Ist *Wert1* ungleich 1 oder 5, enthält *Wert2* den Pfadnamen des Ergebnisdokuments. Dieser Pfad wird benützt, bis ein anderer Pfad angegeben wird. Gibt es bereits eine gleichnamige Datei am Zielort, wird sie ersetzt. Ist der aktuelle Wert nicht in der vordefinierten Liste, enthält *Wert1* mit **GET PRINT OPTION** -1 und die Systemvariable OK wird auf 1 gesetzt. Tritt ein Fehler auf, werden *Wert1* und die Systemvariable OK auf 0 gesetzt. **Hinweise**: Unter Windows können Sie das Druckziel auf 3 (PDF-Datei) setzen, wenn der Treiber PDF Creator installiert ist. Werden die Werte (9;3;Pfad) übergeben, startet 4D automatisch ein "stilles" PDF Drucken, das alle übergebenen Codes für Option berücksichtigt. Übergeben Sie in *Wert2* einen leeren String oder lassen diesen Parameter weg, erscheint beim Drucken ein Sichern-Dialog. Nach dem Drucken wird wieder auf die aktuellen Einstellungen zurückgesetzt. Das vereinfacht das Steuern von Drucken als PDF für 4D und ermöglicht das Schreiben von Multiplattform Code. Sind die Werte (9;3;Pfad) nicht übergeben, wird das Drucken nicht durch 4D gesteuert und alle für PDF Creator übergebenen Optionen werden ignoriert.
Double sided option	Lange Ganzzahl	11	(nur Windows) *Wert1*: 0=Einseitig oder Standard), 1=Doppelseitig. Ist *Wert1*=1, enthält *Wert2* die Bindung: 0=Bindung links (Standard), 1=Bindung oben. **Hinweis:** Diese Option ist nur unter Windows verwendbar.
Spooler document name option	Lange Ganzzahl	12	nur *Wert1*: Namen des aktuellen Druckdokuments, das in der Dokumentenliste des Druck-Servers erscheint. Der hier definierte Name wird für alle Druckdokumente der Sitzung verwendet, solange kein neuer Name oder ein leerer String übergeben wird. Um die Standardoperation wiederherzustellen (Methodenname statt Methode, Tabellenname für Datensatz, etc.), übergeben Sie in *Wert1* einen leeren String.
Mac spool file format option	Lange Ganzzahl	13	(nur Mac) nur *Wert1*: 0= Druckauftrag im PDF-Modus (Standardwert) 1=Druckauftrag im PostScript Modus **Hinweise:** - Diese Option hat keine Auswirkung unter Windows. - Auf Mac OS X wird standardmäßig im PDF-Modus gedruckt. Der PDF-Treiber unterstützt jedoch keine PICT-Bilder mit eingebundener PostScript Information – diese Bilder werden über vektororientierte Programme erstellt. Um dieses Problem zu vermeiden, können Sie über diese Option den Druckmodus für die aktuelle Sitzung auf Mac OS X verändern. Bedenken Sie, dass das Drucken im PostScript Modus unerwünschte Nebenwirkungen haben kann. **64-bit Versionen:** Diese Option wird nicht unterstützt; sie wird ersetzt durch die Option Generic PDF driver des Befehls **SET CURRENT PRINTER**.
Hide printing progress option	Lange Ganzzahl	14	(nur Mac) Nur *Wert1*: 1=Fenster mit dem Druckverlauf ausblenden, 0=Fenster mit dem Druckverlauf anzeigen (Standard). Diese Option ist besonders hilfreich beim Drucken von PDF auf OS X im Hintergrund. **Hinweis**: Es gibt bereits eine Option Druckverlauf im Dialogfenster Einstellungen (Seite **Anwendung/Optionen**). Diese gilt jedoch global für die Anwendung und blendet nicht alle Fenster unter Mac OS X aus.
Page range option	Lange Ganzzahl	15	*Wert1*=erste Seite zum Drucken (Standardwert ist 1) und optional *Wert2*=letzte Seite zum Drucken (Standardwert -1 = Dokumentende)
Legacy printing layer option	Lange Ganzzahl	16	(nur 4D 64-bit Versionen für Windows) nur *Wert1*: 1=auf GDI basierende bisherige Druckebene für nachfolgende Druckaufträge auswählen. 0=die D2D Druckerebene verwenden (Standard) **64-bit Versionen**: Selector wird nur in 4D 64-bit Versionen für Windows (Einzelplatz) unterstützt: auf anderen Plattformen wird er ignoriert. Er dient hauptsächlich dazu, damit bisherige Plug-Ins in 4D Druckaufträgen in 4D 64-bit Anwendungen drucken können.

Die hier definierte Druckoption gilt für alle Druckdokumente und wird für die Dauer der Sitzung und für die komplette 4D Anwendung beibehalten. Sie wird von den Befehlen **PRINT SELECTION**, **PRINT RECORD**, **Print form** und **QR REPORT** verwendet und für alle Druckvorgänge in 4D, inkl. Designmodus.

Hinweise:

- Bei den Befehlen **PRINT SELECTION**, **PRINT RECORD** und **PAGE BREAK** müssen Sie den optionalen Parameter ">" verwenden. Nur so können Sie verhindern, dass die mit **SET PRINT OPTION** definierten Druckeinstellungen neu gesetzt werden.

- Der Befehl **SET PRINT OPTION** unterstützt hauptsächlich PostScript Drucker. Sie können ihn auch mit anderen Druckertypen, wie PCL oder Ink verwenden, dann sind jedoch u.U. einige Optionen nicht verfügbar.

PDF Optionscode (Windows) verwenden

Damit Sie im Parameter *Option* einen PDF Optionscode verwenden können, müssen Sie in Ihrer 4D Umgebung den Treiber PDFCreator installiert haben. Weitere Informationen dazu finden Sie im Abschnitt **Integration des Treibers PDF Creator unter Windows**. Darüberhinaus müssen Sie über folgende Anweisung die Steuerung zum Drucken von PDF für 4D aktivieren:

```
SET PRINT OPTION(Destination option;3;fileName)
```

Andernfalls werden die Optionscodes ignoriert.

Der Parameter *Option* ist ein Wert vom Typ Text, der aus zwei miteinander kombinierten Teilen besteht: "OptionTyp:OptionName". Dieser Code wird nachfolgend beschrieben:

- *OptionTyp* gibt an, ob Sie eine native PDF Creator Option oder eine 4D PDF Administration Option spezifizieren. Es gibt zwei Werte:
  - PDFOptions = native Option
  - PDFInfo = interne Option

- *OptionName* gibt die zu setzende Option an (abhängig vom Wert *OptionTyp*)
  - Ist *OptionTyp* = **PDFOptions**, können Sie in *OptionName* eine der native Optionen von PDFCreator übergeben. Die Option UseAutosave betrifft z.B. das automatische Backup. Um diese Option ändern zu können, übergeben Sie im Parameter Option "PDFOptions:UseAutosave" und im Parameter *Wert1* den Wert, der verwendet werden soll. Eine ausführliche Beschreibung der native Optionen von PDFCreator finden Sie in der Dokumentation zum Treiber PDFCreator.
  - Ist *OptionTyp* = **PDFInfo**, können Sie in *OptionName* einen der nachfolgenden Selektoren übergeben:
    - **Reset print** setzt den internen Wartestatus zurück, insbesondere, um aus einer unendlichen Schleife herauszukommen. In diesem Fall wird *Wert1* nicht verwendet.
    - **Reset standard options** setzt alle Optionen von PDFCreator auf ihre Standardwerte zurück. Läuft gerade ein Druckvorgang, werden die Einstellungen nach Beenden des Druckens angewandt. In diesem Fall wird *Wert1* nicht verwendet.
    - **Start** startet oder stoppt den Spooler von PDFCreator. Zum Stoppen übergeben Sie 0 (Null) in *Wert1*, zum Starten 1.
    - **Reset options** setzt alle Optionen, die seit Beginn der Sitzung über den Befehl **SET PRINT OPTION** und den Selektor **PDFOptions** verändert wurden, wieder zurück.
    - **Version** liest die aktuelle Versionsnummer des Treibers PDFCreator. Dieser Selektor ist nur mit dem Befehl **GET PRINT OPTION** verwendbar. Die Nummer wird im Parameter *Wert1* zurückgegeben.
    - **Last error** liest den zuletzt vom Treiber PDFCreator zurückgegebenen Fehler. Dieser Selektor ist nur mit dem Befehl **GET PRINT OPTION** verwendbar. Die Fehlernummer wird im Parameter *Wert1* zurückgegeben.
    - **Print in progress** stellt fest, ob 4D auf den Druck mit PDFCreator wartet. Dieser Selektor ist nur mit dem Befehl **GET PRINT OPTION** verwendbar. *Wert1* gibt 1 zurück, wenn 4D auf PDFCreator wartet, sonst 0.
    - **Job count** findet die Anzahl der Aufträge, die in der Druckerschleife warten, heraus. Dieser Selektor ist nur mit dem Befehl **GET PRINT OPTION** verwendbar. Die Anzahl der Aufträge wird in *Wert1* zurückgegeben.
    - **Synchronous Mode** setzt den Synchronisierungsmodus zwischen den von 4D gesendeten Druckanfragen und dem Treiber PDFCreator. Da 4D keine Information über den aktuellen Status eines Druckauftrags erhalten kann, der sich in der Warteschleife befindet, können Sie über diese Option die Druckausführung besser steuern, indem Sie Aufträge nur senden, wenn der Status des Treibers PDFCreator "frei" ist. In diesem Fall wird 4D mit dem Treiber synchronisiert. Übergeben Sie 0 (Null) in *Wert1*, wenn 4D Druckanfragen sofort senden soll (Standardeinstellung) und 1, wenn 4D synchronisiert werden und warten soll, bis der Treiber den gerade laufenden Auftrag beendet hat, ehe es einen anderen sendet.

Hinweis: Nach jedem Drucken stellt 4D automatisch wieder die vorige Einstellungen des Treibers PDFCreator her, um Überschneidungen mit anderen Programmen zu vermeiden.

Beispiel 1

Nachfolgende Methode konfiguriert den PDF Treiber, so dass alle Datensätze der Tabelle an der Stelle C:\Test_PDF_X gedruckt werden, wobei X die Sequenznummer des Datensatzes ist:

```
SET CURRENT PRINTER(PDFCreator Printer Name)
 // Verwenden Sie unter Windows den von PDFCreator installierten virtuellen Drucker
If(OK=1)  // ist PDFCreator derzeit installiert

   ALL RECORDS([Table_1])
   For($i;1;Records in selection([Table_1]))
      SET PRINT OPTION(Destination option;3;"C:\\Test\\Test_PDF_"+String($i))
 // Zieloption 3 startet einen PDFCreator Druckauftrag
      PRINT RECORD([Table_1];*)
      NEXT RECORD([Table_1])
   End for
 // Optionen des Treibers PDFCreator zurücksetzen
   SET PRINT OPTION("PDFInfo:Reset standard options";0)
End if
```

Beispiel 2

In 64-bit Versionen lässt sich der Wert von Orientation option innerhalb des gleichen Druckauftrags verändern (Sonderfall). Beachten Sie, dass die Option vor dem Befehl **PAGE BREAK** gesetzt werden muss:

```
ALL RECORDS([People])
PRINT SETTINGS
If(OK=1)
    OPEN PRINTING JOB
    SET PRINT OPTION(Orientation option;1) //Hochformat
    Print form([People];"Vertical_Form")

    SET PRINT OPTION(Orientation option;2) //Querformat
    PAGE BREAK //muss zwingend NACH der Option aufgerufen werden
    Print form([People];"Horiz_Form")
    CLOSE PRINTING JOB
End if
```

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).
Übergeben Sie einen ungültigen Optionscode, z.B. einen Code, der vom PDFCreator nicht erkannt wird, wird OK auf 0 gesetzt.

Fehlerverwaltung

Ist der für eine Option übergebene Wert ungültig oder ist *Option* für den Drucker nicht verfügbar, gibt der Befehl einen Fehler zurück. Sie können ihn ausfindig machen mit einer Fehlerverwaltungsmethode, die der Befehl **ON ERR CALL** aufruft. Der aktuelle Wert der Option bleibt dann unverändert.

SET PRINT PREVIEW

SET PRINT PREVIEW (Zustand)

Parameter	Typ		Beschreibung
Zustand	Boolean	⇒	Ausdruck auf Bildschirm (TRUE) oder kein Ausdruck auf Bildschirm (FALSE)

Beschreibung

Der Befehl **SET PRINT PREVIEW** aktiviert oder deaktiviert das Kontrollkästchen "Ausdruck auf Bildschirm" im Druckdialog. Ist *Zustand* wahr, ist Ausdruck auf Bildschirm aktiv, ist *Zustand* falsch, ist Ausdruck auf Bildschirm inaktiv. Der Befehl ist nur für den aktuellen Prozess gültig. Er gilt, bis er vom Anwender im Druckdialog oder durch erneuten Aufruf geändert wird.

Beispiel

Folgendes Beispiel aktiviert die Option "Ausdruck auf Bildschirm", um das Ergebnis einer Suche auf dem Bildschirm anzuzeigen. Die Option wird anschließend deaktiviert.

```
QUERY([Customers])
If(OK=1)
   SET PRINT PREVIEW(True)
   PRINT SELECTION([Customers];*)
   SET PRINT PREVIEW(False)
End if
```

SET PRINTABLE MARGIN

SET PRINTABLE MARGIN (Links ; Oben ; Rechts ; Unten)

Parameter	Typ		Beschreibung
Links	Lange Ganzzahl	➡	Linker Rand
Oben	Lange Ganzzahl	➡	Oberer Rand
Rechts	Lange Ganzzahl	➡	Rechter Rand
Unten	Lange Ganzzahl	➡	Unterer Rand

Beschreibung

Der Befehl **SET PRINTABLE MARGIN** setzt die Werte der verschiedenen Ränder über die Befehle **Print form**,**PRINT SELECTION** und **PRINT RECORD**.

In den Parametern *Links*, *Oben*, *Rechts* und *Unten* sind folgende Werte möglich:

- 0 = Verwende Papierränder
- -1 = Verwende Druckränder
- Wert > 0 = Rand in Pixel (Zur Erinnerung: 1 Pixel in 72 dpi entspricht ca. 0.4 mm). Die Werte für *Rechts* und *Unten* beziehen sich jeweils auf den rechten und unteren Papierrand.

Hinweis: Weitere Informationen zur Druckverwaltung und 4D Terminologie finden Sie unter dem Befehl **GET PRINTABLE MARGIN**.

4D richtet den Ausdruck standardmäßig an den Druckrändern aus. Wurde der Befehl **SET PRINTABLE MARGIN** ausgeführt, werden die geänderten Parameter für die gesamte Sitzung im selben Prozess beibehalten.

Beispiel 1

Folgendes Beispiel ermittelt die Größe des nicht druckbaren Bereichs:

```
SET PRINTABLE MARGIN(-1;-1;-1;-1)  `Setzt den druckbaren Bereich
GET PRINTABLE MARGIN($l;$t;$r;$b)
  `$l, $t, $r und $b definieren den nicht bedruckbaren Bereich des Blatts.
```

Beispiel 2

Folgendes Beispiel ermittelt die Größe des Papiers:

```
SET PRINTABLE MARGIN(0;0;0;0)  `Setze Papierrand
GET PRINTABLE AREA($height;$width)
  `Für A4 Format: $height=842 ; $width=595 Pixel
```

Subtotal

Subtotal (Feldname {; Seitenumbruch}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Feld, über welches die Zwischensumme gebildet wird
Seitenumbruch	Lange Ganzzahl	➡	Ebene, auf der ein Seitenumbruch erfolgen soll
Funktionsergebnis	Zahl	↩	Zwischensumme von Feldname

Beschreibung

Die Funktion **Subtotal** gibt die Zwischensumme des Feldes *Feldname* für die aktuelle oder die letzte Umbruchebene zurück. Sie kann nur beim Drucken mit dem Befehl **PRINT SELECTION** oder dem Menü **Drucken** in der Anwendungsumgebung verwendet werden.
Feldname muss ein Feld vom Typ Zahl, Ganzzahl oder Lange Ganzzahl sein. Die errechnete Zwischensumme wird mit Hilfe einer Variablen im Umbruchbereich des Formulars dargestellt.

Warnung: Im kompilierten Modus **müssen** Sie vor jedem Bericht mit Umbrüchen die Befehle **BREAK LEVEL** und **ACCUMULATE** ausführen, da diese die Berechnung im Umbruch aktivieren. Weitere Informationen dazu finden Sie in der nachfolgenden Beschreibung.

Mit dem optionalen Parameter *Seitenumbruch* rufen Sie Seitenumbrüche während dem Drucken hervor. Ist *Seitenumbruch* gleich Null (0), erzeugt **Subtotal** keinen Seitenumbruch. Ist *Seitenumbruch* gleich 1, erzeugt **Subtotal** einen Seitenumbruch für jede Umbruchebene 1. Ist *Seitenumbruch* gleich 2, erzeugt **Subtotal** einen Seitenumbruch für jede Umbruchebene 1 und 2, usw..

Tipp: Führen Sie **Subtotal** innerhalb eines Ausgabeformulars auf dem Bildschirm aus, erhalten Sie eine Fehlermeldung, ausgelöst durch eine Endlosschleife von Updates zwischen Formular und Fehlerdialog. Sie können diese Schleife verlassen, wenn Sie unter Windows die Kombination **Alt+Umschalttaste**, auf Macintosh die Kombination **Wahl- + Umschalttaste** drücken und im Fehlerdialog gleichzeitig auf die Schaltfläche **Abbrechen** klicken (evtl. mehrmals). Dies stoppt temporär die Aktualisierung für das Fenster des Formulars. Wählen Sie ein anderes Formular als Ausgabeformular, damit der Fehler nicht erneut auftritt. Gehen Sie zurück in die Designumgebung und isolieren Sie den Aufruf von **Subtotal** in einem Test *Form event=On Printing Break*, wenn Sie das Formular sowohl zum Anzeigen als auch zum Ausdrucken verwenden.

Beispiel

Folgendes Beispiel ist eine online Objektmethode innerhalb eines Umbruchbereichs eines Formulars. Die Variable *vSalary* liegt im Umbruchbereich (der Bereich unterhalb der Markierung B0). Ihr wurde die Zwischensumme des Datenfeldes "Salary" zugewiesen. Die Umbruchbearbeitung muss vor den Befehlen **BREAK LEVEL** und **ACCUMULATE** aktiviert werden:

```
Case of
   :(Form event=On Printing Break)
      vSalary:=Subtotal([Employees]Salary)
End case
```

Weitere Informationen zum Erstellen von Formularen mit Kopf- und Umbruchbereichen finden Sie im Abschnitt **Umbrüche verwenden** des Handbuchs *4D Designmodus*.

Berechnungen im Umbruchbereich von Formularberichten aktivieren

Um Berichte mit Umbrüchen zu erstellen, können Sie Berechnungen im Umbruchbereich von Formularberichten mit **BREAK LEVEL** und **ACCUMULATE** hervorrufen.
Sie müssen beide Befehle vor Drucken eines Formularberichts ausführen. Die Funktion **Subtotal** wird zum Anzeigen von Werten in einem Formular benötigt. Sie müssen mindestens nach soviel Ebenen sortieren, wie Umbrüche benötigt werden.

Mit den Befehlen **BREAK LEVEL** und **ACCUMULATE** läuft das Drucken von Berichten folgendermaßen ab:

1. Sie wählen die Datensätze aus, die gedruckt werden sollen.
2. Sie sortieren die Datensätze mit dem Befehl **ORDER BY**. Die Anzahl der Umbruchebenen muss kleiner oder gleich der Anzahl der Sortierebenen sein.
3. Sie führen die Befehle **BREAK LEVEL** und **ACCUMULATE** aus.
4. Sie drucken den Bericht mit dem Befehl **PRINT SELECTION**.

Die Funktion **Subtotal** benötigen Sie, um Werte in einem Formular anzuzeigen.

Eingabe

- ACCEPT
- ADD RECORD
- CANCEL
- DIALOG
- Modified
- MODIFY RECORD
- Old
- REJECT
- *_o_ADD SUBRECORD*
- *_o_MODIFY SUBRECORD*

ACCEPT

ACCEPT
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **ACCEPT** wird in Formular- bzw. Objektmethoden (oder in Unterroutinen) verwendet, um:

- Einen neuen oder geänderten Datensatz oder Unterdatensatz anzunehmen, der mit **ADD RECORD**, **MODIFY RECORD**, **_o_ADD SUBRECORD** oder **_o_MODIFY SUBRECORD** bearbeitet wurde.
- Ein mit dem Befehl **DIALOG** angezeigtes Formular anzunehmen.
- Ein Formular, das eine Datensatzauswahl anzeigt mit **DISPLAY SELECTION** oder **MODIFY SELECTION** verlassen.

ACCEPT führt die gleiche Aktion wie bei Drücken der Eingabetaste aus. Wurde das Formular angenommen, hat die Systemvariable OK den Wert 1.
ACCEPT wird im allgemeinen nach Aufrufen eines Menübefehls ausgeführt bzw. in der Objektmethode einer Schaltfläche "Keine Aktion".

ACCEPT wird oft auch für die Funktion **Open window** als optionale Schließbox eingesetzt. Gibt es in einem Fenster eine Control-Menübox, können Sie per Doppelklick auf die Control-Menübox oder durch Auswahl des Menübefehls **Schließen**, **ACCEPT** oder **CANCEL** in der auszuführenden Methode aufrufen.

ACCEPT kann nicht gestapelt werden. Sollen als Antwort auf ein Ereignis zwei Befehle **ACCEPT** in einer Spalte innerhalb einer Methode ausgeführt werden, hat es dieselbe Wirkung wie das Ausführen eines Befehls.

ADD RECORD

ADD RECORD ({Tabellenname}{;}{*})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle für die Dateneingabe, oder Standardtabelle ohne Angabe
*		➡	Rollbalken ausblenden

Kompatibilität

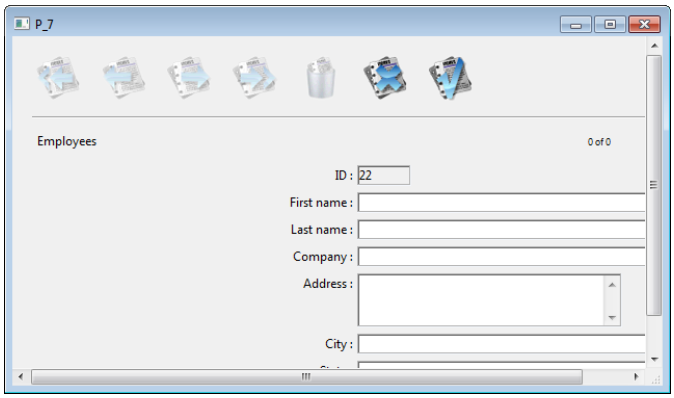
*Dieser Befehl wurde in älteren 4D Releases eingeführt und ist zum Erstellen eines Prototyps oder für grundlegende Entwicklung weiterhin nützlich. Für individuell angepasste und moderne Oberflächen empfehlen wir jedoch, über den Befehl **DIALOG** generische Formulare anzulegen. Denn dieser bietet fortschrittliche Features und eine bessere Kontrolle über den Datenfluss.*

Beschreibung

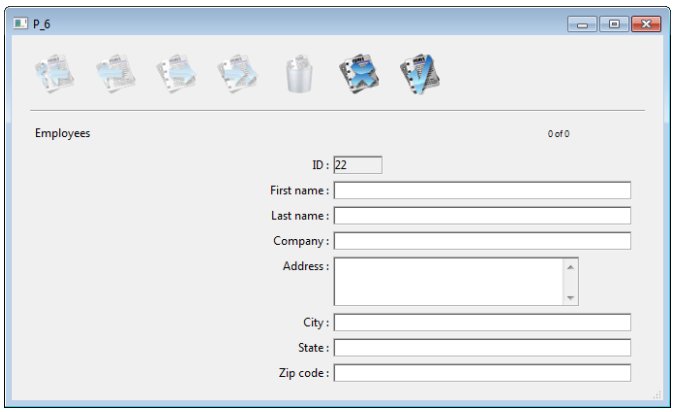
Der Befehl **ADD RECORD** ermöglicht dem Benutzer, für *Tabellenname* einen neuen Datensatz in der Datenbank hinzuzufügen. *Tabellenname* ist optional. Wird dieser Parameter nicht angegeben, wird der Datensatz in der Standardtabelle angelegt.

ADD RECORD erstellt einen neuen Datensatz, macht diesen zum aktuellen Datensatz für den aktuellen Prozess und zeigt ihn im aktuellen Eingabeformular an. Sobald der Benutzer den neuen Datensatz in der Anwendungsumgebung bestätigt, ist er der einzige Datensatz in der aktuellen Auswahl.

Hier sehen Sie ein typisches Eingabeformular:



Das Formular erscheint im obersten Fenster des Prozesses mit Rollbalken und Kästchen für Größeneinstellung. Der optionale Parameter * blendet Rollbalken und Kästchen für Größeneinstellung aus.



ADD RECORD zeigt das Formular an, bis der Benutzer den Datensatz bestätigt oder annulliert. Der Befehl wird für jeden neuen Datensatz einmal ausgeführt.

Der Datensatz wird gesichert, wenn der Benutzer auf die Schaltfläche **Bestätigen** bzw. die **Eingabetaste** im Zahlenblock klickt oder der Befehl **ACCEPT** ausgeführt wird.

Der Datensatz wird nicht gesichert, wenn der Benutzer auf die Schaltfläche **Abbrechen** bzw. die Tastenkombination **strg + Punkt** unter Windows, **Befehl + Punkt** auf Macintosh verwendet oder der Befehl **CANCEL** ausgeführt wird.

Hinweis: Für diesen Befehl muss *Tabellenname* nicht im Lese-/Schreibmodus sein. Er lässt sich auch verwenden, wenn die Tabelle im Nur-Lesen Modus ist (siehe **Überblick zu Datensatz sperren**).

Die Systemvariable OK hat den Wert 1, wenn der Datensatz bestätigt wurde; den Wert 0, wenn er annulliert wurde.

Hinweis: Selbst wenn Sie die Eingabe annullieren, bleibt der Datensatz im Arbeitsspeicher und lässt sich mit **SAVE RECORD** sichern, bevor ein neuer aktueller Datensatz aufgerufen wird.

Beispiel 1

Folgendes Beispiel ist eine gängige Schleife, um in einer Datenbank neue Datensätze hinzuzufügen:

```
FORM SET INPUT([Customers];"Std Input")
  ` Setze Eingabeformular für Tabelle [Customers]
Repeat  ` Durchlaufe, bis der Benutzer abbricht
   ADD RECORD([Customers];*)  ` Füge Datensatz in Tabelle [Customers] hinzu
Until(OK=0)  ` Bis der Benutzer abbricht
```

Beispiel 2

Folgendes Beispiel sucht in der Datenbank nach einem Kunden. Es gibt je nach Suchergebnis zwei Möglichkeiten. Wird kein Kunde gefunden, kann der Benutzer mit **ADD RECORD** einen neuen Kunden hinzufügen. Wird mindestens ein Kunde gefunden, erscheint der zuerst gefundene Datensatz. Der Benutzer kann ihn mit **MODIFY RECORD** ändern:

```
READ WRITE([Customers])
FORM SET INPUT([Customers];"Input")  ` Setze Eingabeformular
vlCustNum:=Num(Request("Gib Kundennr ein:"))  ` Erhalte die Kundennummer
If(OK=1)
   QUERY([Customers];[Customers]CustNo=vlCustNum)  ` Suche nach dem Kunden
   If(Records in selection([Customers])=0)  ` Wird kein Kunde gefunden···
      ADD RECORD([Customers])  ` Füge neuen Kunden hinzu
   Else
      If(Not(Locked([Customers])))
         MODIFY RECORD([Customers])  ` Ändere den Datensatz
         UNLOAD RECORD([Customers])
      Else
         ALERT("Datensatz wird gerade benutzt.")
      End if
   End if
End if
```

Systemvariablen und Mengen

Bestätigen des Datensatzes setzt die Systemvariable OK auf 1, Abbrechen setzt die Systemvariable OK auf 0. Die Systemvariable OK wird nur gesetzt, wenn der Datensatz bestätigt oder annulliert wird.

CANCEL

CANCEL
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **CANCEL** wird in Formular- bzw. Objektmethoden (oder in Unterroutinen) verwendet, um:

- Einen neuen oder geänderten Datensatz oder Unterdatensatz abzubrechen, der mit **ADD RECORD**, **MODIFY RECORD**, **_o_ADD SUBRECORD** oder **_o_MODIFY SUBRECORD** bearbeitet wurde.
- Ein mit dem Befehl **DIALOG** angezeigtes Formular abzubrechen.
- Ein Formular, das eine Datensatzauswahl anzeigt mit **DISPLAY SELECTION** oder **MODIFY SELECTION** verlassen.
- Drucken eines Formulars annullieren, das mit der Funktion **Print form** gedruckt werden soll (siehe unten).

CANCEL führt die gleiche Aktion wie bei Drücken der Kombination **ctrl-Taste+Punkt** unter Windows bzw. **Befehlstaste+Punkt** auf Macintosh aus. Wurde das Formular abgebrochen, hat die Systemvariable OK den Wert 0 (Null).

CANCEL wird im allgemeinen nach Aufrufen eines Menübefehls ausgeführt bzw. in der Objektmethode einer Schaltfläche "Keine Aktion".

CANCEL wird oft auch für die Funktion **Open window** in der optionalen Schließbox eingesetzt. Gibt es in einem Fenster eine Control-Menübox, können Sie per Doppelklick auf die Control-Menübox oder durch Auswahl des Menübefehls Schließen, **ACCEPT** oder **CANCEL** in der auszuführenden Methode aufrufen.

CANCEL kann nicht gestapelt werden. Sollen als Antwort auf ein Ereignis zwei Befehle **CANCEL** in einer Spalte innerhalb einer Methode ausgeführt werden, hat es dieselbe Wirkung wie das Ausführen eines Befehls.

CANCEL lässt sich auch innerhalb der Funktion **Print form** mit dem Formularereignis *On Printing Detail* verwenden. In diesem Kontext widerruft **CANCEL** das Drucken des Formularteils und erlaubt so das Verschieben auf die nächste Seite. Diese Operation eignet sich zur Druckverwaltung von Formularen, wenn Platz fehlt oder ein Seitenumbruch erforderlich ist.

Hinweis: Diese Operation unterscheidet sich vom Befehl **PAGE BREAK**(*). Damit wird der gesamte Druckauftrag abgebrochen.

Beispiel

Siehe Beispiel zum Befehl **SET PRINT MARKER**.

Systemvariablen und Mengen

Wird der Befehl **CANCEL** ausgeführt (Formular oder Drucken abgebrochen), wird die Systemvariable OK auf 0 (Null) gesetzt.

DIALOG

DIALOG ({Tabellenname ;} Formularname {; FormularDaten}{; *})

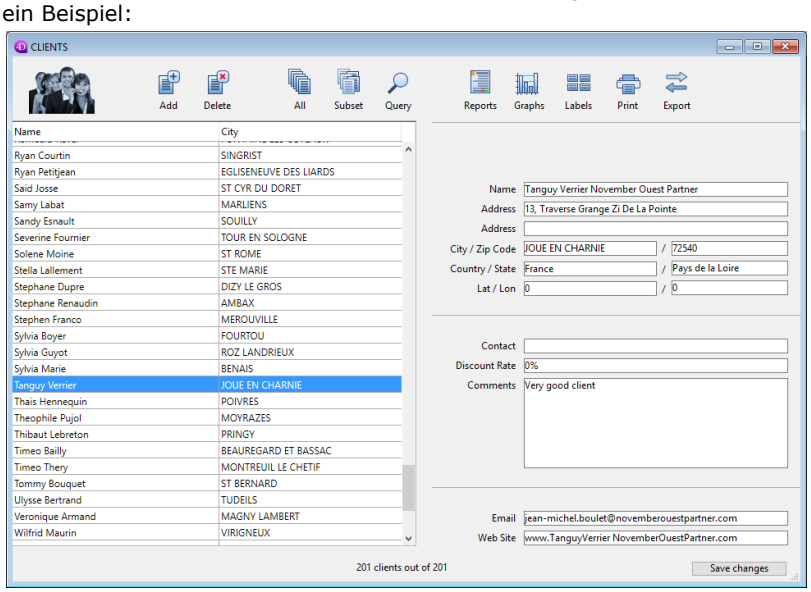
Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, zu der das Formular gehört, ohne Angabe: Standardtabelle oder Verwenden von Projektformular
Formularname	String, Objekt	➡	Name (String) des Tabellen- oder Projektformulars, oder POSIX Pfad (String) zu einer .json Datei mit Beschreibung des Formulars, oder Objekt mit Beschreibung des Formulars.
FormularDaten	Objekt	➡	Daten für das Formular
*	Operator	➡	Verwende gleichen Prozess

Beschreibung

Der Befehl **DIALOG** zeigt dem Benutzer das Formular *Formularname*, optional mit dem Parameter *FormularDaten*.

Dieser Befehl dient zum Arbeiten mit angepassten oder komplexen Benutzeroberflächen, die auf Formularen basieren. Damit können Sie Information aus der Datenbank bzw. von anderen Stellen anzeigen oder Features zur Dateneingabe vermitteln. Im Gegensatz zu **ADD RECORD** oder **MODIFY RECORD** gibt **DIALOG** Ihnen volle Kontrolle über das Formular, seinen Inhalt und die Schaltflächen zum Navigieren und Bestätigen.

Oft wird das Formular zusammen mit der Funktion **Open form window** zum Anzeigen komplexer Formulare aufgerufen. Hierzu ein Beispiel:



Sie können **DIALOG** auch anstatt **ALERT**, **CONFIRM** oder **Request** verwenden, wenn die anzuzeigende Information für diese Befehle zu umfangreich ist.

Hinweis: In konvertierten Datenbanken lässt sich die Dateneingabe in Felder von Dialogboxen unterbinden, d.h. die Dateneingabe ist auf Variablen beschränkt. Diese Option stellen Sie in den Einstellungen der Datenbank auf der Seite Kompatibilität ein. Diese Einschränkung entspricht der Arbeitsweise früherer 4D Versionen.

In *Formularname* definieren Sie, welches Formular (Projekt- oder Tabellenformular) verwendet werden soll.

Im Parameter *Formularname* können Sie folgendes übergeben:

- Name des Formulars (Projektformular oder Tabellenformular)
- Pfad (in POSIX Syntax) zu einer gültigen .json Datei mit der Beschreibung des Formulars. Siehe **Dateipfade für Formulare**;
- Objekt mit der Beschreibung des Formulars. Weitere Informationen dazu finden Sie unter **Dynamische Formulare**.

Optional können Sie für *Formularname* über das Objekt *FormularDaten* bestimmte Eigenschaften übergeben. Alle Eigenschaften von *FormularDaten* sind dann im Formularkontext über die Funktion **Form** verfügbar. Übergeben Sie z.B. ein Objekt mit {"version","12"}, können Sie den Wert der Eigenschaft "Version" im Formular mit folgender Anweisung erhalten:

```
$v:=Form.version //"12"
```

Über eine lokale Variable für *FormularDaten* können Sie Parameter sicher an Ihre Formular übergeben, unabhängig vom aufrufenden Kontext. Insbesondere wenn dasselbe Formular im gleichen Prozess von verschiedenen Stellen aus aufgerufen wird, können Sie durch Aufrufen von **Form**.myProperty immer auf die spezifischen Werte zugreifen. Da Objekte über Referenz übergeben werden, ist ein weiterer Vorteil, dass ein vom Benutzer im Formular geänderter Wert automatisch im Objekt selbst gesichert wird.
Durch Kombination des Objekts *FormularDaten* mit der Funktion **Form** können Sie jederzeit mit sauberem und sicherem Code Parameter an das Formular senden oder Parameter des Formulars ansehen.

Hinweis: Übergeben Sie keinen Parameter *FormularDaten* bzw. ein undefiniertes Objekt, erstellt **DIALOG** automatisch ein neues leeres Objekt, das an *Formularname* angebunden und über die Funktion **Form** verfügbar ist.

Der Dialog wird vom Benutzer geschlossen, entweder über "Bestätigen" (ausgelöst über die Standardaktion ak accept, die **Eingabetaste** oder den Befehl **ACCEPT**) oder "Abbrechen" (ausgelöst über die Standardaktion ak cancel, die **Escape-Taste** oder den Befehl **CANCEL**). Bei Bestätigen wird die Systemvariable OK auf 1 gesetzt, bei Abbrechen auf 0.

Übergeben Sie den optionalen Parameter *, wird *Formularname* geladen und im zuletzt geöffneten Fenster des aktuellen Prozesses angezeigt. Der Befehl beendet seine Ausführung und lässt das aktive Formular auf dem Bildschirm. Dieses Formular reagiert dann normal auf Benutzeraktionen und wird über eine Standardaktion geschlossen oder, wenn die dem Formular zugewiesene Objekt- oder Formularmethode den Befehl **CANCEL** oder **ACCEPT** aufruft. Endet der aktuelle Prozess, werden die auf diese Weise erstellten Formulare automatisch so geschlossen, als ob **CANCEL** aufgerufen worden wäre. Dieser Öffnen-Modus ist besonders hilfreich zum Anzeigen eines Palettenfensters mit einem Dokument und es ist kein weiterer Prozess erforderlich.

Hinweise:

- Sie können die Syntax DIALOG(form;*) mit dem Befehl **CALL FORM** kombinieren, um die Kommunikation zwischen den Formularen einzurichten.
- Sie müssen ein Fenster anlegen, bevor Sie die Anweisung **DIALOG**(Formular;*) aufrufen; Sie können nicht das aktuelle Dialogfenster im Prozess bzw. das standardmäßig für jeden Prozess erstellte Fenster verwenden. In diesem Fall wird der Fehler 9909 erzeugt.
- Mit dem Parameter * wird das Fenster automatisch über eine Standardaktion oder Aufrufen des Befehls **CANCEL** oder **ACCEPT** geschlossen. Sie müssen das Schließen des Fensters selbst nicht verwalten.

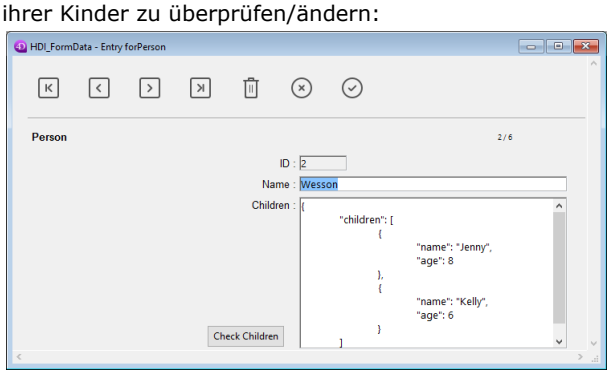
Beispiel 1

Dieses Beispiel erstellt eine Werkzeugpalette

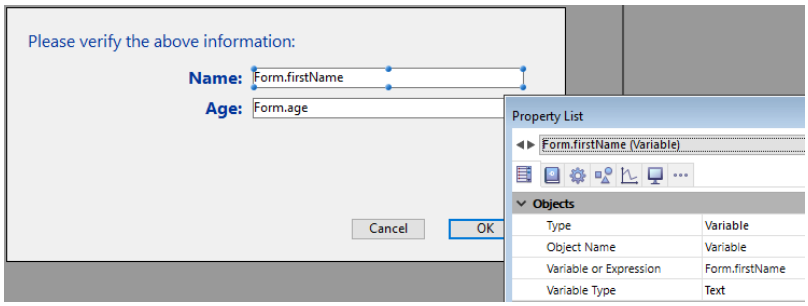
```
  `Zeige Werkzeugpalette
$palette_window:=Open form window("Werkzeuge";Palette form window)
DIALOG("Werkzeuge";*)  `Die Steuerung sofort zurückgeben
  `Zeige Hauptdokumentfenster
$document_window:=Open form window("Dok";Plain form window)
DIALOG("Dok")
```

Beispiel 2

In einen Eingabeformular zu einer Person lässt sich über die Schaltfläche "Check Children" ein Dialog öffnen, um Name und Alter ihrer Kinder zu überprüfen/ändern:



Hinweis: Das Objektfeld "Children" dient nur dazu, um seine Struktur für dieses Beispiel anzuzeigen.
Im Fenster zur Überprüfung haben Sie den **Form** Objekteigenschaften Variablen zugewiesen:

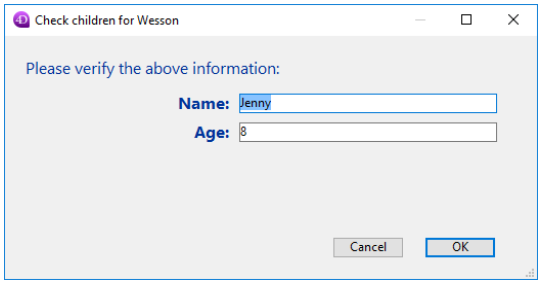


Der Code für die Schaltfläche "Check Children" lautet:

```
C_LONGINT($win;$n;$i)
C_BOOLEAN($save)
ARRAY OBJECT($children;0)
OB GET ARRAY([Person]Children;"children";$children)  //die  Collection children erhalten
$save:=False  //die Variable save initialisieren

$n:=Size of array($children)
If($n>0)
   $win:=Open form window("Edit_Children";Movable form dialog box)
   SET WINDOW TITLE("Check children for "+[Person]Name)
   For($i;1;$n)  //für jedes Kind
      DIALOG("Edit_Children";$children{$i})  //den mit Werten gefüllten Dialog anzeigen
      If(OK=1)  //der Benutzer hat auf OK geklickt
         $save:=True
```

```
        End if
    End for
    If($save=True)
        [Person]Children:=[Person]Children  //Update des Objektfeldes erzwingen
    End if
    CLOSE WINDOW($win)
Else
    ALERT("No child to check.")
End if
```

Hinweis: Für dieses Beispiel muss die Objektnotation in der Anwendung aktiviert sein (siehe **Seite Kompatibilität**).
Das Formular zeigt Information zu jedem Kind an:



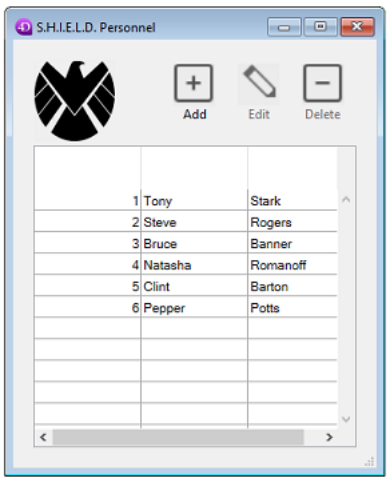
Werden die Werte bearbeitet und auf die Schaltfläche OK geklickt, wird das Feld aktualisiert (der übergeordnete Datensatz muss anschließend gesichert werden).

Beispiel 3

Dieses Beispiel verwendet den Pfad eines .json Formulars zum Anzeigen der Datensätze in einer Liste der Angestellten:

```
Open form window("/RESOURCES/OutputPersonnel.json";Plain form window)
ALL RECORDS([Personnel])
DIALOG("/RESOURCES/OutputPersonnel.json";*)
```

Das ergibt folgendes Formular:

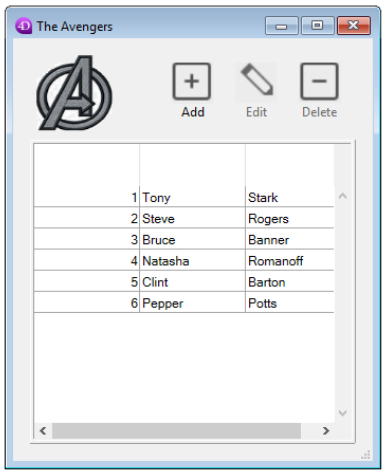


Beispiel 4

Dieses Beispiel verwendet eine .json Datei als ein Objekt und ändert ein paar Eigenschaften:

```
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;*)
```

Das ergibt dieses Formular mit geändertem Titel, Logo und Rahmen:



Systemvariablen und Mengen

Bestätigen des Dialogs nach Aufrufen von **DIALOG** setzt die Systemvariable OK auf 1, Abbrechen setzt sie auf 0.

Modified

Modified (Feldname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Feldname	Feld	⇒	Datenfeld, das auf Änderung abgefragt werden soll
Funktionsergebnis	Boolean	⇐	Wahr, wenn dem Datenfeld ein neuer Wert zugewiesen wurde, sonst Falsch

Beschreibung

Die Funktion **Modified** gibt Wahr zurück, wenn *Feldname* per Programmierung ein Wert zugewiesen wurde oder während der Dateneingabe bearbeitet wurde. **Modified** darf nur in einer Formularmethode bzw. in einer Unterroutine, die von einer Formularmethode aufgerufen wird, verwendet werden.

Beachten Sie, dass diese Funktion nur im gleichen Ausführungszyklus einen signifikanten Wert zurückgibt. Für alle Formularereignisse, die zum davorliegenden Ausführungszyklus **_o_During** gehören (On Clicked, On After Keystroke, etc.), gibt sie *Falsch* zurück.

Bei der Dateneingabe gilt ein Datenfeld als geändert, das der Benutzer bearbeitet - unabhängig davon, ob der ursprüngliche Wert geändert wird - und durch Wechseln zum nächsten Datensatz oder Anklicken einer Steuerung verlässt. Nur das Verlassen eines Datenfeldes ohne Bearbeitung setzt **Modified** nicht auf *Wahr*.

Beim Ausführen einer Methode gilt ein Datenfeld als geändert, wenn ein Wert zugewiesen wurde (geändert oder nicht).

Hinweis: Modified gibt nach Ausführen der Befehle **PUSH RECORD** und **POP RECORD** immer *Wahr* zurück.

Testen Sie in beiden Fällen mit dem Befehl **Old**, ob der Wert des Datenfeldes geändert wurde.

Hinweis: Sie können **Modified** zwar für jeden Datenfeldtyp verwenden. Beachten Sie jedoch in Kombination mit dem Befehl **Old** die dafür geltenden Einschränkungen. Weitere Informationen dazu finden Sie unter dem Befehl **Old**.

Bei der Dateneingabe lassen sich Operationen normalerweise einfacher in Objektmethoden als über **Modified** in Formularmethoden anzeigen. Da einer Objektmethode bei jeder Datenfeldänderung ein Ereignis On Data Change übergeben wird, ist die Verwendung von **Modified** in Formularmethoden gleichwertig mit Objektmethoden.

Beispiel 1

Folgendes Beispiel testet, ob das Datenfeld *[Orders]Quantity* oder das Datenfeld *[Orders]Price* geändert wurde. Wenn ja, wird das Datenfeld *[Orders]Total* neu berechnet.

```
If((Modified([Orders]Quantity)|(Modified([Orders]Price))
   [Orders]Total :=[Orders]Quantity*[Orders]Price
End if
```

Beachten Sie, dass derselbe Vorgang ausgeführt wird, wenn Sie die zweite Zeile als Unterroutine verwenden, die von einer Objektmethode für die Datenfelder *[Orders]Quantity* und *[Orders]Price* innerhalb des Formularereignisses *On Data Change* aufgerufen wird.

Beispiel 2

Wählen Sie einen Datensatz für die Tabelle *[anyTable]*, rufen Sie dann mehrere Unterroutinen auf, die das Datenfeld *[anyTable]Important field* ändern, jedoch nicht den Datensatz sichern. Am Ende der Hauptmethode können Sie dann mit dem Befehl **Modified** prüfen, ob Sie den Datensatz sichern müssen:

```
   ` Hier wurde der Datensatz als aktueller Datensatz ausgewählt.
   ` Führen Sie dann Aktionen mit Unterroutinen aus
MACH DIES
MACH DAS
MACH JENES
   ` ...
   ` Testen Sie nun das Datenfeld, um herauszufinden, ob der Datensatz gesichert
   ` werden muss
If(Modified([anyTable]Important field))
   SAVE RECORD([anyTable])
End if
```

MODIFY RECORD

MODIFY RECORD ({Tabellenname}{;}{*})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle für Dateneingabe oder Standardtabelle ohne Angabe
*		➡	Rollbalken ausblenden

Beschreibung

Der Befehl **MODIFY RECORD** öffnet den aktuellen Datensatz von *Tabellenname* zur Änderung. Er benutzt das Eingabeformular der Tabelle. Ist kein aktueller Datensatz vorhanden, geschieht beim Aufruf von **MODIFY RECORD** nichts. Dateiname ist optional. Wird dieser Parameter nicht angegeben, bezieht sich **MODIFY RECORD** auf die Standardtabelle. Der Befehl beeinträchtigt nicht die aktuelle Auswahl.

Das Formular erscheint im obersten Fenster des Prozesses mit Rollbalken und Kästchen für Größeneinstellung. Der optionale Parameter * blendet Rollbalken und Kästchen für Größeneinstellung aus.

Um den Befehl **MODIFY RECORD** zu verwenden, muss der aktuelle Datensatz im Lesen-Schreibmodus sein und darf nicht gesperrt sein.
Enthält das Formular Schaltflächen zum Bewegen innerhalb der Datensatzauswahl, kann der Benutzer über **MODIFY RECORD** auf die Schaltflächen klicken, um Datensätze zu ändern und zu anderen Datensätze zu gehen.

Der Datensatz wird gesichert, wenn der Benutzer auf die Schaltfläche **Bestätigen** bzw. die **Eingabetaste** im Zahlenblock klickt oder der Befehl **ACCEPT** ausgeführt wird.
Der Datensatz wird nicht gesichert, wenn der Benutzer auf die Schaltfläche **Abbrechen** bzw. die Tastenkombination **strg-Taste+Punkt** unter Windows, **Befehlstaste+Punkt** auf Macintosh verwendet oder der Befehl **CANCEL** ausgeführt wird.

Hinweis: Selbst wenn der Datensatz abgebrochen wird, bleibt er solange im Arbeitsspeicher, bis ein neuer aktueller Datensatz aufgerufen wird. Er kann gesichert werden, wenn **SAVE RECORD** ausgeführt wird, bevor der Zeiger auf den aktuellen Datensatz geändert wird.

Verwenden Sie **MODIFY RECORD**, ohne dass der Benutzer Daten im Datensatz ändert, wird er als nicht geändert gewertet, d.h. durch Bestätigen wird er nicht erneut gesichert. Aktionen wie Variable ändern, Ankreuzfeld markieren oder Optionsfeld auswählen, gelten nicht als Änderung. Nur wenn Daten in einem Feld geändert werden, sei es durch Dateneingabe oder über eine Methode, wird der Datensatz gesichert.

Nach Aufrufen von **MODIFY RECORD** hat die Systemvariable OK den Wert 1, wenn der Datensatz bestätigt wurde; den Wert 0, wenn er annulliert wurde.

Beispiel

Siehe Beispiel für den Befehl **ADD RECORD**.

Systemvariablen und Mengen

Bestätigen des Datensatzes setzt die Systemvariable OK auf 1, Abbrechen setzt die Systemvariable OK auf 0. Die Systemvariable OK wird nur gesetzt, wenn der Datensatz bestätigt oder annulliert wird.

Old

Old (Feldname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Datenfeld, für das der alte Wert zurückgeben werden soll
Funktionsergebnis	Ausdruck	⮌	Ursprünglicher Datenfeldwert

Beschreibung

Die Funktion **Old** gibt in *Feldname* den Wert zurück, bevor dem Datenfeld per Programmierung ein Wert zugewiesen oder der Wert in Dateneingabe geändert wurde.

Immer wenn Sie den aktuellen Datensatz für eine Tabelle ändern, erstellt und unterhält 4D im Speicher ein Duplikat des neuen aktuellen Datensatzes, wenn er in den Speicher geladen wird. Wenn Sie einen Datensatz ändern, arbeiten Sie mit dem eigentlichen Datensatz, nicht mit dem Duplikat. Das Duplikat wird verworfen, sobald Sie erneut den aktuellen Datensatz ändern.

Old gibt den Wert des Duplikats zurück. Die Funktion gibt also für ein Datenfeld den Wert zurück, so wie er auf der Festplatte gespeichert ist. Bei einem neuen Datensatz gibt **Old** den leeren Standardwert für *Feldname* gemäß seinem Typ zurück. Ist *Feldname* z.B. vom Typ Alphanumerisch, wird eine leere Zeichenkette zurückgegeben, ist *Feldname* vom Typ numerisch, wird Null (0) zurückgegeben.

Old funktioniert für *Feldname*, egal, ob eine Methode oder der Benutzer bei der Dateneingabe das Datenfeld ändert. **Old** gilt für alle Datenfeldtypen.

Um den ursprünglichen Wert eines Datenfeldes wiederherzustellen, weisen Sie den Wert zu, der von **Old** zurückgegeben wird.

Hinweis: Aus technischen Gründen lässt sich der von **Old** zurückgegebene Ausdruck bei Feldern vom Typ Bild und BLOB nicht direkt als Parameter für einen anderen Befehl verwenden. Der Wert muss über eine Zwischenvariable übergeben werden. Zum Beispiel:

```
  `Schreiben Sie NICHT (erzeugt einen Syntaxfehler):
$size :=BLOB size(Old([theTable]theBlob))  `INKORREKT

  `sondern schreiben Sie:
$oldBLOB:=Old([theTable]theBlob)
$size :=BLOB size($oldBLOB)  `KORREKT
```

REJECT

REJECT {(Feldname)}

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Feld, dessen Eingabe verweigert werden soll

Beschreibung

Der Befehl **REJECT** hat zwei Arten. Die erste Art verwendet keine Parameter. Sie verweigert die komplette Dateneingabe, der Benutzer muss im Formular bleiben. Die zweite Art verweigert nur die Eingabe in *Feldname*, der Benutzer muss im Datenfeld bleiben.

Hinweis: Verwenden Sie zuerst die in 4D integrierten Eingabekontrollen.

Die erste Art verhindert das Bestätigen von unvollständigen Datensätzen. Sie erhalten dasselbe Ergebnis, wenn Sie der Eingabetaste die Schaltfläche **Keine Aktion** zuordnen und den Datensatz je nach Eingabe mit den Befehlen **ACCEPT** und **CANCEL** bestätigen bzw. annullieren. Wir empfehlen diese Vorgehensweise anzuwenden und nicht die erste Art von **REJECT**.

Verwenden Sie **REJECT** ohne Parameter, kann der Benutzer einen Datensatz nicht bestätigen, weil er entweder unvollständig ist oder unzulässige Eingaben enthält. Versucht der Benutzer, den Datensatz zu bestätigen, wird die Aktion durch **REJECT** verhindert. Der Benutzer muss mit der Dateneingabe fortfahren, bis der Datensatz akzeptiert wird oder den Datensatz annullieren.

Der beste Platz für **REJECT** ohne Parameter ist die Objektmethode der Schaltfläche **Bestätigen**, der die Eingabetaste zugeordnet ist. So erfolgt die Bestätigung nur, wenn der Datensatz bestätigt wird. Der Benutzer kann diese nicht durch Betätigen der Eingabetaste umgehen.

Die zweite Art von **REJECT** wird mit dem Parameter *Feldname* ausgeführt. Der Cursor bleibt im Datenfeldbereich, das zwingt den Benutzer, einen korrekten Wert einzugeben. Mit dieser Syntax müssen Sie **REJECT** im Formularereignis On Data Change aufrufen. Sie müssen diese Syntax von **REJECT** in die Formular- oder Objektmethode des Eingabebereichs setzen. Verwenden Sie **REJECT** für das Eingabeformular eines Unterformulars in einer Tabelle, setzen Sie den Befehl in die Formular- oder Objektmethode für das Eingabeformular. Dieser Befehl hat keine Auswirkung auf Felder von Unterformularen.

Mit dem Befehl **HIGHLIGHT TEXT** können Sie Daten in dem zurückgewiesenen Feld auswählen.

Beispiel 1

Folgendes Beispiel gehört zu einem Datensatz für eine Banküberweisung. Es zeigt die erste Art von **REJECT** in der Objektmethode für die Schaltfläche **Bestätigen**. Die Eingabetaste ist der Schaltfläche gleichgesetzt, d.h., die Objektmethode wird auch dann ausgeführt, wenn der Benutzer die Eingabetaste zum Bestätigen benutzt. Bei einer Überweisung per Scheck muss eine Schecknummer eingegeben werden, sonst wird die Bestätigung verweigert:

```
Case of
   :(([Operation]Transaction="Check") & ([Operation]Check Number=""))  ` Ist es ein Scheck ohne Nummer...
      ALERT("Bitte die Schecknummer eintragen.") ` Warnung an Benutzer
      REJECT ` Verweigere die Eingabe
      GOTO AREA([Operation]Check Number) ` Gehe in das Datenfeld Schecknr.
End case
```

Beispiel 2

Folgendes Beispiel ist Teil einer Objektmethode für das Datenfeld *[Employees]Salary*. Die Objektmethode prüft dieses Datenfeld und verweigert die Eingabe, wenn der Wert unter Euro 3.000 liegt. Sie können dieselbe Operation auch mit einem Mindestwert im Formulareditor ausführen:

```
Case of
   :(Form event=On Data Change)
      If([Employees]Salary<3000)
         ALERT("Gehalt muss über Euro 3.000 liegen")
         REJECT([Employees]Salary)
      End if
End case
```

_o_ADD SUBRECORD

_o_ADD SUBRECORD (Untertabelle ; Formularname {; *})

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle für Dateneingabe
Formularname	String	➡	Formular für Dateneingabe
*		➡	Rollbalken ausblenden

Hinweis zur Kompatibilität

Untertabellen werden ab Version 11 von 4D nicht mehr unterstützt. Eine Vorkehrung zur Kompatibilität sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_MODIFY SUBRECORD

_o_MODIFY SUBRECORD (Untertabelle ; Formularname {; *})

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle für Dateneingabe
Formularname	String	➡	Formular für Dateneingabe
*		➡	Rollbalken ausblenden

Hinweis zur Kompatibilität

Untertabellen werden ab Version 11 von 4D nicht mehr unterstützt. Eine Vorkehrung zur Kompatibilität sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

Eingabekontrolle

- EDIT ITEM
- FILTER KEYSTROKE
- Get edited text
- GET HIGHLIGHT
- GOTO OBJECT
- HIGHLIGHT TEXT
- Keystroke

EDIT ITEM

EDIT ITEM ({* ;} Objekt {; Eintrag})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist Objektname (String) Ohne *: Objekt ist Tabelle oder Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, Ohne *: Tabelle oder Variable
Eintrag	Lange Ganzzahl	➡	Nummer des Eintrags

Beschreibung

Der Befehl **EDIT ITEM** ermöglicht, den aktuellen Eintrag oder die in *Eintrag* angegebene Nummer im Array oder der Liste, definiert in *Objekt*, zu bearbeiten. Sie können also den gewählten Eintrag ändern, wobei ein eingegebenes Zeichen ganz den Inhalt des Eintrags ersetzt.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname ist. In diesem Fall übergeben Sie einen String. Ohne * geben Sie an, dass *Objekt* ein Feld oder eine Tabelle ist. In diesem Fall übergeben Sie eine Referenz auf eine Tabelle oder eine Variable.

EDIT ITEM gilt für folgende eingebbare Objekte:

- Hierarchische Listen
- Listboxen
- Unterformulare (in diesem Fall lässt sich in *Objekt* nur ein Objektname des Unterformulars übergeben,
- Listenformulare, die über die Befehle **DISPLAY SELECTION** oder **MODIFY SELECTION** angezeigt werden.

Bei Verwendung mit einem eingebbaren Objekt, das keine Liste ist, arbeitet **EDIT ITEM** genauso wie **GOTO OBJECT**.
Der Befehl führt nichts aus, wenn die Liste bzw. das Array leer oder ausgeblendet ist. Ist die Liste bzw. das Array nicht eingebbar, wählt der Befehl die angegebenen Einträge nur aus, ohne in den Bearbeitungsmodus zu wechseln.
Erlauben in Listboxen die Spalten keine Texteingabe (nur Eingabe über Optionsfelder oder DropDown-Listen), erhält das angegebene Element den Fokus.

Mit dem optionalen Parameter *Eintrag* setzen Sie die Position des Eintrags in der hierarchischen Liste bzw. die Zeilennummer in der Listbox, dem Listenformular und dem Unterformular im Modus mehrfache Auswahl, um in den Bearbeitungsmodus zu wechseln. Ohne diesen Parameter gilt der Befehl für den aktuellen Eintrag für *Objekt*. Gibt es keinen aktuellen Eintrag, wechselt der erste Eintrag von *Objekt* in den Bearbeitungsmodus.

Hinweise:

- In Listen- und Unterformularen setzt **EDIT ITEM** das erste Feld einer angegebenen Zeile in den Bearbeitungsmodus, gemäß der Reihenfolge der Eingabe.
- In Listboxen im hierarchischen Modus wird diese Ebene inkl. untergeordneter Ebenen automatisch aufgeklappt, wenn der anvisierte Eintrag zu einer zugeklappten hierarchischen Ebene gehört, so dass der Eintrag sichtbar wird.

Beispiel 1

Dieser Befehl eignet sich besonders zum Erstellen eines neuen Eintrags in einer hierarchischen Liste. Bei Aufruf des Befehls wird der zuletzt ein- oder angefügte Eintrag automatisch editierbar, also ohne Zutun des Benutzers.
Nachfolgender Code dient als Methode einer Schaltfläche, über die Sie in eine vorhandene Liste einen neuen Eintrag einfügen. Der Standardtext "Neuer_Eintrag" lässt sich automatisch verändern:

```
vlUniqueRef:=vlUniqueRef+1
INSERT IN LIST(hList;*;"Neuer_Eintrag";vlUniqueRef)
EDIT ITEM(*;"MeineListe")
```

Beispiel 2

Wir gehen aus von zwei Spalten in einer Listbox mit den Variablennamen "Array1" und "Array2".
Der folgende Code fügt einen neuen Eintrag in die beiden Arrays ein und setzt den neuen Eintrag von Array2 in den Bearbeitungsmodus:

```
$vlRowNum:=Size of array(Array1)+1
INSERT LISTBOX ROW(*;"MeineListBox";$vlRowNum)
Array1{$vlRowNum}:="NeuerWert 1"
Array2{$vlRowNum}:="NeuerWert 2"
EDIT ITEM(Array2;$vlRowNum)
```

Table1	Table2
Julie	Smith
Sam	Jones
William	Thomas
Smeldorf	Goldman
Bart	Winkler

Table1	Table2
Julie	Smith
Sam	Jones
William	Thomas
Smeldorf	Goldman
Bart	Winkler
New value 1	New value 2

FILTER KEYSTROKE

FILTER KEYSTROKE (Zeichen)

Parameter	Typ		Beschreibung
Zeichen	String	➡	Gefilterte Zeichen oder Leerer String, um Anschlag zu annullieren

Beschreibung

Der Befehl **FILTER KEYSTROKE** ersetzt das Zeichen, das der Benutzer in ein Datenfeld oder einen eingebbaren Bereich getippt hat, durch das erste Zeichen des übergebenen String *Zeichen*.

Übergeben Sie einen leeren String, wird der Anschlag annulliert und nicht berücksichtigt.

Sie rufen **FILTER KEYSTROKE** im allgemeinen bei einem Formularereignis *On Before Keystroke* in einem Formular oder einer Objektmethode auf. Mit der Funktion **Form event** finden Sie solche Ereignisse. Mit der Funktion **Keystroke** finden Sie den aktuellen Tastaturanschlag.

WICHTIGER HINWEIS: Mit dem Befehl **FILTER KEYSTROKE** können Sie das vom Benutzer eingetippte Zeichen annullieren oder durch ein anderes ersetzen. Wollen Sie für einen bestimmten Tastaturanschlag mehr als ein Zeichen eingeben, beachten Sie, dass der Text auf dem Bildschirm NOCH NICHT der Wert des Quelldatenfeldes oder der Variablen für den Bereich in Bearbeitung ist. Der eingegebene Wert wird dem Quelldatenfeld oder der Variablen erst zugewiesen, wenn die Dateneingabe für den Bereich bestätigt wurde. Sie können zu diesem Zweck die Dateneingabe in einer Variablen spiegeln und mit diesem Wert arbeiten. Anschließend weisen Sie den eingebbaren Bereich erneut zu (Siehe Beispiel).

Sie verwenden den Befehl **FILTER KEYSTROKE**, um:

- Zeichen auf eigene Weise zu filtern
- Die Dateneingabe zu filtern, die Sie nicht über Dateneingabefilter ausführen können.
- Dynamische Nachschlag- oder Fortschreibbereiche (lookup oder type-ahead) zu integrieren.

WARNUNG: Rufen Sie nach dem Befehl **FILTER KEYSTROKE** die Funktion **Keystroke** auf, wird das hier übergebene Zeichen und nicht das gerade eingegebene zurückgegeben.

Beispiel 1

Mit dem Code:

```
  ` Objektmethode Eingebbarer Bereich myObject
Case of
   :(Form event=On Load)
      myObject:=""
   :(Form event=On Before Keystroke)
      If(Position(Keystroke;"0123456789")>0)
         FILTER KEYSTROKE("*")
      End if
End case
```

werden alle im Bereich *myObject* eingetragenen Ziffern in Sternchen umgewandelt.

Beispiel 2

Der folgende Code integriert einen eingebbaren Bereich, in dem wie bei Kennwörtern alle eingegebenen Zeichen auf dem Bildschirm durch Blindzeichen ersetzt werden:

```
  ` Objektmethode Eingebbarer Bereich vsPassword
:(Form event=On Load)
 vsPassword:=""
 vsActualPassword:=""
:(Form event=On Before Keystroke)
 Handle keystroke(->vsPassword;->vsActualPassword)
 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
```

Nach Bestätigen der Dateneingabe finden Sie das vom Benutzer eingetragene Kennwort in der Variablen *vsActualPassword*.

Hinweis: Die Methode *Handle keystroke* finden Sie im Beispiel zur Funktion **Keystroke**.

Beispiel 3

In Ihrer Anwendung gibt es Textbereiche für die Eingabe von mehreren Sätzen sowie ein Wörterbuch mit häufig vorkommenden Einträgen. Beim Eingeben von Text sollen nun anhand von markierten Zeichen schnell dazu passende Einträge aus dem

Wörterbuch gefunden und integriert werden. Es gibt dafür zwei Lösungen:
- Sie legen einige Schaltflächen mit Tastaturkürzeln an oder
- Sie filtern während der Eingabe in den Textbereich spezifische Tastaturanschläge heraus.

Nachfolgendes Beispiel erläutert die zweite Lösung anhand der Taste **Hilfe**.

Wie Sie bereits wissen, wird beim Bearbeiten des Textbereichs der eingegebene Wert der Datenquelle erst nach Bestätigen der Dateneingabe zugewiesen. Damit Sie Einträge aus dem Wörterbuch während der Eingabe von Text in den Textbereich finden und einsetzen können, müssen Sie die Dateneingabe spiegeln. Sie übergeben als Parameter Zeiger auf den eingebbaren Bereich und die Spiegelvariable. Als dritten Parameter übergeben Sie einen String für die unzulässigen Zeichen. Es spielt nun keine Rolle, wie die Zeichen eingegeben werden, die Methode gibt immer die Originalschreibweise zurück. Die unzulässigen Zeichen sollen nicht in den eingebbaren Bereich integriert, sondern als spezielle Zeichen behandelt werden.

```
  ` Projektmethode Spiegelanschlag
  ` Spiegelanschlag ( Zeiger ; Zeiger ; String ) -> String
  ` Spiegelanschlag ( -> Quellbereich ; -> curWert ; Filter ) -> Alter Anschlag
C_STRING(1;$0)
C_POINTER($1;$2)
C_TEXT($vtNewValue)
C_STRING(255;$3)
  ` Gib Originalanschlag zurück
$0:=Keystroke
  ` Hole ausgewählten Textbereich aus dem eingebbaren Bereich
GET HIGHLIGHT($1->;$vlStart;$vlEnd)
  ` Beginne, mit dem aktuellen Wert zu arbeiten
$vtNewValue:=$2->
  ` Führe je nach gedrückter Taste oder eingegebenem Zeichen geeignete Aktionen aus
Case of
  ` Die Rückschrittaste (Löschen) wurde gedrückt
  :(ASCII($0)=Backspace)
  ` Lösche die ausgewählten Zeichen oder die Zeichen links vom Cursor
     $vtNewValue:=Delete text($vtNewValue;$vlStart;$vlEnd)
  ` Eine Pfeiltaste wurde gedrückt
  ` Tue nichts, aber akzeptiere den Anschlag
  :(ASCII($0)=Left arrow key)
  :(ASCII($0)=Right arrow key)
  :(ASCII($0)=Up arrow key)
  :(ASCII($0)=Down arrow key)

  ` Ein zulässiges Zeichen wurde eingegeben
  :(Position($0;$3)=0)
     $vtNewValue:=Insert text($vtNewValue;$vlStart;$vlEnd;$0)
  Else
  ` Das Zeichen wird nicht angenommen
     FILTER KEYSTROKE("")
End case
  ` Gib Wert für die Bearbeitung des nächsten Anschlags zurück
$2->:=$vtNewValue
```

Diese Methode arbeitet mit folgenden beiden Untermethoden:

```
  ` Projektmethode Text löschen
  ` Lösche Text ( String ; Lang ; Lang ) -> String
  ` Lösche Text ( -> Text ; AuswStart ; AuswEnde ) -> Neuer Text
C_TEXT($0;$1)
C_LONGINT($2;$3)
$0:=Substring($1;1;$2-1-Num($2=$3))+Substring($1;$3)

  ` Projektmethode Text einfügen
  ` Text einfügen ( String ; Lang ; Lang ; String ) -> String
  ` Text einfügen ( -> Quelltext ; AuswStart ; AuswEnde ; einzufügender Text ) -> Neuer Text
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
```

Sie können diese Projektmethoden folgendermaßen einsetzen:

```
` Objektmethode Eingebbarer Bereich vsDescription
Case of
   :(Form event=On Load)
      vsDescription:=""
      vsShadowDescription:=""
` Erstelle Liste der unzulässigen Zeichen, die als spezielle Tasten zu behandeln sind (Hier wird nur die Taste Hilfe gefiltert)
      vsSpecialKeys:=Char(HelpKey)
   :(Form event=On Before Keystroke)
      $vsKey:=Shadow keystroke(->vsDescription;->vsShadowDescription;vsSpecialKeys)
      Case of
         :(ASCII($vsKey)=Help key)
` Tue etwas, wenn die Taste Hilfe gedrückt wird
` Hier muss ein Eintrag aus dem Wörterbuch gesucht und eingesetzt werden
            LOOKUP DICTIONARY(->vsDescription;->vsShadowDescription)
      End case
End case
```

Die Projektmethode ***LOOKUP DICTIONARY*** sehen Sie im Folgenden. Damit soll die Spiegelvariable verwendet werden, um den eingebbaren Bereich in Bearbeitung neu zuzuweisen:

```
` Projektmethode LOOKUP DICTIONARY
` LOOKUP DICTIONARY ( Zeiger ; Zeiger )
` LOOKUP DICTIONARY ( -> Eingebbarer Bereich ; ->Spiegelvariable )

C_POINTER($1;$2)
C_LONGINT($vlStart;$vlEnd)

` Hole ausgewählten Textbereich aus dem eingebbaren Bereich
GET HIGHLIGHT($1->;$vlStart;$vlEnd)
` Hole ausgewählten Text oder Wort links vom Cursor
$vtHighlightedText:=Get highlighted text($2->;$vlStart;$vlEnd)
` Gibt es etwas, wonach gesucht werden soll?
If($vtHighlightedText#"")
` War der ausgewählte Text der Cursor, startet die Auswahl jetzt mit dem vor dem Cursor liegenden Wort.
   If($vlStart=$vlEnd)
      $vlStart:=$vlStart-Length($vtHighlightedText)
   End if
` Suche nach dem ersten möglichen Eintrag im Wörterbuch
   QUERY([Dictionary];[Dictionary]Entry=$vtHighlightedText+"@")
` Ist einer vorhanden?
   If(Records in selection([Dictionary])>0)
` Wenn ja, füge ihn in den Spiegeltext ein
      $2->:=Insert text($2->;$vlStart;$vlEnd;[Dictionary]Entry)
` Kopiere Spiegeltext in den eingebbaren Bereich in Bearbeitung
      $1->:=$2->
` Setze Auswahl direkt hinter die Eingabe aus dem Wörterbuch
      $vlEnd:=$vlStart+Length([Dictionary]Entry)
      HIGHLIGHT TEXT(vsComments;$vlEnd;$vlEnd)
   Else  ` Es gibt keinen entsprechenden Eintrag im Wörterbuch
      BEEP
   End if
Else  ` Es gibt keinen markierten Text
   BEEP
End if
```

Die Methode ***Hole markierten Text*** sieht folgendermaßen aus:

```
` Projektmethode Hole markierten Text
<gen9>  ` Hole markierten Text ( String ; Lang ; Lang ) -> String
` Hole markierten Text ( Text ; AuswStart ; AuswEnde ) -> markierter Text
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</gen9>
```

Get edited text

Get edited text -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Text	Eingegebener Text

Beschreibung

Die Funktion **Get edited text** wird hauptsächlich mit dem Formularereignis On After Keystroke eingesetzt. Damit finden Sie den Text genauso wieder, wie Sie ihn eingegeben haben. Sie können die Funktion auch mit dem Formularereignis On Before Keystroke verwenden. Weitere Informationen zu diesen Formularereignissen finden Sie in der Beschreibung der Funktion **Form event**.

Die Kombination dieser Funktion mit den Formularereignissen On Before Keystroke und On After Keystroke funktioniert wie folgt:

- Sobald ein Zeichen auf der Tastatur getippt wird, wird das Ereignis On Before Keystroke generiert. In diesem Fall gibt **Get edited text** den Inhalt des Bereichs zurück, bevor die letzte Tasteneingabe passiert. Enthält der Bereich z.B. "PA" und tippt der Benutzer ein "R", gibt **Get edited text** "PA" im Ereignis On Before Keystroke zurück. Ist der Bereich anfangs leer, gibt **Get edited text** einen leeren String zurück.
- Als nächstes wird das Formularereignis generiert. In diesem Fall gibt **Get edited text** den Inhalt des Bereichs einschließlich der letzten Tasteneingabe zurück. Enthält der Bereich z.B. "PA" und tippt der Benutzer ein "R", gibt **Get edited text** "PAR" im Ereignis On After Keystroke zurück.

Diese Funktion gibt in einem Formularobjekt in einem anderen Kontext als Texteingabe einen leeren String zurück.

Beispiel 1

Folgende Methode wandelt die eingegebenen Zeichen automatisch in Großbuchstaben um:

```
If(Form event=On After Keystroke)
   [Trips]Agencies:=Uppercase(Get edited text)
End if
```

Beispiel 2

Folgendes Beispiel zeigt, wie Sie in ein Textfeld eingegebene Zeichen per Programmierung "on the fly" entnehmen. Die Idee dabei ist, alle Wörter des eingegebenen Satzes in ein anderes Textfeld mit Namen "Words" zu setzen. Dazu schreiben Sie in der Objektmethode für dieses Feld folgenden Code:

```
If(Form event=On After Keystroke)
   $RealTimeEntry:=Get edited text
   PLATFORM PROPERTIES($platform)
   If($platform#3) ` MacOS
      Repeat
         $DecomposedSentence:=Replace string($RealTimeEntry;Char(32);Char(13))
      Until(Position(" ";$DecomposedSentence)=0)
   Else ` Windows
      Repeat
         $DecomposedSentence:=Replace string($RealTimeEntry;Char(32);Char(13)+Char(10))
      Until(Position(" ";$DecomposedSentence)=0)
   End if
   [Example]Words:=$DecomposedSentence
End if
```

Hinweis: Dieses Beispiel ist nicht voll ausgeschöpft, da wir lediglich die Wörter berücksichtigt haben, die durch Leerzeichen (Char (32)) voneinander getrennt sind. Für eine vollständige Lösung müssen Sie weitere Filter hinzufügen, um auch Wörter zu entnehmen, die durch andere Zeichen begrenzt sind, wie z.B. Kommas, Strichpunkte, Apostrophe.

GET HIGHLIGHT

GET HIGHLIGHT ({* ;} Objekt ; StartAusw ; EndeAusw)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *, Objekt ist ein Objektname (String), Ohne *, Objekt ist Feld oder Variable
Objekt	Feld, Variable, Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
StartAusw	Lange Ganzzahl	⬅	Startpunkt der aktuellen Textauswahl
EndeAusw	Lange Ganzzahl	⬅	Endpunkt der aktuellen Auswahl

Beschreibung

Der Befehl **GET HIGHLIGHT** gibt die Position des ausgewählten Textes des aktuellen Objekts zurück.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (Zeichenkette) ist. Ohne den Parameter * geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie anstelle einer Zeichenkette die Feld- oder Variablenreferenz (nur Formularfelder oder Variablen).

Hinweis: Sie können diesen Befehl nicht für Felder eines Ausgabeformulars in einem Unterformular verwenden.

Der Text kann entweder vom Benutzer oder über den Befehl **HIGHLIGHT TEXT** markiert werden.
Der Parameter *StartAusw* gibt die Position des ersten markierten Zeichens zurück. Der Parameter *EndeAusw* gibt die Position nach dem letzten markierten Zeichen zurück. Haben *StartAusw* und *EndeAusw* denselben Wert, wird die Einfügemarke vor dem ersten markierten Zeichen gesetzt. In diesem Fall hat der Benutzer keinen Text ausgewählt, es werden keine Zeichen markiert.

Wird das im Parameter *Objekt* angegebene Objekt nicht im Formular gefunden, gibt der Befehl in *StartAusw* -1 und in *EndeAusw* -2 zurück.

Beispiel 1

Folgendes Beispiel erhält den markierten Text aus dem Datenfeld *[Products]Comments*:

```
GET HIGHLIGHT([Products]Comments;vFirst;vLast)
If(vFirst<vLast)
   ALERT("Der ausgewählte Text ist: "+Substring([Products]Comments;vFirst;vLast-vFirst))
End if
```

Beispiel 2

Siehe Beispiel zum Befehl **FILTER KEYSTROKE**.

Beispiel 3

Markierten Textstil verändern:

```
GET HIGHLIGHT(*;"meinText";$startsel,$endsel)
ST SET ATTRIBUTES(*;"meinText";$startsel,$endsel;Attribute underline style;1;Attribute bold style;1)
```

GOTO OBJECT

GOTO OBJECT ({* ;} Objekt)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Angabe = Objekt ist ein Objektname (string) Ohne Angabe = Objekt ist Feld oder Variable
Objekt	Feld, Variable	➡	Objektname (wenn * angegeben) oder Feld bzw. Variable (wenn * nicht angegeben)

Beschreibung

Der Befehl **GOTO OBJECT** wählt das Objekt zum Einfügen von Daten, definiert in *Objekt*, als den aktiven Bereich des Formulars. Das entspricht dem Klicken des Benutzers in das Datenfeld oder die Variable.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen an, d.h. einen String. Ohne den optionalen Parameter * geben Sie in *Objekt* eine Feld- oder eine Variablenreferenz an, also keinen String. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Um im aktuellen Formular jeden Fokus zu entfernen, übergeben Sie in *Objekt* einen leeren Objektnamen (siehe Beispiel 2).

Der Befehl **GOTO OBJECT** lässt sich auch im Kontext eines Unterformulars verwenden. Wird er von einem Unterformular aufgerufen, sucht er erst nach Objekt im Unterformular, wird dort nichts gefunden, weitet er die Suche auf Objekte des Elternformulars aus.

Beispiel 1

Sie können **GOTO OBJECT** auf beide Arten anwenden:

```
GOTO OBJECT([People]Name)  ` Referenz auf Feld
GOTO OBJECT(*;"AgeArea")  ` Objektname
```

Beispiel 2

Kein Objekt des Formulars soll Fokus haben

```
GOTO OBJECT(*;"")
```

Beispiel 3

Siehe Beispiel für den Befehl **REJECT**.

HIGHLIGHT TEXT

HIGHLIGHT TEXT ({* ;} Objekt ; StartAusw ; EndeAusw)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Feld oder Variable
Objekt	Feld, Variable, Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
StartAusw	Lange Ganzzahl	➡	Startpunkt der neuen Textauswahl
EndeAusw	Lange Ganzzahl	➡	Endpunkt der neuen Textauswahl

Beschreibung

Der Befehl **HIGHLIGHT TEXT** markiert einen Bereich des Texts, angegeben in *Objekt*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (Zeichenkette) ist. Ohne den Parameter * geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie anstelle einer Zeichenkette die Feld- oder Variablenreferenz (nur Formularfelder oder Variablen).

Ist *Objekt* nicht das aktuell bearbeitete Objekt, erhält es den Fokus.

Hinweis: Sie können diesen Befehl nicht auf Felder in einem Unterformular anwenden.

StartAusw ist die Position des ersten zu markierenden Zeichens, *EndeAusw* ist das letzte zu markierende Zeichen plus eins. Sind *StartAusw* und *EndeAusw* gleich, wird der Einfügepunkt vor das in *StartAusw* angegebene Zeichen gesetzt und es werden keine Zeichen markiert.

Ist *EndeAusw* größer als die Anzahl Zeichen in *Objekt*, werden alle Zeichen zwischen *StartAusw* und *EndeAusw* und das Textende markiert.

Beispiel 1

Folgendes Beispiel wählt alle Zeichen des eingebbaren Feldes*[Products]Comments*:

```
HIGHLIGHT TEXT([Products]Comments;1;Length([Products]Comments)+1)
```

Beispiel 2

Folgendes Beispiel legt die Einfügemarke an den Anfang des eingebbaren Feldes *[Products]Comments*:

```
HIGHLIGHT TEXT([Products]Comments;1;1)
```

Beispiel 3

Folgendes Beispiel legt die Einfügemarke an das Ende des eingebbaren Feldes*[Products]Comments*:

```
$vLen:=Length([Products]Comments)+1
HIGHLIGHT TEXT([Products]Comments;$vLen;$vLen)
```

Beispiel 4

Siehe Beispiel zum Befehl **FILTER KEYSTROKE**.

Keystroke

Keystroke -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	String	Vom Benutzer eingegebene Zeichen

Beschreibung

Die Funktion **Keystroke** gibt das Zeichen zurück, das der Benutzer in ein Datenfeld oder einen eingebbaren Bereich eingetippt hat.

Sie rufen **Keystroke** im allgemeinen bei einem Formularereignis *On Before Keystroke* in einem Formular oder einer Objektmethode auf. Mit der Funktion **Form event** können Sie Tastaturanschläge finden.

Mit dem Befehl **CHANGE CURRENT USER** können Sie soeben vom Benutzer eingegebene Anschläge durch andere ersetzen.

WICHTIGER HINWEIS: Wollen Sie einige Operationen "on the fly" ausführen, beachten Sie, dass der Text auf dem Bildschirm NOCH NICHT der Wert des Quelldatenfeldes oder der Variablen für den zu bearbeitenden Bereich ist. Der eingegebene Wert, sei es der aktuelle Wert des bearbeiteten eingebbaren Bereichs oder ein neu eingetragenes Zeichen, wird dem Quelldatenfeld oder der Variablen erst zugewiesen, wenn die Dateneingabe für den Bereich bestätigt wurde. Das kann ein Tab in einen anderen Bereich, das Anklicken einer Schaltfläche uvm. sein. Sie können zu diesem Zweck die Dateneingabe in einer Variablen spiegeln und mit diesem Wert arbeiten. Diese Vorgehensweise ist erforderlich, wenn Sie zum Ausführen spezifischer Aktionen den aktuellen Textwert wissen müssen.

Sie verwenden die Funktion **Keystroke**, um:

- Zeichen auf eigene Weise zu filtern
- Die Dateneingabe zu filtern, die Sie nicht über Dateneingabefilter ausführen können.
- Dynamische Nachschlag- oder Fortschreibbereiche (lookup oder type-ahead) zu integrieren

Beispiel 1

Siehe Beispiele für den Befehl **FILTER KEYSTROKE**.

Beispiel 2

Beim Durchführen eines Ereignisses *On Before Keystroke* arbeiten Sie mit dem aktuellen Textbereich, d.h. wo der Cursor steht und nicht mit "dem zukünftigen Wert" der Datenquelle (Datenfeld oder Variable) für diesen Bereich. Mit der Projektmethode ***Handle keystroke*** können Sie jede Eingabe in den Textbereich in eine zweite Variable spiegeln. Damit können Sie dann die Aktionen während der Eingabe von Zeichen in den Bereich ausführen. Sie übergeben als ersten Parameter einen Zeiger auf die Datenquelle für den Bereich und als zweiten Parameter einen Zeiger auf die Spiegelvariable. Die Methode gibt den neuen Wert des Textbereichs in der Spiegelvariablen zurück und meldet **True**, wenn sich der Wert von dem unterscheidet, was vor dem zuletzt eingegebenen Zeichen eingefügt wurde.

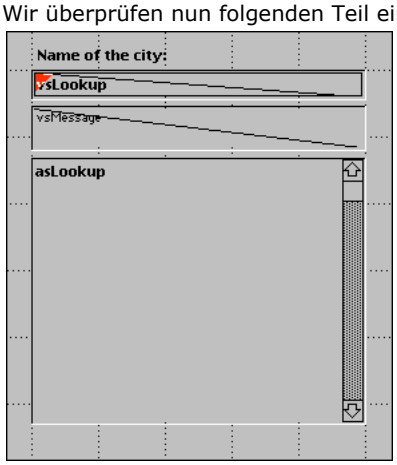
```
  ` Verwalte Projektmethode Tastaturanschlag
  ` Verwalte Tastaturanschlag ( Zeiger ; Zeiger ) -> Boolean
  ` Verwalte Tastaturanschlag ( -> srcArea ; -> curValue ) -> Ist neuer Wert

C_POINTER($1;$2)
C_TEXT($vtNewValue)

  ` Hole Textauswahl innerhalb des eingebbaren Bereichs
GET HIGHLIGHT($1->;$vlStart;$vlEnd)
  ` Beginne mit dem aktuellen Wert zu arbeiten
$vtNewValue:=$2->
  ` Führe nach gedrückter Taste oder eingegebenem Zeichen,
  ` die entsprechenden Aktionen aus
Case of

  ` Die Rückschrittaste (Löschen) wurde gedrückt
   :(Character code(Keystroke)=Backspace)
  ` Lösche die oder das links vom Cursor ausgewählte Zeichen
      $vtNewValue:=Substring($vtNewValue;1;$vlStart-1-Num($vlStart=$vlEnd))+Substring($vtNewValue;$vlEnd)

  ` Ein zulässiges Zeichen wurde eingegeben.
   :(Position(Keystroke;"abcdefghjiklmnopqrstuvwxyz -0123456789")>0)
      If($vlStart#$vlEnd)
  ` Ein oder mehrere Zeichen wurden ausgewählt, der Tastaturanschlag wird sie überspringen
         $vtNewValue:=Substring($vtNewValue;1;$vlStart-1)+Keystroke+Substring($vtNewValue;$vlEnd)
      Else
  ` Die Textauswahl ist der Cursor
         Case of
```

```
    ` Der Cursor ist derzeit am Textbeginn
            :($vlStart<=1)
    ` Füge Zeichen am Textbeginn ein
            $vtNewValue:=Keystroke+$vtNewValue
    ` Der Cursor ist derzeit am Textende
            :($vlStart>=Length($vtNewValue))
    ` Hänge Zeichen an Textende an
            $vtNewValue:=$vtNewValue+Keystroke
          Else
    ` Der Cursor steht mitten im Text, füge hier neues Zeichen ein
            $vtNewValue:=Substring($vtNewValue;1;$vlStart-1)+Keystroke
            +Substring($vtNewValue;$vlStart)
       End case
    End if

    ` Eine Pfeiltaste wurde gedrückt
    ` Tue nichts, nimm jedoch den Tastaturanschlag an
    :(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
    ` Akzeptiere als Zeichen nur Buchstaben, Ziffern, Leerzeichen und Bindestrich
      FILTER KEYSTROKE("")
End case
  ` Unterscheidet sich der Wert vom vorigen?
$0:=($vtNewValue#$2->)
  ` Gib den Wert für die Bearbeitung des nächsten Tastaturanschlags zurück
$2->:=$vtNewValue
```

Sie können diese Projektmethode folgendermaßen einsetzen:

```
  ` Objektmethode Eingebbarer Bereich MyObject
Case of
   :(Form event=On Load)
      MyObject:=""
      MyShadowObject:=""
   :(Form event=On Before Keystroke)
      If(Handle keystroke(->MyObject;->MyShadowObject))
   ` Führe die entsprechenden Aktionen mit dem in MyShadowObject gespeicherten Wert aus
      End if
End case
```

Wir überprüfen nun folgenden Teil eines Formulars:



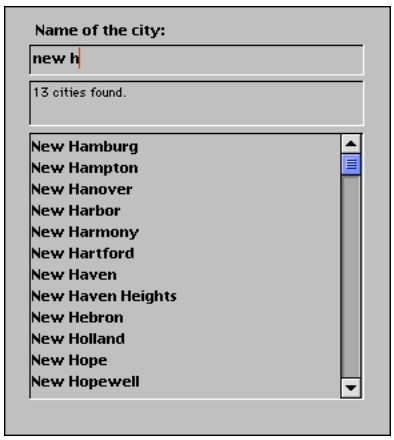
Es besteht aus einem eingebbaren Bereich *vsLookup*, einem nicht eingebbaren Bereich *vsMessage* und einem rollbaren Bereich *asLookup*. Beim Eingeben von Zeichen in *vsLookup* startet die Methode für dieses Objekt eine Suche auf die Tabelle [PLZ]. Der Benutzer muss nur die ersten Buchstaben eintippen, um eine bestimmte Stadt in Deutschland zu finden.

Die Objektmethode *vsLookup* lautet folgendermaßen:

```
  ` Objektmethode Eingebbarer Bereich vsLookup
Case of
   :(Form event=On Load)
      vsLookup:=""
      vsResult:=""
      vsMessage:="Gib die ersten Buchstaben der gesuchten Stadt ein."
```

```
            CLEAR VARIABLE(asLookup)
        :(Form event=On Before Keystroke)
            If(Handle keystroke(->vsLookup;->vsResult))
                If(vsResult#"")
                    QUERY([PLZ];[PLZ]Stadt=vsResult+"@")
                    MESSAGES OFF
                    DISTINCT VALUES([PLZ]Stadt;asLookup)
                    MESSAGES ON
                    $vlResult:=Size of array(asLookup)
                    Case of
                        :($vlResult=0)
                            vsMessage:="Es wurde keine Stadt gefunden."
                        :($vlResult=1)
                            vsMessage:="Es wurde eine Stadt gefunden."
                        Else
                            vsMessage:=String($vlResult)+" gefundene Städte."
                    End case
                Else
                    DELETE FROM ARRAY(asLookup;1;Size of array(asLookup))
                    vsMessage:="Gib die ersten Buchstaben der gesuchten Stadt ein."
                End if
            End if
    End case
```

Hier ist das Formular in der Anwendungsumgebung:



Mit den Funktionalitäten der Interprozesskommunikation von 4D können Sie ähnliche Benutzeroberflächen gestalten, die in Palettenfenstern Features zum Nachschlagen anzeigen, mit Prozessen kommunizieren, Datensätze auflisten oder bearbeiten.

Fenster

- Fenster verwalten
- Fenstertypen
- 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*
- Fenstertypen (Kompatibilität)

Fenster verwalten

Fenster haben die Funktion, Informationen für den Benutzer anzuzeigen. Es gibt drei Hauptbereiche: Daten eingeben, Daten anzeigen und den Benutzer durch Meldungen und Dialoge informieren.

Es ist immer mindestens ein Fenster offen. Bei Bedarf werden Rollbalken hinzugefügt, damit der Benutzer im Formular scrollen kann, wenn es größer als das Fenster ist. Dieses Fenster zeigt in der Designumgebung entweder die Liste der Datensätze (Ausgabeformular) oder einen Datensatz für die Dateneingabe (Eingabeformular) an. In der Anwendungsumgebung ist es der Startbildschirm mit dem 4D Logo oder einem eigenen Bild.

Rufen Sie einen Menübefehl in einem Anwendungsprozess auf, wird der Startbildschirm durch Daten aus dem Formular ersetzt. Sind die Befehle ausgeführt, erscheint standardmäßig wieder der Startbildschirm.

WinRef

Mit der Funktion **Open window** oder **Open form window** können Sie verschiedene Arten angepasster Fenster öffnen. Weitere Informationen dazu finden Sie im Abschnitt **Fenstertypen (Kompatibilität)**. Auf alle so geöffneten Fenster wird über einen Ausdruck *WinRef* Bezug genommen. Das ist eine einmalige ID vom Typ Lange Ganzzahl für jedes geöffnete Fenster. Alle Befehle, die mit angepassten Fenstern arbeiten, erwarten einen Parameter *WinRef*.

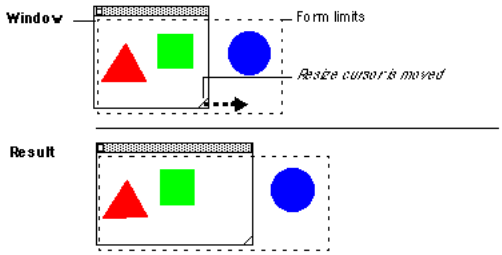
Wenn Sie das angepasste Fenster nicht mehr benötigen, schließen Sie es mit dem Befehl **CLOSE WINDOW** oder klicken - sofern vorhanden - unter Windows in die Control-Menübox, auf Macintosh in das Schließkästchen.

Bestimmte Befehle wie **QR REPORT** und **PRINT LABEL** öffnen ihr eigenes Fenster und legen es am Bildschirm ganz nach vorne.

Starten Sie einen neuen Prozess und öffnen kein Fenster zu Beginn der Prozessmethode, öffnet 4D automatisch ein Standardfenster, sobald ein Formular angezeigt wird.

Seitenschieber

Die rechte und untere Seite von Fenstern sind standardmäßig mit "Seitenschiebern" ausgestattet. Das heißt, Objekte rechts oder unterhalb der Begrenzung eines Fensters auf dem Bildschirm werden beim Vergrößern des Fensters automatisch nach rechts oder unten geschoben:



Diese Funktionsweise ermöglicht, zurücknehmbare Fenster wie das Explorer-Fenster zu verwalten (siehe Beispiel zum Befehl **FORM SET SIZE**).

Hinweis: Das funktioniert nicht mit Fenstern, die Rollbalken haben.

Koordinaten für Fenster und der Modus "rechts-nach-links"

Bei den Befehlen zur Fensterverwaltung werden die Koordinaten des Fensters in Bezug auf einen Ursprungspunkt bestimmt. Er liegt in der Regel in der linken oberen Ecke des Fensters/Bildschirms.
Ist dagegen der Modus "rechts-nach-links" für die Anwendung aktiviert, werden die Koordinaten umgekehrt und der Ursprungspunkt wechselt in die rechte obere Ecke des Fensters/Bildschirms. Folglich müssen dann auch die horizontalen Koordinaten umgekehrt werden, die folgende Routinen verwenden:

Open window
Open form window
_o_Open external window
GET WINDOW RECT
SET WINDOW RECT
Find window

Hinweis: Weitere Informationen zum "rechts-nach-links" Modus finden Sie im Handbuch *4D Designmodus* und in der Beschreibung zum Befehl **SET DATABASE PARAMETER**.

Fenstertypen

Beschreibung

Mit folgenden vordefinierten Konstanten unter dem Thema **Open form window** legen Sie den Typ des Fensters fest, das Sie mit der Funktion **Open form window** öffnen möchten:

Konstante	Typ	Wert
Modal form dialog box	Lange Ganzzahl	1
Movable form dialog box	Lange Ganzzahl	5
Plain form window	Lange Ganzzahl	8
Pop up form window	Lange Ganzzahl	32
Sheet form window	Lange Ganzzahl	33
Toolbar form window	Lange Ganzzahl	35
Palette form window	Lange Ganzzahl	1984
Form has no menu bar	Lange Ganzzahl	2048
Form has full screen mode Mac	Lange Ganzzahl	65536
Controller form window	Lange Ganzzahl	133056

Dieser Abschnitt zeigt die verschiedenen Fenstertypen unter Windows (links) und macOS (rechts).

Modales Fenster

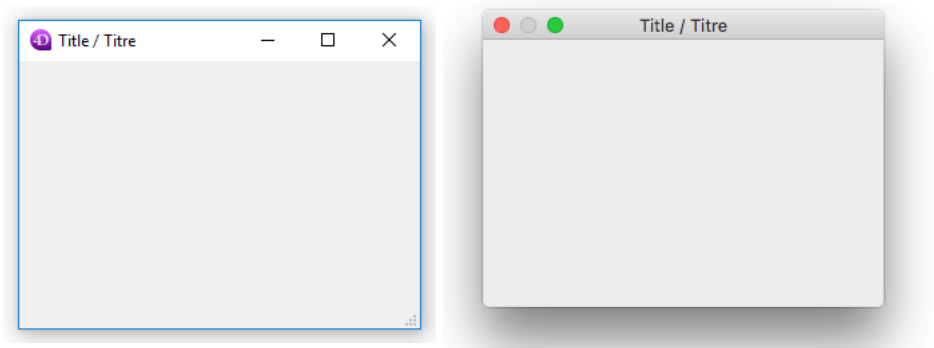
Ein modales Fenster beschränkt die Aktionen des Benutzers auf dieses Fenster. Solange es angezeigt wird, sind die Menübefehle und andere Anwendungsfenster nicht zugänglich. Der Benutzer muss erst das modale Fenster schließen: Er kann es bestätigen, annullieren oder eine angebotene Option wählen. Dialogfenster mit Warnungen oder Meldungen sind ein typisches Beispiel für modale Fenster.
In 4D sind die Fenster vom Typ 1 und 5 modale Fenster.

Hinweis: Ein modales Fenster bleibt immer im Vordergrund. Folglich erscheint ein nicht-modales Fenster, das vom modalen Fenster aufgerufen wird, im Hintergrund, selbst wenn es nach dem modalen Fenster aufgerufen wird. Deshalb sollten Sie eine derartige Operation vermeiden.
Ruft dagegen ein modales Fenster ein anderes modales Fenster auf, erscheint dieses Fenster im Vordergrund.

Modal dialog box (1)

- Kann Titel haben: Nein
- Kann Schließbox o.ä. haben: Nein
- Kann in der Größe verändert werden: Nein
- Kann auf max. bzw. min. Größe oder auf Bildschirmgröße gestellt werden: Nein
- Eignet sich für Rollbalken: Nein
- Verwendung: **DIALOG**, **ADD RECORD**(...;...*) o.ä.
- Fenster dieses Typs sind modal.

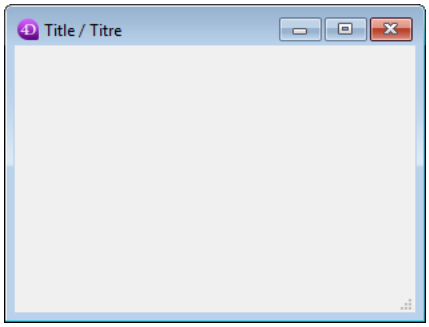
Movable dialog box (5)

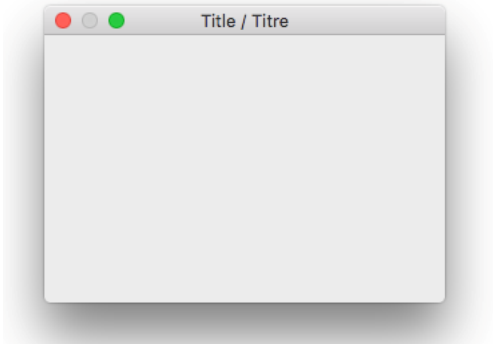


- Kann einen Titel haben: Ja
- Kann eine Schließbox o.ä. haben: Ja
- Kann in der Größe verändert werden: Ja (nicht verfügbar in 4D 32-bit Versionen unter Windows)

- Kann auf max. bzw. min. Größe oder auf Bildschirmgröße gestellt werden: Ja (nicht verfügbar in 4D 32-bit Versionen unter Windows)
- Eignet sich für Rollbalken: Nein
- Verwendung: **DIALOG**, **ADD RECORD**(...;...*) o.ä.
- Fenster dieses Typs sind modal, können jedoch bewegt werden.

Plain window (8)





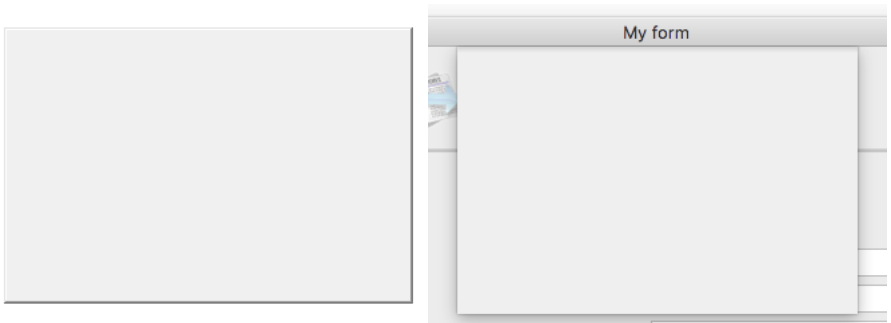
- Kann Titel haben: Ja
- Kann Schließbox o.ä. haben: Ja
- Kann in der Größe verändert werden: Ja
- Kann auf max. bzw. min. Größe oder auf Bildschirmgröße gestellt werden: Ja
- Eignet sich für Rollbalken: Ja
- Verwendung: Dateneingabe mit Rollbalken, **DISPLAY SELECTION**, **MODIFY SELECTION**, etc.

Pop up window (32)

Dieses Fenster unterstützt folgende weitere Eigenschaften:

- Das Fenster kann keine Schließbox haben, wird aber automatisch geschlossen und das Ereignis “Abbrechen” wird übergeben, wenn:
  - Ein Klick außerhalb des Fensters erfolgt,
  - Das Hintergrund- bzw. MDI (Multiple Document Interface) Fenster bewegt wird,
  - Der Benutzer auf die Taste **Esc** klickt.
- Dieses Fenster erscheint vor seinem Hauptfenster (es darf nicht als Hauptfenster des Prozesses verwendet werden). Das Hintergrundfenster wird bei Anzeige des Fensters nicht deaktiviert. Es empfängt jedoch keine Ereignisse mehr.
- Sie können das Fenster mit der Maus weder anpassen noch bewegen; führen Sie diese Aktionen jedoch per Programmierung aus, wird das Neuzeichnen der Hintergrundelemente optimiert.
- Verwendung: Dieser Fenstertyp dient hauptsächlich zum Verwalten von PopUp-Menüs, die mit 3D Schaltflächen vom Typ “bevel” oder mit Icons der Werkzeugpalette verknüpft sind.
- Einschränkungen:
  - Innerhalb díeses Fensters lassen sich keine PopUp-Menü Objekte anzeigen
  - Dieser Fenstertyp unterstützt ab 4D Version 13 nicht mehr die Anzeige von Hilfetipps auf Mac OS.

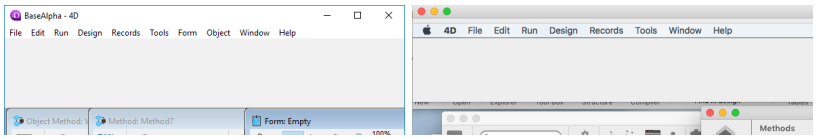
Sheet window (33)



Sheet Fenster sind spezifisch für macOS. Sie rollen animiert unter der Titelleiste des Hauptfensters auf und liegen vor diesem Fenster. Sie haben ähnliche Eigenschaften wie modale Dialogfenster. Sie dienen dazu, eine Operation auszuführen, die mit der Aktion im Hauptfenster zusammenhängt. In 4D wird z.B. ein Sheet Fenster ausgelöst, wenn der Benutzer im Etiketteneditor auf die Schaltfläche **Sichern** klickt.

- Sie können ein Sheet Fenster auf macOS nur erstellen, wenn das letzte offene Fenster sichtbar und ein Dokumenttyp (Formular) ist.
- Die Funktion öffnet ein Fenster vom Typ 1 (Modal dialog box) statt vom Typ 33:
  - wenn das zuletzt geöffnete Fenster nicht sichtbar ist oder kein Dokumenttyp ist,
  - unter Windows.
- Da ein Sheet Fenster über ein Formular gezeichnet wird, wird seine Anzeige im Ereignis On Load des ersten in das Fenster geladenen Formulars zurückgesetzt (siehe Beispiel 3 unter der Funktion **Open form window**).
- Verwendung: **DIALOG**, **ADD RECORD**(...;...*) o.ä, auf macOS (nicht Standard unter Windows).

Toolbar form window (35)



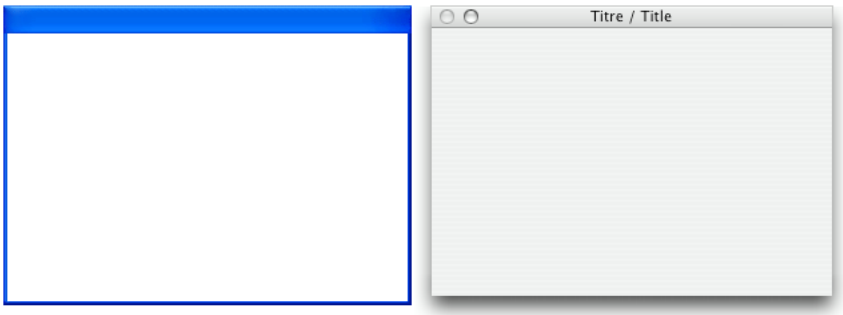
Ein Formularfenster mit Toolbar wird mit der Position, Größe und den grafischen Eigenschaften einer Toolbar erstellt, dabei gilt folgendes:

Ist die Konstante Toolbar form window übergeben, wird das Fenster mit der Position, Größe und den grafischen Eigenschaften der Toolbar erstellt. Hier ein paar Beispiele:

- Das Fenster erscheint immer direkt unter der Menüleiste.
- Die Breite des Fensters wird automatisch an den horizontalen Bereich angepasst, der auf dem Desktop (auf macOS und unter Windows im SDI Modus) oder im Hauptfenster von 4D (unter Windows im MDI Modus) verfügbar ist. Die Höhe richtet sich wie bei allen anderen Arten von Formularfenstern nach den Formulareigenschaften.
- Das Fenster hat keinen Rand. Es lässt sich weder bewegen noch manuell größer oder kleiner ziehen.
- Im selben Prozess können nicht gleichzeitig zwei unterschiedliche Fenster mit Toolbar angelegt werden. In diesem Fall wird der Fehler -10613 ("Cannot create two form windows of type toolbar") zurückgegeben.

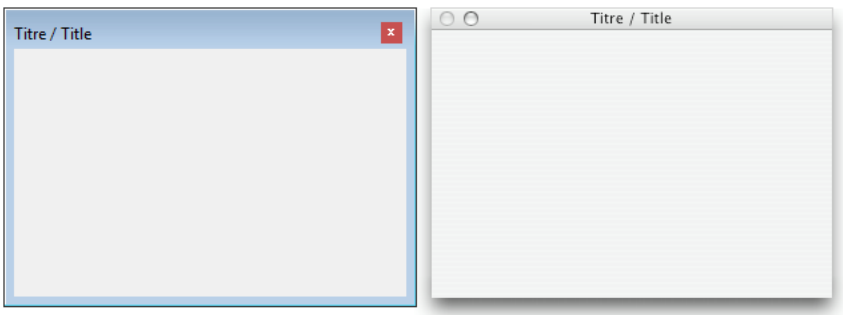
Formularfenster mit Toolbar und Vollbildmodus in macOS: Zeigt Ihre Anwendung ein Fenster mit Toolbar und ein Standardfenster an, das den Vollbildmodus unterstützt (Option Form has full screen mode Mac), muss die Toolbar gemäß den Oberflächenregeln ausgeblendet werden, wenn das Standardfenster in den Vollbildmodus geht. Ob das Fenster im Vollbildmodus ist, erkennen Sie daran, dass seine vertikale Größe genau der Höhe des Bildschirms entspricht (siehe Befehl **HIDE TOOL BAR**).

Palette window (1984)



Mit diesem Fenstertyp können Sie **Palettenfenster** einrichten. Ihr Hauptmerkmal ist, dass sie im Vordergrund erhalten bleiben, auch wenn der Benutzer auf ein anderes Fenster des Prozesses klickt. Sie dienen im allgemeinen zur Anzeige dauerhafter Informationen oder Toolbars.

Controller form window (133056)



Dieser Fenstertyp ist ähnlich zum Palette form window mit folgender Besonderheit: Unter Windows wird über ein Icon in der Werkzeugleiste eine Referenz auf das Palettenfenster gesetzt (unter Windows erscheinen reguläre Palettenfenster nicht in der Werkzeugleiste).

Dieser Fenstertyp ist hilfreich, wenn die Anwendung im **SDI Modus unter Windows** läuft. In diesem Modus werden Palettenfenster ausgeblendet, wenn das übergeordnete Anwendungsfenster in den Hintergrund gesetzt wird. Basiert ihre Anwendungsoberfläche nur auf einem einzigen Palettenfenster (z.B. zum Anzeigen einer Bildschirmansicht), müssen Sie ein Controller form window verwenden, um in der Werkzeugleiste eine Referenz auf die Anwendung zu setzen und sicherzustellen, dass es erreichbar bleibt, auch wenn es im Hintergrund liegt.

Beispiel:

```
$win:=Open form window("myMonitor";Controller form window;On the left;Vertically centered)
```

Hinweis: Auf macOS verhält sich dieses Fenster wie ein reguläres Palette form window.

Form has no menu bar (2048)

Diese Variante wird verwendet, wenn die Anwendung im **SDI Modus unter Windows** läuft.

In diesem Kontext zeigen alle Fenster Ihrer Anwendung standardmäßig die Menüleiste des aktuellen Prozesses. Wollen Sie ein Fenster ohne Menüleiste öffnen, müssen Sie im Parameter *Typ* die Konstante Form has no menu bar hinzufügen. Beispiel: Nachfolgender Code erstellt in einer SDI Applikation unter Windows ein Fenster vom Typ "plain form window" ohne Menüleiste:

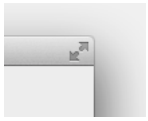
```
$win:=Open form window("myPanel";Plain form window+Form has no menu bar;Horizontally centered;At the top)
```

Hinweis: Diese Option hat keine Auswirkung:

- in einer macOS Anwendung
- in einer Windows Anwendung im MDI Modus

Has full screen mode Mac (65536)

Ab 4D v14 auf OS X gibt es für Fenster vom Typ Dokument die Option "full screen". Mit dieser Option erscheint in der rechten oberen Ecke des Fensters das Icon für Vollbildmodus:



Klickt der Benutzer auf dieses Icon, wechselt das Fenster auf Bildschirmgröße und 4D blendet automatisch die Hauptwerkzeugleiste aus.
Dazu fügen Sie im Parameter *Typ* die Konstante Form has full screen mode Mac hinzu. Der folgende Code erstellt ein Formularfenster mit Icon für Vollbildmodus auf macOS:

```
$win:=Open form window([Interface];"User_Choice";Plain form window+Form has full screen mode Mac)
DIALOG([Interface];"User_Choice")
```

Hinweis: Unter Windows hat diese Option keine Auswirkung.

CLOSE WINDOW

CLOSE WINDOW {(FensterRef)}

Parameter	Typ		Beschreibung
FensterRef	WinRef	⇒	Referenznr des Fensters, ohne Angabe vorderstes Fenster des aktuellen Prozesses

Beschreibung

Der Befehl **CLOSE WINDOW** schließt das zuletzt mit der Funktion **Open window** oder **Open form window** geöffnete Fenster im aktuellen Prozess. Wenn kein eigenes Fenster geöffnet ist, wird **CLOSE WINDOW** nicht ausgeführt. Der Befehl schließt weder Standardfenster, noch Fenster, wenn ein Formular aktiv ist. Ein mit **Open window** oder **Open form window** geöffnetes Fenster müssen Sie mit **CLOSE WINDOW** schließen.

Sie müssen für **CLOSE WINDOW** keine Nummer übergeben, da dieser Befehl immer das zuletzt mit **Open window** oder **Open form window** geöffnete Fenster schließt.

Übergeben Sie im Parameter *FensterRef* eine Referenznummer auf ein externes Fenster, das mit der Funktion **_o_Open external window** erstellt wurde, schließen Sie das angegebene Fenster.
Der Parameter ist optional. Weitere Informationen zu externen Fenstern finden Sie in der Beschreibung zu **_o_Open external window**.

Beispiel

Folgendes Beispiel öffnet ein Formularfenster und fügt mit dem Befehl **ADD RECORD** neue Datensätze hinzu. Anschließend wird das Fenster mit **CLOSE WINDOW** wieder geschlossen.

```
FORM SET INPUT([Employees];"Entry")
$winRef:=Open form window([Employees];"Entry")
Repeat
   ADD RECORD([Employees]) //Datensatz mit neuem Angestellten hinzufügen
Until(OK=0) //Die Schleife durchlaufen, bis der Benutzer abbricht
CLOSE WINDOW //Das Fenster schließen
```

CONVERT COORDINATES

CONVERT COORDINATES (xKoord ; YKoord ; Von ; Nach)

Parameter	Typ		Beschreibung
xKoord	Variable Lange Ganzzahl	➡	Horizontale Koordinate eines Punkts (Ausgang)
		⬅	Horizontale Koordinate eines Punkts (konvertiert)
YKoord	Variable Lange Ganzzahl	➡	Vertikale Koordinate eines Punkts (Ausgang)
		⬅	Vertikale Koordinate eines Punkts (konvertiert)
Von	Lange Ganzzahl	➡	Ausgehendes Koordinatensystem
Nach	Lange Ganzzahl	➡	Koordinatensystem, in das konvertiert wird

Beschreibung

Der Befehl **CONVERT COORDINATES** konvertiert die (x;y) Koordinaten eines Punkts von einem Koordinatensystem in ein anderes. Unterstützte Koordinatensysteme für Ein- und Ausgabe sind Formulare und Unterformulare, Fenster und Bildschirm. Mit diesem Befehl können Sie z.B. die Koordinaten im Hauptfenster für ein Objekt in einem Unterformular erhalten. Das macht das Erstellen eines Kontextmenüs an einer beliebigen eigenen Stelle einfacher.

In *xKoord* und *yKoord* übergeben Sie als Variablen die (x;y) Koordinaten des betreffenden Punkts. Nach Ausführen des Befehls enthalten diese Variablen die konvertierten Werte.

Im Parameter *Von* übergeben Sie das ausgehende Koordinatensystem, das den Eingabepunkt nutzt. Im Parameter *Nach* übergeben Sie das Koordinatensystem, in das konvertiert werden soll. Beide Parameter können eine der folgende Konstanten unter dem Thema "Fenster" nutzen:

Konstante	Typ	Wert	Kommentar
XY Current form	Lange Ganzzahl	1	Ursprung ist die linke obere Ecke des aktuellen Formulars
XY Current window	Lange Ganzzahl	2	Ursprung ist die linke obere Ecke des aktuellen Fensters
XY Screen	Lange Ganzzahl	3	Ursprung ist die linke obere Ecke des Hauptbildschirms (genauso wie für den Befehl **SCREEN COORDINATES**)
XY Main window	Lange Ganzzahl	4	Unter Windows: Ursprung ist die linke obere Ecke des Hauptfensters; auf OS X: genauso wie XY Screen

Wird dieser Befehl von der Methode oder einem Objekt des Unterformulars aufgerufen, und ist einer der Selektoren XY Current form, beziehen sich die Koordinaten auf das Unterformular selbst und nicht auf das übergeordnete Formular.

Beim Konvertieren aus/in die Position eines Formularfensters, beispielsweise beim Konvertieren der Ergebnisse von **GET WINDOW RECT**, oder in Werte, übergeben in **Open form window**, sollte XY Main window verwendet werden, da es sich hier um das Koordinatensystem handelt, das von Befehlen für Fenster unter Windows verwendet wird. Er lässt sich zu diesem Zweck auch auf OS X verwenden, hier entspricht er XY Screen.

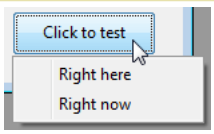
Stammt *Von* von XY Current form und liegt der Punkt im Hauptteil eines Listenformulars, richtet sich das Ergebnis nach dem jeweiligen Kontext des Befehls:

- Wird der Befehl während einem Ereignis On Display Detail aufgerufen, liegt der Ergebnispunkt in dem Datensatz, der auf dem Bildschirm angezeigt wird.
- Wird der Befehl außerhalb einem Ereignis On Display Detail aufgerufen, jedoch während der Bearbeitung eines Datensatzes, liegt der Ergebnispunkt in dem Datensatz, der bearbeitet wird.
- Ansonsten liegt der Ergebnispunkt im ersten angezeigten Datensatz.

Beispiel 1

Ein Pop-Up Menü in der linken unteren Ecke des Objekts "MyObject" aufrufen.

```
// OBJECT GET COORDINATES arbeitet im Koordinatensystem des aktuellen Formulars
// Das dynamische Pop-Up Menü verwendet das Koordinatensystem des aktuellen Fensters
// Wir müssen die Werte konvertieren
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)
```



Beispiel 2

Ein Pop-Up Fenster an der Position des Mauszeigers öffnen. Unter Windows müssen Sie die Koordinaten konvertieren, da **GET MOUSE** (mit dem Parameter *) Werte gemäß der Position des MDI Fensters zurückgibt:

```
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 -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	WinRef		Referenznummer des aktuellen Formularfensters

Beschreibung

Die Funktion **Current form window** gibt die Nummer des aktuellen Formularfensters zurück. Wurde für das aktuelle Formular kein Fenster definiert, gibt sie 0 (Null) zurück.

Das aktuelle Formularfenster lässt sich automatisch anlegen über einen Befehl wie **ADD RECORD**, in Folge einer Benutzeraktion oder über die Funktionen **Open window** bzw. **Open form window**.

DRAG WINDOW

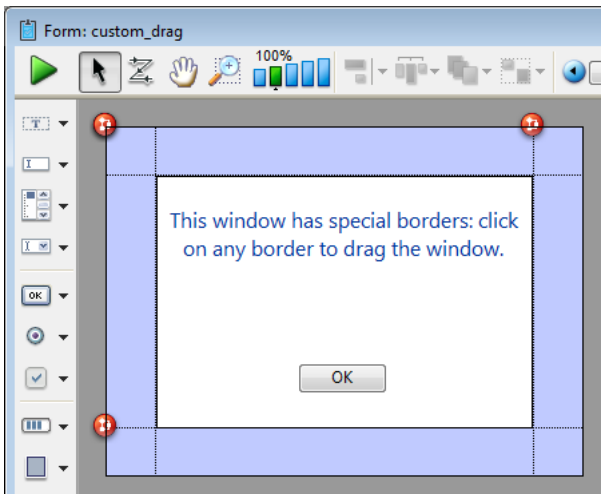
DRAG WINDOW
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **DRAG WINDOW** ermöglicht Benutzern, das angeklickte Fenster gemäß den Bewegungen mit der Maus zu ziehen. Sie können diesen Befehl normalerweise innerhalb einer Objektmethode eines Objekts aufrufen, das auf Mausklicks reagiert, wie z.B. unsichtbare Schaltflächen.

Beispiel

Folgendes Formular in der Designumgebung enthält einen Rahmen, der als statisches Bild erstellt wurde und in jeder Seite vier unsichtbare Schaltflächen enthält:



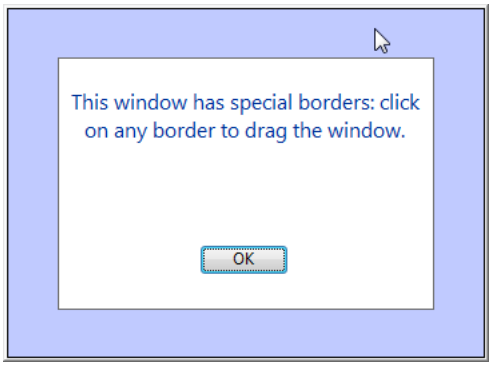
Jeder Schaltfläche ist folgende Methode zugeordnet:

```
DRAG WINDOW  //Beginne Fenster zu ziehen, wenn angeklickt
```

Nach Ausführen der Projektmethode:

```
$winRef:=Open form window("custom_drag";Modal form dialog box)
DIALOG("custom_drag")
CLOSE WINDOW
```

könnte Ihr Fenster in der Design- bzw. Anwendungsumgebung so aussehen:



Klicken Sie nun im Rahmen an eine beliebige Stelle, können Sie dieses Fenster frei ziehen.

ERASE WINDOW

ERASE WINDOW {(FensterRef)}

Parameter	Typ		Beschreibung
FensterRef	WinRef	⇒	Referenznr. des Fensters, ohne Angabe vorderstes Fenster des aktuellen Prozesses

Beschreibung

Der Befehl **ERASE WINDOW** löscht den Inhalt des aktuellen Fensters mit der Referenznr *FensterRef*.

Der Parameter *FensterRef* ist optional. Geben Sie ihn nicht an, löscht **ERASE WINDOW** den Inhalt des vordersten Fensters des aktuellen Prozesses.

Sie verwenden **ERASE WINDOW** im allgemeinen in Verbindung mit den Befehlen **MESSAGE** und **GOTO XY**. In diesem Fall löscht **ERASE WINDOW** den Inhalt des Fensters und setzt den Cursor in die obere linke Ecke, die Position **GOTO XY** (0;0).

Verwechseln Sie den Befehl **ERASE WINDOW** (löscht den Inhalt eines Fensters) nicht mit dem Befehl **CLOSE WINDOW** (entfernt das Fenster vom Bildschirm).

Find window

Find window (Links ; Oben {; Fensterteil}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Links	Lange Ganzzahl	➡	Linke Position
Oben	Lange Ganzzahl	➡	Obere Position
Fensterteil	Lange Ganzzahl	⬅	3, wenn Fensterteil berührt wurde, sonst 0
Funktionsergebnis	WinRef	↩	Referenznummer des Fensters

Beschreibung

Die Funktion **Find window** gibt (sofern vorhanden) die Referenznummer des Fensters zurück, das zuerst von den in *Links* und *Oben* übergebenen Koordinaten berührt wird.

Ausgangspunkt für die Koordinaten ist unter Windows die linke obere Ecke des Anwendungsfensters, auf Macintosh die linke obere Ecke des Hauptbildschirms.

Der Parameter *Fensterteil* gibt 3 zurück, wenn das Fenster berührt wird, andernfalls 0.

Hinweis zur Kompatibilität: Ab 4D v14 sind die Konstanten unter dem Thema **Find window** obsolet.

Frontmost window

Frontmost window {(*)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Palettenfenster berücksichtigen Ohne *: Palettenfenster ignorieren
Funktionsergebnis	WinRef	➲	Referenznummer des Fensters

Beschreibung

Die Funktion **Frontmost window** gibt die Referenznummer des vordersten Fensters zurück.

GET WINDOW RECT

GET WINDOW RECT (Links ; Oben ; Rechts ; Unten {; FensterRef})

Parameter	Typ		Beschreibung
Links	Lange Ganzzahl	⇐	Linke Koordinate vom Innenbereich des Fensters
Oben	Lange Ganzzahl	⇐	Obere Koordinate vom Innenbereich des Fensters
Rechts	Lange Ganzzahl	⇐	Rechte Koordinate vom Innenbereich des Fensters
Unten	Lange Ganzzahl	⇐	Untere Koordinate vom Innenbereich des Fensters
FensterRef	WinRef	⇒	Referenznummer des Fensters oder vorderstes Fenster des aktuellen Prozesses Ohne Angabe oder MDI Fenster, wenn -1 (Windows)

Beschreibung

Der Befehl **GET WINDOW RECT** gibt die Koordinaten des Fensters mit der in *FensterRef* übergebenen Referenznummer zurück. Gibt es das Fenster nicht, bleiben die Parameter unverändert.

Der Parameter *FensterRef* ist optional. Geben Sie diesen Parameter nicht an, gilt **GET WINDOW RECT** für das vorderste Fenster des aktuellen Prozesses.

Ausgangspunkt für die Koordinaten ist die linke obere Ecke vom Innenbereich des Anwendungsfensters (Windows MDI Modus) oder vom Hauptbildschirm (Mac OS und Windows SDI Modus). Die Koordinaten geben den Innenbereich des Fensters an, d.h. ohne Titelleisten und Ränder.

Hinweis: Übergeben Sie unter Windows für *FensterRef* -1, gibt **GET WINDOW RECT** die Koordinaten des Anwendungsfensters zurück (MDI Fenster). Diese Koordinaten entsprechen dem Innenbereich des Fensters, d.h. ohne Menüleisten und Ränder. Bei einem Fenster im SDI Modus gibt **GET WINDOW RECT** als Koordinaten (0;0;0;0) zurück.

Beispiel

Siehe Beispiel zum Befehl **WINDOW LIST**.

Get window title

Get window title {(FensterRef)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
FensterRef	WinRef	⇒	Referenznummer des Fensters Ohne Angabe vorderstes Fenster des aktuellen Prozesses
Funktionsergebnis	String	⇐	Fenstertitel

Beschreibung

Die Funktion **Get window title** gibt den Titel des Fensters mit der Referenznummer *FensterRef* zurück. Gibt es das Fenster nicht, wird ein leerer String zurückgegeben.

Der Parameter *FensterRef* ist optional. Geben Sie Ihn nicht an, gibt **Get window title** den Titel des vordersten Fensters des aktuellen Prozesses zurück.

Beispiel

Siehe Beispiel zum Befehl **SET WINDOW TITLE**.

HIDE TOOL BAR

HIDE TOOL BAR
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **HIDE TOOL BAR** steuert die Anzeige eigener Toolbars, die über die Funktion **Open form window** für den aktuellen Prozess erstellt wurden.

Wurde über **Open form window** mit der Option Toolbar form window ein Fenster mit Toolbar angelegt, blendet der Befehl diesen Fenstertyp aus. Ist das Toolbar Fenster bereits ausgeblendet oder wurde kein Fenster dieses Typs angelegt, führt der Befehl nichts aus.

Beispiel

Sie haben auf OS X eine eigene Toolbar und ein Standardfenster mit der Option Has full screen mode Mac definiert. Zieht ein Benutzer ein Standardfenster auf Bildschirmgröße, während das Toolbar Fenster angezeigt wird, soll die Toolbar nicht das maximierte Fenster überlappen.

Dazu müssen Sie im Formularereignis On Resize des Standardfensters herausfinden, wenn das Fenster auf vollen Bildschirm geht und dann **HIDE TOOL BAR** aufrufen:

```
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 {(FensterRef)}

Parameter	Typ		Beschreibung
FensterRef	WinRef	➡	Referenznummer des Fensters Ohne Angabe vorderstes Fenster des aktuellen Prozesses

Beschreibung

Der Befehl **HIDE WINDOW** blendet das Fenster mit der in *FensterRef* übergebenen Nummer aus. Wird dieser Parameter nicht angegeben, erscheint das vorderste Fenster des aktuellen Prozesses. Mit diesem Befehl können Sie zum Beispiel in einem Prozess mit mehreren Unterprozessen nur das aktive Fenster anzeigen.
Das Fenster erscheint nicht mehr auf dem Bildschirm, bleibt aber geöffnet. Sie können weiterhin per Programmierung von 4D Fenstern unterstützte Änderungen anwenden.

Um ein mit **HIDE WINDOW** ausgeblendetes Fenster wieder einzublenden, gehen Sie folgendermaßen vor:

- Verwenden Sie den Befehl **SHOW WINDOW** und übergeben Sie die Referenznummer des Fensters.
- Verwenden Sie im **Runtime-Explorer** die Seite **Prozess**. Wählen Sie den Prozess, der das Fenster verwaltet und dann in der Werkzeugleiste die Schaltfläche **Einblenden**.

Wollen Sie alle Fenster eines Prozesses ausblenden, wählen Sie den Befehl **HIDE PROCESS**.

Beispiel

Dieses Beispiel zeigt die Methode für eine Schaltfläche in einem Eingabeformular. Die Schaltfläche öffnet ein Dialogfenster in einem neuen Fenster desselben Prozesses. Der Benutzer möchte die anderen Fenster des Prozesses (Eingabeformular und Werkzeugpalette) ausblenden, während dieses Fenster angezeigt wird. Bestätigt er es, erscheinen die anderen Fenster des Prozesses wieder.

```
  ` Objektmethode für die Schaltfläche "Information"
HIDE WINDOW(Eingabe)  ` Eingabefenster ausblenden
HIDE WINDOW(Palette)  ` Palette ausblenden
$Infos:=Open window(20;100;500;400;8)  ` Erstelle Fenster Information
...  ` Setze hier Anweisungen zum Verwalten des Dialogs
CLOSE WINDOW($Infos)  ` Schließe den Dialog
SHOW WINDOW(Eingabe)
SHOW WINDOW(Palette)  ` Zeige die anderen Fenster an
```

MAXIMIZE WINDOW

MAXIMIZE WINDOW {(FensterRef)}

Parameter	Typ		Beschreibung
FensterRef	WinRef	➡	Referenznummer des Fensters Ohne Angabe alle Fenster (Windows) oder vorderstes Fenster des aktuellen Prozesses (Mac OS)

Beschreibung

Der Befehl **MAXIMIZE WINDOW** erweitert das Fenster mit der in *FensterRef* übergebenen Referenznummer. Wird dieser Parameter nicht angegeben, wird ebenfalls vergrößert. Der Befehl gilt unter Windows für alle Fenster der Anwendung, auf Macintosh für das vorderste Fenster des aktuellen Prozesses.

Dieser Befehl hat dieselbe Wirkung wie ein Klick in die Vergrößerungsbox eines 4D Anwendungsfensters. Fenster, die Sie maximieren wollen, müssen eine Vergrößerungsbox haben. Bei Fenstertypen ohne Vergrößerungsbox führt der Befehl nichts aus.

Klicken Sie später auf die Vergrößerungsbox des Fensters oder rufen Sie den Befehl **MINIMIZE WINDOW** auf, wird das Fenster wieder auf seine Ausgangsgröße gesetzt.

Ist *FensterRef* bereits maximiert, führt der Befehl nichts aus.

Unter Windows

Das Fenster wird auf die aktuelle Größe des Anwendungsfensters (MDI Modus) oder des Bildschirms (SDI Modus) gesetzt. Dieses Fenster wird zum vordersten Fenster. Wird der Parameter *FensterRef* nicht übergeben, gilt der Befehl für alle Fenster der Anwendung.

Vergrößerungsbox unter Windows

Für Fenster mit Größenbeschränkungen (zum Beispiel ein Formularfenster) gilt folgendes:

- Liegen die Größenbeschränkungen innerhalb der Zielgröße, wird das Fenster auf die maximale Größe gesetzt. Es wird z.B. an die Ausmaße des übergeordneten Anwendungsfensters (MDI Modus) oder des Bildschirms (SDI Modus) angepasst; Titelleiste und Ränder werden ausgeblendet, die Icons der Titelleiste (Minimieren, Maximieren und Schließbox) werden rechts neben die Menüleiste des Hauptbildschirms gesetzt.
- Liegt jedoch mindestens ein Wert außerhalb der Zielgröße, hat z.B. das MDI Fenster die Breite 100 und ist die maximale Breite des Formularfensters 80, wird das Fenster nicht auf die maximalen Ausmaße gesetzt, sondern nur auf die maximal zugelassene Größe. Die Größe wird entweder über die Maße des MDI-Fensters oder die jeweilige Beschränkung definiert. Auf diese Weise bleibt die Konsistenz der Oberfläche erhalten.

Auf Mac OS

Das Fenster wird auf die Größe seines Inhalts gesetzt. Übergeben Sie in *FensterRef* keinen Parameter, gilt der Befehl für das vorderste Fenster des aktuellen Prozesses.

Vergrößerungsbox auf Mac OS

- Die Vergrößerung richtet sich nach dem Inhalt des Fensters; deshalb muss der Befehl in einem Kontext aufgerufen werden, der den Inhalt des Fensters definiert, z.B. in einer Formularmethode. Sonst führt der Befehl nichts aus.
- **MAXIMIZE WINDOW** setzt ein Fenster auf seine "maximale" Größe. Bei einem Formular, dessen Größe in den Formulareigenschaften festgelegt wurde, wird das Fenster auf diese Größe gesetzt. Hat das Fenster bereits die maximale Größe, hat der Befehl keine Auswirkung.

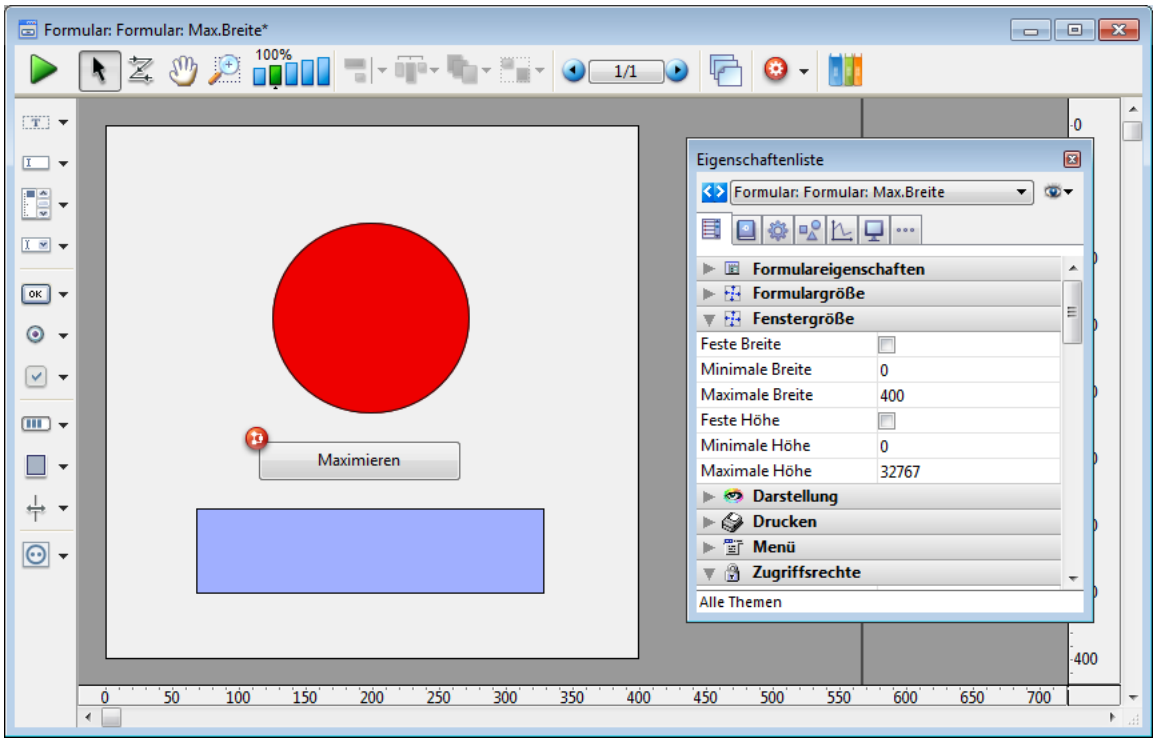
Beispiel 1

Dieses Beispiel setzt die Fenstergröße Ihres Formulars beim Öffnen auf die volle Größe des Bildschirms. Schreiben Sie dazu folgenden Code in der Formularmethode:

```
` Formularmethode

MAXIMIZE WINDOW
```

Beispiel 2

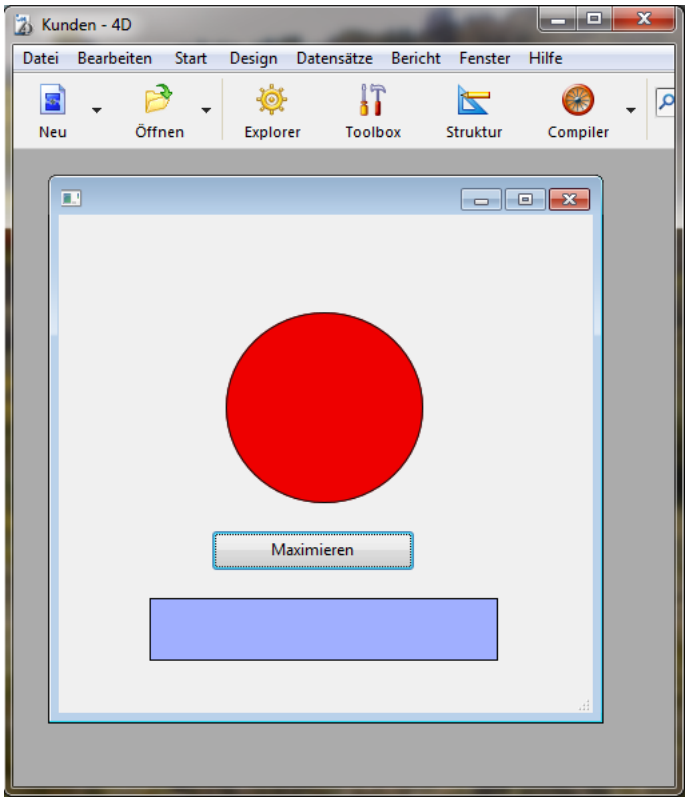
Dieses Beispiel zeigt, wie die Größeneinschränkung unter Windows (MDI Modus) gehandhabt wird. Nachfolgendes Formular hat die Größeneinschränkung: maximale Breite=400:



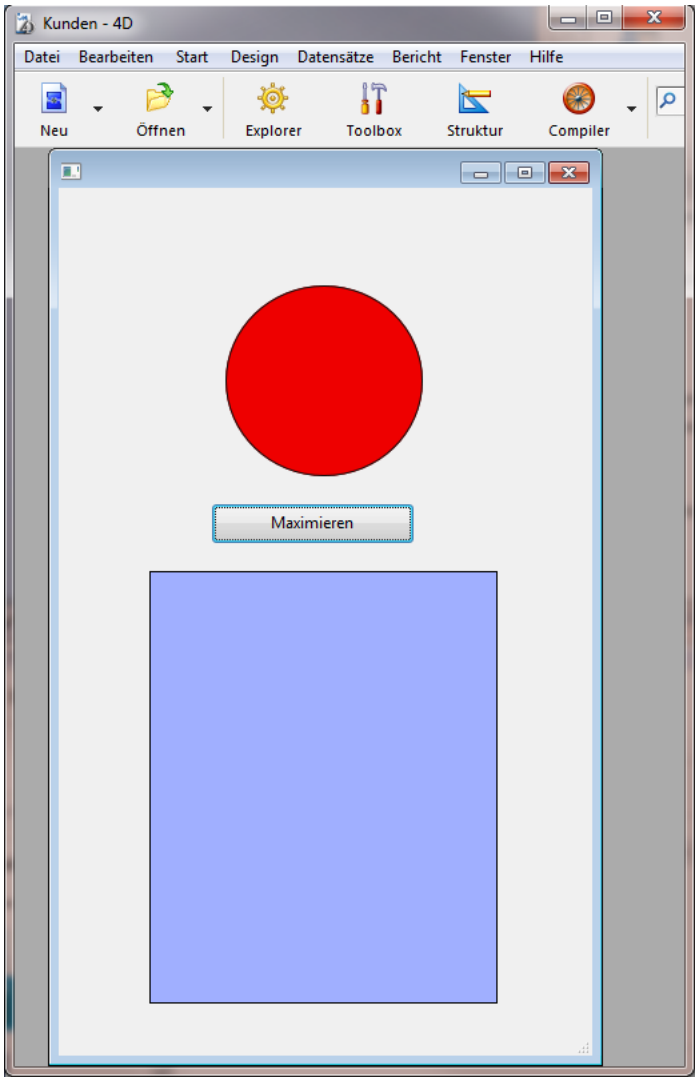
Der Code der Schaltfläche lautet:

```
MAXIMIZE WINDOW(Current form window)
```

Klickt der Benutzer im folgenden Kontext auf die Schaltfläche:



... wird das Fenster nicht auf die maximalen Ausmaße gesetzt, sondern nur in der Höhe vergrößert:


Kunden - 4D
Datei
Bearbeiten
Start
Design
Datensätze
Bericht
Fenster
Hilfe
Neu
Öffnen
Explorer
Toolbox
Struktur
Compiler
Maximieren

MINIMIZE WINDOW

MINIMIZE WINDOW {(FensterRef)}

Parameter	Typ		Beschreibung
FensterRef	WinRef	➡	Referenznummer des Fensters Ohne Angabe alle Fenster (Windows) oder vorderstes Fenster des aktuellen Prozesses (Mac OS)

Beschreibung

Der Befehl **MINIMIZE WINDOW** setzt die Größe des Fensters mit der in *FensterRef* übergebenen Nummer auf die Größe vor der Maximierung. Wird dieser Parameter nicht angegeben, gilt der Befehl unter Windows für alle Fenster der Anwendung, auf Macintosh für das vorderste Fenster des Prozesses.
Dieser Befehl hat dieselbe Wirkung wie ein Klick auf die Verkleinerungsbox im Fenster der 4D Anwendung:

Unter Windows

Das Fenster wird auf seine Ausgangsgröße gesetzt, z.B. auf die Größe vor der Maximierung. Wird der Parameter *FensterRef* nicht angegeben, werden alle Fenster der Anwendung auf ihre Ausgangsgröße gesetzt.

Verkleinerungsbox unter Windows

Auf Mac OS

Das Fenster wird auf seine Ausgangsgröße gesetzt, z.B. auf die Größe vor der Maximierung. Wird der Parameter *FensterRef* nicht angegeben, wird das vorderste Fenster des aktuellen Prozesses auf seine Ausgangsgröße gesetzt.

Verkleinerungs-/Vergrößerungsbox auf Mac OS

Der Befehl hat keine Auswirkung, wenn die Fenster, auf die der Befehl angewandt wird, nicht zuvor maximiert waren (manuell oder über den Befehl **MAXIMIZE WINDOW**) oder der Fenstertyp keine Vergrößerungsbox besitzt. Weitere Informationen dazu finden Sie unter **Fenstertypen (Kompatibilität)**.

Hinweis: Verwechseln Sie diese Funktion nicht mit der Minimierung eines Fensters auf eine Schaltfläche. Dies wird ausgeführt, wenn Sie auf folgende Box klicken:

Windows

Mac OS

Next window

Next window (FensterRef) -> Funktionsergebnis

Parameter	Typ		Beschreibung
FensterRef	WinRef	⇒	Referenznummer des Fensters oder vorderstes Fenster des aktuellen Prozesses
Funktionsergebnis	WinRef	⇐	Referenznummer des Fensters

Beschreibung

Die Funktion **Next window** gibt die Referenznummer des Fensters zurück, das hinter dem in *FensterRef* übergebenen Fenster liegt (gemäß der von vorne nach hinten laufenden Anordnung).

Open form window

Open form window ({Tabellenname ;} FormularName {; Typ {; hPos {; vPos {; *}}}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle des Formulars, ohne Angabe Standardtabelle
FormularName	String, Objekt	➡	Name (String) des Tabellen- oder Projektformulars, oder POSIX Pfad (String) zu einer .json Datei mit Beschreibung des Formulars, oder Objekt mit Beschreibung des Formulars.
Typ	Lange Ganzzahl	➡	Fenstertyp
hPos	Lange Ganzzahl	➡	Horizontale Position des Fensters
vPos	Lange Ganzzahl	➡	Vertikale Position des Fensters
*	Operator	➡	Sichere aktuelle Fensterposition und -größe
Funktionsergebnis	WinRef	↩	Referenznummer des Fensters

Beschreibung

Die Funktion **Open form window** öffnet ein neues Fenster mit der Größe und den Eigenschaften zur Größensteuerung des Formulars *FormularName*.

Hinweis: Über den Befehl **FORM GET PROPERTIES** können Sie die Haupteigenschaften eines Formulars abfragen.

Im Parameter *Formularname* können Sie folgendes übergeben:

- Name des Formulars (Projektformular oder Tabellenformular)
- Pfad (in POSIX Syntax) zu einer gültigen .json Datei mit der Beschreibung des Formulars. Siehe **Dateipfade für Formulare**;
- Objekt mit der Beschreibung des Formulars. Weitere Informationen dazu finden Sie unter **Dynamische Formulare**

FormularName wird nicht im Fenster angezeigt. Wollen Sie es anzeigen, müssen Sie einen Befehl aufrufen, der ein Formular lädt, z.B. **ADD RECORD**.

Mit dem optionalen Parameter *Typ* legen Sie einen Typ für das Fenster fest. Sie müssen eine der vordefinierten Konstanten unter dem Thema **Open form window** übergeben:

Konstante	Typ	Wert
Controller form window	Lange Ganzzahl	133056
Form has full screen mode Mac	Lange Ganzzahl	65536
Form has no menu bar	Lange Ganzzahl	2048
Modal form dialog box	Lange Ganzzahl	1
Movable form dialog box	Lange Ganzzahl	5
Palette form window	Lange Ganzzahl	1984
Plain form window	Lange Ganzzahl	8
Pop up form window	Lange Ganzzahl	32
Sheet form window	Lange Ganzzahl	33
Toolbar form window	Lange Ganzzahl	35

Weitere Informationen zu den Fenstertypen finden Sie im Abschnitt **Fenstertypen**.

Hinweis: Die Konstanten Form has full screen mode Mac und Form has no menu bar müssen zu einer der anderen Konstanten hinzugefügt werden.

Standardmäßig, d.h. der Parameter *Typ* ist nicht übergeben, wird ein Fenster vom Typ Plain form window übergeben.

Schließbox
Fenster vom Typ Movable form dialog box, [#cst id="5185"/] und Palette form window haben eine Schließbox. Dieser ist keine Methode zugewiesen. Klicken in diese Schließbox annulliert und schließt das Fenster, außer das Formularereignis [#cst id="845790"/] wurde für das Formular aktiviert. In diesem Fall wird der Code für dieses Ereignis ausgeführt.

Größensteuerung
Sind die Eigenschaften für die Fenstergröße von *FormularName* nicht auf "fest" gesetzt, kann der Benutzer das Fenster anpassen. Je nach Fenstertyp kann auch eine Zoombox vorhanden sein. Ist in den Formulareigenschaften die Option **Feste Breite** bzw. **Feste Höhe** markiert, lässt sich die Fenstergröße nicht verändern.

Hinweis: Einige Attribute des angelegten Fensters, wie Schließbox, Vergrößerungskästchen, etc. richten sich nach den Spezifikationen des Betriebssystems für den gewählten *Typ*. Deshalb kann das Ergebnis je nach verwendeter Plattform unterschiedlich sein.

Mit dem optionalen Parameter *hPos* definieren Sie die horizontale Position des Fensters. Sie können eine eigene Position in Pixel angeben oder eine der vordefinierten Konstanten unter dem Thema **Open form window** verwenden:

Konstante	Typ	Wert
Horizontally centered	Lange Ganzzahl	65536
On the left	Lange Ganzzahl	131072
On the right	Lange Ganzzahl	196608

Mit dem optionalen Parameter *vPos* definieren Sie die vertikale Position des Fensters. Sie können eine eigene Position in Pixel angeben oder eine der vordefinierten Konstanten unter dem Thema **Open form window** verwenden:

Konstante	Typ	Wert
At the bottom	Lange Ganzzahl	393216
At the top	Lange Ganzzahl	327680
Vertically centered	Lange Ganzzahl	262144

Ausgangspunkt für die Koordinaten ist die linke obere Ecke vom Innenbereich des Anwendungsfensters (Windows MDI Modus) oder vom Hauptbildschirm (Mac OS und Windows SDI Modus). Sie berücksichtigen das Vorhandensein der Werkzeugleiste und der Menüleiste.

Übergeben Sie den optionalen Parameter *, werden die aktuelle Position und Größe des Fensters beim Schließen gespeichert. Beim Wiederöffnen erscheint das Fenster dann mit diesen Festlegungen. In diesem Fall werden die Parameter *vPos* und *hPos* nur beim ersten Öffnen des Fensters verwendet.

Hinweis: Ist der Parameter * übergeben und Sie wollen ein Fenster erneut mit den ursprünglichen Koordinaten in *vPos* und *hPos* öffnen, öffnen Sie das Fensters mit gedrückter **Shift**-Taste.

Beispiel 1

Folgender Code öffnet ein Standardfenster mit Schließkästchen und passt es automatisch an, so dass es dieselbe Größe wie das Eingabeformular hat. Da die Größe des Formularfensters nicht auf "fest" gesetzt ist, hat das Fenster auch ein Kästchen zum Vergrößern und Zoomen:

```
$winRef :=Open form window([Table1];"Eingabe")
```

Beispiel 2

Folgender Code öffnet ein Palettenfenster im oberen linken Bereich des Bildschirms, basierend auf dem Projektformular mit Namen "Tools". Beim Wiederöffnen verwendet dieses Fenster immer die zuletzt angegebene Position:

```
$winRef :=Open form window("Tools";Palette window;On the left;At the top;*)
```

Beispiel 3

Dieser Code muss auf macOS beim Anzeigen eines Dokumentfensters aufgerufen werden, um ein Sheet Fenster zu öffnen:

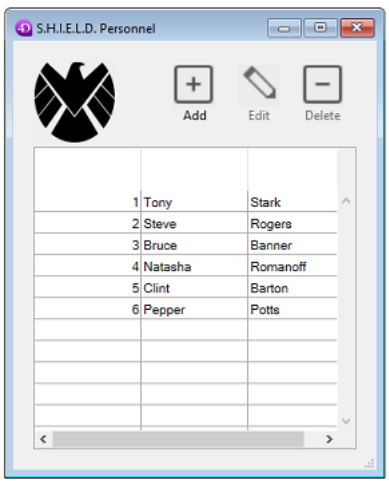
```
$myWin:=Open form window("sheet form";Sheet form window)
 // Momentan wird das Fenster erstellt, bleibt aber unsichtbar
DIALOG([aTable];"dialForm")
 //Das Ereignis On Load wird generiert, dann erscheint das "Sheet" Fenster;
 //es rollt unterhalb der Titelleise auf
```

Beispiel 4

Dieses Beispiel verwendet den Pfad eines .json Formulars zum Anzeigen der Datensätze in einer Liste der Angestellten:

```
Open form window("/RESOURCES/OutputPersonnel.json";Plain form window)
ALL RECORDS([Personnel])
DIALOG("/RESOURCES/OutputPersonnel.json";*)
```

Das ergibt folgendes Formular:



Open window

Open window (Links ; Oben ; Rechts ; Unten {; Typ {; Titel {; Schließen}}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Links	Lange Ganzzahl	➡	Linke Position des Fensters
Oben	Lange Ganzzahl	➡	Obere Position des Fensters
Rechts	Lange Ganzzahl	➡	Rechte Position des Fensters oder -1 für Standardgröße des Formulars
Unten	Lange Ganzzahl	➡	Untere Position des Fensters oder -1 für Standardgröße des Formulars
Typ	Lange Ganzzahl	➡	Typ des Fensters
Titel	String	➡	Titel des Fensters oder "" für Standardtitel des Formulars
Schließen	String	➡	Aufzurufende Methode bei Doppelklick auf die Kontrollmenü-Box oder Klick auf Schließbox
Funktionsergebnis	WinRef	↩	Referenznummer des Fensters

Beschreibung

Die Funktion **Open window** öffnet ein neues Fenster mit den in den ersten vier Parametern angegebenen Werten.

- *Links* ist der Abstand in Pixel vom linken Rand des Anwendungsfensters zum linken inneren Rand des Fensters.
- *Oben* ist der Abstand in Pixel vom oberen Rand des Anwendungsfensters zum oberen inneren Rand des Fensters.
- *Rechts* ist der Abstand in Pixel vom linken Rand des Anwendungsfensters zum rechten inneren Rand des Fensters.
- *Unten* ist der Abstand in Pixel vom oberen Rand des Anwendungsfensters zum unteren inneren Rand des Fensters.

Hinweis zur Kompatibilität: Open window bietet eine Reihe von Optionen, die über die Jahre weiterentwickelt wurden und nur zur Wahrung der Kompatibiltät beibehalten werden. Schreiben Sie neuen Code zum Verwalten von Fenstern, empfehlen wir, die Funktion **Open form window** zu verwenden, da sie sich für die aktuellen Oberflächen besser eignet.

Übergeben Sie für *Rechts* und *Unten* den Wert -1, legt 4D unter folgenden Bedingungen automatisch die Größe des Fensters fest:

- Sie haben im Designmodus ein Formular erstellt und seine Ausmaße im Fenster Formulareigenschaften festgelegt.
- Sie haben vor dem Aufruf von **Open window** das Formular mit dem Befehl **FORM SET INPUT** ausgewählt und den optionalen Parameter * übergeben.

Wichtig: Das Fenster wird nur automatisch angepasst, wenn Sie zuvor für das anzuzeigende Formular den Befehl **FORM SET INPUT** aufgerufen und den optionalen Parameter * übergeben haben.

- Der Parameter *Typ* ist optional. Damit bestimmen Sie den Fenstertyp. Die verschiedenen Typen werden im Abschnitt **Fenstertypen (Kompatibilität)** beschrieben. Geben Sie diesen Wert nicht an, wird standardmäßig der Typ 1 verwendet. Übergeben Sie einen negativen Wert, verhält sich das Fenster wie ein Palettenfenster. Es liegt immer auf dem Bildschirm ganz vorne, außer es erscheint ein Dialogfenster.
- Der Parameter *Titel* ist der optionale Titel für das Fenster. Bei Fenstertypen ohne Titelleiste hat er keine Wirkung.

Übergeben Sie in *Titel* einen leeren String (""), verwendet 4D für das anzuzeigende Formular den Fenstertitel, der im Designmodus im Fenster Formulareigenschaften festgelegt wurde.

Wichtig: Der Standardtitel des Formulars wird nur eingesetzt, wenn Sie zuvor für das anzuzeigende Formular den Befehl **FORM SET INPUT** aufgerufen und den optionalen Parameter * übergeben haben.

- Der Parameter *Schließen* ist ebenfalls optional. Er erwartet einen Methodennamen. Wird der Parameter angegeben, erscheint unter Windows eine Kontrollmenü-Box, auf Macintosh eine Schließbox. Mit einem Doppelklick auf die Kontrollmenübox bzw. einem Klick auf die Schließbox ruft der Anwender die in *Schließen* angegebene Methode auf. Sie muss eine Bestätigen- oder Abbrechen-Anweisung enthalten.

Hinweis: Sie können auch das Schließen des Fensters über eine Formularmethode des im Fenster angezeigten Formulars verwalten, wenn ein Ereignis *On Close Box* eintritt. Weitere Informationen dazu finden Sie unter der Funktion **Form event**.

Öffnen Sie während eines Prozesses mehrere Fenster, ist das zuletzt geöffnete das aktive Fenster. Sie können nur die Informationen in diesem Fenster verändern. Die Daten in den anderen Fenstern werden jedoch weiterhin angezeigt. Gibt der Anwender Daten ein, kommt das für diesen Prozess aktive Fenster ganz nach vorne.

Formulare werden innerhalb eines offenen Fensters angezeigt. Text vom Befehl **MESSAGE** erscheint ebenso im Fenster.

Open window gibt eine Referenz vom Typ *WinRef* zurück, welche die Befehle zur Fensterverwaltung nutzen können. Weitere Informationen dazu finden Sie im Abschnitt **WinRef**.

Beispiel 1

Folgende Projektmethode öffnet unter Windows innerhalb des Hauptfensters, auf Macintosh im Hauptbildschirm ein zentriertes Fenster. Beachten Sie, dass es zwei, drei oder vier Parameter haben kann:

```
` Projektmethode OPEN CENTERED WINDOW
` $1 – Fensterbreite
` $2 – Fensterhöhe
` $3 – Fenstertyp (optional)
` $4 – Fenstertitel (optional)
$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
```

Diese Projektmethode können Sie beispielsweise so einsetzen:

```
OPEN CENTERED WINDOW(400;250;Movable dialog box;"Update Archives")
DIALOG([Utility Table];"UPDATE OPTIONS")
CLOSE WINDOW
If(OK=1)
   ` ...
End if
```

Beispiel 2

Folgendes Beispiel öffnet ein Palettenfenster, das eine Methode für die Kontrollmenü-Box (Windows) bzw. Schließbox (Macintosh) enthält. Dieses Fenster öffnet sich in der oberen rechten Ecke des Anwendungsfensters.

```
$myWindow=Open window(Screen width-149;33;Screen width-4;178;-Palette window;"";"Schließe Farbpalette")
DIALOG([Dialogs];"Farbpalette")
```

Die Methode ***CloseColorPalette*** ruft den Befehl **CANCEL** auf:

```
CANCEL
```

Beispiel 3

Folgendes Beispiel öffnet ein Fenster, das die Größe und Titel über die Eigenschaften des im Fenster angezeigten Formulars erhält:

```
FORM SET INPUT([Customers];"Füge Datensätze hinzu";*)
$myWindow=Open window(10;80;-1;-1;Plain window;"")
Repeat
   ADD RECORD([Customers])
Until(OK=0)
```

Bitte beachten: Open window verwendet nur dann automatisch die Einstellungen des Formulars, wenn Sie zuvor den Befehl **FORM SET INPUT** mit dem optionalen Parameter * aufgerufen haben und in der Designumgebung entsprechende Formulareigenschaften festgelegt haben.

Beispiel 4

Folgendes Beispiel zeigt unter Mac OS X, wie sich die Anzeige eines Sheet Fensters verzögern lässt:

```
$myWindow:=Open window(10;10;400;400;Sheet window)
  `Das Fenster wird erstellt, bleibt jedoch ausgeblendet.
DIALOG([Table];"dialForm")
  `Das Ereignis On Load wird erstellt, dann rollt das Sheet Fenster von der Titelleiste her auf.
```

REDRAW WINDOW

REDRAW WINDOW {(FensterRef)}

Parameter	Typ		Beschreibung
FensterRef	WinRef	➡	Referenznr des Fensters Ohne Angabe vorderstes Fenster des aktuellen Prozesses

Beschreibung

Der Befehl **REDRAW WINDOW** aktualisiert die Darstellung des Fensters mit der in *FensterRef* übergebenen Referenznummer.

Der Parameter *FensterRef* ist optional. Geben Sie ihn nicht an, gilt **REDRAW WINDOW** für das vorderste Fenster des aktuellen Prozesses.

Hinweis: 4D aktualisiert die Darstellung des Fensters immer, wenn Sie ein Fenster bewegen, in der Größe verändern bzw. nach vorne bringen. Dasselbe gilt, wenn Sie das Formular und/oder die angezeigten Werte ändern. Von daher verwenden Sie diesen Befehl nur selten.

RESIZE FORM WINDOW

RESIZE FORM WINDOW (Breite ; Höhe)

Parameter	Typ		Beschreibung
Breite	Lange Ganzzahl	⇒	Pixel, die von der Breite des aktuellen Formularfensters hinzugefügt oder abgezogen werden sollen
Höhe	Lange Ganzzahl	⇒	Pixel, die von der Höhe des aktuellen Formularfensters hinzugefügt oder abgezogen werden sollen

Beschreibung

Der Befehl **RESIZE FORM WINDOW** ermöglicht, die Größe des aktuellen Formularfensters zu verändern.

In den Parametern *Breite* und *Höhe* geben Sie die Anzahl Pixel an, um die das aktuelle Formularfenster vergrößert werden soll. Übergeben Sie 0 (Null), wenn die Größe gleich bleiben soll. Verwenden Sie negative Werte, um das Formularfenster zu verkleinern.

Dieser Befehl liefert dasselbe Ergebnis wie die manuelle Anpassung über den Anpassungsdialog – sofern der Fenstertyp das zulässt. Darüberhinaus berücksichtigt er auch die Eigenschaften für Objekte und die Größenbeschränkungen, die in den Formulareigenschaften definiert wurden. So hat der Befehl keine Auswirkung, wenn er das Fenster größer machen will als in den Formulareigenschaften zugelassen ist.
Beachten Sie, dass dieser Befehl anders arbeitet wie **SET WINDOW RECT**. Dieser berücksichtigt beim Anpassen der Fenstergröße weder die Formulareigenschaften noch den Inhalt.

RESIZE FORM WINDOW verändert auch nicht unbedingt die Formulargrößen. Um ein Formular per Programmierung anzupassen, verwenden Sie den Befehl **FORM SET SIZE**.

Beispiel

Wir gehen von folgendem Fenster aus (Felder und Rahmen haben die Eigenschaft “Wachse horizontal”):

Nach Ausführen der Code-Zeile:

```
RESIZE FORM WINDOW(25;0)
```

... sieht das Fenster folgendermaßen aus:

SET WINDOW RECT

SET WINDOW RECT (Links ; Oben ; Rechts ; Unten {; FensterRef}{; *})

Parameter	Typ		Beschreibung
Links	Lange Ganzzahl	➡	Globale linke Koordinate vom Innenbereich des Fensters
Oben	Lange Ganzzahl	➡	Globale obere Koordinate vom Innenbereich des Fensters
Rechts	Lange Ganzzahl	➡	Globale rechte Koordinate vom Innenbereich des Fensters
Unten	Lange Ganzzahl	➡	Globale untere Koordinate vom Innenbereich des Fensters
FensterRef	WinRef	➡	Referenznummer des Fensters Ohne Angabe vorderstes Fenster des aktuellen Prozesses
*	Operator	➡	Ohne * (Standard): Fenster nach vorne setzen Mit *: Ebene des Fensters nicht verändern

Beschreibung

Der Befehl **SET WINDOW RECT** ändert die globalen Koordinaten des Fensters mit der in *FensterRef* übergebenen Referenznummer. Gibt es das Fenster nicht, hat der Befehl keine Auswirkung.

Der Parameter *FensterRef* ist optional. Geben Sie diesen Parameter nicht an, gilt **SET WINDOW RECT** für das vorderste Fenster des aktuellen Prozesses.

Dieser Befehl kann das Fenster, je nach den neuen Koordinaten in der Größe anpassen und verschieben.

Ausgangspunkt für die Koordinaten ist die linke obere Ecke vom Innenbereich des Anwendungsfensters (Windows MDI Modus) oder vom Hauptbildschirm (Mac OS und Windows SDI Modus). Die Koordinaten geben den Innenbereich des Fensters an, d.h. ohne Titelleisten und Ränder.

Warnung: Verwenden Sie diesen Befehl mit Vorsicht, denn damit können Sie das Fenster auch über die Grenzen des Hauptfensters bzw. -bildschirms hinausbewegen. Um das zu verhindern, prüfen Sie die neuen Koordinaten zusätzlich mit den Funktionen **Screen width** und **Screen height**.

Standardmäßig, d.h. ohne den Parameter *, setzt der Befehl das im Parameter *Fenster* angegebene Fenster in den Vordergrund. Das können Sie ändern, wenn Sie * als letzten Parameter übergeben. Dann verändert der Befehl nicht die ursprüngliche Ebene des Fensters ("z" Koordinate).

Dieser Befehl gilt nicht für Formularobjekte. Enthält das Fenster ein Formular, bewegt oder verändert der Befehl nicht die Formularobjekte, unabhängig von ihren Eigenschaften. Nur das Fenster wird verändert. Um ein Formularfenster unter Berücksichtigung seiner Anpassungseigenschaften und Objekte zu verändern, müssen Sie den Befehl **RESIZE FORM WINDOW** verwenden.

Beispiel 1

Siehe Beispiel zum Befehl **WINDOW LIST**.

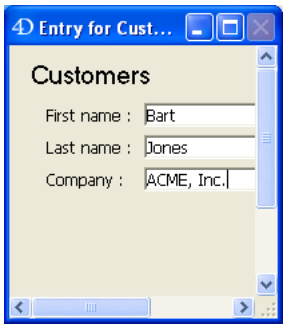
Beispiel 2

Nehmen wir folgendes Fenster:

Nach Ausführung der Code-Zeile:

```
SET WINDOW RECT(100;100;300;300)
```

sieht das Fenster folgendermaßen aus:


Entry for Cust...
Customers
First name :
Bart
Last name :
Jones
Company :
ACME, Inc.

SET WINDOW TITLE

SET WINDOW TITLE (Titel {; FensterRef})

Parameter	Typ		Beschreibung
Titel	String	➡	Titel des Fensters
FensterRef	WinRef	➡	Referenznummer des Fensters Ohne Angabe vorderstes Fenster des aktuellen Prozesses

Beschreibung

Der Befehl **SET WINDOW TITLE** ändert den Titel des Fensters mit der Referenznr. *FensterRef* für den in *Titel* übergebenen Text (max. Länge 80 Zeichen).

Gibt es das Fenster nicht, wird **SET WINDOW TITLE** nicht ausgeführt.

Der Parameter *FensterRef* ist optional. Geben Sie ihn nicht an, ändert **SET WINDOW TITLE** den Titel des vordersten Fensters für den aktuellen Prozess.

Hinweis: In der Designumgebung ändert 4D die Fenstertitel automatisch — z.B., "Tabelleneingabe" bei der Dateneingabe. Ändern Sie einen Fenstertitel, überschreibt ihn 4D wahrscheinlich. In der Anwendungsumgebung ändert 4D dagegen nicht die Titel der Fenster.

Beispiel

Bei der Dateneingabe in ein Formular klicken Sie auf eine Schaltfläche, die eine länger dauernde Operation ausführt, zum Beispiel per Programmierung verknüpfte Datensätze aus einem Unterformular durchläuft. Über den Titel des aktuellen Fensters bleiben Sie über den Verlauf der Operation informiert:

```
  `Objektmethode für Schaltfläche bAnalysis
Case of
   :(Form event=On Clicked)
  ` Sichere aktuellen Fenstertitel in lokaler Variable
      $vsCurTitle:=Get window title
  ` Starte die länger dauernde Operation
      FIRST RECORD([Invoice Line Items])
      For($vlRecord;1;Records in selection([Invoice Line Items]))
         FÜHRE ETWAS AUS
  ` Zeige Information über Verlauf
         SET WINDOW TITLE("Processing Line Item #"+String($vlRecord))
      End for
  ` Speichere Original-Fenstertitel
      SET WINDOW TITLE($vsCurTitle)
End case
```

SHOW TOOL BAR

SHOW TOOL BAR
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **SHOW TOOL BAR** steuert die Anzeige eigener Toolbars, die über die Funktion **Open form window** für den aktuellen Prozess erstellt wurden.

Wurde über **Open form window** mit der Option Toolbar form window ein Fenster mit Toolbar angelegt, zeigt der Befehl diesen Fenstertyp an. Ist das Toolbar Fenster bereits sichtbar oder wurde kein Fenster dieses Typs angelegt, führt die Funktion nichts aus.

Beispiel

Siehe Beispiel unter dem Befehl **HIDE TOOL BAR**.

SHOW WINDOW

SHOW WINDOW {(FensterRef)}

Parameter	Typ		Beschreibung
FensterRef	WinRef	⇒	Referenznummer des Fensters Ohne Angabe vorderstes Fenster des aktuellen Prozesses

Beschreibung

Der Befehl **SHOW WINDOW** zeigt das Fenster mit der in *FensterRef* übergebenen Nummer an. Wird dieser Parameter nicht angegeben, erscheint das vorderste Fenster des aktuellen Prozesses. Um **SHOW WINDOW** zu benutzen, muss das Fenster zuvor mit dem Befehl **HIDE WINDOW** ausgeblendet worden sein. Wird das Fenster bereits angezeigt, hat der Befehl keine Auswirkung.

Beispiel

Siehe Beispiel zum Befehl **HIDE WINDOW**.

Tool bar height

Tool bar height -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl	➲	Höhe der Werkzeugleiste in Pixel oder 0 bei ausgeblendeter Leiste

Beschreibung

Die Funktion **Tool bar height** gibt die Höhe der Werkzeugleiste in Pixel zurück.

Das kann je nach Kontext entweder die Toolbar des 4D Designmodus sein oder eine eigene Toolbar, die mit dem Befehl **Open form window** angelegt wurde. Gibt es eine eigene Toolbar, wird die Toolbar des Designmodus automatisch ausgeblendet.

Wird keine Toolbar angezeigt, gibt sie 0 (Null) zurück.

Window kind

Window kind {(FensterRef)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
FensterRef	WinRef	➡	Referenznummer des Fensters Ohne Angabe vorderstes Fenster des aktuellen Prozesses
Funktionsergebnis	Lange Ganzzahl	➲	Fenstertyp

Beschreibung

Die Funktion **Window kind** gibt den 4D Typ des Fensters mit der in *FensterRef* übergebenen Referenznummer zurück. Gibt es das Fenster nicht, gibt **Window kind** den Wert Null (*0*) zurück.

Window kind gibt eine der folgenden vordefinierten Konstanten unter dem Thema **Fenster** zurück:

Konstante	Typ	Wert
External window	Lange Ganzzahl	5
Floating window	Lange Ganzzahl	14
Modal dialog	Lange Ganzzahl	9
Regular window	Lange Ganzzahl	8

Der Parameter *FensterRef* ist optional. Geben Sie den Parameter nicht an, gibt **Window kind** den Typ des vordersten Fensters für den aktuellen Prozess zurück.

Beispiel

Siehe Beispiel zum Befehl **WINDOW LIST**.

WINDOW LIST

WINDOW LIST (Fenster {; *})

Parameter	Typ		Beschreibung
Fenster	Array	⇐	Array der Fenster Referenznummern
*	Operator	⇒	Mit *: Palettenfenster berücksichtigen Ohne *: Palettenfenster ignorieren

Beschreibung

Der Befehl **WINDOW LIST** füllt das Array *Fenster* mit den Referenznummern der Fenster, die derzeit in allen laufenden Prozessen (Kernel oder Benutzer) offen sind. Nur "sichtbare" Fenster werden zurückgegeben, d.h. ausgeblendete Fenster werden nicht berücksichtigt.

Übergeben Sie keinen optionalen Parameter *, werden Palettenfenster ignoriert.

Beispiel

Folgende Projektmethode verteilt alle aktuellen offenen Fenster, mit Ausnahme von Palettenfenstern und Dialogboxen:

```
  ` Projektmethode TILE WINDOWS
WINDOW LIST($alWnd)
$vlLeft:=10
$vlTop:=80  ` Genügend Platz für die Werkzeugleiste lassen
For($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
```

Hinweis: Diese Methode lässt sich verbessern, wenn Sie Tests über die Größe des Hauptfensters (unter Windows) bzw. die Größe und Anordnung der Bildschirmoberfläche (auf Macintosh) hinzufügen.

Window process

Window process {(FensterRef)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
FensterRef	WinRef	⇒	Referenznummer des Fensters
Funktionsergebnis	Lange Ganzzahl	⇐	Referenznummer des Prozesses

Beschreibung

Die Funktion **Window process** gibt die Prozessnummer zurück, die das Fenster mit der Referenznummer *FensterRef* öffnet. Gibt es das Fenster nicht, wird der Wert Null (0) zurückgegeben.

Der Parameter *FensterRef* ist optional. Geben Sie diesen Parameter nicht an, gibt **Window process** den Prozess des aktuellen vordersten Fensters zurück.

_o_Open external window

_o_Open external window (Links ; Oben ; Rechts ; Unten ; Typ ; Titel ; PlugInBereich) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Links	Lange Ganzzahl	➡	Linke Position des Fensters
Oben	Lange Ganzzahl	➡	Obere Position des Fensters
Rechts	Lange Ganzzahl	➡	Rechte Position des Fensters
Unten	Lange Ganzzahl	➡	Untere Position des Fensters
Typ	Lange Ganzzahl	➡	Typ des Fensters
Titel	String	➡	Titel des Fensters
PlugInBereich	String	➡	Funktion für externen Bereich
Funktionsergebnis	WinRef	↩	Referenznummer für Fenster und externen Bereich

Beschreibung

Die Funktion **_o_Open external window** ist in 4D ab Version 16 überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Beachten Sie, dass sie in 64-bit Versionen von 4D nicht unterstützt wird.

Fenstertypen (Kompatibilität)

Hinweis zur Kompatibilität

Einige der hier beschriebenen Fenstertypen gelten für alte 4D Versionen und alte Betriebssysteme. Dieses Kapitel sowie die Funktion **Open window** werden nur zur Wahrung der Kompatibilität beibehalten. Wenn Sie neuen Code zum Verwalten von Fenstern schreiben, raten wir dringend, die Funktion **Open form window** zu verwenden, die besser zu den aktuellen Oberflächen passt.

Einführung

Mit folgenden vordefinierten Konstanten legen Sie den Typ des Fensters fest, das Sie mit **Open Window** öffnen möchten:

Konstante	Typ	Wert	Kommentar
Plain no zoom box window	Lange Ganzzahl	0	
Modal dialog box	Lange Ganzzahl	1	
Plain dialog box	Lange Ganzzahl	2	Kann Palettenfenster sein
Alternate dialog box	Lange Ganzzahl	3	Kann Palettenfenster sein
Plain fixed size window	Lange Ganzzahl	4	
Movable dialog box	Lange Ganzzahl	5	Kann Palettenfenster sein
Plain window	Lange Ganzzahl	8	
Round corner window	Lange Ganzzahl	16	
Pop up window	Lange Ganzzahl	32	
Sheet window	Lange Ganzzahl	33	
Resizable sheet window	Lange Ganzzahl	34	
Palette window	Lange Ganzzahl	1984	Kann Palettenfenster sein

Palettenfenster

Übergeben Sie eine dieser Konstanten, öffnen Sie ein normales Fenster. Um ein Palettenfenster zu öffnen, übergeben Sie in **Open window** einen negativen Wert. Das ist ein Fenster, das immer vorderstes Fenster bleibt, auch wenn der Benutzer auf ein anderes Fenster klickt. Dieser Fenstertyp eignet sich zur dauerhaften Anzeige von Informationen oder Werkzeugleisten.

Modales Fenster

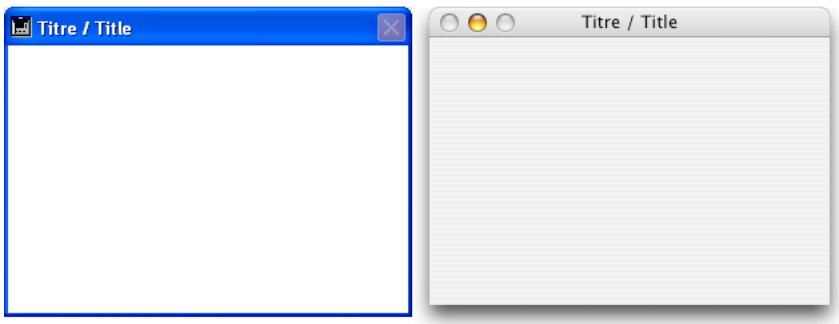
Ein modales Fenster beschränkt die Aktionen des Benutzers auf dieses Fenster. Solange es angezeigt wird, sind die Menübefehle und andere Anwendungsfenster nicht zugänglich. Der Benutzer muss erst das modale Fenster schließen: Er kann es bestätigen, annullieren oder eine angebotene Option wählen. Dialogfenster mit Warnungen oder Meldungen sind ein typisches Beispiel für modale Fenster.
In 4D sind die Fenster vom Typ 1 und 5 modale Fenster.

Hinweis: Ein modales Fenster bleibt immer im Vordergrund. Folglich erscheint ein nicht-modales Fenster, das vom modalen Fenster aufgerufen wird, im Hintergrund, selbst wenn es nach dem modalen Fenster aufgerufen wird. Deshalb sollten Sie eine derartige Operation vermeiden.
Ruft dagegen ein modales Fenster ein anderes modales Fenster auf, erscheint dieses Fenster im Vordergrund.

Beschreibung

Im folgenden sehen Sie die verschiedenen Fenstertypen. Die Darstellung unter Windows steht links, die Darstellung auf Macintosh rechts.

Plain fixed size window (4)

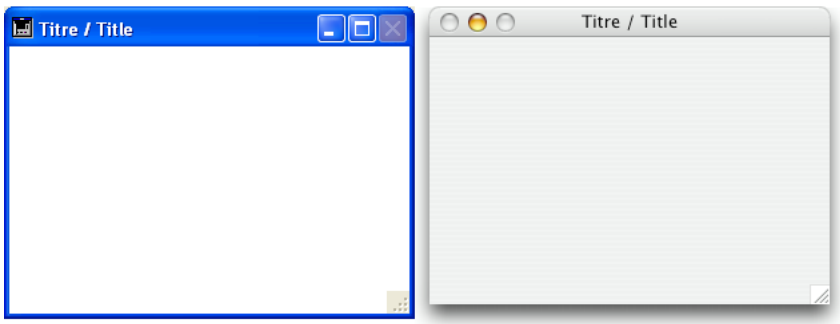


- Kann Titel haben: Ja
- Kann Schließbox o.ä. haben: Ja
- Kann in der Größe verändert werden: Ja
- Kann auf max. bzw. min. Größe oder auf Bildschirmgröße gestellt werden: Nein
- Eignet sich für Rollbalken: Ja/Nein
- Verwendung: Dateneingabe mit **ADD RECORD** o.ä.

Modal dialog box (1)

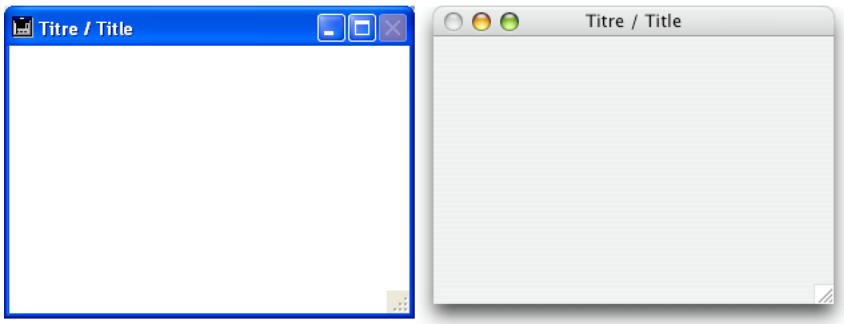
- Kann Titel haben: Nein
- Kann Schließbox o.ä. haben: Nein
- Kann in der Größe verändert werden: Nein
- Kann auf max. bzw. min. Größe oder auf Bildschirmgröße gestellt werden: Nein
- Eignet sich für Rollbalken: Nein
- Verwendung: **DIALOG**, **ADD RECORD** o.ä.
- Fenster dieses Typs sind modal.

Plain no zoom box window (0)



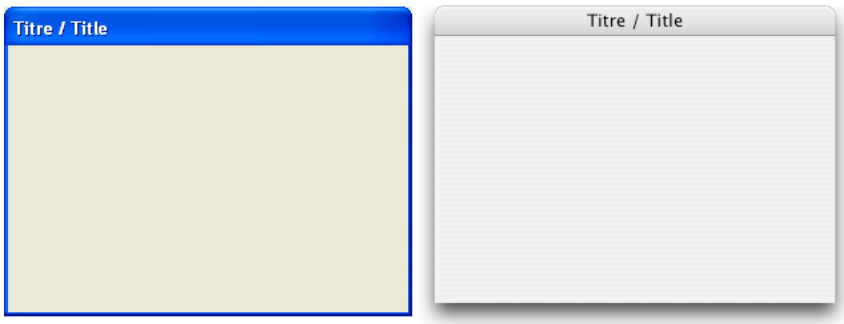
- Kann Titel haben: Ja
- Kann Schließbox o.ä. haben: Ja
- Kann in der Größe verändert werden: Ja
- Kann auf max. bzw. min. Größe oder auf Bildschirmgröße gestellt werden: Nicht auf Macintosh
- Eignet sich für Rollbalken: Ja
- Verwendung: Dateneingabe mit Rollbalken, **DISPLAY SELECTION**, **MODIFY SELECTION**, etc.

Plain window (8)



- Kann Titel haben: Ja
- Kann Schließbox o.ä. haben: Ja
- Kann in der Größe verändert werden: Ja
- Kann auf max. bzw. min. Größe oder auf Bildschirmgröße gestellt werden: Ja
- Eignet sich für Rollbalken: Ja
- Verwendung: Dateneingabe mit Rollbalken, **DISPLAY SELECTION**, **MODIFY SELECTION**, etc.

Movable dialog box (5)



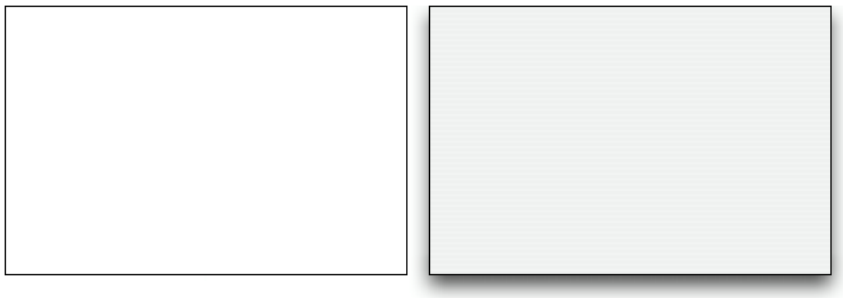
- Kann einen Titel haben: Ja
- Kann eine Schließbox o.ä. haben: Nein
- Kann in der Größe verändert werden: Nein

- Kann auf max. bzw. min. Größe oder auf Bildschirmgröße gestellt werden: Nein
- Eignet sich für Rollbalken: Nein
- Verwendung: **DIALOG**, **ADD RECORD** o.ä.
- Fenster dieses Typs sind modal, können jedoch bewegt und als "floating window" verwendet werden.

Alternate dialog box (3)

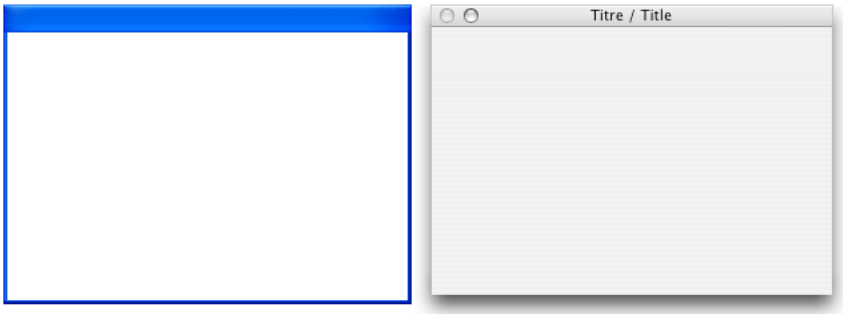
- Kann Titel haben: Nein
- Kann Schließbox o.ä. haben: Nein
- Kann in der Größe verändert werden: Nein
- Kann auf max. bzw. min. Größe oder auf Bildschirmgröße gestellt werden: Nein
- Eignet sich für Rollbalken: Nein
- Verwendung: **DIALOG**, **ADD RECORD** o.ä.
- Fenster dieses Typs sind modal, außer bei Verwendung als "floating window"

Plain dialog box (2)



- Kann Titel haben: Nein
- Kann Schließbox o.ä. haben: Nein
- Kann in der Größe verändert werden: Nein
- Kann auf max. bzw. min. Größe oder Bildschirmgröße gestellt werden: Nein
- Eignet sich für Rollbalken: Nein
- Verwendung: **DIALOG**, **ADD RECORD** o.ä., Startbildschirme
- Fenster dieses Typs sind modal, außer bei Verwendung als "floating window"

Palette window (1984)



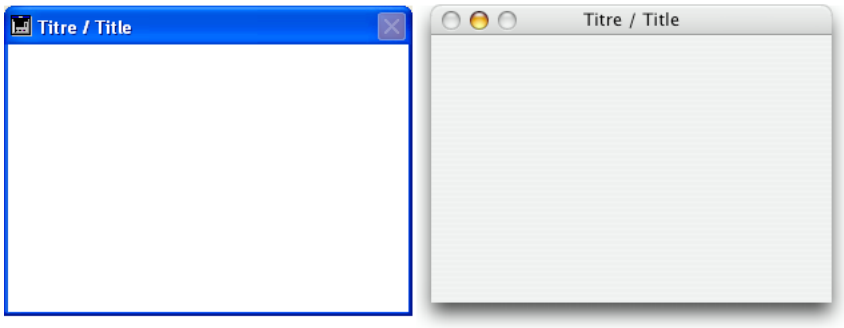
Bei diesem Fenstertyp können Sie **Palettenfenster** mit oder ohne anpassbare Größe einrichten. Es werden nur folgende Optionen unterstützt:

Option	Wert unter Windows	Wert auf macOS
Nicht anpassbar	-(Palette window+2)	-Palette window
Anpassbar	-(Palette window+6)	-(Palette window+6)

- Kann Titel haben: Ja, wenn angegeben
- Kann in der Größe verändert werden: Ja, wenn der entsprechende Wert angegeben ist
- Verwendung: Palettenfenster mit **DIALOG** oder **DISPLAY SELECTION** (Keine Dateneingabe)

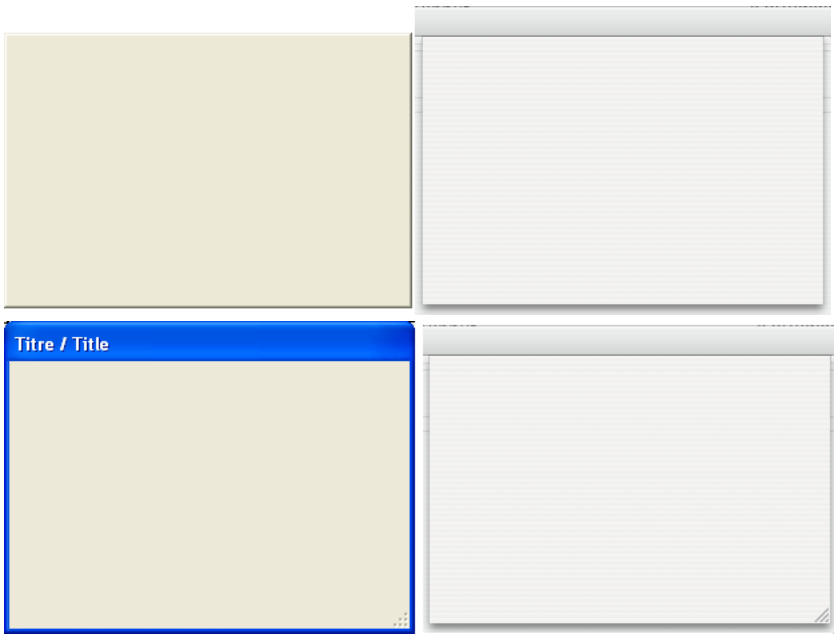
HInweis: Bei diesem Fenstertyp muss die Angabe (Konstante+Option) immer als negativer Wert übergeben werden. Hierzu ein Beispiel: Sie müssen -(Palette window+6) und nicht (-Palette window+6) übergeben.

Round corner window (16)



- Kann Titel haben: Ja
- Kann Schließbox o.ä. haben: Ja
- Kann in der Größe verändert werden: Nicht auf Macintosh
- Kann auf max. bzw. min. Größe oder Bildschirmgröße gestellt werden: Nein
- Eignet sich für Rollbalken: Nicht auf Macintosh
- Verwendung: Selten

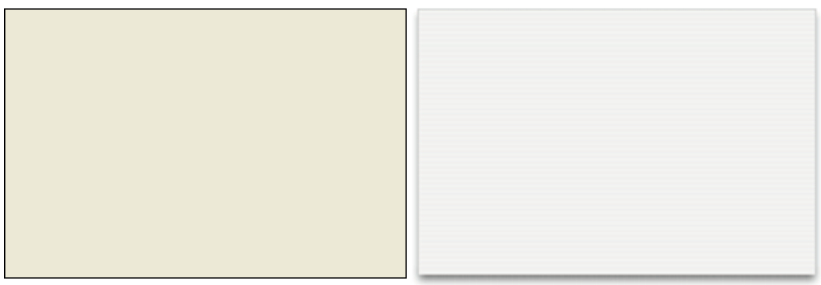
Sheet window (33) und Resizable sheet window (34)



Sheet Fenster sind spezifisch für Mac OS X. Sie rollen animiert unter der Titelleiste des Hauptfensters auf und liegen vor diesem Fenster. Sie haben ähnliche Eigenschaften wie modale Dialogfenster. Sie dienen dazu, eine Operation auszuführen, die mit der Aktion im Hauptfenster zusammenhängt. In 4D wird z.B. ein Sheet Fenster ausgelöst, wenn der Benutzer im Etikетteneditor auf die Schaltfläche **Sichern** klickt.

- Sie können ein Sheet Fenster auf Mac OS X nur erstellen, wenn das letzte offene Fenster ein Dokumenttyp (Formular) und sichtbar ist.
- Die Funktion öffnet ein Fenster vom Typ 1 (Modal dialog box) statt vom Typ 33 bzw. ein Fenster vom Typ 8 (Plain) statt vom Typ 34:
  - wenn das zuletzt geöffnete Fenster nicht sichtbar ist oder kein Dokumenttyp ist,
  - unter Windows.
- Da ein Sheet Fenster über ein Formular gezeichnet wird, wird seine Anzeige im Ereignis *On load* des ersten in das Fenster geladenen Formulars zurückgesetzt (siehe Beispiel 4 unter der Funktion **Open window**).
- Verwendung: **DIALOG**, **ADD RECORD** o.ä, auf Mac OS (kein Standard unter Windows).

Pop up window (32)

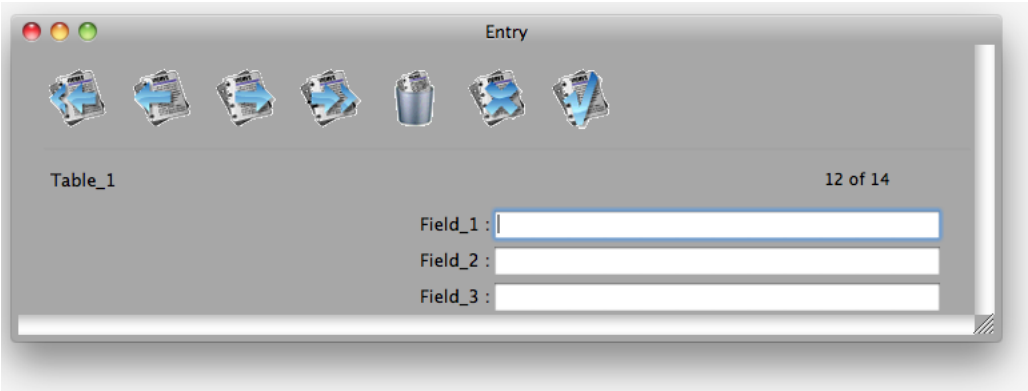


Dieses Fenster hat die gleichen Merkmale wie Fenstertyp 2 (hat Fenstertitel) und folgende weitere Eigenschaften:

- Das Fenster wird automatisch geschlossen und das Ereignis “Abbrechen” wird übergeben, wenn:
  - Ein Klick außerhalb des Fensters erfolgt,
  - Das Hintergrund- bzw. MDI (Multiple Document Interface) Fenster bewegt wird,
  - Der Benutzer auf die Taste **Esc** klickt.
- Dieses Fenster erscheint vor seinem Hauptfenster (es darf nicht als Hauptfenster des Prozesses verwendet werden). Das Hintergrundfenster wird bei Anzeige des Fensters nicht deaktiviert. Es empfängt jedoch keine Ereignisse mehr.

- Sie können das Fenster mit der Maus weder anpassen noch bewegen; führen Sie diese Aktionen jedoch per Programmierung aus, wird das Neuzeichnen der Hintergrundelemente optimiert.
- Verwendung: Dieser Fenstertyp dient hauptsächlich zum Verwalten von PopUp-Menüs, die mit 3D Schaltflächen vom Typ "bevel" oder mit Icons der Werkzeugpalette verknüpft sind.
- Einschränkungen:
  - Innerhalb díeses Fensters lassen sich keine PopUp-Menü Objekte anzeigen
  - Dieser Fenstertyp unterstützt ab 4D Version 13 nicht mehr die Anzeige von Hilfetipps auf Mac OS.

Texture appearance (2048)



Auf Mac OS können Sie Fenster mit metallischem Aussehen anlegen. Diese Darstellungsart gilt für die Mac OS Oberfläche, unter Windows hat sie keine Auswirkung.

Um ein Fenster, das mit der Funktion **Open window** erstellt wurde, metallisch darzustellen, fügen Sie die Konstante Texture appearance zum Fenstertyp hinzu, der im Parameter *Typ* gesetzt wurde. Beispiel:

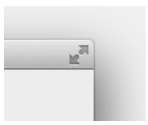
```
$win:=Open window(10;80;-1;-1;Plain window+Texture appearance;"")
```

Diese Darstellung ist für folgende Fenstertypen möglich:

Plain window
Plain no zoom box window
Plain fixed size window
Movable dialog box
Round corner window

Has full screen mode Mac (65536)

Ab 4D v14 gibt es auf OS X für Fenster vom Typ Dokument die Option "full screen". Mit dieser Option erscheint in der rechten oberen Ecke des Fensters das Icon für Vollbildmodus:



Klickt der Benutzer auf dieses Icon, wechselt das Fenster auf Bildschirmgröße und 4D blendet automatisch die Hauptwerkzeugleiste aus.
Dazu fügen Sie für die Funktionen **Open window**, **Open form window** und **_o_Open external window** im Parameter *Typ* die Konstante Form has full screen mode Mac hinzu. Der folgende Code erstellt ein Formularfenster mit Icon für Vollbildmodus auf OS X:

```
$win:=Open form window([Interface];"User_Choice";Plain form window+Form has full screen mode Mac)
DIALOG([Interface];"User_Choice")
```

Hinweis: Unter Windows hat diese Option keine Auswirkung.

Formel

- Tokens in Formeln verwenden
- EDIT FORMULA
- EXECUTE FORMULA
- GET ALLOWED METHODS
- Parse formula Neu 17.0
- SET ALLOWED METHODS

Tokens in Formeln verwenden

Überblick

Die Programmiersprache von 4D enthält ein einmaliges System zur "Tokenisierung" für alle Objektnamen der Programmierung, die im Code vorkommen (Konstanten, Befehle, Tabellen, Felder und Schlüsselwörter). Die Namen als Token setzen bedeutet, dass sie beim Tippen im Code-Editor intern als absolute Referenzen (Nummern) gespeichert werden und dann je nach Kontext beim Ausführen oder Anzeigen als Text wiederhergestellt werden. Auf diese Weise können Sie sicherstellen, dass der Code immer korrekt interpretiert wird, selbst wenn Sie Ihre Tabellen oder Felder umbenennen oder wenn 4D Befehle im Laufe der verschiedenen Versionen umbenannt werden.

Hinweis: Das sorgt auch für die automatische Übersetzung des Code, wenn Sie in den 4D Einstellungen auf der **Seite Methoden** unter Optionen "Verwende regionale Systemeinstellungen" markiert haben und Ihre Anwendungen mit 4D Versionen in verschiedenen Sprachen öffnen.

Die *Tokenisierung* läuft für 4D Entwickler beim Arbeiten im Code-Editor vollkommen transparent. Dieser Mechanismus ist dagegen nicht automatisch in 4D Formeln implementiert, da sie aus Text bestehen, der beim Ausführen und nicht beim Eintippen interpretiert wird. Das ist immer der Fall, wenn 4D Code als Rohtext dargestellt wird, insbesondere wenn Code exportiert und dann über die Befehle **METHOD GET CODE** und **METHOD SET CODE** importiert, kopiert/eingefügt oder über **4D HTML Tags** interpretiert wird.
Damit Sie die *Tokenisierung* auch in diesen Kontexten nutzen können, müssen Sie die jeweiligen Token als Endung der Objektnamen von der Programmiersprache setzen (siehe unten).

Syntax mit Tokens

Tokens sind nicht standardmäßig in 4D Formeln bzw. in Szenarien enthalten, wo 4D Code als Rohtext dargestellt wird. 4D bietet jedoch in Ausdrücken für Elemente mit Namen eine spezielle Syntax, um direkt Tokens zuzuweisen. Dazu müssen Sie dem Elementnamen die passende Endung für den Typ (Befehl, Feld, etc.) und die Referenz geben. Die Syntax mit Tokens lautet wie folgt:

Element	Beispiel (Standardsyntax)	Endung	Beispiel (Syntax mit Token)	Kommentar
4D Befehl	String(a)	:Cxx	String:C10(a)	xx ist die Befehlsnummer
Tabelle	[Angestellte]	:xx	[Angestellte:1]	xx ist die Tabellennummer
Feld	[Angestellte]Name	:xx	[Angestellte:1]Name:2	xx ist die Feldnummer
4D Plug-In	PV PRINT(Bereich)	:Pxx:yy	PV PRINT:P13000:229(Bereich)	xx ist die ID des Plug-In, yy ist die Befehlsnummer

Beachten Sie, dass in Endungen Großbuchstaben (C, P) verwendet werden müssen; sonst werden sie nicht korrekt interpretiert.

Mit dieser Syntax sorgen Sie dafür, dass Ihre Formeln korrekt interpretiert werden, auch bei Umbenennung einzelner Elemente oder Ausführung in einer anderen Sprache.

Hinweis: Auch Konstanten werden als Token gesetzt. In Formeln können Sie jedoch einfach ihren Wert übergeben, um sie vom Kontext unabhängig zu machen.

Diese Syntax wird in allen 4D Formeln (oder 4D Ausdrücken) akzeptiert, unabhängig vom jeweiligen Kontext:

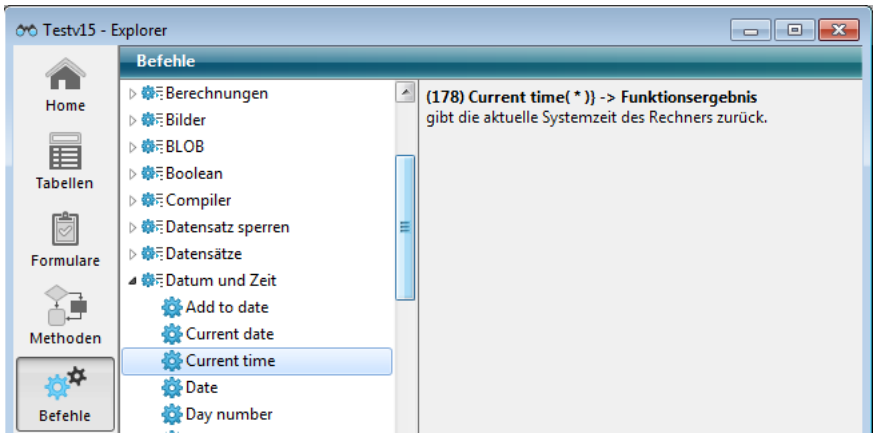
- 4D Formeln, die mit dem **Formeleditor** ausgeführt werden oder Befehle wie **EXECUTE FORMULA**, **APPLY TO SELECTION**, **QUERY BY FORMULA**, **LISTBOX INSERT COLUMN FORMULA**, etc.
- Ausdrücke, die in Rich Text Bereiche eingefügt werden (siehe **ST INSERT EXPRESSION** und **Unterstützte Tags**),
- Ausdrücke, die in Transformation Tags berechnet werden (siehe **4D HTML Tags**),
- Ausdrücke, die in Plug-In Bereiche eingefügt werden,
- Ausdrücke, die in 4D Write Pro Bereiche eingefügt werden (ab 4D v15).

Die Elementnummern finden

Die Syntax mit Token benötigt die Referenznummer von verschiedenen Elementen, die je nach Typ an unterschiedlichen Stellen liegen.

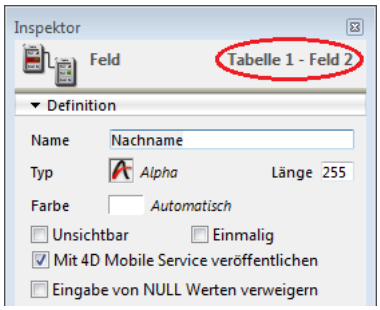
4D Befehle

Befehlsnummern erscheinen in diesem Handbuch *4D Programmiersprache* rechts oben unter **Eigenschaften** und im Explorer auf der Seite **Befehle**:



Tabellen und Felder

Tabellen- und Feldnummern erhalten Sie über die Funktionen **Table** und **Field**.
Sie erscheinen auch im **Inspektorfenster** des Strukturedit ors:



4D Plug-In Befehle

Für Tokens von Befehlen eines 4D Plug-Ins gibt es einen Trick: Sie geben den gewünschten Code im Methodeneditor ein, deaktivieren das Plug-In, indem Sie z.B. seinen Ordner verschieben, und starten 4D neu. Dann erscheinen nur Tokens im Methodeneditor und Sie können die benötigten Tokens kopieren.

Code mit installiertem Plug-In:

```
12      PV SET ROWS HEIGHT (Area;1;10;PV Get row height (Area;10)+$Height)
```

Derselbe Code nach Deaktivierung des Plug-In:

```
12      Ô13000;75Ô (Area;1;10;Ô13000;76Ô (Area;10)+$Height)
```

EDIT FORMULA

EDIT FORMULA (Tabellenname ; Formel)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, die standardmäßig im Formeleditor angezeigt wird
Formel	Text	➡	Variable mit der Formel zur Anzeige im Formeleditor oder "", um nur den Editor anzuzeigen
		⬅	Vom Benutzer bestätigte Formel

Beschreibung

Der Befehl **EDIT FORMULA** öffnet den Formeleditor und zeigt ihn standardmäßig an:

- In der linken Liste die Felder der Tabelle, übergeben in *Tabellenname*.
- Im Bereich Formel die Formel, übergeben in der Variablen *Formel*. Übergeben Sie einen leeren String, erscheint der Formeleditor ohne Formel.

Erscheint der Editor, kann der Benutzer *Formel* wie definiert anzeigen – sofern die Syntax korrekt ist – oder diese verändern, eine Formel schreiben, eine neue laden oder sogar sichern. Bestätigt der Benutzer das Dialogfenster, wird die Systemvariable OK auf 1 gesetzt, die Variable *Formel* enthält die aktuell ausgeführte Formel. Bricht der Benutzer die Formel ab, wird die Systemvariable auf 0 (Null) gesetzt, die Variable *Formel* bleibt unverändert.

Hinweise:

- Der Zugriff auf Methoden und Befehle ist standardmäßig für alle Benutzer eingeschränkt. Davon ausgenommen sind Designer und Administrator in Datenbanken, die mit 2004.4 oder höher erstellt wurden. Über den Befehl **SET ALLOWED METHODS** müssen Sie explizit die Elemente angeben, auf welche die Benutzer zugreifen können. Ruft *Formel* Methoden auf, die nicht zuvor über den Befehl **SET ALLOWED METHODS** im Formeleditor zugelassen wurden, wird ein Syntaxfehler generiert und Sie können das Dialogfenster nicht bestätigen.
- Der Formeleditor ist standardmäßig keiner Menüleiste zugeordnet. Damit Benutzer im Formeleditor die Tastenkombinationen für Ausschneiden/Kopieren/Einsetzen nutzen können, müssen Sie im aufrufenden Prozess ein Standardmenü **Bearbeiten** installieren.

Beachten Sie, dass der Befehl beim Bestätigen des Dialogfensters nicht die *Formel* ausführt; er bestätigt und aktualisiert lediglich den Inhalt der Variablen. Um *die Formel* auszuführen, müssen Sie den Befehl **EXECUTE FORMULA** verwenden.

Beispiel

Formeleditor mit der Tabelle [Angestellte] ohne vorab eingegebene Formel anzeigen:

```
$myFormula:=""
EDIT FORMULA([Angestellte];$myFormula)
If(OK=1)
   APPLY TO SELECTION([Angestellte];EXECUTE FORMULA($myFormula))
End if
```

Systemvariablen und Mengen

Führt der Benutzer die Formel aus, wird die Systemvariable OK auf 1 gesetzt. Bricht der Benutzer die Formel ab, wird die Systemvariable OK auf 0 (Null) gesetzt.

EXECUTE FORMULA

EXECUTE FORMULA (Anweisung)

Parameter	Typ		Beschreibung
Anweisung	String	➡	Auszuführender Code

Beschreibung

Der Befehl **EXECUTE FORMULA** führt *Anweisung* als eine Codezeile aus. Dieser Befehl kommt zum Einsatz, wenn Sie eine Anweisung bewerten müssen, die ein Benutzer erstellen oder ändern kann.

Die Anweisung muss in einer Zeile sein. Ist *Anweisung* ein leerer String, führt **EXECUTE FORMULA** nichts aus. Als Faustregel gilt: *Anweisung* wird korrekt ausgeführt, wenn sie als einzeilige Methode ausführbar ist.
Sie sollten den Befehl nur selten verwenden, da er die Ausführungsgeschwindigkeit verlangsamt. In einer kompilierten Datenbank ist die Codezeile nicht kompiliert, d.h. *Anweisung* wird ausgeführt, jedoch nicht vom Compiler in Kompilierzeit geprüft. Eine Case of-Schleife ist in jedem Fall schneller.

Hinweis: Die Ausführung von Formeln im kompilierten Modus lässt sich durch Verwendung des Cache optimieren (siehe nächsten Absatz **Cache für Formeln im kompilierten Modus**).

Anweisung kann folgende Elemente enthalten:

- Aufruf einer Funktion (Projektmethode, die einen Wert zurückgibt)
- Aufruf eines 4D Befehls
- Eine Zuweisung

Hinweise:

- Ist *Anweisung* eine Projektmethode, sollten Sie **EXECUTE METHOD** verwenden, da Sie hier auch Parameter übergeben können.
- Sie sollten in *Anweisung* keine Befehle zur Variablendeklaration aufrufen, wie z.B. **C_DATE**, da dies zu Konflikten im Code führen kann.

Anweisung kann Prozessvariablen und Interprozessvariablen enthalten, jedoch keine Elemente für Befehlsfolgen, wie If, Case of, Else, da nur einzeiliger Code möglich ist.

Um sicherzustellen, dass *Anweisung* unabhängig von der 4D Programmiersprache oder Version korrekt bewertet wird, empfehlen wir die Syntax mit Tokens bei Elementen zu verwenden, deren Name sich zwischen verschiedenen Versionen ändern kann (Befehle, Tabellen, Felder, Konstanten). Um beispielsweise den Befehl **Current time** einzufügen, geben Sie '**Current time:C178**' ein. Weitere Informationen dazu finden Sie im Abschnitt **Tokens in Formeln verwenden**.

Cache für Formeln im kompilierten Modus

Für Optimierungszwecke lässt sich jede Formel, die über den Befehl **EXECUTE FORMULA** im kompilierten Modus ausgeführt wird, in einem neuen spezifischen Cache im Speicher beibehalten. Die Formel wird als Token abgespeichert. Liegt sie einmal im Cache, sind nachfolgende Ausführungen in hohem Maße optimiert, da die Tokenisierung übersprungen wird.

Die Größe des Cache ist standardmäßig Null (kein Cache); sie muss über den Befehl **SET DATABASE PARAMETER** erstellt bzw. angepasst werden.

```
SET DATABASE PARAMETER(Number of formulas in cache;0)  //Kein Cache für Formeln
SET DATABASE PARAMETER(Number of formulas in cache;3)  //Bis zu drei Formeln können für alle Prozesse im Cache gespeichert werden
```

Der Befehl **EXECUTE FORMULA** verwendet den Cache nur, wenn er über eine kompilierte Datenbank oder Komponente aufgerufen wird.

Beispiel

Sie wollen Formeln ausführen, die Aufrufe von 4D Befehlen und Tabellen enthalten. Da diese u.U. umbenannt werden, können Sie für die korrekte Ausführung in zukünftigen Versionen sorgen, wenn Sie die Syntax mit Tokens verwenden:

```
EXECUTE FORMULA("Year of:C25 ([Products:5]Creation_Date:2])+$add")
```

GET ALLOWED METHODS

GET ALLOWED METHODS (MethodeArray)

Parameter	Typ		Beschreibung
MethodeArray	Array String	⇐	Array der Methodennamen

Beschreibung

Der Befehl **GET ALLOWED METHODS** gibt in *MethodeArray* die Namen der Methoden zurück, die zum Schreiben von Formeln verwendbar sind. Sie erscheinen im Editor am Ende der Befehlsliste.

Standardmäßig lassen sich keine Methoden im Formeleditor verwenden. Die Methoden müssen über den Befehl **SET ALLOWED METHODS** explizit zugelassen werden. Wurde dieser Befehl nicht ausgeführt, gibt **GET ALLOWED METHODS** ein leeres Array zurück.

GET ALLOWED METHODS gibt genau das zurück, was im Befehl **SET ALLOWED METHODS** übergeben wurde; bei Bedarf erstellt und passt der Befehl das Array an. Wurde über das Joker-Zeichen (@) eine Methodengruppe festgelegt, wird der String mit dem Zeichen @ zurückgegeben, und nicht der Name der Methodengruppe.

Dieser Befehl ist hilfreich, um die Einstellungen des aktuellen Satzes zugelassener Methoden vor Ausführen der Formel in einem spezifischen Kontext zu speichern, z.B. ein Bericht.

Beispiel

Dieser Code authorisiert einen Satz spezifischer Methoden zum Erstellen eines Berichts:

```
  `Speichere aktuelle Parameter
GET ALLOWED METHODS(methodsArray)
  `Definiere Methoden für Schnellbericht
methodsarr_Reports{1}:="Reports_@"
SET ALLOWED METHODS(methodsarr_Reports)
QR REPORT([People];"MyReport")
  `Stelle aktuelle Parameter wieder her
SET ALLOWED METHODS(methodsArray)
```

Parse formula

Parse formula (formula {; options}{; errorMessage}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
formula	Text	➡	Formel in Rohtext (plain text)
options	Lange Ganzzahl	➡	Anweisungen für Eingabe / Ausgabe
errorMessage	Text	⬅	Fehlermeldung (ohne Fehler leerer String)
Funktionsergebnis	Text	↩	Umgewandelte Formel (plain text)

Beschreibung

Die Funktion **Parse formula** analysiert die 4D *formula*, prüft ihre Syntax und gibt ihre generische Form zurück. So bleibt die Formel auch gültig, wenn ein Element der 4D Programmiersprache oder Struktur umbenannt wird (Befehl, Konstante, Tabelle, Feld oder 4D Plug-In).

Mit **Parse formula** lassen sich Formeln folgendermaßen bewerten und übersetzen:

- Reale Tabellen-/Feldnamen lassen sich in virtuelle Strukturnamen* (eigene Namen) oder tokenisierte Entsprechungen** konvertieren
- Tokenisierte Entsprechungen für Tabelle/Feld lassen sich in virtuelle Strukturnamen oder reale Tabellen-/Feldnamen konvertieren
- Virtuelle Strukturen lassen sich in reale Tabellen-/Feldnamen oder tokenisierte Entsprechungen konvertieren
- Elemente der 4D Programmiersprache lassen sich in tokenisierte Entsprechungen der 4D Programmiersprache konvertieren
- Tokenisierte Entsprechungen der 4D Programmiersprache lassen sich in Elemente der 4D Programmiersprache konvertieren

** Virtuelle Strukturen werden mit den Befehlen **SET TABLE TITLES** und **SET FIELD TITLES** definiert (der Parameter * ist erforderlich).*

*** Tokenisierte Entsprechungen sind Elemente der 4D Programmiersprache und Struktur in Rohtext (plain text), dargestellt in Token Syntax (siehe auch **Tokens in Formeln verwenden**). Hierzu ein Beispiel:*

```
[Table:1]Field:1+String:C10(1)
```

In *formula* übergeben Sie eine 4D Formel in Rohtext. Sie kann reale oder virtuelle Strukturnamen, sowie tokenisierte Entsprechungen verwenden.

Unabhängig vom verwendeten Namenstyp in *formula* gibt **Parse formula** standardmäßig die Namen der aktuellen Elemente der 4D Programmiersprache oder Struktur ohne Text-Tokens zurück.

Mit dem optionalen Parameter *options* können Sie angeben, wie *formula* übergeben bzw. zurückgegeben wird. Dazu verwenden Sie eine der Konstanten unter dem Thema **Formulas**. Sie können auch Konstanten kombinieren, um sowohl Eingabe- als auch Ausgabeformat der zurückgegebenen Formel anzugeben.

Konstante	Wert	Kommentar
Formula in with virtual structure	1	Formel enthält eigene (virtuelle) Namen). Die zurückgegebene Formel enthält standardmäßig reale Namen.
Formula out with virtual structure	2	Die zurückgegebene Formel muss eigene (virtuelle) Namen enthalten.
Formula out with tokens	4	Die zurückgegebene Formel muss Text Tokens enthalten (z.B. :Cxx).

Der optionale Parameter *errorMessage* empfängt eine Fehlermeldung, wenn in *formula* ein Syntaxfehler auftritt. Gibt es keinen Fehler, wird ein leerer String zurückgegeben.

Beispiel 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;*)

 //Virtuelle Struktur in Tabellen- und Feldnamen Entsprechung setzen
$parsedFormula:=Parse formula("[Virtual table]Virtual field";Formula in with virtual structure;$errorMessage)
 //ergibt [Table_1]Field_2

 //Tabellen- und Feldname in virtuelle Struktur Entsprechung setzen
$parsedFormula:=Parse formula("[Table_1]Field_2";Formula out with virtual structure;$errorMessage)
 //ergibt [Virtual table]Virtual field
```

```
 //Tabellen- und Feldname in tokenisierte Entsprechung setzen
$parsedFormula:=Parse formula("String([Table_1]Field_2)";Formula out with tokens;$errorMessage)
 //ergibt String:C10([Table_1:1]Field_2:2)
```

Beispiel 2

Mit den Tabellen aus **Beispiel 1** folgendes ausführen:

```
 //den Benutzer nach seiner bevorzugten Formeldarstellung fragen
$formula:=""
EDIT FORMULA([Table_1];$formula)

 //die verwendete Formel für später sichern
CREATE RECORD([users_preferences])
$persistentFormula:=Parse formula($formula;Formula out with tokens)
[users_preferences]formula:=$persistentFormula

 //später: die zuvor gesicherte Formel ausführen
CREATE RECORD([Table_1])
EXECUTE FORMULA([users_preferences]formula)
```

SET ALLOWED METHODS

SET ALLOWED METHODS (MethodeArray)

Parameter	Typ		Beschreibung
MethodeArray	Array Text	➡	Array mit den Methodennamen

Beschreibung

Der Befehl **SET ALLOWED METHODS** definiert die Projektmethoden, die direkt in der Anwendung aufrufbar sind.
4D enthält einen Sicherheitsmechanismus, der aufrufbare Projektmethoden in folgenden Kontexten filtert:

- Formeleditor - die zugelassenen Methoden erscheinen am Ende der Liste der Standardbefehle und lassen sich in Formeln verwenden (siehe Abschnitt **Beschreibung des Formeleditors**).
- 4D Write Pro Dokumente - zugelassene Methoden lassen sich in eingefügten, dynamischen Ausdrücken verwenden (siehe Abschnitt **Filtern von Ausdrücken in einem 4D Write Pro Dokument**).
- Etiketteneditor in 64-bit - die zugelassenen Methoden erscheinen im Menü **Anwenden**, wenn sie gemeinsam mit der Komponente verwendet werden (siehe Abschnitt **Beschreibung des Etiketteneditors**).

Standardmäßig, d.h. wenn dieser Befehl nicht verwendet wird, ist keine Methode aufrufbar. Verwendet ein Ausdruck eine nicht zugelassene Methode, wird ein Syntaxfehler generiert.

Im Parameter *MethodeArray* übergeben Sie den Namen des Array mit der Liste der Methoden, die für den Formeleditor vorgeschlagen werden. Das Array muss natürlich zuvor angelegt werden.
Mit dem “Joker” Zeichen (@) in Methodennamen können Sie eine oder mehrere zugelassene Methodengruppen definieren.

Soll der Benutzer auch die Möglichkeit haben, 4D oder Plug-In Befehle aufzurufen, die standardmäßig nicht allgemein zugänglich sind, müssen Sie dafür spezifische Methoden verwenden.

Hinweis: Auf der Seite **Sicherheit** der Datenbank-Eigenschaften lässt sich der eingeschränkte Zugriff auf Befehle und Methoden im Formeleditor für alle Benutzer oder für Designer und Administrator deaktivieren. Ist die Option "Deaktivieren für alle" markiert, hat der Befehl **SET ALLOWED METHODS** keine Auswirkung.

Beispiel

Dieses Beispiel lässt alle Methoden zu, die mit “formula” beginnen und die Methode “Total_general” im Formeleditor:

```
ARRAY TEXT(methodsArray;2)
methodsArray{1}:="formula@"
methodsArray{2}:="Total_general"
SET ALLOWED METHODS(methodsArray)
```

Formulare

- 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 -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Text	Name des aktuellen Projektformulars oder des aktuellen Tabellenformulars im Prozess

Beschreibung

Die Funktion **Current form name** gibt den Namen des aktuellen Formulars zurück, das für den Prozess definiert wurde. Das kann ein Projektformular oder ein Tabellenformular sein.

Standardmäßig, d.h. ohne Aufrufen des Befehls **FORM LOAD** im aktuellen Prozess, ist das aktuelle Formular das gerade angezeigte oder gedruckte. Haben Sie den Befehl **FORM LOAD** m Prozess aufgerufen, wird das hier gesetzte Formular das aktuelle Formular und bleibt es, bis Sie den Befehl **FORM UNLOAD** (oder **CLOSE PRINTING JOB**) aufrufen.

Die Funktion gibt folgendes zurück:

- Den Namen des Formulars oder
- Den Dateinamen ohne Endung, wenn das aktuelle Formular über eine eine .json Datei erstellt wurde.
- Das Attribut "name", wenn das aktuelle Formular über ein 4D Objekt erstellt wurde, oder
- Einen leeren String, wenn für den Prozess kein aktuelles Formular definiert wurde.

Beispiel 1

Eingabeformular mit Code in einer Schaltfläche:

```
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"
 //...
```

Beispiel 2

Sie wollen das aktuelle Formular abfragen, aber nur wenn es ein Projektformular ist:

```
$PointerTable:=Current form table
If(Nil($PointerTable))  // Dies ist ein Projektformular
    $FormName:=Current form name
    ... // Bearbeitung
End if
```

Form

Form -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Objekt	Dem aktuellen Formular zugeordnete Formulardaten

Beschreibung

Die Funktion **Form** gibt das dem aktuellen Formular zugeordnete Objekt zurück, falls vorhanden. 4D weist dem aktuellen Formular automatisch ein Objekt zu, wenn:

- das aktuelle Formular über den Befehl **DIALOG** angezeigt wird
  oder
- das aktuelle Formular ein Unterformular ist

Formular über DIALOG

Wird das aktuelle Formular über den Befehl **DIALOG** aufgerufen, gibt **Form** das Objekt *FormularData* zurück, wenn es als Parameter an **DIALOG** übergeben wurde, sonst ein leeres Objekt.

Unterformular

Ist das aktuelle Formular ein Unterformular, richtet sich das zurückgegebene Objekt nach der Variablen des übergeordneten Container:

- Ist die Variable als Objekt (**C_OBJECT**) typisiert, gibt **Form** den Wert dieser Variablen zurück.
  Dann wird über **Form** dasselbe Objekt zurückgegeben wie über die Anweisung:

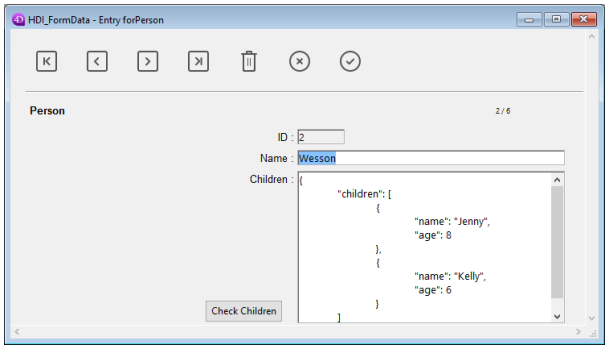
```
(OBJECT Get pointer(Object subform container))->
```

- Ist die Variable nicht als ein Objekt typisiert, gibt **Form** im Unterformularkontext ein leeres Objekt zurück.

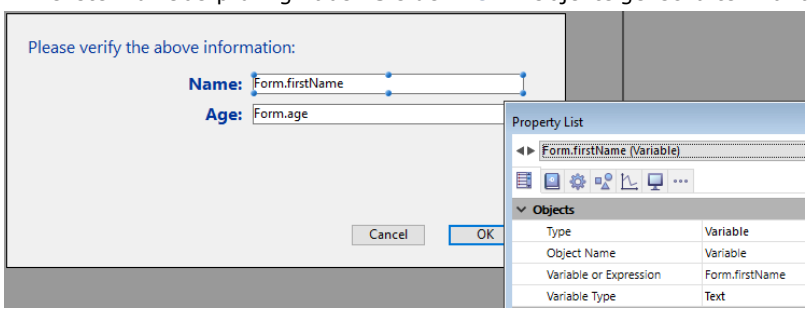
Weitere Informationen dazu finden Sie im Abschnitt **Unterformulare als Seite**.

Beispiel

In einen Eingabeformular zu einer Person lässt sich über die Schaltfläche "Check Children" ein Dialog öffnen, um Name und Alter ihrer Kinder zu überprüfen/ändern:



Hinweis: Das Objektfeld "Children" dient nur dazu, um seine Struktur für dieses Beispiel anzuzeigen.
Im Fenster zur Überprüfing haben Sie den **Form** Objekteigenschaften Variablen zugewiesen:

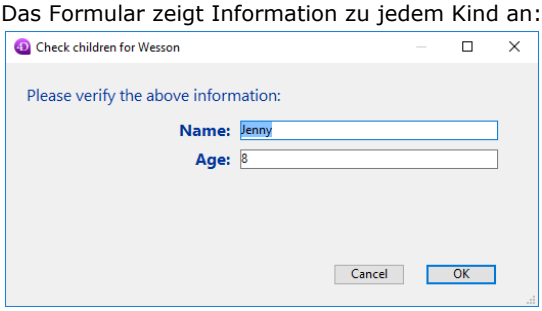


Der Code für die Schaltfläche "Check Children" lautet:

```
C_LONGINT($win;$n;$i)
C_BOOLEAN($save)
ARRAY OBJECT($children;0)
OB GET ARRAY([Person]Children;"children";$children)  //die  Collection children erhalten
$save:=False  //die Variable save initialisieren
```

```
$n:=Size of array($children)
If($n>0)
    $win:=Open form window("Edit_Children";Movable form dialog box)
    SET WINDOW TITLE("Check children for "+[Person]Name)
    For($i;1;$n)  //für jedes Kind
        DIALOG("Edit_Children";$children{$i})  //den mit Werten gefüllten Dialog anzeigen
        If(OK=1)  //der Benutzer hat auf OK geklickt
            $save:=True
        End if
    End for
    If($save=True)
        [Person]Children:=[Person]Children  //Update des Objektfeldes erzwingen
    End if
    CLOSE WINDOW($win)
Else
    ALERT("No child to check.")
End if
```

Hinweis: Für dieses Beispiel muss die Objektnotation in der Anwendung aktiviert sein (siehe **Seite Kompatibilität**).
Das Formular zeigt Information zu jedem Kind an:



Werden die Werte bearbeitet und auf die Schaltfläche OK geklickt, wird das Feld aktualisiert (der übergeordnete Datensatz muss anschließend gesichert werden).

FORM FIRST PAGE

FORM FIRST PAGE
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **FORM FIRST PAGE** wechselt von der aktuellen Seite eines Formulars auf seine erste Seite. Wird kein Formular angezeigt bzw. über den Befehl **FORM LOAD** geladen oder ist die angezeigte Seite bereits die erste Seite des Formulars, wird der Befehl nicht ausgeführt.

Beispiel

Folgendes Beispiel ist eine einzeilige Methode, die über Menübefehl aufgerufen wird. Sie zeigt die erste Seite des Formulars an:

```
FORM FIRST PAGE
```

FORM Get current page

FORM Get current page {(*)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	⇒	Gibt die Nummer der aktuellen Seite des Unterformulars zurück
Funktionsergebnis	Lange Ganzzahl	⇐	Nummer der aktuell angezeigten Formularseite

Beschreibung

Die Funktion **FORM Get current page** gibt die Nummer der aktuell angezeigten Formularseite oder des aktuellen Formulars, geladen über den Befehl **FORM LOAD** zurück.

Der Parameter * ist hilfreich, wenn die Funktion im Rahmen einer Seite vom Typ Unterformular mit mehreren Seiten aufgerufen wird. Mit * gibt die Funktion die aktuelle Seite des aktuellen Unterformulars, d.h. das die Funktion aufgerufen hat, zurück. Standardmäßig, also ohne *, wird die Funktion immer auf das Elternformular angewandt.

Beispiel

Wenn Sie in einem Formular einen Menübefehl in einem Menü auswählen, oder das Formular einen Aufruf von einem anderen Prozess erhält, können Sie im Formular je nach aktuell angezeigter Formularseite eine andere Aktion aufrufen:

```
  ` Formularmethode[myTable];"myForm"
Case of
   :(Form event=On Load)
` ...
   :(Form event=On Unload)
` ...
   :(Form event=On Menu Selected)
      $vlMenuNumber:=Menu selected>>16
      $vlItemNumber:=Menu selected  & 0xFFFF
      Case of
         :($vlMenuNumber=...)
            Case of
               :($vlItemNumber=...)
               :(FORM Get current page=1)
` Führe geeignete Aktion für Seite 1 aus
               :(FORM Get current page=2)
` Führe geeignete Aktion für Seite 2 aus
` ...
               :($vlItemNumber=...)
` ...
            End case
         :($vlMenuNumber=...)
` ...
      End case
   :(Form event=On Outside Call)
      Case of
         :(FORM Get current page=1)
` Führe geeignete Anwort für Seite 1 aus
         :(FORM Get current page=2)
` Führe geeignete Anwort für Seite 2 aus
      End case
` ...
End case
```

FORM GET ENTRY ORDER

FORM GET ENTRY ORDER (ObjektNamen {; SeitenNr | *})

Parameter	Typ		Beschreibung
ObjektNamen	Array Text	⇐	ObjektNamen sortiert nach Eingabereihenfolge
SeitenNr | *	Lange Ganzzahl, Operator	⇒	Nummer der Seite zum Erhalten der definierten Eingabereihenfolge (ohne Angabe aktuelle Seite) oder * zum Erhalten der aktuellen Eingabereihenfolge der aktuellen Seite

Beschreibung

Der Befehl **FORM GET ENTRY ORDER** gibt in *ObjektNamen* die sortierten Namen der Objekte zurück, die die Eingabereihenfolge im Formular definieren.

- Ohne den Parameter * gibt **FORM GET ENTRY ORDER** die Eingabereihenfolge zurück, wie sie zuvor mit dem Befehl **FORM SET ENTRY ORDER** definiert wurde. Den Parameter *SeitenNr* können Sie weglassen oder übergeben:
  - Ohne den Parameter *SeitenNr* gibt das Array *ObjektNamen* die Eingabereihenfolge für die aktuelle Seite zurück
  - Mit dem Parameter *SeitenNr* gibt das Array *ObjektNamen* die Eingabereihenfolge für die Seite *SeitenNr* zurück

  In beiden Fällen wird das Array *ObjektNamen* leer zurückgegeben, wenn nicht zuvor für das aktuelle Formular der Befehl **FORM SET ENTRY ORDER** aufgerufen wurde.

- Mit dem Parameter * gibt **FORM GET ENTRY ORDER** die aktuelle Eingabereihenfolge der aktuellen Seite zurück, z.B. enthält das Array *ObjektNamen* nur **gültige** Objektnamen. Weitere Informationen dazu finden Sie unter dem Befehl **FORM SET ENTRY ORDER**. Die aktuelle Eingabefolge des Formulars kann folgendes sein:
  - Die standardmäßige Eingabereihenfolge gemäß der Anordnung der Objekte
  - Oder die Eingabefolge des Formulareditors (siehe **Eingabereihenfolge der Daten ändern**), falls verwendet
  - Oder die Eingabereihenfolge, gesetzt durch Aufrufen von **FORM SET ENTRY ORDER** in aktuellen Prozess, falls verwendet

  Die aktuelle Eingabereihenfolge enthält immer Objekte von der Seite 0 und von vererbten Formularen.

Hinweis: Bei Anwendung auf ein Hauptformular gibt dieser Befehl nicht die Eingabereihenfolge in einem Unterformular zurück.

Beispiel

Einige Objekte aus der aktuellen Eingabefolge ausschließen:

```
ARRAY TEXT($arrTabOrderObject;0)
C_LONGINT($vElem)

FORM GET ENTRY ORDER($arrTabOrderObject;*)  //die aktuelle Eingabefolge erhalten
Repeat
   $vElem:=Find in array($arrTabOrderObject;"vTax@")
   If($vElem>0)  //Objekte aus der Eingabefolge ausschließen, deren Name mit "vTax" beginnt
      DELETE FROM ARRAY($arrTabOrderObject;$vElem)
   End if
Until($vElem<0)
FORM SET ENTRY ORDER($arrTabOrderObject)  //die neue Eingabefolge anwenden
```

FORM GET HORIZONTAL RESIZING

FORM GET HORIZONTAL RESIZING (Anpassen {; minBreite {; maxBreite}})

Parameter	Typ		Beschreibung
Anpassen	Boolean	⇐	Wahr: Formular ist horizontal anpassbar Falsch: Formular ist nicht horizontal anpassbar
minBreite	Lange Ganzzahl	⇐	Kleinstmögliche Formularbreite (Pixel)
maxBreite	Lange Ganzzahl	⇐	Größtmögliche Formularbreite (Pixel)

Beschreibung

Der Befehl **FORM GET HORIZONTAL RESIZING** gibt die horizontalen Anpassungseigenschaften des aktuellen Formulars in den Variablen *Anpassen*, *minBreite* und *maxBreite* zurück. Diese Eigenschaften können für das Formular im Formulareditor des Designmodus oder für den aktuellen Prozess über den Befehl **FORM SET HORIZONTAL RESIZING** gesetzt worden sein.

FORM GET OBJECTS

FORM GET OBJECTS (ArrayObjekte {; ArrayVariablen {; ArraySeiten}} {; FormSeiteOption})

Parameter	Typ		Beschreibung
ArrayObjekte	Array String	⇐	Name der Formularobjekte
ArrayVariablen	Array Zeiger	⇐	Zeiger auf Variablen oder Felder, die Objekten zugeordnet sind
ArraySeiten	Array Ganzzahl	⇐	Seitennummer für jedes Objekt
FormSeiteOption	Lange Ganzzahl, Operator	⇒	1=Form current page, 2=Form all pages, 4=Form inherited Mit * (überholt) = Aktuelle Seite mit vererbten Objekten

Beschreibung

Der Befehl **FORM GET OBJECTS** gibt die Liste aller Objekte im aktuellen Formular als Array zurück. Diese Liste lässt sich auf die aktuelle Formularseite beschränken. Der Befehl lässt sich mit Eingabe- und Ausgabeformularen verwenden.

Wurde ein als Parameter übergebenes Array nicht zuvor deklariert, legt der Befehl dieses an und setzt seine Größe automatisch. Es empfiehlt sich jedoch, jedes Array explizit zu deklarieren, da Sie in der Regel Ihre Anwendung kompilieren.
In *ArrayObjekte* übergeben Sie den Namen des String Array mit den Objektnamen. Jeder Objektname innerhalb eines Formulars ist einmalig. Die Reihenfolge der Objekte ist ohne Bedeutung.
Die weiteren Arrays füllt der Befehl bei Bedarf, sie werden mit dem ersten Array synchronisiert.

Im optionalen Parameter *ArrayVariablen* übergeben Sie den Namen des Array mit Zeigern. Es enthält bereits die Zeiger auf die Variablen/Felder, die Objekten zugeordnet sind. Für Objekte ohne Variable wird der Zeiger **Is nil pointer** zurückgegeben. Für Objekte vom Typ Unterformular wird ein Zeiger auf die Tabelle des Unterformulars zurückgegeben.

Der optionale Parameter *ArraySeiten* enthält die jeweiligen Seitennummern zu den Objekten im Formular. Jede Zeile dieses Array enthält die Seitennummer des dazugehörigen Objekts.
Objekte aus einem vererbten Formular gehören zur Seite 0 der aktuellen Seite.

Mit dem optionalen Parameter * können Sie die Liste der zurückgegebenen Objekte auf die aktuelle Seite des Formulars beschränken. Dann gibt der Befehl nur die Objekte der aktuellen Seite, von Seite 0 und den vererbten Seiten zurück, d.h. der Befehl bearbeitet alle Objekte auf der aktuellen Seite, egal ob sichtbar oder nicht.

Mit dem optionalen Parameter *FormSeiteOption* können Sie bestimmte Teile des Formulars angeben, aus denen Sie Objekte erhalten wollen. Standardmäßig, also ohne den Parameter *FormSeiteOption* (und ohne den Parameter *), werden Objekte von allen Seiten zurückgegeben, inkl. vererbte Objekte. Um die Reichweite des Befehls zu reduzieren, übergeben Sie einen Wert in *FormSeiteOption*. Sie können eine oder mehrere kombinierte Konstanten aus dem Thema **Formularobjekte (Zugriff)** übergeben:

Konstante	Typ	Wert	Kommentar
Form all pages	Lange Ganzzahl	2	gibt alle Objekte von allen Seiten ohne vererbte Objekte zurück
Form current page	Lange Ganzzahl	1	Gibt alle Objekte der aktuellen Seite zurück, einschließlich der Seite 0, aber ohne vererbte Objekte
Form inherited	Lange Ganzzahl	4	Gibt nur die vererbten Objekte zurück

Hinweis zur Kompatibilität: Die Übergabe des Parameters * ist dasselbe wie Übergeben der Konstanten Form current page+Form inherited. Die Syntax mit dem Parameter * ist jetzt überholt und sollte nicht mehr verwendet werden.

Beispiel 1

Information auf allen Seiten erhalten, einschließlich Objekten von vererbten Formularen (sofern vorhanden):

```
//offenes Formular
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray)
```

Oder:

```
//geladenes Formular
FORM LOAD([Table1];"MyForm")
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray;Form all pages+Form inherited)
```

Beispiel 2

Nur Information auf der aktuellen Seite erhalten, mit Seite 0 des geladenen Formulars und vererbten Formularobjekten (sofern vorhanden):

```
FORM LOAD("MyForm")
FORM GOTO PAGE(2)
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray;Form current page+Form inherited)
```

Beispiel 3

Information von allen Objekten im vererbten Formular (sofern vorhanden) erhalten. Gibt es kein vererbtes Formular, werden die Arrays leer zurückgegeben.

```
FORM LOAD("MyForm")
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray;Form inherited)
```

Beispiel 4

Information von Objekten auf Seite 4 erhalten, einschließlich Objekten von Seite 0, aber ohne vererbte Formularobjekte (sofern vorhanden):

```
FORM LOAD([Table1];"MyForm")
FORM GOTO PAGE(4)
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray;Form current page)
```

Beispiel 5

Information von Objekten auf allen Seiten erhalten, jedoch ohne vererbte Formularobjekte (sofern vorhanden):

```
FORM LOAD([Table1];"MyForm")
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray;Form all pages)
```

FORM GET PARAMETER

FORM GET PARAMETER ({Tabellenname ;} Formularname ; Selector ; Wert1...N)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Formulartabelle, ohne Angabe Standardtabelle
Formularname	String	⇒	Formularname
Selector	Lange Ganzzahl	⇒	Code des Parameters
Wert1...N	Lange Ganzzahl	⇐	Aktueller Wert des Parameters

Beschreibung

Der Befehl **FORM GET PARAMETER** liefert den aktuellen Wert eines Parameters im Formular, definiert durch *Tabellenname* und *Formularname*.
Selector gibt den Parameter des Formulars an, dessen Wert Sie finden wollen. Sie können dazu folgende Konstante unter dem Thema **Formularoptionen** verwenden:

Konstante	Typ	Wert
NonInverted objects	Lange Ganzzahl	0

Verwenden Sie die Konstante NonInverted Objects als Selector, gibt der Befehl in *Wert* die aktuelle Anzeige des Formulars im Anwendungsmodus unter Windows an. Dieser Parameter wird für Anwendungen verwendet, die mit rechts-nach-links laufenden Sprachen eingesetzt werden. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

- Gibt *Wert* 0 (Null) zurück, werden die Formularobjekte invertiert,
- Gibt *Wert* 1 zurück, werden die Formularobjekte nicht invertiert.

Wird der Befehl nicht im Anwendungsmodus unter Windows aufgerufen, gibt er immer 1 zurück.

Beachten Sie, dass sich die aktuelle Umkehrung der Formularobjekte nach der Kombination verschiedener Parameter richtet: Der Option „Inversion der Objekte im Anwendungsmodus", der Option "Objekte nicht invertieren" auf Formularebene und dem Betriebssystem, auf dem die Datenbank läuft. Nachfolgende Tabelle zeigt die verschiedenen Kombinationen und welchen Wert der Befehl **FORM GET PARAMETER** zurückgibt:

Voreinstellungen: "Umkehrung der Objekte im Anwendungsmodus" (1)	**Formular-eigenschaften: "Objekte nicht umkehren"**	**Rechts-nach-links Anzeige unter Windows**	**Von FORM GET PARAMETER zurückgegebener Wert**
Nein	X	X	1
		X	1
	X		1
			1
Automatisch	X	X	1
		X	0
	X		1
			1
Ja	X	X	1
		X	0
	X		1
			0

(1) Diese Voreinstellung lässt sich auch mit den Routinen **SET DATABASE PARAMETER** und **Get database parameter** setzen bzw. erhalten.

FORM GET PROPERTIES

FORM GET PROPERTIES ({Tabellenname ;} FormularName ; Breite ; Höhe {; AnzSeiten {; FesteBreite {; FesteHöhe {; Titel}}}})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle des Formulars, ohne Angabe Standardtabelle
FormularName	String	⇒	Name des Formulars
Breite	Lange Ganzzahl	⇐	Breite des Formulars (in Pixel)
Höhe	Lange Ganzzahl	⇐	Höhe des Formulars (in Pixel)
AnzSeiten	Lange Ganzzahl	⇐	Anzahl der Seiten im Formular
FesteBreite	Boolean	⇐	Wahr = Feste Breite, Falsch = Variable Breite
FesteHöhe	Boolean	⇐	Wahr = Feste Höhe, Falsch = Variable Höhe
Titel	Text	⇐	Titel des Formularfensters

Beschreibung

Der Befehl **FORM GET PROPERTIES** gibt die Eigenschaften des Formulars *FormularName* zurück.
Die Parameter *Breite* und *Höhe* geben die Breite und Höhe des Formulars in Pixel zurück. Diese Werte werden über die Größenangaben des Standardfensters für das Formular festgelegt:

- Ist die Formulargröße **automatisch**, werden Breite und Höhe so kalkuliert, dass alle Formularobjekte sichtbar sind. Dabei werden der angegebene horizontale und vertikale Rand berücksichtigt.
- Ist eine bestimmte Formulargröße **gesetzt**, richten sich Breite und Höhe nach den in den jeweiligen Bereichen manuell eingegebenen Werten.
- Ist die Formulargröße **objektbezogen**, werden Breite und Höhe in Bezug auf die Position dieses Objektes berechnet.

Der Parameter *AnzSeiten* gibt die Anzahl der Seiten im Formular zurück, mit Ausnahme der Seite 0 (Null). Die Parameter *FesteBreite* und *FesteHöhe* geben an, ob das Formular in Breite und Höhe veränderbar ist (der Parameter gibt **Falsch** zurück) oder nicht veränderbar (der Parameter gibt **Wahr** zurück).
Der Parameter *Titel* gibt den Titel des Formularfensters zurück, wie in der Eigenschaftenliste des Formulareditors definiert wurde. Wurde kein Name festgelegt, gibt er einen leeren String zurück.

FORM GET VERTICAL RESIZING

FORM GET VERTICAL RESIZING (Anpassen {; minHöhe {; maxHöhe}})

Parameter	Typ		Beschreibung
Anpassen	Boolean	⇐	Wahr: Formular ist vertikal anpassbar, Falsch: Formular ist nicht vertikal anpassbar
minHöhe	Lange Ganzzahl	⇐	Kleinstmögliche Formularhöhe (Pixel)
maxHöhe	Lange Ganzzahl	⇐	Größtmögliche Formularhöhe (Pixel)

Beschreibung

Der Befehl **FORM GET VERTICAL RESIZING** gibt die vertikalen Anpassungseigenschaften des aktuellen Formulars in den Variablen *Anpassen*, *minHöhe* und *maxHöhe* zurück. Diese Eigenschaften können für das Formular im Formulareditor des Designmodus oder für den aktuellen Prozess über den Befehl **FORM SET VERTICAL RESIZING** gesetzt worden sein.

FORM GOTO PAGE

FORM GOTO PAGE (SeitenNr {; *})

Parameter	Typ		Beschreibung
SeitenNr	Lange Ganzzahl	➡	Nummer der anzuzeigenden Formularseite
*	Operator	➡	Seite des aktuellen Unterformulars ändern

Beschreibung

Der Befehl **FORM GOTO PAGE** wechselt von der aktuellen Seite eines Formulars auf die Seite mit der Nummer *SeitenNr*.

Wird kein Formular angezeigt bzw. über den Befehl **FORM LOAD** geladen, oder entspricht *SeitenNr* der aktuellen Seite des Formulars, hat **FORM GOTO PAGE** keine Auswirkung. Ist *SeitenNr* größer als die Anzahl der Seiten, wird die letzte Seite angezeigt. Ist *SeitenNr* kleiner als Eins, wird die erste Seite angezeigt.

Der Parameter * ist hilfreich, wenn der Befehl im Rahmen einer Seite vom Typ Unterformular mit mehreren Seiten aufgerufen wird. Mit * ändert der Befehl die Seite des aktuellen Unterformulars, d.h. das den Befehl aufgerufen hat. Standardmäßig, also ohne *, wird der Befehl immer auf das Elternformular angewandt.

Über Befehle zur Verwaltung von Seiten

Schaltflächen mit automatischen Aktionen führen dieselben Tasks wie die Befehle **FORM FIRST PAGE**, **FORM LAST PAGE**, **FORM NEXT PAGE**, **FORM PREVIOUS PAGE** und **FORM GOTO PAGE** durch. Diese können Sie für Registerkarten, Dropdown-Listen, usw. verwenden. Wir empfehlen, statt Befehlen vorrangig Schaltflächen mit automatischen Aktionen zu verwenden.

Befehle für Seiten können Sie in Eingabeformularen oder in Formularen innerhalb von Dialogen verwenden. Ausgabeformulare arbeiten nur mit der ersten Seite. Ein Formular hat mindestens eine Seite – die erste Seite. Beachten Sie, dass für ein Formular – unabhängig von seiner Seitenanzahl – immer nur eine Formularmethode existiert.

- Über die Funktion **FORM Get current page** können Sie herausfinden, welche Seite angezeigt wird
- Verwenden Sie das Formularereignis On Page Change, das jedes Mal, wenn sich die aktuelle Seite des Formulars ändert, generiert wird

Hinweis: Wenn Sie ein Formular einrichten, können Sie mit den Seiten 1 bis N arbeiten, und mit der Seite 0. Hier setzen Sie Objekte, die auf allen Seiten erscheinen sollen.
Die Seitenbefehle rufen im Formular nur die Seiten 1 bis N auf; Seite Null (0) wird automatisch mit der angezeigten Seite kombiniert.

Beispiel

Folgendes Beispiel ist eine Objektmethode für eine Schaltfläche. Sie zeigt eine spezifische Seite, hier Seite 3 an:

```
FORM GOTO PAGE(3)
```

FORM LAST PAGE

FORM LAST PAGE
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **FORM LAST PAGE** wechselt von der aktuellen Seite eines Formulars auf die letzte Seite. Wird das Formular nicht angezeigt bzw. über den Befehl **FORM LOAD** geladen, oder ist die angezeigte Seite bereits die letzte Seite des Formulars, wird der Befehl nicht ausgeführt.

Beispiel

Folgendes Beispiel ist eine einzeilige Methode, die über Menübefehl aufgerufen wird. Sie zeigt die letzte Seite des Formulars an:

```
FORM LAST PAGE
```

FORM LOAD

FORM LOAD ({Tabellenname ;} Formularname {; *})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Zu ladendes Tabellenformular (wenn weggelassen, wird Projektformular geladen)
Formularname	String, Objekt	➡	Name (String) des Projekt- oder Tabellenformulars oder POSIX Pfad (String) zu einer .json Datei mit Beschreibung des Formulars oder Objekt mit Beschreibung des zu öffnenden Formulars
*	Operator	➡	Mit * wird der Befehl auf Host Datenbank angewandt, wenn er von einer Komponente ausgeführt wird (außerhalb dieses Kontexts wird der Parameter ignoriert)

Beschreibung

Der Befehl **FORM LOAD** lädt das Formular *Formularname* (Projekt oder Tabelle) im Speicher in den aktuellen Prozess, um seine Daten zu drucken oder den Inhalt zu analysieren . Es gibt immer nur ein aktuelles Formular pro Prozess.

Im Parameter *Formularname* übergeben Sie:

- Den Namen des Formulars oder
- Den Pfad (in POSIX Syntax) zu einer gültigen .json Datei mit der Beschreibung des Formulars (siehe **Dateipfade für Formulare**) oder
- Ein Objekt mit der Beschreibung des Formulars. Weitere Informationen dazu finden Sie unter **Dynamische Formulare**.

Daten mit diesem Formular drucken

Um diesen Befehl auszuführen, muss zuvor mit dem Befehl **OPEN PRINTING JOB** ein Druckauftrag geöffnet werden. Das geladene Formular wird zum aktuellen Druckformular. Alle Befehle zur Objektverwaltung sowie die Funktion **Print object** arbeiten mit diesem Formular.

Wurde bereits zuvor ein Druckformular geladen (über einen früheren Aufruf von **FORM LOAD**), wird es geschlossen und durch *Formularname* ersetzt. Sie können in der gleichen Drucksitzung verschiedene Projektformulare öffnen und schließen. Beim Ändern des Druckformulars über **FORM LOAD** werden keine Umbrüche generiert. Der Entwickler muss selbst die Seitenumbrüche verwalten. Übergeben Sie einen leeren String in *Formularname*, wird das aktuelle Projektformular zum Drucken geschlossen.
Nur das Formularereignis On Load wird beim Öffnen des Projektformulars ausgeführt. Die anderen Formularereignisse werden ignoriert. Das Formularereignis On Unload wird am Ende des Druckens ausgeführt.

Um die grafische Konsistenz von Formularen beizubehalten, empfehlen wir, die Darstellungseigenschaft "Drucken" unabhängig von der Plattform anzuwenden.

Das aktuelle Druckformular wird automatisch geschlossen, wenn der Befehl **CLOSE PRINTING JOB** aufgerufen wird.

Hinweis zur Kompatibilität: Früher hieß dieser Befehl OPEN PRINTING FORM und akzeptierte im Parameter *Formularname* auch einen leeren String, um das aktuelle Projektformular zu schließen. Diese Syntax wird nicht mehr unterstützt, sie gibt einen Fehler zurück. Verwenden Sie jetzt zum Schließen des Formulars die Befehle **FORM UNLOAD** oder **CLOSE PRINTING JOB**.

Inhalt des Formulars analysieren (parsen)

Hier wird ein off-screen Formular zur Analyse des Inhalts geladen. Dazu rufen Sie **FORM LOAD** außerhalb eines Druckauftrags auf. In diesem Anwendungsfall werden die Formularereignisse nicht ausgeführt.

FORM LOAD lässt sich zusammen mit den Befehlen **FORM GET OBJECTS** und **OBJECT Get type** verwenden, um den Inhalt des Formulars abzufragen. Sie müssen zum Schluss den Befehl **FORM UNLOAD** aufrufen, um das Formular aus dem Speicher zu entfernen.

Beachten Sie, dass ein bereits angezeigtes Formular auf dem Bildschirm dabei geladen bleibt (es ist vom Befehl **FORM LOAD** nicht betroffen), es muss also nach Aufruf von **FORM UNLOAD** nicht erneut geladen werden.

Wird der Befehl über eine Komponente ausgeführt, lädt er standardmäßig die Formulare der Komponente. Mit dem Parameter * lädt die Methode die Formulare der Host Datenbank.

Zur Erinnerung: Damit es zu keiner Speicherüberlastung kommt, sollten Sie im im off-screen Kontext den Befehl **FORM UNLOAD** aufrufen.

Beispiel 1

Ein Projektformular in einem Druckauftrag aufrufen:

```
OPEN PRINTING JOB
FORM LOAD("Druckformular")
  // Ereignisse und Objektmethoden ausführen
```

Beispiel 2

Ein Tabellenformular in einem Druckauftrag aufrufen:

```
OPEN PRINTING JOB
FORM LOAD([People];"Druckformular")
  // Ereignisse und Objektmethoden ausführen
```

Beispiel 3

Inhalt des Formulars analysieren (parsen), um Eingabebereiche für Text zu bearbeiten:

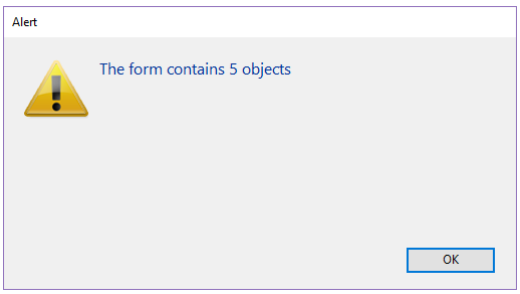
```
FORM LOAD([People];"my_form")
 // Formular ohne Ausführen von Ereignissen oder Methoden auswählen
FORM GET OBJECTS(arrObjNames;arrObjPtrs;arrPages;*)
For($i;1;Size of array(arrObjNames))
   If(OBJECT Get type(*;arrObjNames{$i})=Object type text input)
 //··· Bearbeitungen
   End if
End for
FORM UNLOAD  //Nicht vergessen, das Formular wieder zu entladen
```

Beispiel 4

Dieses Beispiel gibt die Anzahl der Objekte in einem JSON Formular zurück:

```
ARRAY TEXT(objectsArray;0)  //sortiert Formulareinträge in Arrays
ARRAY POINTER(variablesArray;0)
ARRAY INTEGER(pagesArray;0)

FORM LOAD("/RESOURCES/OutputForm.json")  //lädt das Formular
FORM GET OBJECTS(objectsArray;variablesArray;pagesArray;Form all pages+Form inherited)

ALERT("The form contains "+String(size of array(objectsArray))+" objects")  //gibt die Anzahl Obhekte zurück
```

Das Ergebnis lautet:



FORM NEXT PAGE

FORM NEXT PAGE
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **FORM NEXT PAGE** wechselt von der aktuellen Seite eines Formulars zur nächsten Seite. Wird kein Formular angezeigt bzw. über den Befehl **FORM LOAD** geladen, oder ist die angezeigte Seite bereits die letzte Seite des Formulars, wird der Befehl nicht ausgeführt.

Beispiel

Folgendes Beispiel ist eine einzeilige Methode, die über Menübefehl aufgerufen wird. Sie zeigt die nächste Seite des Formulars an, d.h. die Seite, die auf die aktuell angezeigte Seite folgt:

```
FORM NEXT PAGE
```

FORM PREVIOUS PAGE

FORM PREVIOUS PAGE
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **FORM PREVIOUS PAGE** wechselt von der aktuellen Seite eines Formulars zur vorherigen Seite. Wird kein Formular angezeigt bzw. vom Befehl **FORM LOAD** geladen oder ist die angezeigte Seite bereits die erste Seite des Formulars, wird der Befehl nicht ausgeführt.

Beispiel

Folgendes Beispiel ist eine einzeilige Methode, die über Menübefehl aufgerufen wird. Sie zeigt die vorige Seite des Formulars an, d.h. die Seite, die vor der aktuell angezeigten Seite liegt:

```
FORM PREVIOUS PAGE
```

FORM SCREENSHOT

FORM SCREENSHOT ({{TabellenName ;} FormularName ;} FormularBild {; SeitenNr})

Parameter	Typ		Beschreibung
TabellenName	Tabelle	➡	Tabelle zum Formular
FormularName	Text	➡	Name des Formulars
FormularBild	Bild	⬅	Ohne die beiden ersten Parameter: Bild des gerade ausgeführten Formulars Mit Formularname: Bild des Formulars im Formulareditor
SeitenNr	Lange Ganzzahl	➡	Seitennummer des Formulars

Beschreibung

Der Befehl **FORM SCREENSHOT** gibt ein Formular als Bild zurück. Dieser Befehl erlaubt zwei Syntaxarten: Je nach der verwendeten Syntax erhalten Sie entweder ein Bild des ausgeführten Formulars oder ein Bild des Formulars im Formulareditor.

- **FORM SCREENSHOT** (*FormularBild*)
  Diese Syntax erhält ein Screenshot der aktuellen Seite des Formulars, das gerade ausgeführt oder über den Befehl **FORM LOAD** geladen wird: Das Bild, das im Parameter *FormularBild* zurückgegeben wird, enthält alle sichtbaren Objekte des Formulars mit den aktuellen Werten von Feld und Variable des Formulars, Unterformulars, etc. Das Formular wird komplett zurückgegeben, ohne Berücksichtigung der Größe des Fensters, in dem das Formular enthalten ist.
  Beachten Sie, dass diese Syntax nur mit Eingabeformularen funktioniert.

- **FORM SCREENSHOT** ({*TabellenName* ;} *FormularName* ; *FormularBild* {; *SeitenNr*})
  Diese Syntax erhält ein Screenshot einer sog. Formularvorlage, wie sie im Formulareditor angezeigt wird. Alle sichtbaren Objekte werden so gezeichnet, wie sie im Editor vorkommen; diese Syntax berücksichtigt auch vererbte Formulare und Objekte, die auf Seite 0 liegen.
  Wollen Sie ein Abbild für ein Tabellenformular, übergeben Sie im Parameter *TabellenName* die Formulartabelle und dann ihren Namen als String in *FormularName*. Für ein Projektformular übergeben Sie den Formularnamen direkt in *FormularName*.
  Der Befehl gibt standardmäßig ein Screenshot von Seite 1 des Formulars zurück. Wollen Sie nur ein Bild von Seite 0 oder von einer anderen Seite des Formulars, übergeben Sie die gewünschte Seitennummer im Parameter *SeitenNr*.

Hinweise:

- Web Areas werden im zurückgegebenen Screenshot nicht gerendert.
- Da die beiden ersten Parameter dieses Befehls optional sind, können Sie eine Funktion, die einen Zeiger zurückgibt, wie z.B. **Current form table**-> oder **Table**->, nicht direkt als Argument übergeben. Diese Syntax funktioniert im interpretierten Modus, wird aber beim Kompilieren abgewiesen. Hier müssen Sie stattdessen eine dazwischengesetzte Zeiger-Variable verwenden. Weitere Information dazu finden Sie im Abschnitt **Befehle mit Zeigern direkt verwenden**.

FORM SET ENTRY ORDER

FORM SET ENTRY ORDER (ObjektNamen {; SeitenNummer})

Parameter	Typ		Beschreibung
ObjektNamen	Array Text	➡	Array mit Objektnamen in vorgesehener Reihenfolge
SeitenNummer	Lange Ganzzahl	➡	Nummer der Seite zum Setzen der Eingabefolge (ohne Angabe aktuelle Seite)

Beschreibung

Der Befehl **FORM SET ENTRY ORDER** setzt die Eingabereihenfolge des aktuellen Formulars für den aktuellen Prozess gemäß dem Array *ObjektNamen*.

In *ObjektNamen* übergeben Sie ein Array mit den Namen der Formularobjekte für die Eingabereihenfolge. Die Reihenfolge der Objekte im Array bestimmt die Eingabefolge im Formular. Alle gültigen Formularobjekte im aktuellen Formular lassen sich auflisten. Ein Objekt ist gültig, wenn es:

- die Eigenschaft fokusfähig hat(**Hinweis:** Der Befehl ignoriert die Objekteigenschaft **Tabfähig**)
- im Formular existiert, d.h. sein Name definiert ist
- auf der aktuellen Seite verwendet wird, bzw. auf der Seite *SeitenNr* (siehe unten). Beachten Sie, dass eine Formularseite auch Objekte auf Seite 0 und vererbte Objekte enthält.

Wird in Echtzeit ein ungültiges Objekt gefunden, wird es einfach ignoriert und 4D geht im Array *ObjektNamen* zum nächsten gültigen Objekt. Um die aktuelle Eingabefolge der aktuellen Seite, basierend auf gültigen Objekten zu erfahren, verwenden Sie den Befehl **FORM GET ENTRY ORDER** mit dem Parameter *.

Optional können Sie in *SeitenNr* eine bestimmte Seite für die Eingabefolge angeben. Ohne diesen Parameter verwendet der Befehl die aktuelle Seite.

Hinweise:

- Die Eingabereihenfolge eines Unterformulars wird im Unterformular selbst defniert. Sie müssen dafür **FORM SET ENTRY ORDER** im Kontext des Unterformulars aufrufen.
- Dieser Befehl definiert nicht das erste Objekt mit Fokus im Formular in Echtzeit. Dazu müssen Sie den Befehl **GOTO OBJECT** im Ereignis On Load des Formulars verwenden. Mit dem Befehl **OBJECT DUPLICATE** können Sie das duplizierte Objekt als das erste setzen, wenn Sie im Parameter *GehenZu* die Konstante Object First in entry order setzen.

Über die Eingabereihenfolge von Daten
Die Eingabefolge der Daten ist die Reihenfolge, in der Felder, Unterformulare und alle anderen aktiven Objekte ausgewählt werden, wenn der Benutzer im Formular die **Tabulatortaste** oder die **Zeilenschaltung** drückt. Mit der Tastenkombination **Shift+Tab** oder **Shift+Zeilenschaltung** läuft die Reihenfolge in umgekehrter Richtung. Die Eingabereihenfolge lässt sich im Formulareditor standardmäßig setzen oder ändern. Weitere Informationen dazu finden Sie im Abschnitt **Eingabereihenfolge der Daten ändern** des Handbuchs *4D Designmodus*.

Beispiel

Die Eingabereihenfolge von Objekten im Formular basierend auf ihren Namen setzen:

```
ARRAY TEXT(tabNames;0)

FORM GET OBJECTS(tabNames;Form current page+Form inherited)  //Namen der Formularobjekte erhalten
SORT ARRAY(tabNames;>)  //Namen in aufsteigender Reihenfolge sortieren
FORM SET ENTRY ORDER(tabNames)  //alphabetische Reihenfolge für die Eingabefolge verwenden
 //nicht-fokusfähige Objekte werden ignoriert
```

FORM SET HORIZONTAL RESIZING

FORM SET HORIZONTAL RESIZING (Anpassen {; minBreite {; maxBreite}})

Parameter	Typ		Beschreibung
Anpassen	Boolean	➡	Wahr: Formular lässt sich horizontal anpassen Falsch: Formular lässt sich nicht horizontal anpassen
minBreite	Lange Ganzzahl	➡	Kleinste zugelassene Formularbreite (Pixel)
maxBreite	Lange Ganzzahl	➡	Größte zugelassene Formularbreite (Pixel)

Beschreibung

Der Befehl **FORM SET HORIZONTAL RESIZING** verändert die Eigenschaften zur horizontalen Anpassung des aktuellen Formulars per Programmierung. Diese Eigenschaften werden standardmäßig im Formulareditor der Designumgebung eingestellt. Die neuen Eigenschaften gelten nur für den aktuellen Prozess, sie werden nicht mit dem Formular gespeichert.

Der Parameter *Anpassen* legt fest, ob das Formular horizontal anpassbar ist, d.h. ob die Breite veränderbar ist – manuell vom Benutzer oder per Programmierung.
Bei *Wahr* kann der Benutzer die Formularbreite verändern; 4D verwendet als Marken die in *minBreite* und *maxBreite* übergebenen Werte.
Bei *Falsch* lässt sich die aktuelle Formularbreite nicht verändern; in diesem Fall müssen in den Parametern *minBreite* und *maxBreite* keine Werte eingetragen werden.
Haben Sie *Wahr* im ersten Parameter übergeben, können Sie in den optionalen Parametern *minBreite* und *maxBreite* neue Mindest- und Maximumwerte in Pixel eintragen. Geben Sie hier keine Werte an, gelten die Werte aus der Designumgebung – sofern vorhanden.

Beispiel

Siehe Beispiel zum Befehl **FORM SET SIZE**.

FORM SET INPUT

FORM SET INPUT ({Tabellenname ;} Formularname {; Benutzerformular {; *}})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, in der das Eingabeformular geändert werden soll Ohne Angabe Haupttabelle
Formularname	String, Objekt	➡	Name (String) des Tabellenformulars, oder POSIX Pfad (String) zu einer .json Datei mit Beschreibung des Formulars, oder Objekt mit Beschreibung des Formulars.
Benutzerformular	String	➡	Name des zu verwendenden Benutzerformulars
*		➡	Automatische Fenstergröße

Beschreibung

Der Befehl**FORM SET INPUT** deklariert *Formularname* bzw. *Benutzerformular* als aktuelles Eingabeformular. Das Formular muss zu *Tabellenname* gehören.

Die Reichweite dieses Befehls ist der aktuelle Prozess. Jede Tabelle hat in jedem Prozess ein eigenes Eingabeformular.

Im Parameter *Formularname* können Sie folgendes übergeben:

- Name des Formulars
- Pfad (in POSIX Syntax) zu einer gültigen .json Datei mit der Beschreibung des Formulars. Siehe **Dateipfade für Formulare**;
- Objekt mit der Beschreibung des Formulars. Weitere Informationen dazu finden Sie unter **Dynamische Formulare**

Hinweis: Dieser Befehl ist aus strukturellen Gründen nicht kompatibel mit Projektformularen.

Das Standardeingabeformular definieren Sie im Explorerfenster für jede Tabelle. Sie erkennen es am Buchstaben E, der dem Formularnamen gegenübersteht. 4D nimmt dieses Formular als Eingabeformular, wenn von **FORM SET INPUT** kein anderes Formular bestimmt wird oder ein Formular angegeben ist, das nicht existiert.

Mit dem optionalen Parameter *Benutzerformular* können Sie ein Benutzerformular als standardmäßiges Eingabeformular festlegen. Übergeben Sie einen gültigen Namen, wird dieses Formular standardmäßig statt des Eingabeformulars im aktuellen Prozess verwendet. Auf diese Weise können Sie gleichzeitig mehrere verschiedene Benutzerformulare haben, die mit dem Befehl **CREATE USER FORM** erstellt wurden, und je nach Kontext das passende Formular verwenden.
Weitere Informationen zu Benutzerformularen finden Sie im Abschnitt **Einführung in Benutzerformulare**.

Eine Reihe von Befehlen zeigen ein Eingabeformular an, um dem Benutzer zu ermöglichen, neue Daten einzugeben oder bestehende Daten zu ändern. Folgende Befehle zeigen ein Eingabeformular für Dateneingaben und Suchläufe an:

- **ADD RECORD**
- **DISPLAY RECORD**
- **MODIFY RECORD**
- **QUERY BY EXAMPLE**

Die Befehle **DISPLAY SELECTION** und **MODIFY SELECTION** zeigen die Daten als Liste im Ausgabeformular. Durch Doppelklick auf einen Datensatz wird das Eingabeformular angezeigt.

Die Befehle **IMPORT TEXT**, **IMPORT SYLK** und **IMPORT DIF** verwenden das aktuelle Eingabeformular für den Import von Datensätzen.

Der optionale Parameter * wird zusammen mit den in der Designumgebung festgelegten Formulareigenschaften und der Funktion **Open window** verwendet. Geben Sie den Parameter * an, wird das Fenster beim nächsten Aufrufen des Formulars (als Eingabeformular oder Dialogbox) automatisch angepasst. Weitere Informationen dazu finden Sie unter der Funktion **Open window**.

Hinweis: FORM SET INPUT ändert – egal, ob mit und ohne optionalen Parameter * – in jedem Fall das Eingabeformular für die Tabelle.

Beispiel 1

Folgendes Beispiel zeigt eine typische Verwendung von **FORM SET INPUT**:

```
FORM SET INPUT([Companies];"Neue Firma")  ` Formular zum Hinzufügen neuer Firmen
ADD RECORD([Companies])  ` Neue Firma hinzufügen
```

Beispiel 2

In einer Rechnungsdatenbank mit verschiedenen Firmen muss die Rechnung mit einem entsprechenden Benutzerformular erstellt werden:

```
Case of
   :(company="ACE")
      FORM SET INPUT([Invoices];"Eingabe";"ACE")
   :(company="Kranz GmbH")
      FORM SET INPUT([Invoices];"Eingabe";"Kranz GmbH")
   :(company="Merl-Software")
      FORM SET INPUT([Invoices];"Eingabe";"Merl-Software")
```

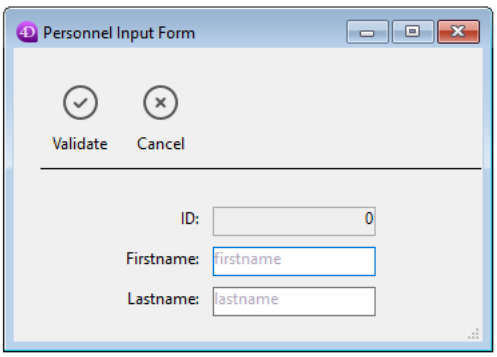
```
End Case
ADD RECORD([Invoices])
```

Beispiel 3

Dieses Beispiel verwendet eine .json Datei mit der Beschreibung des Formulars zur Eingabe der Datensätze in eine Liste der Angestellten:

```
FORM SET INPUT([Personnel];"/RESOURCES/PersonnelForm.json")
ADD RECORD([Personnel])
```

Das ergibt folgendes Formular:



FORM SET OUTPUT

FORM SET OUTPUT ({Tabellenname ;} Formularname {; Benutzerformular})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Setzen des Ausgabeformulars, ohne Angabe Standardformular
Formularname	String, Objekt	➡	Name (String) des Tabellenformulars, oder POSIX Pfad (String) zu einer .json Datei mit Beschreibung des Formulars, oder Objekt mit Beschreibung des Formulars.
Benutzerformular	String	➡	Name des zu verwendenden Benutzerformulars

Beschreibung

Der Befehl **FORM SET OUTPUT** setzt das aktuelle Ausgabeformular in *Formularname* oder *Benutzerformular*. Das Formular muss zu *Tabellenname* gehören.

Die Reichweite dieses Befehls ist der aktuelle Prozess. Jede Tabelle hat in jedem Prozess ein eigenes Ausgabeformular.

Im Parameter *Formularname* können Sie folgendes übergeben:

- Name des Formulars
- Pfad (in POSIX Syntax) zu einer gültigen .json Datei mit der Beschreibung des Formulars. Siehe **Dateipfade für Formulare**;
- Objekt mit der Beschreibung des Formulars. Weitere Informationen dazu finden Sie unter **Dynamische Formulare**.

Hinweis: FORM SET OUTPUT ist aus strukturellen Gründen nicht mit Projektformularen kompatibel.

FORM SET OUTPUT zeigt nicht das Formular an; es bestimmt lediglich, welches Formular gedruckt, angezeigt oder von einem anderen Befehl verwendet wird. Weitere Informationen zum Erstellen von Formularen finden Sie im Handbuch *4D Designmodus*.

Das Standardausgabeformular für jede Tabelle definieren Sie im Explorer der Designumgebung. Sie erkennen es am Buchstaben A, der dem Formularnamen gegenübersteht. 4D nimmt dieses Formular als Ausgabeformular, wenn kein anderes Formular von **FORM SET OUTPUT** bestimmt wird oder ein Formular angegeben ist, das nicht existiert.

Mit dem optionalen Parameter *Benutzerformular* können Sie ein Benutzerformular als standardmäßiges Ausgabeformular festlegen. Übergeben Sie einen gültigen Namen, wird dieses Formular standardmäßig statt des Ausgabeformulars im aktuellen Prozess verwendet. Auf diese Weise können Sie gleichzeitig mehrere verschiedene Benutzerformulare haben, die mit dem Befehl **CREATE USER FORM** erstellt wurden, und je nach Kontext das passende Formular verwenden.
Weitere Informationen zu Benutzerformularen finden Sie im Abschnitt **Einführung in Benutzerformulare**.

Ausgabeformulare werden von drei Befehlsgruppen verwendet: Die erste Gruppe zeigt eine Liste von Datensätzen auf dem Bildschirm an, die zweite erstellt Berichte, die dritte exportiert Daten. Die Befehle **DISPLAY SELECTION** und **MODIFY SELECTION** zeigen die Datensätze als Liste im Ausgabeformular an. Verwenden Sie zum Erstellen von Berichten mit den Befehlen **PRINT LABEL** und **PRINT SELECTION** ein Ausgabeformular. Die Befehle **EXPORT DIF**, **EXPORT SYLK** und **EXPORT TEXT** verwenden ebenfalls das Ausgabeformular für den Export.

Beispiel 1

Folgendes Beispiel zeigt eine typische Verwendung von **FORM SET OUTPUT**. Hier steht **FORM SET OUTPUT** direkt vor dem Aufruf des Ausgabeformulars. Das ist jedoch nicht unbedingt erforderlich. Der Befehl kann auch in einer ganz anderen Methode ausgeführt werden, so lange er vor dieser Methode ausgeführt wird:

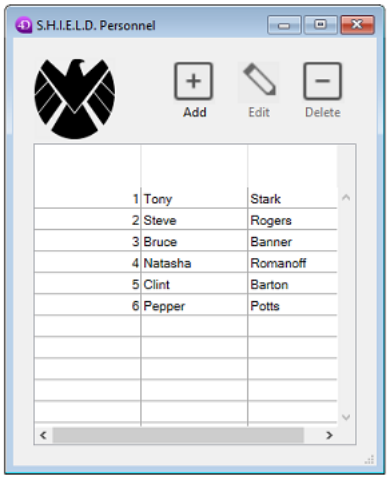
```
FORM SET INPUT([Parts];"Artikel eingeben")  //Wähle das Eingabeformular
FORM SET OUTPUT([Parts];"Artikelliste")  //Wähle das Ausgabeformular
MODIFY SELECTION([Parts])  //Dieser Befehl verwendet beide Formulare
```

Beispiel 2

Dieses Beispiel verwendet den Pfad eines .json Formulars zum Drucken der Datensätze in einer Liste der Angestellten:

```
FORM SET OUTPUT([Personnel];"/RESOURCES/PersonnelPrintForm.json")
ALL RECORDS([Personnel])
PRINT SELECTION([Personnel])
```

Das druckt folgendes Formular:


S.H.I.E.L.D. Personnel
Add
Edit
Delete
1 Tony Stark
2 Steve Rogers
3 Bruce Banner
4 Natasha Romanoff
5 Clint Barton
6 Pepper Potts

FORM SET SIZE

FORM SET SIZE ({Objekt ;} horizontal ; vertikal {; *})

Parameter	Typ		Beschreibung
Objekt	String	➡	Objektname mit Angabe der Formulargrenzen
horizontal	Lange Ganzzahl	➡	Mit *: Horizontaler Rand (Pixel) Ohne *: Breite (Pixel)
vertikal	Lange Ganzzahl	➡	Mit *: Vertikaler Rand (Pixel) Ohne *: Höhe (Pixel)
*	Operator	➡	Mit *: Fügt Formularränder hinzu, definiert durch Horizontal und Vertikal Ohne Stern: Verwendet Horizontal und Vertikal als Breite und Höhe des Formulars. Gilt nicht für objektbasierte Größe.

Beschreibung

Der Befehl **FORM SET SIZE** ändert die Größe des aktuellen Formulars per Programmierung. Die neue Größe wird für den aktuellen Prozess definiert; sie wird nicht mit dem Formular gespeichert. Analog zur Designumgebung können Sie diesen Befehl zum Setzen der Formulargröße folgendermaßen verwenden:

- Automatisch — 4D bestimmt die Größe des Formulars so, dass alle Objekte sichtbar sind — und fügt evtl. einen horizontalen und vertikalen Rand hinzu.
- An der Stelle, wo ein Formularobjekt gefunden wird, dem ein horizontaler und vertikaler Rand hinzugefügt werden kann.
- Durch Eingeben "fester" Größen (Breite und Höhe).
- Automatische Größe

Weitere Informationen dazu finden Sie im Abschnitt **Formulargröße** des Handbuchs *4D Designmodus*.

Automatische Größe
Um die Größe des Formulars automatisch zu setzen, verwenden Sie folgende Syntax:

```
FORM SET SIZE(horizontal;vertikal;*)
```

In diesem Fall übergeben Sie in *horizontal* und *vertikal* die Ränder in Pixel, die Sie im Formular rechts und unten hinzufügen wollen.

Objektbasierte Größe
Um die Größe des Formulars objektbezogen zu setzen, verwenden Sie folgende Syntax:

```
FORM SET SIZE(Objekt;horizontal;vertikal)
```

In diesem Fall müssen Sie in *horizontal* und *vertikal* die Ränder in Pixel übergeben, die Sie dem Objekt rechts und unten hinzufügen wollen. Sie können nicht den Parameter * übergeben.

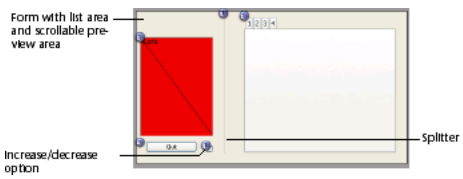
Feste Größe
Um die Größe des Formulars mit fester Größe einzurichten, verwenden Sie folgende Syntax:

```
FORM SET SIZE(horizontal;vertikal)
```

In diesem Fall übergeben Sie in *horizontal* und *vertikal* die Breite und Höhe (in Pixel) des Formulars.
FORM SET SIZE verändert die Größe des Formulars, berücksichtigt aber auch die Eigenschaften zum Anpassen. Ist z.B. für ein Formular als Mindestbreite 500 Pixel angegeben und setzt der Befehl die Breite auf 400 Pixel, wird die neue Formularbreite auf 500 Pixel gesetzt.
Beachten Sie auch, dass dieser Befehl nicht die Größe des Formularfensters verändert. (Sie können ein Formular ohne Verändern der Größe und umgekehrt anpassen). Um die Größe des Formulars anzupassen, verwenden Sie den Befehl **RESIZE FORM WINDOW**.

Beispiel

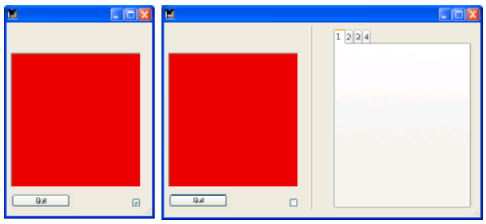
Folgendes Beispiel zeigt, wie ein Fenster vom Typ Explorer eingerichtet wird. In der Designumgebung wird folgendes Formular erstellt:



Die Größe des Formulars ist "automatisch".
Das Fenster wird über folgenden Code angezeigt:

```
$ref:=Open form window([Table 1];"Form1";Plain form window;Horizontally centered;
Vertically centered;*)
DIALOG([Table 1];"Form1")
```

Sie können den rechten Teil des Fensters über das Kontrollkästchen am unteren Rand ein- bzw. ausblenden:



Die dazugehörige Objektmethode lautet:

```
Case of
    :(Form event=On Load)
        C_BOOLEAN(b1;<>zugeklappt)
        C_LONGINT(margin)
        margin:=15
        b1:=<>zugeklappt
        If(<>zugeklappt)
            FORM SET HORIZONTAL RESIZING(False)
            FORM SET SIZE("b1";margin;margin)
        Else
            FORM SET HORIZONTAL RESIZING(True)
            FORM SET SIZE("tab";margin;margin)
        End if

    :(Form event=On click)
        <>collapsed:=b1
        If(b1)
 `zugeklappt
            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+margin;$tf+$b+margin;Current form window)
            FORM SET HORIZONTAL RESIZING(False)
            FORM SET SIZE("b1";margin;margin)

        Else
 `aufgeklappt
            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+margin;$tf+$b+margin;
            Current form window)
            FORM SET HORIZONTAL RESIZING(True)
            FORM SET SIZE("tab";margin;margin)
        End if
End case
```

FORM SET VERTICAL RESIZING

FORM SET VERTICAL RESIZING (Anpassen {; minHöhe {; maxHöhe}})

Parameter	Typ		Beschreibung
Anpassen	Boolean	➡	Wahr: Formular lässt sich vertikal anpassen Falsch: Formular lässt sich nicht vertikal anpassen
minHöhe	Lange Ganzzahl	➡	Kleinste zugelassene Formularhöhe (Pixel)
maxHöhe	Lange Ganzzahl	➡	Größte zugelassene Formularhöhe (Pixel)

Beschreibung

Der Befehl **FORM SET VERTICAL RESIZING** verändert die Eigenschaften zur vertikalen Anpassung des aktuellen Formulars per Programmierung. Diese Eigenschaften werden standardmäßig im Formulareditor der Designumgebung eingestellt. Die neuen Eigenschaften gelten nur für den aktuellen Prozess, sie werden nicht mit dem Formular gespeichert.

Der Parameter *Anpassen* legt fest, ob das Formular vertikal anpassbar ist, d.h. ob die Höhe veränderbar ist – manuell vom Benutzer oder per Programmierung.
Bei *Wahr* kann der Benutzer die Formularhöhe verändern; 4D verwendet als Marken die in *minHöhe* und *maxHöhe* übergebenen Werte.
Bei *Falsch* lässt sich die aktuelle Formularhöhe nicht verändern; in diesem Fall müssen in den Parametern *minHöhe* und *maxHöhe* keine Werte eingetragen werden.

Haben Sie *Wahr* im ersten Parameter übergeben, können Sie in den optionalen Parametern *minHöhe* und *maxHöhe* neue Mindest- und Maximumwerte in Pixel eintragen. Geben Sie hier keine Werte an, gelten die Werte aus der Designumgebung – sofern vorhanden.

Beispiel

Siehe Beispiel zum Befehl **FORM SET SIZE**.

FORM UNLOAD

FORM UNLOAD
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **FORM UNLOAD** entfernt das aktuelle Formular aus dem Speicher, das über den Befehl **FORM LOAD** geladen wurde. Sie müssen diesen Befehl verwenden, wenn **FORM LOAD** außerhalb eines Druckauftrags aufgerufen wird. Beim Drucken wird das aktuelle Formular automatisch wieder geschlossen, wenn der Befehl **CLOSE PRINTING JOB** aufgerufen wird.

Formularereignisse

- 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 -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		Gibt WAHR zurück, wenn die Ausführungsphase eine Aktivierung ist

Beschreibung

Die Funktion **Activated** gibt in der Formularmethode den Wert WAHR zurück, wenn das Fenster mit dem Formular an vorderster Stelle des vordersten Prozesses kommt.

Damit die Ausführungsphase **Activated** generiert wird, müssen Sie zuvor in der Designumgebung die Formulareigenschaft On Activate ausgewählt haben. Das wird beim Konvertieren einer Datenbank automatisch ausgeführt.

Hinweis: Diese Funktion entspricht der Verwendung von **Form event** und Testen, ob sie das Ereignis On Activate zurückgibt.

WARNUNG: Verwenden Sie nicht die Befehle **TRACE** oder **ALERT**in der Phase **Activated** des Formulars, denn das verursacht eine Endlosschleife.

After

After -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Gibt Wahr zurück, wenn die Ausführungsphase AFTER ist.

Beschreibung

After gibt Wahr für die Ausführungsphase After zurück.

Damit die Ausführungsphase **After** generiert wird, müssen Sie zuvor in der Designumgebung für das Formular und/oder die Objekte als Eigenschaft das Ereignis *On Validate* ausgewählt haben.

Hinweis: Diese Funktion entspricht der Verwendung von **Form event** und Testen, ob sie das Ereignis *On Validate* zurückgibt.

Before

Before -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		Gibt Wahr zurück, wenn die Ausführungsphase BEFORE ist.

Beschreibung

Before gibt Wahr für die Ausführungsphase Before zurück.

Damit die Ausführungsphase **Before** generiert wird, müssen Sie zuvor in der Designumgebung für das Formular und/oder die Objekte als Eigenschaft das Ereignis *On Load* ausgewählt haben.

Hinweis: Diese Funktion entspricht der Verwendung von **Form event** und Testen, ob sie das Ereignis *On Load* zurückgibt.

CALL FORM

CALL FORM (Fenster ; Methode {; Param}{; Param2 ; ... ; ParamN})

Parameter	Typ		Beschreibung
Fenster	WinRef	➡	Referenznummer des Fensters
Methode	Text	➡	Name der aufzurufenden Projektmethode
Param	Ausdruck	➡	Parameter zur Methode

Beschreibung

Der Befehl **CALL FORM** führt die Projektmethode mit dem in *Methode* übergebenen Namen aus, mit dem optionalen *Param* im Kontext eines Formulars, das in *Fenster* angezeigt wird, unabhängig vom Prozess, zu dem das Fenster gehört.

Wie bei der auf Workern basierenden Interprozesskommunikation (siehe **Über Worker**), ist dem Fenster eine Nachrichtenbox zugeordnet, die verwendet werden kann, wenn das Fenster ein Formular anzeigt (nach dem Formularereignis On Load). **CALL FORM** bindet Methodenname und Befehlsargumente in eine Nachricht ein, die in die Nachrichtenbox gelegt werden. Das Formular führt die Nachricht dann in einem eigenen Prozess aus.

In *Fenster* übergeben Sie die Referenznummer des Fensters, welches das aufgerufene Formular anzeigt.

In *Methode* übergeben Sie den Namen der Projektmethode, die im Kontext des Elternprozesses für das *Fenster* ausgeführt werden soll.

In *Param* können Sie einen oder mehrere Parameter für die Methode übergeben. Das funktioniert genauso wie Parameter für eine Unterroutine übergeben, siehe Abschnitt **Parameter in Methoden** . Beim Starten der Ausführung im Rahmen des Formulars empfängt die Methode die Parameterwerte in *$1*, *$2*, etc. Beachten Sie auch, dass sich Arrays nicht als Parameter für eine Methode übergeben lassen. Außerdem müssen Sie für den Befehl **CALL FORM** folgendes beachten:

- Zeiger auf Tabellen oder Felder sind erlaubt.
- Zeiger auf Variablen, insbesondere lokale und Prozessvariablen werden nicht empfohlen, da sie zum Zeitpunkt, wo die Prozessmethode auf sie zugreift, undefiniert sein können.
- Übergeben Sie einen Parameter vom Typ Objekt oder Collection, erstellt 4D im Zielprozess eine Kopie des Objekts oder der Collection (anstelle einer Referenz), wenn das Formular nicht in dem Prozess liegt, den der Befehl **CALL FORM** aufruft.

Beispiel 1

Im gleichen Formular zwei unterschiedliche Fenster öffnen (mit anderen Hintergrundfarben und anderen Meldungen), die Nachrichten dann senden und im jeweiligen Dialogfenster anzeigen.

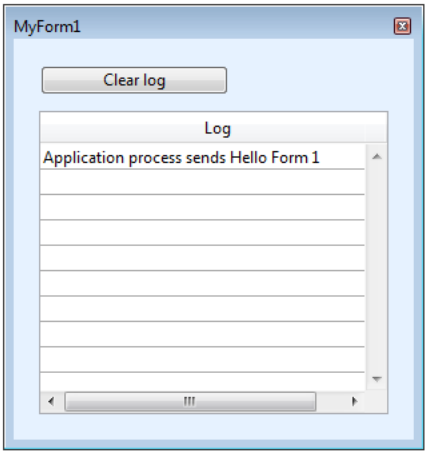
Eine Schaltfläche im Hauptformular öffnet die beiden Dialoge:

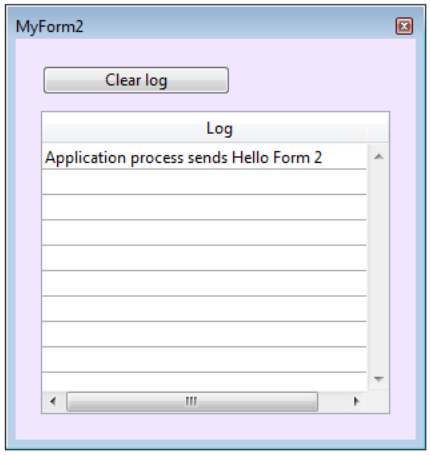
```
 //Objektmethode des Formulars erstellen
 // erstes Fenster
formRef1:=Open form window("FormularNachricht";Palette form window;On the left)
SET WINDOW TITLE("MeinFormular1";formRef1)
DIALOG("FormularNachricht";*)
SHOW WINDOW(formRef1)

 //zweites Fenster
formRef2:=Open form window("FormularNachricht";Palette form window;On the left+500)
SET WINDOW TITLE("MeinFormular2";formRef2)
DIALOG("FormularNachricht";*)
SHOW WINDOW(formRef2)

 //Farben senden
CALL FORM(formRef1;"doSetColor";0x00E6F2FF)
CALL FORM(formRef2;"doSetColor";0x00F2E6FF)
 //Meldung aufbauen
CALL FORM(formRef1;"doAddMessage";Current process name;"Hello Form 1")
CALL FORM(formRef2;"doAddMessage";Current process name;"Hello Form 2")
```

Die Methode *doAddMessage* fügt nur eine Zeile in einer Listbox im Formular "FormularNachricht" hinzu:

```
C_TEXT($1) // Name des Aufrufers (caller)
C_TEXT($2) // Nachricht zum Anzeigen
 // Nachricht von $2 empfangen und die Nachricht in die Listbox setzen
$p:=OBJECT Get pointer(Object named;"Spalte1")
INSERT IN ARRAY($p->;1)
$p->{1}:=$1+" sendet "+$2
```

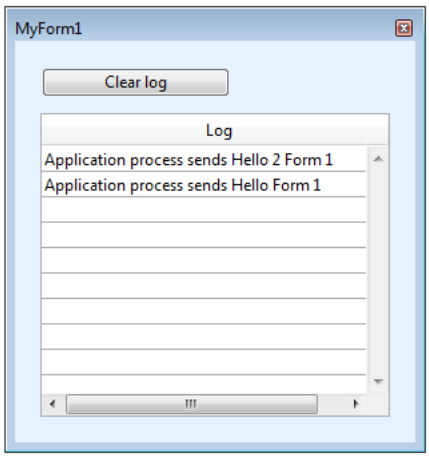
Beim Ausführen erhalten Sie folgendes Ergebnis:

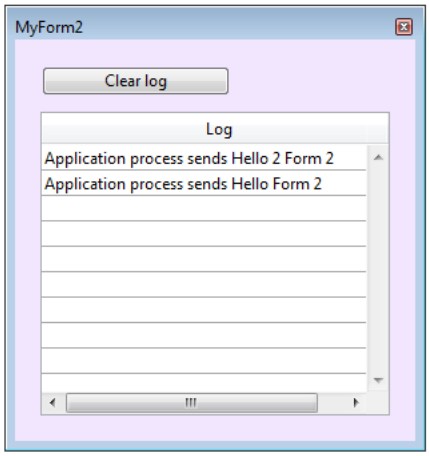




Sie können dann andere Nachrichten hinzufügen. Dazu müssen Sie lediglich den Befehl **CALL FORM** erneut ausführen:

```
CALL FORM(formRef1;"doAddMessage";Current process name;"Hello 2 Form 1")
CALL FORM(formRef2;"doAddMessage";Current process name;"Hello 2 Form 2")
```





Beispiel 2

Über den Befehl **CALL FORM** eigene Einstellungen für ein Formular setzen, z.B. Werte zur Konfiguration - ohne Verwenden von Prozessvariablen:

```
$win:=Open form window("form")
CALL FORM($win;"configure";param1;param2)
DIALOG("form")
```

CALL SUBFORM CONTAINER

CALL SUBFORM CONTAINER (Ereignis)

Parameter	Typ		Beschreibung
Ereignis	Lange Ganzzahl	➡	Zu sendendes Ereignis

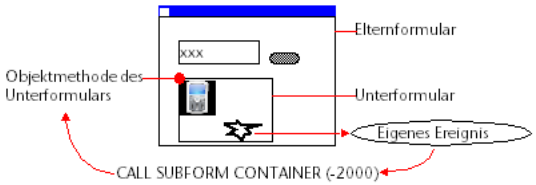
Beschreibung

Der Befehl **CALL SUBFORM CONTAINER** lässt eine Instanz des Unterformulars ein *Ereignis* an das Objekt Unterformular senden, das diese enthält. Das Objekt Unterformular kann dann *Ereignis* im Kontext des Elternformulars bearbeiten.

Dieser Befehl muss in die Formularmethode des Unterformulars oder in die Objektmethode eines Unterformulars gesetzt werden. Das Ereignis wird nur in der Objektmethode des Unterformular Containers empfangen.

Im Parameter *Ereignis* können Sie ein beliebiges vordefiniertes Formularereignis von 4D (Konstanten unter dem Thema **Formularereignisse**) oder einen beliebigen Wert eines eigenen Ereignisses übergeben. Bei vordefinierten Ereignissen muss das Ereignis für das Unterformular markiert sein. Bei eigenen Ereignissen empfehlen wir, einen negativen Wert zu übergeben, um eine Überschneidung mit vorhandenen oder zukünftigen 4D Ereignisnummern zu vermeiden.

Beispiel für Ausführung des Befehls **CALL SUBFORM CONTAINER**:



Clickcount

Clickcount -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl	➲	Anzahl aufeinanderfolgender Klicks

Beschreibung

Die neue Funktion **Clickcount** gibt bei Ereignissen mit Mausklicks an, wie oft der Benutzer mit derselben Maustaste in rascher Abfolge/kurz nacheinander geklickt hat. So gibt sie z.B. bei einem Doppelklick den Wert 2 zurück.

Mit dieser Funktion lassen sich Doppelklicks in Kopf- oder Fußzeilen einer Listbox herausfinden oder Sequenzen von Dreifachklicks oder mehr verwalten.

Jeder Mausklick generiert ein eigenes Klick-Ereignis. Macht der Benutzer z.B. einen Doppelklick, wird für den ersten Klick ein Ereignis gesendet, in dem **Clickcount** 1 zurückgibt. Dann wird ein anderes Ereignis für den zweiten Klick gesendet, in dem **Clickcount** 2 zurückgibt.

Diese Funktion ist nur für die Formularereignisse On Clicked, On Header Click oder On Footer Click verwendbar. Deshalb muss zuvor im Designmodus geprüft werden, ob in den Formulareigenschaften bzw. im spezifischen Objekt das passende Ereignis ausgewählt ist.

Sind beide Formularereignisse On Clicked und On Double Clicked aktiviert, gibt die Funktion **Clickcount** folgendes zurück:

- 1 beim Ereignis On Clicked
- 2 beim Ereignis On Double Clicked
- 2+n beim Ereignis On Clicked

Beispiel 1

Dieser Code könnte im Kopfteil einer Listbox stehen, um einfache und doppelte Klicks zu verwalten:

```
Case of
   :(Form event=On Header Click)
      Case of
         :(Clickcount=1)
            ... //einfacher Klick
         :(Clickcount=2)
            ... //Doppelklick
      End case
End case
```

Beispiel 2

Sie wollen Benutzern erlauben, Etiketten bei Bedarf zu bearbeiten. Sie sind nicht eingebbar, werden aber nach dreifachem Klick eingebbar. Die Objektmethode lautet folgendermaßen:

```
If(Form event=On Clicked)
   Case of
      :(Clickcount=3)
         OBJECT SET ENTERABLE(*;"Etikett";True)
         EDIT ITEM(*;"Bezeichnung")
   End case
End if
```

Contextual click

Contextual click -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Wahr, wenn Kontext-Klick gefunden, wurde, sonst Falsch

Beschreibung

Die Funktion **Contextual click** gibt Wahr zurück, wenn ein Kontext-Klick ausgeführt wurde:

- Unter Windows und auf Mac OS führen Sie einen Kontext-Klick mit der rechten Maustaste aus.
- Auf Mac OS ist auch die Kombination **Ctrl-Taste+Mausklick** möglich. Das gilt insbesondere, wenn keine rechte Maustaste vorhanden ist.

Diese Funktion sollte nur in Zusammenhang mit dem Formularereignis On clicked verwendet werden. Von daher muss im Strukturmodus geprüft werden, ob das Ereignis in den Formulareigenschaften bzw. im entsprechenden Objekt korrekt ausgewählt wurde.

Beispiel

Mit nachfolgender Methode können Sie, in Kombination mit einem rollbaren Bereich, den Wert eines Array-Elements über ein Kontextmenü ändern:

```
If(Contextual click)
   If(Pop up menu("True;False")=1)
      myArray{myArray}:="True"
   Else
      myArray{myArray}:="False"
   End if
End if
```

Deactivated

Deactivated -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Gibt WAHR zurück, wenn die Ausführungsphase eine Deaktivierung ist

Beschreibung

Die Funktion **Deactivated** gibt in einer Formular- oder Objektmethode den Wert WAHR zurück, wenn das vorderste Fenster des vordersten Prozesses mit dem Formular nach hinten gelegt wird.

Damit die Ausführungsphase **Deactivated** generiert wird, müssen Sie zuvor in der Designumgebung für das Formular und/oder die Objekte als Eigenschaft das Ereignis On Deactivate ausgewählt haben.

Hinweis: Diese Funktion entspricht der Verwendung von **Form event** und Testen, ob sie das Ereignis On Deactivate zurückgibt.

EXECUTE METHOD IN SUBFORM

EXECUTE METHOD IN SUBFORM (UnterformularObjekt ; MethodenName {; Rückgabe {; Parameter} {; Parameter2 ; ... ; ParameterN}})

Parameter	Typ		Beschreibung
UnterformularObjekt	Text	➡	Name des Objekts Unterformular
MethodenName	Text	➡	Name der auszuführenden Projektmethode
Rückgabe	Operator, Variable	➡	Von der Methode zurückgegebener Wert oder
		⬅	* wenn Methode keinen Wert zurückgibt.
Parameter	Ausdruck	➡	Parameter, die der Methode übergeben werden sollen

Beschreibung

Der Befehl **EXECUTE METHOD IN SUBFORM** führt die Projektmethode *MethodenName* im Kontext des Objekts *UnterformularObjekt* aus.
Die aufgerufene Projektmethode kann in *Parameter* Werte von 1 bis X empfangen und in Rückgabe einen Wert zurückgeben. Übergeben Sie * im Parameter *Rückgabe*, wenn die Methode keine Parameter zurückgibt.
Sie können den Namen jeder Projektmethode übergeben, die über die Anwendung oder die Komponente, welche den Befehl ausführt, verfügbar ist. Der Kontext der Ausführung wird in der aufgerufenen Methode beibehalten, d.h. das aktuelle Formular sowie Formularereignis bleiben definiert. Kommt das Unterformular von einer Komponente, muss die Methode zur Komponente gehören und die Eigenschaft "von Komponente und Host Datenbank gemeinsam genutzt" haben.

Dieser Befehl muss im Kontext des Elternformulars aufgerufen werden, welches das Objekt *UnterformularObjekt* enthält, z.B. über die Formularmethode.
Hinweis: Die Methode *MethodenName* wird nicht ausgeführt, wenn *UnterformularObjekt* in der aktuellen Seite nicht gefunden wird.

Beispiel 1

Wir gehen aus vom Formular "KundeDetail", das im Elternformular "Firma" als Unterformular verwendet wird. Das Objekt *Unterformular* mit dem Formular *KundeDetail* lautet "KundeUnterformular". Wir wollen nun die Darstellung bestimmter Elemente im Unterformular je nach Wert des Feldes *Firma* verändern: "Kundenname" soll in Rot wechseln, wenn gilt [Firma]Stadt="Berlin" und in Blau, wenn gilt [Firma]Stadt="Frankfurt". Diese Operation wird über die Methode ***SetToColor*** integriert. Um dieses Ergebnis zu erreichen, kann die Methode ***SetToColor*** nicht direkt über das Formularereignis "On Load" des Elternformulars *Firma* ausgeführt werden, da das Objekt "Kundenname" nicht zum aktuellen Formular, sondern zum Objekt Unterformular "KundeUnterformular" gehört. Von daher muss die Methode, damit sie korrekt funktioniert, über den Befehl **EXECUTE METHOD IN SUBFORM** ausgeführt werden.

```
Case of
   :(Form event=On Load)
      Case of
         :([Firma]Stadt ="Berlin")
            $Color:=$Red
         :([Firma]Stadt ="Frankfurt")
            $Color:=$Blue
         Else
            $Color:=$Black
      End case
      EXECUTE METHOD IN SUBFORM("KundeUnterformular";"SetToColor";*;$Color)
End case
```

Beispiel 2

Sie entwickeln eine Datenbank, die als Komponente verwendet wird. Sie enthält ein gemeinsam genutztes Projektformular, z.B. *Kalender*. Es enthält dynamische Variablen sowie eine öffentliche Projektmethode, um den Kalender anzupassen: SetCalendarDate(varDate).
Würde diese Methode direkt in der Formularmethode *Kalender* verwendet, könnten Sie sie direkt im Formularereignis "On Load" aufrufen:

```
SetCalendarDate(Current date)
```

Im Kontext der Host Datenbank kann es passieren, dass ein Projektformular zwei "Kalender" Unterformulare, z.B. in den Unterformularobjekten mit Namen "Kal1" und "Kal2" enthält. Sie müssen das Datum von Kal1 auf 01/01/10 und das Datum von Kal2 auf 05/05/10 setzen. Sie können die Methode ***SetCalendarDate*** nicht direkt aufrufen, weil die Methode nicht weiß, welche Formulare und Variablen sie verwenden soll. Deshalb müssen Sie sie über folgenden Code aufrufen:

```
EXECUTE METHOD IN SUBFORM("Cal1";"SetCalendarDate";*;!01/01/10!)
EXECUTE METHOD IN SUBFORM("Cal2";"SetCalendarDate";*;!05/05/10!)
```

Beispiel 3

Fortgeschrittenes Beispiel: Im selben Kontext wie oben zeigt dieses Beispiel eine generische Methode:

```
 // Inhalt der Methode SetCalendarDate
C_DATE($1)
C_TEXT($2)
Case of
   :(Count parameters=1)
 // Standardausführung der Methode (als ob sie vom Formular selbst ausgeführt wurde)
 // oder speziell für einen Kontext (siehe Fall 2)

   :(Count parameters=2)
 //  Externer Aufruf, benötigt einen Kontext
 // Rekursiver Aufruf mit nur einem Parameter
      EXECUTE METHOD IN SUBFORM($2;"SetCalendarDate";*;$1)
End case
```

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

Form event

Form event -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Nummer Formularereignis

Beschreibung

Die Funktion **Form event** gibt einen numerischen Wert zurück, der die Art des gerade abgelaufenen Formularereignisses kennzeichnet. Sie verwenden **Form event** normalerweise innerhalb eines Formulars oder einer Objektmethode.

4D bietet unter dem Thema **Formularereignisse** vordefinierte Konstanten, um die durch die Funktion **Form event** zurückgegebenen Werte zu vergleichen.

Einige Ereignisse sind generisch, d.h. sie gelten für jede Art von Objekt, andere gelten nur für einen bestimmten Objekttyp.

Konstante	Typ	Wert	Kommentar
On Load	Lange Ganzzahl	1	Das Formular wird gleich angezeigt oder gedruckt.
On Mouse Up	Lange Ganzzahl	2	(*nur Bilder*) Der Benutzer hat gerade in einem Bildobjekt die linke Maustaste losgelassen
On Validate	Lange Ganzzahl	3	Die Eingabe in den Datensatz wurde bestätigt.
On Clicked	Lange Ganzzahl	4	Das Objekt wurde angeklickt.
On Header	Lange Ganzzahl	5	Der Kopfteil des Formulars wird gleich gedruckt oder angezeigt.
On Printing Break	Lange Ganzzahl	6	Ein Umbruchbereich im Formular wird gleich gedruckt.
On Printing Footer	Lange Ganzzahl	7	Der Fußteil des Formulars wird gleich gedruckt.
On Display Detail	Lange Ganzzahl	8	Ein Datensatz wird gleich in einer Liste bzw. eine Zeile in einer Listbox angezeigt.
On VP Ready	Lange Ganzzahl	9	(*nur 4D View Pro Areas*) Laden des 4D View Pro Bereichs ist komplett
On Outside Call	Lange Ganzzahl	10	Das Formular hat einen Aufruf **POST OUTSIDE CALL** erhalten.
On Activate	Lange Ganzzahl	11	Das Formularfenster wird zum vordersten Fenster.
On Deactivate	Lange Ganzzahl	12	Das Formularfenster ist nicht mehr das vorderste Fenster.
On Double Clicked	Lange Ganzzahl	13	Auf ein Objekt wurde ein Doppelklick ausgeführt.
On Losing Focus	Lange Ganzzahl	14	Ein Formularobjekt verliert den Fokus.
On Getting Focus	Lange Ganzzahl	15	Ein Formularobjekt erhält den Fokus.
On Drop	Lange Ganzzahl	16	Daten werden in ein Objekt gezogen.
On Before Keystroke	Lange Ganzzahl	17	Ein Zeichen wird gerade in das Objekt mit Fokus eingegeben. **Get edited text** gibt den Text im Objekt ohne dieses Zeichen zurück.
On Menu Selected	Lange Ganzzahl	18	Ein Menüeintrag wurde ausgewählt.
On Plug in Area	Lange Ganzzahl	19	Ein externes Objekt hat angefragt, seine Objektmethode auszuführen.
On Data Change	Lange Ganzzahl	20	Daten im Objekt wurden geändert.
On Drag Over	Lange Ganzzahl	21	Daten werden in ein Objekt gezogen.
On Close Box	Lange Ganzzahl	22	Die Schließbox des Fensters wurde angeklickt.
On Printing Detail	Lange Ganzzahl	23	Der Detailbereich des Formulars wird gleich gedruckt.
On Unload	Lange Ganzzahl	24	Das Formular wird gerade verlassen oder erneuert.
On Open Detail	Lange Ganzzahl	25	Ein dem Ausgabeformular oder der Listbox zugeordnetes Eingabeformular wird gerade geöffnet.
On Close Detail	Lange Ganzzahl	26	Sie haben das Eingabeformular verlassen und gehen zurück zum Ausgabeformular.
On Timer	Lange Ganzzahl	27	Die Anzahl der durch **SET TIMER** definierten Ticks wurde überschritten.
On After Keystroke	Lange Ganzzahl	28	Ein Zeichen wird gerade in das Objekt mit Fokus eingegeben. **Get edited text** gibt den Text im Objekt inkl. diesem Zeichen zurück
On Resize	Lange Ganzzahl	29	Das Formularfenster wird angepasst.
On After Sort	Lange Ganzzahl	30	(*nur Listbox*) In einer Spalte der Listbox wurde gerade eine Standard-Sortierung ausgeführt.
On Selection Change	Lange Ganzzahl	31	• *Listbox*: Die aktuelle Auswahl der Zeilen oder Spalten wurde geändert. • *Datensätze in Liste*: Der aktuelle Datensatz oder die aktuelle Auswahl der Zeilen in einem Listen- bzw. Unterformular wurde geändert. • *Hierarchische Liste*: Die Auswahl in der Liste wurde nach einem Mausklick oder Tastenanschlag geändert. • *Eingebbares Feld oder Variable*: Die Textauswahl oder Position des Cursors im Bereich wurde nach einem Mausklick oder Tastenanschlag geändert.
On Column Moved	Lange Ganzzahl	32	(*nur Listbox*) Der Benutzer hat eine Spalte der Listbox per Drag and Drop bewegt.
On Column Resize	Lange Ganzzahl	33	(*nur Listbox*) Der Benutzer hat die Breite einer Spalte der Listbox mit der Maus geändert.

Konstante	Typ	Wert	Kommentar
On Row Moved	Lange Ganzzahl	34	(*nur Listbox*) Der Benutzer hat eine Zeile der Listbox per Drag-and-Drop bewegt.
On Mouse Enter	Lange Ganzzahl	35	Der Mauszeiger geht in den grafischen Bereich eines Objekts.
On Mouse Leave	Lange Ganzzahl	36	Der Mauszeiger verlässt den grafischen Bereich eines Objekts.
On Mouse Move	Lange Ganzzahl	37	Der Mauszeiger bewegt sich (mindestens 1 Pixel) ODER eine Modifier-Taste (Shift, Alt, Shift Lock) wurde gedrückt. Wurde das Ereignis nur für ein Objekt markiert, wird es nur generiert, wenn der Cursor im grafischen Bereich eines Objekts liegt.
On Alternative Click	Lange Ganzzahl	38	• *3D Schaltflächen:* Der Pfeilbereich einer 3D Schaltfläche ist angeklickt • Listboxen: In einer Spalte eines Objekt Arrays ist eine Schaltfläche [...] angeklickt (Attribut "alternateButton")
On Long Click	Lange Ganzzahl	39	(*nur 3D buttons*) Eine 3D Schaltfläche wird angeklickt und die Maustaste bleibt für eine gewisse Zeit gedrückt.
On Load Record	Lange Ganzzahl	40	Bei der Eingabe in die Liste, wird ein Datensatz während der Änderung geladen (Der Benutzer klickt auf eine Zeile im Datensatz und ein Feld wechselt in den Editiermodus).
On Before Data Entry	Lange Ganzzahl	41	(*nur Listbox*) Eine Zelle der Listbox wechselt gerade in den Editiermodus
On Header Click	Lange Ganzzahl	42	(*nur Listbox*) Ein Spaltentitel der Listbox wird angeklickt.
On Expand	Lange Ganzzahl	43	(*hierarchische Listen und hierarchische Listboxen*) Ein Element der hierarchischen Liste bzw. Listbox wurde per Mausklick oder Tastenanschlag aufgeklappt.
On Collapse	Lange Ganzzahl	44	(*hierarchische Listen und hierarchische Listboxen*) Ein Element der hierarchischen Liste bzw. Listbox wurde über Mausklick oder Tastenanschlag zugeklappt.
On After Edit	Lange Ganzzahl	45	Der Inhalt des eingebbaren Objekts mit Fokus wurde gerade geändert.
On Begin Drag Over	Lange Ganzzahl	46	Ein Objekt wird gerade bewegt (Drag)
On Begin URL Loading	Lange Ganzzahl	47	(*nur Web Areas*) Eine neue URL wird in den Webbereich geladen.
On URL Resource Loading	Lange Ganzzahl	48	(*nur Web Areas*) Eine neue Ressource wird in den Web Bereich geladen.
On End URL Loading	Lange Ganzzahl	49	(*nur Web Areas*) Alle Ressourcen des URL wurden geladen.
On URL Loading Error	Lange Ganzzahl	50	(*nur Web Areas*) Beim Laden der URL ist ein Fehler aufgetreten.
On URL Filtering	Lange Ganzzahl	51	(*nur Web Areas*) Der Web Bereich hat eine URL geblockt.
On Open External Link	Lange Ganzzahl	52	(*nur Web Areas*) Im Browser wurde eine externe URL geöffnet.
On Window Opening Denied	Lange Ganzzahl	53	(*nur Web Areas*) Ein PopUp-Fenster wurde blockiert.
On bound variable change	Lange Ganzzahl	54	Die dem Unterformular zugewiesene Variable wird geändert.
On Page Change	Lange Ganzzahl	56	Aktuelle Seite im Formular wird geändert
On Footer Click	Lange Ganzzahl	57	(*nur Listboxen*) Der Fußteil einer Listbox oder einer Spalte der Listbox ist angeklickt
On Delete Action	Lange Ganzzahl	58	(*nur hierarchische Listen und Listboxen*) Ein Benutzer möchte ein Element löschen
On Scroll	Lange Ganzzahl	59	Der Benutzer scrollt den Inhalt eines Feldes vom Typ Bild oder Variable mit der Maus oder Tastatur.

Hinweis: Ereignisse, die für Ausgabeformulare spezifisch sind, sind für **Projektformulare** nicht verwendbar. Dazu gehören: On Display Detail, On Open Detail, On Close Detail, On Load Record, On Header, On Printing Detail, On Printing Break, On Printing Footer.

Ereignisse und Methoden

Tritt ein Formularereignis ein, führt 4D folgende Aktionen aus:

- Zuerst werden die Objekte des Formulars durchlaufen und die zum Ereignis gehörenden Objektmethoden aufgerufen.
- Wurde das entsprechende Formularereignis ausgewählt, wird die Formularmethode aufgerufen.

Die Objektmethoden, sofern vorhanden, werden nicht in einer bestimmten Reihenfolge aktiviert. Als Faustregel gilt, dass die Objektmethoden immer vor den Formularmethoden aufgerufen werden. Ist das Objekt ein Unterformular, werden auch hier

zuerst die Objektmethoden und dann die Formularmethoden des Ausgabeformulars aufgerufen. Die Faustregel gilt also auch für Objekte innerhalb eines Unterformulars.

Wird für ein gegebenes Ereignis kein Formularereignis ausgewählt, können für Objekte mit demselben Ereignis trotzdem Objektmethoden aufgerufen werden. Mit anderen Worten, Aktivieren oder Deaktivieren eines Ereignisses auf Formularebene hat keine Auswirkung auf die Ereignisse für Objekte. Davon ausgenommen sind die Ereignisse On Load und On Unload.

Die Anzahl der Objekte in einem Ereignis ist je nach Ereignis unterschiedlich:

- On Load - Alle Objekte des Formulars (von jeder Seite), für die Ereignisse On Load für Objekte ausgewählt wurden, sind mit den entsprechenden Objektmethoden beteiligt. Bei Auswahl von Ereignissen On Load für Formulare werden die entsprechenden Formularmethoden angerufen.
- On Activate oder On Resize - Es wird keine Objektmethode aufgerufen, da dieses Ereignis für das gesamte Formular gilt und nicht für ein spezifisches Objekt. Folglich werden bei Auswahl von Ereignissen On Activate für Formulare im Formular nur die Formularmethoden aufgerufen.
- On Timer - Dieses Ereignis wird nur erzeugt, wenn in der Formularmethode zuvor der Befehl **SET TIMER** aufgerufen wurde. Ist die Eigenschaft Formularereignis On Timer ausgewählt, erhält nur die Formularmethode das Ereignis. Es wird keine Objektmethode ausgelöst.
- On Drag Over - Nur die Objektmethode für das dropfähige Objekt innerhalb des Ereignisses wird aufgerufen,wenn die Eigenschaft "dropfähig" dafür ausgewählt wurde. Die Formularmethode wird nicht aufgerufen.
- Umgekehrt wird für das Ereignis On Begin Drag Over die Objektmethode oder die Formularmethode des bewegten Objekts aufgerufen, wenn das Ereignis "dragfähig" dafür ausgewählt ist.

WARNUNG: Während einem Ereignis On Drag over wird - im Gegensatz zu allen anderen Ereignissen - die Objektmethode für ein Objekt im Kontext des Drag-and-Drop Quellobjekts und nicht des Zielobjekts ausgeführt. Weitere Informationen dazu finden Sie im Abschnitt **Einführung in Drag and Drop**.

- Wurden die Ereignisse On Mouse Enter, On Mouse Move und On Mouse Leave für das Formular markiert, werden sie für jedes Formularobjekt erzeugt. Sind sie für ein einzelnes Objekt markiert, werden sie nur für dieses Objekt generiert. Bei übereinander liegenden Objekten wird das Ereignis vom ersten Objekt erzeugt, das dieses verwalten kann, ausgehend von oben nach unten. Objekte, die über den Befehl **OBJECT SET VISIBLE** ausgeblendet sind, erzeugen diese Ereignisse nicht. Wird ein Objekt eingegeben, können – je nach Position der Maus – andere Objekte diese Art Ereignisse empfangen. Beachten Sie, dass das Ereignis On Mouse Move nicht nur bei Bewegen des Mauszeigers generiert wird, sondern auch wenn der Benutzer eine Modifier-Taste, wie **Shift**, **Shift Lock**, **Alt** oder **Option**, drückt. So lassen sich auch Drag-and-Drop Operationen vom Typ Kopieren oder Bewegen verwalten.
- Datensätze in Liste: Die Abfolge der Aufrufe von Methoden und Formularereignissen in Listenformularen, die über die Befehle **DISPLAY SELECTION** und **MODIFY SELECTION** und Unterformulare angezeigt werden ist folgendermaßen:
  Für jedes Objekt im Kopfteil:
  - Objektmethode mit Ereignis On Header

  Formularmethode mit Ereignis On Header

  Für jeden Datensatz:
  - Für jedes Objekt im Detail-Bereich:
  - Objektmethode mit Ereignis On Display Detail
  - Formularmethode mit Ereignis On Display Detail

- Es ist nicht erlaubt, einen 4D Befehl aufzurufen, der ein Dialogfenster über die Ereignisse On Display Detail und On Header anzeigt. Das erzeugt einen Syntaxfehler. Das gilt insbesondere für die Befehle **ALERT**, **DIALOG**, **CONFIRM**, **Request**, **ADD RECORD**, **MODIFY RECORD**, **DISPLAY SELECTION** und **MODIFY SELECTION**.
- On Page Change: Auf Formularebene verfügbar (wird in der Formularmethode aufgerufen).
  Es wird jedes Mal generiert, wenn sich die aktuelle Seite des Formulars ändert, entweder nach Aufrufen des Befehls **FORM GOTO PAGE** oder über standardmäßige Navigationsaktionen. Beachten Sie, dass dieses Ereignis erst generiert wird, wenn die Seite komplett geladen ist, z.B. wenn alle enthaltenen Objekte, inkl. Web Areas, initialisiert sind. Dieses Ereignis ist hilfreich zum Ausführen von Code, für den zuvor alle Objekte initialisiert sein müssen. Sie können damit Ihre Anwendung auch optimieren, wenn Code nicht bereits nach dem Laden der 1. Seite ausgeführt werden soll, sondern erst nach Anzeigen einer bestimmten Seite im Formular, beispielsweise eine Suche.

Folgende Tabelle zeigt, für welche Ereignisse Objekte und Methoden aufgerufen werden:

Ereignis	Objektmethoden	Formularmethoden	Welche Objekte
On Load	Ja	Ja	Alle Objekte
On Unload	Ja	Ja	Alle Objekte
On Validate	Ja	Ja	Alle Objekte
On Clicked	Ja (wenn anklickbar oder eingebbar) (*)	Ja	Nur betroffene Objekte
On Double Clicked	Ja (wenn anklickbar oder eingebbar) (*)	Ja	Nur betroffene Objekte
On Before Keystroke	Ja (wenn eingebbar) (*)	Ja	Nur betroffene Objekte
On After Keystroke	Ja (wenn eingebbar) (*)	Ja	Nur betroffene Objekte
On Getting Focus	Ja (wenn tabfähig) (*)	Ja	Nur betroffene Objekte
On Losing Focus	Ja (wenn tabfähig) (*)	Ja	Nur betroffene Objekte
On Activate	Nie	Ja	Keine
On Deactivate	Nie	Ja	Keine
On Outside Call	Nie	Ja	Keine
On Begin Drag Over	Ja (wenn drag-fähig (**)	Ja	Nur betroffene Objekte
On Drop	Ja (wenn drop-fähig) (**)	Ja	Nur betroffene Objekte
On Drag Over	Ja (wenn drop-fähig) (**)	Nie	Nur betroffene Objekte
On Mouse Enter	Ja	Ja	Alle Objekte
On Mouse Move	Ja	Ja	Alle Objekte
On Mouse Leave	Ja	Ja	Alle Objekte
On Mouse Up	Ja	Nie	Nur betroffene Objekte
On Menu Selected	Nie	Ja	Keine
On Data Change	Ja (wenn änderbar) (*)	Ja	Nur betroffene Objekte
On Plug-In Area	Ja	Ja	Nur betroffene Objekte
On Header	Ja	Ja	Alle Objekte
On Printing Details	Ja	Ja	Alle Objekte
On Printing Break	Ja	Ja	Alle Objekte
On Printing Footer	Ja	Ja	Alle Objekte
On Close Box	Nie	Ja	Keine
On Display Detail	Ja	Ja	Alle Objekte
On Open Detail	Nein, außer für Listboxen	Ja	Keine, außer Listboxen
On Close Detail	Nie	Ja	Keine
On Resize	Nein, außer für Listboxen	Ja	Keine, außer Listboxen
On Selection Change	Ja (***)	Ja	Nur betroffenes Objekt
On Load Record	Nie	Ja	Keine
On Timer	Nie	Ja	Keine
On Scroll	Ja	Ja	Nur betroffenes Objekt
On Before Data Entry	Ja (Listbox)	Nie	Nur betroffenes Objekts
On Column Moved	Ja (Listbox)	Nie	Nur betroffenes Objekt
On Row Moved	Ja (Listbox)	Nie	Nur betroffenes Objekt
On Column Resize	Ja (Listbox)	Nie	Nur betroffenes Objekt
On Header Click	Ja (Listbox)	Nie	Nur betroffenes Objekt
On After Sort	Ja (Listbox)	Nie	Nur betroffenes Objekt
On Long Click	Ja (3D Schaltfläche)	Ja	Nur betroffenes Objekt
On Alternative Click	Ja (3D Schaltfläche und Listbox)	Nie	Nur betroffenes Objekt
On Expand	Ja (hierarch. Liste)	Nie	Nur betroffenes Objekt
On Collapse	Ja (hierarch. Liste)	Nie	Nur betroffenes Objekt
On Begin URL Loading	Ja (Web Area)	Nie	Nur betroffenes Objekt
On URL Resource Loading	Ja (Web Area)	Nie	Nur betroffenes Objekt
On End URL Loading	Ja (Web Area)	Nie	Nur betroffenes Objekt
On URL Loading Error	Ja (Web Area)	Nie	Nur betroffenes Objekt
On URL Filtering	Ja (Web Area)	Nie	Nur betroffenes Objekt
On Open External Link	Ja (Web Area)	Nie	Nur betroffenes Objekt
On Window Opening Denied	Ja (Web Area)	Nie	Nur betroffenes Objekt
On VP Ready	Ja (4D View Pro Area)	Ja	Nur betroffenes Objekt

(*) Weitere Informationen finden Sie im folgenden Abschnitt **Ereignisse, Objekte und Eigenschaften**.
(**) Weitere Informationen finden Sie im Kapitel **Drag and Drop**.
(***) Dieses Ereignis unterstützen nur Objekte vom Typ Listbox, hierarchische Liste und Unterformular

WICHTIG: Beachten Sie, dass die Methode eines Formulars oder Objekts nur dann aufgerufen wird, wenn das entsprechende Ereignis für das Formular bzw. die Objekte ausgewählt wurde. Sie können aber auch in der Designumgebung Ereignisse im Fenster Formular- und Objekteigenschaften deaktivieren. Dadurch können Sie die Anzahl der aufgerufenen Methoden erheblich verringern und so die Ausführungsgeschwindigkeit Ihrer Formulare optimieren.

WARNUNG: Die Ereignisse On Load und On Unload werden für Objekte nur generiert, wenn sowohl die Ereignisse, die zu den Objekten gehören, als auch die Ereignisse, die zu den Formularen gehören, aktiviert sind. Sind nur die Ereignisse für Objekte aktiviert, werden sie nicht ausgeführt; On Load und On Unload müssen sowohl auf Objekt- als auch auf Formularebene aktiviert sein.

Ereignisse, Objekte und Eigenschaften

Eine Objektmethode wird aufgerufen, wenn das Ereignis für das Objekt eintreten kann. Das ist abhängig vom Typ und den Eigenschaften. Im folgenden werden die Ereignisse beschrieben, die Sie im allgemeinen zum Verwalten der verschiedenen Objekttypen verwenden können.
Beachten Sie, dass die Eigenschaftenliste im Formulareditor nur die Ereignisse anzeigt, die mit dem ausgewählten Objekt oder dem Formular kompatibel sind.

Klickbare Objekte

Klickbare Objekte werden mit der Maus verwaltet. Dazu gehören:

- Eingebbare Datenfelder oder Variablen vom Typ Boolean
- Schaltflächen, Standardschaltflächen, Optionsfelder, Kontrollkästchen, Schaltflächengitter
- 3D Schaltflächen, 3D Optionsfelder, 3D Kontrollkästchen
- PopUp-Menüs, hierarchische PopUp-Menüs, Bildmenüs
- Dropdown-Listen, Menüs/Dropdown-Listen
- Rollbare Bereiche, hierarchische Listen, Listboxen und Spalten von Listboxen
- Unsichtbare Schaltflächen, hervorgehobene Schaltflächen, Optionsbilder
- Thermometer, Lineale, Halbkreisskalen (Objekte mit Regler)
- Registerkarten
- Splitter

Haben Sie für eines dieser Objekte als Eigenschaft On Clicked und On Double Clicked ausgewählt, können Sie mit der Funktion **Form event** die Klicks innerhalb oder auf das Objekt herausfinden und verwalten. Sie gibt je nach Fall On Clicked oder On Double Clicked zurück. Haben Sie für ein Objekt beide Ereignisse gewählt, wird zuerst On Clicked und dann On Double Clicked erzeugt, wenn der Benutzer auf das Objekt doppelklickt.

Das Ereignis On Clicked tritt für all diese Objekte ein, sobald die Maustaste losgelassen wird. Es gibt jedoch bestimmte Ausnahmen:

- Unsichtbare Schaltflächen - Das Ereignis On Clicked tritt bereits beim Klick ein, und nicht erst, wenn die Maustaste losgelassen wird.
- Objekte mit Schieberegler (Thermometer, Lineale und Halbkreisskalen) - Gibt das Anzeigeformat an, dass die Objektmethode aufgerufen werden muss, während der Regler verschoben wird, tritt das Ereignis On Clicked bereits beim Klicken ein.

Beim Ereignis On Clicked können Sie über den Befehl **Clickcount** die Anzahl der Klicks eines Benutzers testen.

Hinweis: Einige dieser Objekte können über die Tastatur aktiviert werden. Beispiel: Ein Kontrollkästchen mit Fokus kann mit der Taste Leerschritt eingegeben werden. In diesem Fall wird auch das Ereignis On Clicked generiert.

WARNUNG: Comboboxen gelten nicht als klickbare Objekte, sondern als eingebbarer Textbereich mit zugeordneter Dropdown-Liste, die Standardwerte liefert. Von daher verwalten Sie die Dateneingabe in Comboboxen über die Ereignisse On Before Keystroke, On After Keystroke und On Data Change.

Hinweis: Ab 4D v13 können die Objekte PopUp-Menü/DropDown-Liste und hierarchisches PopUp-Menü das Ereignis On Data Change generieren. Damit können Sie die Aktivierung des Objekts feststellen, wenn ein anderer Wert als der aktuelle ausgewählt wird.

Über Tastatur eingebbare Objekte

Bei über Tastatur eingebbaren Objekten geben Sie Daten über Tastatur ein. Dafür filtern Sie die Dateneingabe auf der untersten Ebene, nämlich mit den Ereignissen On After Edit, On Before Keystroke, On After Keystroke und On Selection Change. Sie können diese Ereignisse mit der Funktion **Get edited text** nutzen. Solche Objekte und Datentypen sind:

- Alle eingebbaren Datenfeldobjekte vom Typ Alpha, Text, Datum, Zeit, Zahl oder Bild (nur On After Edit)
- Alle eingebbaren Variablen vom Typ Alpha, Text, Datum, Zeit, Zahl oder Bild (nur On After Edit)
- Comboboxen (außer On Selection Change)
- Listboxen

Hinweis: Ab 4D v14 können eingebbare Felder und Variablen mit Text (vom Typ Text, Datum, Zeit oder Zahl) ebenfalls die Ereignisse On Clicked oder On Double Clicked generieren.

Hinweis: Auch wenn hierarchische Listen "eingebbare" Objekte sind, verwalten sie nicht die Formularereignisse On After Edit, On Before Keystroke und On After Keystroke. Siehe hierzu auch den späteren Absatz "Hierarchische Listen".

- On Before Keystroke and On After Keystroke
  Hinweis: Das Ereignis On After Keystroke kann generell durch das Ereignis On After Edit ersetzt werden (siehe unten).
  Haben Sie für ein Objekt die Ereignisse On Before Keystroke und On After Keystroke ausgewählt, können Sie die Tastaturanschläge im Objekt mit der Funktion **Form event** herausfinden und verwalten. Sie gibt On Before Keystroke und dann On After Keystroke zurück. Weitere Informationen dazu finden Sie in der Beschreibung zur Funktion **Get edited text**.
  Diese Ereignisse werden auch über Befehle aktiviert, die eine Benutzeraktion simulieren, z.B. **POST KEY**.
  Beachten Sie, dass Benutzeraktionen, die nicht über Tastatur ausgeführt werden, z.B. Kopieren, Drag & Drop, nicht berücksichtigt werden. Diese Ereignisse verwalten Sie mit On After Edit.
  Hinweis: Die Ereignisse On Before Keystroke und On After Keystroke werden beim Verwenden einer Eingabemethode nicht erzeugt. Eine Eingabemethode (oder IME, Eingabemethodeneditor) ist ein Programm oder eine Systemkomponente, um komplexe Zeichen oder Symbole mit einer Standardtastatur einzugeben, z.B. Japanisch oder Chinesisch.

- On After Edit
  Dieses Ereignis wird nach jeder Änderung am Inhalt eines Eingabebereichs (Feld oder Variable) generiert, unabhängig von der Aktion, welche die Änderung verursacht hat.
  Aktionen, welche dieses Ereignis auslösen, sind:
  - Standardaktionen für Bearbeiten, welche den Inhalt verändern, wie Einsetzen, Ausschneiden, Löschen oder Abbrechen;
  - Wert ziehen (Drop-Aktion ähnlich wie Einsetzen);
  - Jede vom Benutzer gemachte Eingabe per Tastatur; in diesem Fall wird das Ereignis On After Edit nach den Ereignissen On Before Keystroke und On After Keystroke erzeugt, sofern sie verwendet werden.
  - Jede Änderung, die über einen Befehl der Programmiersprache ausgeführt wird, der eine Benutzeraktion simuliert, z.B. **POST KEY**.
  Achten Sie darauf, dass folgende Aktionen NICHT dieses Ereignis auslösen:
  - Aktionen für Bearbeiten, welche den Inhalt des Bereichs nicht verändern, wie Kopieren, Alle auswählen oder Wert

markieren (Drag-Aktion ähnlich wie Kopieren); diese Aktionen verändern nicht die Position des Cursors und lösen so das Ereignis On Selection Change aus.
- Jede Änderung am Inhalt, die per Programmierung ausgeführt wird, mit Ausnahme der Befehle, die eine Benutzeraktion simulieren.
Über dieses Ereignis können Sie Benutzeraktionen steuern, z.B. um zu verhindern, dass ein zu langer Text eingesetzt, ein Feld mit Kennwort abgeschnitten wird oder um bestimmte Zeichen zu blockieren.

- On Selection Change
Dieses Ereignis wird bei Anwendung auf ein Textfeld bzw. eine Variable (eingebbar oder nicht eingebbar) immer ausgelöst, wenn sich die Position des Cursors ändert. Das passiert z.B., sobald der Benutzer Text mit der Maustaste oder über die Pfeiltasten der Tastatur auswählt oder Text eingibt. Auf diese Weise können Sie z.B. Befehle wie **GET HIGHLIGHT** auswählen.

Änderbare Objekte

Die Werte in änderbaren Objekten können mit der Maus oder über Tastatur geändert werden; sie sind eigentlich keine echten Steuerelemente der Benutzeroberfläche, die mit dem Ereignis On Clicked verwaltet werden. Solche Objekte sind:

- Alle eingebbaren Datenfeldobjekte (mit Ausnahme von BLOB)
- Alle eingebbaren Variablen (mit Ausnahme von BLOB, Zeiger und Array)
- Comboboxen
- Externe Objekte (für die die komplette Dateneingabe über das 4D Plug-In bestätigt wird)
- Hierarchische Listen
- Listboxen und Spalten von Listboxen

Diese Objekte erhalten On Data Change Ereignisse. Wurde für eines dieser Objekte das Ereignis On Data Change ausgewählt, können Sie mit der Funktion **Form event** das Ändern des ursprünglichen Werts herausfinden und verwalten. Sie gibt On Data Change zurück. Das Ereignis wird generiert, sobald 4D die dem Objekt zugewiesene Variable intern aktualisiert, z.B. wenn der Eingabebereich eines Objekts den Fokus verliert.

Tabfähige Objekte

Tabfähige Objekte erhalten Fokus, wenn Sie diese mit der Tabulatortaste aufrufen bzw. anklicken. Das Objekt mit Fokus erhält Zeichen über Tastatur. Es können jedoch keine Abkürzungen (Windows) bzw. Tastaturkürzel (Mac OS) eines Menübefehls oder Objekts wie z.B. eine Schaltfläche sein.

Folgende Objekte sind NICHT tabfähig:

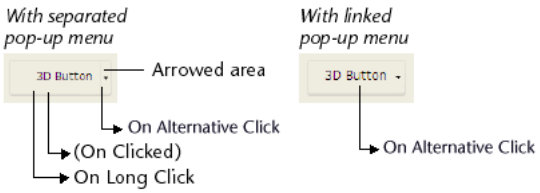
- Nicht eingebbare Datenfelder oder Variablen
- Schaltflächengitter
- 3D Schaltflächen, 3D Optionsfelder, 3D Kontrollkästchen
- PopUp-Menüs, hierarchische PopUp-Menüs
- Menüs/Dropdown-Listen
- Bildmenüs
- Rollbare Bereiche
- Unsichtbare Schaltflächen, hervorgehobene Schaltflächen, Optionsbilder
- Diagramme
- Externe Objekte (für die das 4D Plug-In keine komplette Dateneingabe zulässt)
- Registerkarten
- Splitter

Haben Sie für ein tabfähiges Objekt das Ereignis On Getting Focus bzw. On losing Focus auf Objektebene ausgewählt, können Sie mit der Funktion **Form event** eine Fokusänderung herausfinden und verwalten. Sie gibt entsprechend On Getting Focus oder On losing Focus zurück.

3D Schaltflächen

Mit 3D Schaltflächen können Sie ausgeklügelte grafische Oberflächen einrichten. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*. Zusätzlich zu den generischen Ereignissen gibt es für 3D Schaltflächen zwei spezifische Formularereignisse:

- On Long Click: Dieses Ereignis tritt ein, wenn auf eine 3D Schaltfläche geklickt wird und die Maustaste eine gewisse Zeit gedrückt bleibt. Theoretisch entspricht die Zeitspanne, für die das Ereignis generiert wird, der max. Zeit, die zwischen einem Doppelklick liegt, so wie in den Systemeinstellungen definiert.
Dieses Ereignis lässt sich für alle Arten von 4D Schaltflächen, 3D Optionsfeldern und 3D Kontrollkästchen generieren. Davon ausgenommen sind die "vorherige Generation" der 3D Schaltflächen (z.B. Background offset) und die Pfeilbereiche von 3D Schaltflächen mit PopUp-Menü (siehe unten).
Dieses Ereignis dient in der Regel zum Anzeigen von PopUp-Menüs bei langen Klicks auf die Schaltfläche. Das Ereignis On Clicked wird erzeugt – sofern aktiviert, wenn der Benutzer die Maustaste loslässt, bevor die Zeit für den langen Klick abgelaufen ist.
- On Alternative Click: Einige 3D Schaltflächentypen können mit einem PopUp-Menü verknüpft sein und einen Pfeil anzeigen. Klicken Sie darauf, erscheint ein PopUp-Menü zum Auswählen alternativer Aktionen in Bezug auf die primäre Aktion der Schaltfläche.
Diese Art Schaltfläche lässt sich mit dem Ereignis On Alternative Click verwalten. Es wird erzeugt, wenn der Benutzer auf den Pfeil klickt und die Maustaste gedrückt hält.
- Ist das PopUp-Menü "getrennt", wird das Ereignis nur ausgelöst bei Anklicken des Bereichs mit dem Pfeil.
- Ist das PopUp-Menü "verbunden", wird das Ereignis ausgelöst bei Anklicken eines beliebigen Teils der Schaltfläche.
Beachten Sie, dass das Ereignis On Long Click nicht mit diesem Schaltflächentyp erzeugt werden kann.



Folgende Typen für 3D Schaltflächen, 3D Optionsfelder und 3D Kontrollkästchen akzeptieren die Eigenschaft "Mit PopUp-Menü": Keine, Toolbar Schaltfäche, Bevel, Bevel (abgerundet) und Office XP.

Listboxen

Für diesen Objekttyp gibt es verschiedene Formularereignisse. Damit lassen sich folgende Situationen verwalten:

- On Before Data Entry: Dieses Ereignis wird unmittelbar vor Bearbeiten einer Zelle in der Listbox generiert (vor Anzeige des Eingabe-Cursors). Damit kann der Entwickler z.B. einen anderen Text anzeigen, je nachdem ob der Benutzer im Modus Anzeigen oder Bearbeiten ist.
- On Selection Change: Dieses Ereignis wird bei jeder Änderung der aktuellen Auswahl von Zeilen und Spalten in der Listbox generiert, sowie in Datensatzlisten und hierarchischen Listen.
- On Column Moved: Dieses Ereignis wird generiert, wenn der Benutzer eine Spalte in der Listbox per Drag&Drop bewegt. Es wird nicht generiert, wenn die Spalte per Drag&Drop an ihre Ausgangsposition bewegt wird. Der Befehl **LISTBOX MOVED COLUMN NUMBER** gibt die neue Position der Spalte zurück.
- On Row Moved: Dieses Ereignis wird generiert, wenn der Benutzer eine Zeile der Listbox per Drag&Drop bewegt. Es wird nicht generiert, wenn die Spalte per Drag&Drop an ihre Ausgangsposition bewegt wird.
- On Column Resize: Dieses Ereignis wird generiert, wenn der Benutzer die Breite einer Spalte in der Listbox ändert.
  Ab 4D v16 wird das Ereignis "live" ausgelöst, es wird beispielsweise während dem Ereignis kontinuierlich gesendet, solange die Größe der betroffenen Listbox oder Spalte verändert wird. Die Aktion erfolgt manuell durch einem Benutzer oder ergibt sich durch Ändern der Listbox und seiner Spalten zusammen mit dem Formularfenster (manuelle Anpassung oder über den Befehl **RESIZE FORM WINDOW**).
  Hinweis: Das Ereignis On Column Resize wird nicht ausgelöst beim Ändern einer "fake" Spalte. Weitere Informationen dazu finden Sie im Abschnitt **Gruppe Vergrößerungsoptionen**.
- On Expand und On Collapse: Diese Ereignisse werden generiert, wenn eine Zeile in der hierarchischen Listbox auf- bzw. zugeklappt wird.
- On Header Click: Dieses Ereignis wird bei einem Klick in den Kopfteil einer Spalte in der Listbox generiert. In diesen Fall erfahren Sie über die Funktion **Self**, welcher Kopfteil angeklickt wurde. Mit der Funktion **Clickcount** können Sie die Anzahl Klicks durch den Benutzer testen.
  Ist in der Listbox die Eigenschaft **Sortierbar** angeklickt, können Sie eine Standardsortierung der Spalte über die Variable $0 zulassen oder verweigern:
  - Bei $0 gleich 0 wird die Standardsortierung ausgeführt.
  - Bei $0 gleich -1 wird die Standardsortierung nicht ausgeführt, der Kopfteil zeigt keinen Sortierpfeil an. Über die Befehle zur Array-Verwaltung kann der Entwickler weiterhin eine Spalte nach eigenen Kriterien sortieren.
  Wurde die Eigenschaft **Sortierbar** nicht für die Listbox ausgewählt, wird die Variable $0 nicht verwendet.
- On Footer Click: Dieses Ereignis ist für eine Listbox bzw. die Spalte einer Listbox verfügbar. Es wird generiert, wenn in den Fußteil einer Listbox bzw. einer Spalte in der Listbox geklickt wird. Dann gibt der Befehl **OBJECT Get pointer** einen Zeiger auf die Variable des angeklickten Fußteils zurück. Das Ereignis wird für Klicks mit der rechten und der linken Maustaste generiert. Mit der Funktion **Clickcount** können Sie die Anzahl Klicks durch den Benutzer testen.
- On After Sort: Dieses Ereignis wird direkt nach einer Standardsortierung generiert (es wird z.B. nicht generiert, wenn die Variable $0 im Ereignis On Header Click den Wert -1 zurückgibt.) Dieses Ereignis ist hilfreich, um die Richtung der zuletzt vom Benutzer ausgeführten Sortierung zu speichern. Bei diesem Ereignis gibt die Funktion **Self** einen Zeiger auf die Variable des Kopfteils der sortierten Spalte zurück.
- On Delete Action: Dieses Ereignis wird immer erzeugt, wenn ein Benutzer versucht, die ausgewählten Einträge über die Taste **Löschen** bzw. **Rückschritt** oder den Menübefehl **Bearbeiten > Löschen** zu löschen. Es ist nur auf der Listbox-Ebene verfügbar und wird jedes Mal erzeugt, wenn ein Benutzer versucht, eine Zeile zu löschen. Beachten Sie, dass 4D nur das Ereignis generiert, d.h. das Programm löscht keine Einträge. Der Entwickler muss den Löschvorgang mit entsprechenden Meldungen selbst verwalten.
- On Scroll (neu in v15): Dieses Ereignis wird erzeugt, sobald ein Benutzer in den Zeilen oder Spalten der Listbox scrollt. Es wird nur erzeugt, wenn das Scrollen durch einen Benutzer erfolgt, also Rollbalken bzw. Cursor, Mausrädchen oder Tastatur verwendet werden. Es wird nicht generiert bei Scrollen durch Ausführen des Befehls **OBJECT SET SCROLL POSITION**. Dieses Ereignis wird nach jedem anderen Benutzerereignis im Zusammenhang mit Scrollen ausgelöst (On Clicked, On After Keystroke, etc.). Es wird nur in der Objektmethode und nicht in der Formularmethode generiert. Siehe hierzu Beispiel 15.
- On Alternative Click (neu in v15): Dieses Ereignis wird in Spalten von Listboxen des Typs Objekt Array erzeugt, wenn der Benutzer auf eine Schaltfläche [...] klickt (Attribut "alternateButton"). Weitere Informationen dazu finden Sie im Abschnitt **Objekt Arrays in Spalten von Listboxen (4D View Pro)**.

Für Listboxen vom Typ Auswahl gibt es zwei generische Ereignisse:

- On Open Detail: Dieses Ereignis wird generiert, wenn gerade ein Datensatz im Detailformular einer Listbox vom Typ Auswahl angezeigt wird (und bevor dieses Formular geöffnet wird).
- On Close Detail: Dieses Ereignis wird generiert, wenn gerade ein Datensatz im Detailformular einer Listbox vom Typ Auswahl geschlossen wird (egal, ob er geändert wurde oder nicht).

Hierarchische Listen

Es gibt vier spezifische Formularereignisse, um Benutzeraktionen in hierarchischen Listen zu verwalten:

- On Selection Change: Dieses Ereignis wird immer erzeugt, wenn die Auswahl in der hierarchischen Liste über Mausklick oder Tastenkürzel geändert wird. Es wird auch in Listboxen und Datensatzlisten generiert.
- On Expand: Dieses Ereignis wird immer erzeugt, wenn ein Element der hierarchischen Liste über Mausklick oder Tastenkürzel aufgeklappt wird.
- On collapse: Dieses Ereignis wird immer erzeugt, wenn ein Element der hierarchischen Liste über Mausklick oder Tastenkürzel zugeklappt wird.

- On Delete Action: Dieses Ereignis wird immer erzeugt, wenn ein Benutzer versucht, die ausgewählten Einträge über die Taste **Löschen** bzw. **Rückschritt** oder den **Menübefehl Bearbeiten > Löschen** zu löschen.
  Beachten Sie, dass 4D nur das Ereignis generiert, d.h. das Programm löscht keine Einträge. Der Entwickler muss den Löschvorgang mit entsprechenden Meldungen selbst verwalten (siehe Beispiel).

Diese Ereignisse schließen sich nicht gegenseitig aus. Sie lassen sich für eine hierarchische Liste der Reihe nach erzeugen:

- Nach Tastenkürzel (chronologisch):

Ereignis	**Kontext**
On data change	Element wurde bearbeitet
On expand / On collapse	Unterliste über die Pfeile g oder f öffnen/schließen
On selection change	Neues Element wählen
On clicked	Liste über Tastatur aktivieren

- Nach Mausklick (chronologisch):

Ereignis	**Kontext**
On data change	Element wurde bearbeitet
On expand / On collapse	Unterliste über die Icons Auf-/Zuklappen, Öffnen/Schließen oder Doppelklick auf nicht-editierbare Unterliste
On selection change	Neues Element wählen
On clicked / On double clicked	Liste über Klick oder Doppelklick aktivieren

Bildfelder und Variablen

- Das Formularereignis On Picture Scroll wird generiert, sobald ein Benutzer ein Bild in einem Bereich (Feld oder Variable) scrollt. Sie können den Inhalt eines Bildbereichs scrollen, wenn der Bereich kleiner als sein Inhalt ist und beim Anzeigeformat "**Abgeschnitten (nicht-zentriert)**". Weitere Informationen dazu finden Sie im Abschnitt **Bildformate**.
  Das Ereignis wird generiert, wenn das Scrollen durch einen Benutzer erfolgt, also Rollbalken bzw. Cursor, Mausrädchen oder Tastatur verwendet werden. Weitere Informationen zum Scrollen über die Tastatur finden Sie im Abschnitt **Rollbalken**. Es wird nicht generiert, wenn ein Bild durch Ausführen des Befehls **OBJECT SET SCROLL POSITION** gescrollt wird.
  Dieses Ereignis wird nach jedem anderen Benutzerereignis im Zusammenhang mit Scrollen ausgelöst (On Clicked, On After Keystroke, etc.). Es wird nur in der Objektmethode und nicht in der Formularmethode generiert. Siehe hierzu Beispiel 14.

- (Neu in v16) Das Ereignis On Mouse Up wird generiert, wenn der Benutzer beim Ziehen in einem Bildbereich (Feld oder Variable) gerade die linke Maustaste losgelassen hat. Dieses Ereignis ist z.B. hilfreich, wenn der Benutzer die Möglichkeit haben soll, in einem SVG Bereich Objekte zu bewegen, zeichnen oder zu vergrößern/verkleinern.
  Wird das Ereignis On Mouse Up generiert, können Sie dort, wo die Maustaste losgelassen wurde, die lokalen Koordinaten erhalten. Sie werden in den **Systemvariablen** **MouseX** und **MouseY** zurückgegeben und in Pixel angegeben, ausgehend von der oberen linken Ecke des Bildes (0,0).
  Mit diesem Ereignis müssen Sie auch den Befehl **Is waiting mouse up** verwenden für Fälle, wo der "Statusmanager" des Formulars desynchronisiert ist, z.B. wenn das Ereignis On Mouse Up nicht nach einem Klick empfangen wird. Das passiert beispielsweise, wenn ein Meldefenster über den Formular angezeigt wird, während die Maustaste nicht losgelassen wurde. Weitere Informationen dazu finden Sie unter dem Befehl **Is waiting mouse up**.

  Hinweis: Ist für das Bildobjekt die Option "Dragfähig" markiert, wird nie das Ereignis On Mouse Up generiert.

Unterformulare

Ein Objekt Unterformular Container (im Elternformular enthaltenes Objekt, das eine Unterformular Instanz enthält) unterstützt folgende Ereignisse:

- On Load und On Unload: Öffnet und schließt jeweis das Unterformular (Diese Ereignisse müssen auf Ebene des Elternformulars aktiviert worden sein, damit sie berücksichtigt werden). Beachten Sie, dass diese Ereignisse vor denen des Elternformulars generiert werden. Entsprechend zur Funktionsweise von Formularereignissen werden diese Ereignisse bei Unterformularen, die nicht auf Seite 0 oder 1 liegen, nur erzeugt, wenn diese Seite angezeigt bzw. geschlossen ist, und nicht wenn das Formular angezeigt bzw. geschlossen wird.
- On Validate: Bestätigt die Dateneingabe im Unterformular.
- On Data Change: Der Wert der Variablen des Objekts Unterformular wurde geändert.
- On Getting Focus und On Losing Focus: Unterformular Container hat gerade Fokus erhalten bzw. verloren. Diese Ereignisse werden in der Methode des Objekts Unterformular erzeugt, wenn sie markiert sind. Sie werden an die Formularmethode des Unterformulars gesendet, d.h. zum Beispiel, dass Sie die Anzeige der Schaltflächen zum Navigieren im Unterformular gemäß dem Fokus verwalten können.
  Beachten Sie dass Unterformular Objekte selbst Fokus haben können.
- On bound variable change: Dieses spezifische Ereignis wird im Rahmen der Formularmethode des Unterformulars erzeugt, sobald der Variablen, die mit dem Unterformular im Elternformular verbunden ist, ein Wert zugewiesen wird (selbst wenn der gleiche Wert erneut zugewiesen wird) und das Unterformular zur aktuellen Formularseite oder zur Seite 0 gehört. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Hinweise:

- Sie können einen beliebigen eigenen Ereignistyp spezifizieren, der sich über den Befehl **CALL SUBFORM CONTAINER** im Unterformular generieren lässt. Dieser Befehl ermöglicht, die Objektmethode des Containers aufzurufen und ihr einen Ereigniscode zu übergeben.
- Die im Unterformular generierten Ereignisse On Clicked und On Double Clicked werden zuerst von der Formularmethode des Unterformulars und dann von der Formularmethode des Host Formulars empfangen.

Web Areas

Speziell für Web Areas sind sieben Ereignisse verfügbar

- On Begin URL Loading: Dieses Ereignis wird erzeugt, wenn im Web Bereich das Laden einer neuen URL startet. Über die dem Web Bereich zugeordnete Variable "URL" lässt sich die URL herausfinden, die gerade geladen wird.

Hinweis: Die gerade geladene URL unterscheidet sich von der aktuellen URL (siehe Beschreibung zur Funktion **WA Get current URL**)

- On URL Resource Loading: Dieses Ereignis wird immer erzeugt, wenn eine neue Ressource, z.B. ein Bild, ein Rahmen,etc. in die aktuelle Web Seite geladen wird.
  Über die dem Bereich zugewiesene Variable "Ablauf" können Sie den aktuellen Status des Ladevorgangs herausfinden.
- On End URL Loading: Dieses Ereignis wird erzeugt, wenn alle Ressourcen der aktuellen URL geladen sind.
  Über die Funktion **WA Get current URL** können Sie die geladene URL herausfinden.
- On URL Loading Error: Dieses Ereignis wird erzeugt, wenn beim Laden einer URL ein Fehler gefunden wird.
  Über den Befehl **WA GET LAST URL ERROR** erhalten Sie Informationen über den Fehler.
- On URL Filtering: Dieses Ereignis wird erzeugt, wenn der Web Bereich das Laden einer URL blockiert, da über den Befehl **WA SET URL FILTERS** Filter eingerichtet wurden.
  Über die Funktion **WA Get last filtered URL** können Sie die blockierte URL herausfinden.
- On Open External Link: Dieses Ereignis wird erzeugt, wenn das Laden einer URL vom Web Bereich blockiert und die URL mit dem aktuellen Browser des Systems geöffnet wird, da über den Befehl **WA SET EXTERNAL LINKS FILTERS** ein Filter eingerichtet ist.
  Über die Funktion **WA Get last filtered URL** können Sie die blockierte URL herausfinden.
- On Window Opening Denied: Dieses Ereignis wird erzeugt, wenn das Öffnen eines PopUp-Fensters vom Web Bereich blockiert wird. 4D Web Bereiche erlauben nicht das Öffnen von PopUp-Fenstern.
  Über die Funktion **WA Get last filtered URL** können Sie die blockierte URL herausfinden.

4D View Pro areas

The On VP Ready event is available for 4D View Pro areas only. It is generated when the 4D View Pro area loading is complete. You need to use this event to write initialization code for the area. For more information, please refer to the **On VP Ready form event** section.

Beispiel 1

Dieses Beispiel setzt das Ereignis *On Validate* ein, um dem Datenfeld automatisch anzuzeigen, wann der Datensatz geändert wurde:

```
  ` Formularmethode
Case of
  ` ...
  :(Form event=On Validate)
     [aTable]Last Modified On:=Current date
End case
```

Beispiel 2

Dieses Beispiel zeigt die komplette Verwaltung einer Dropdown-Liste (Initialisierung, Benutzerklicks und Objekt loslassen) innerhalb einer Objektmethode:

```
  ` Objektmethode asBurgerSize für DropDown Liste
Case of
  :(Form event=On Load)
     ARRAY TEXT(31;asBurgerSize;3)
     asBurgerSize{1}:="Small"
     asBurgerSize{1}:="Medium"
     asBurgerSize{1}:="Large"
  :(Form event=On Clicked)
     If(asBurgerSize#0)
        ALERT("You chose a "+asBurgerSize{asBurgerSize}+" burger.")
     End if
  :(Form event=On Unload)
     CLEAR VARIABLE(asBurgerSize)
End case
```

Beispiel 3

Dieses Beispiel zeigt, wie eine Drag&Drop-Operation für ein Feldobjekt, das nur Bilder zulässt, angenommen und später verwaltet wird.

```
  ` Objektmethode für Datenfeld, das nur Bilder zulässt
Case of
  :(Form event=On Drag Over)
` Eine Drag&Drop-Operation hat begonnen, die Maus ist gerade im Datenfeld
` Hole Information über das Quellobjekt
     DRAG AND DROP PROPERTIES($vpSrcObject;$vlSrcElement;$lSrcProcess)
` Die ID Nummer des Quellprozesses muss hier nicht geprüft werden, da die Objektmethode innerhalb desselben Prozesses ausgeführt wird.
     $vlDataType:=Type($vpSrcObject->)
` Ist das Quelldokument ein Bild (Datenfeld, Variable oder Array) ?
     If(($vlDataType=Is picture)|($vlDataType=Picture array))
```

```
   ` Wenn ja, akzeptiere das Bewegen (drag).
         $0:=0
      Else
   ` Wenn ja, verweigere das Bewegen (Drag)
         $0:=-1
      End if
   :(Form event=On Drop)
   ` Die Quelldaten wurden bewegt und losgelassen,
   ` sie müssen also in das Objekt kopiert werden
   ` Hole Information über das Quellobjekt
      DRAG AND DROP PROPERTIES($vpSrcObject;$vlSrcElement;$lSrcProcess)
      $vlDataType:=Type($vpSrcObject->)
      Case of
   ` Quellobjekt ist Bilddatenfeld oder Variable
         :($vlDataType=ls picture)
   ` Stammt das Quellobjekt aus demselbem Prozess
   ` (also aus dem gleichen Fenster und Formular)?
            If($lSrcProcess=Current process)
   ` Wenn ja, kopiere den Quellwert
               [aTable]aPicture:=$vpSrcObject->
            Else
   ` Wenn nein, ist das Quellobjekt eine Variable?
               If(Is a variable($vpSrcObject))
   ` Wenn ja, hole Wert aus dem Quellprozess
                  GET PROCESS VARIABLE($lSrcProcess;$vpSrcObject->;$vgDraggedPict)
                  [aTable]aPicture:=$vgDraggedPict
               Else
   ` Wenn nein, hole mit CALL PROCESS den Feldwert aus dem Quellprozess
               End if
            End if
   ` Quellobjekt ist ein Array vom Typ Bild
         :($vlDataType=Picture array)
   ` Stammt das Quellobjekt aus demselbem Prozess
   ` (also aus dem gleichen Fenster und Formular)?
            If($lSrcProcess=Current process)
   ` Wenn ja, kopiere den Quellwert
               [aTable]aPicture:=$vpSrcObject->{$vlSrcElement}
            Else
   ` Wenn nein, hole Wert aus dem Quellprozess
               GET PROCESS VARIABLE($lSrcProcess;$vpSrcObject
               ->{$vlSrcElement};$vgDraggedPict)
               [aTable]aPicture:=$vgDraggedPict
            End if
      End case
End case
```

Hinweis: Weitere Beispiele zum Verwalten von Ereignissen On Drag Over und On Drop finden Sie unter dem Befehl **DRAG AND DROP PROPERTIES**.

Beispiel 4

Dieses Beispiel ist Vorlage für eine Formularmethode. Es zeigt alle Ereignisse, die während einem Summenbericht eintreten können, der ein Formular als Ausgabeformular verwendet:

```
   ` Formularmethode für ein Ausgabeformular für einen Summenbericht
$vpFormTable:=Current form table
Case of
   ` ...
   :(Form event=On Header)
   ` Ein Kopfteil soll gedruckt werden
      Case of
         :(Before selection($vpFormTable->))
   ` Code für den ersten Umbruch im Kopfteil folgt
         :(Level=1)
   ` Code für den Umbruch im Kopfteil Ebene 1 folgt
         :(Level=2)
   ` Code für den Umbruch im Kopfteil Ebene 2 folgt
   ` ...
      End case
   :(Form event=On Printing Details)
   ` Ein Datensatz soll gedruckt werden
```

```
   ` Code für jeden Datensatz folgt
     :(Form event=On Printing Break)
   ` Ein Umbruchbereich soll gedruckt werden
        Case of
           :(Level=0)
   ` Code für den Umbruch Ebene 0 folgt
           :(Level=1)
   ` Code für den Umbruch Ebene 1 folgt
   ` ...
        End case
     :(Form event=On Printing Footer)
        If(End selection($vpFormTable->))
   ` Code für den letzten Fußteil folgt
        Else
   ` Code für den Fußteil folgt
        End if
End case
```

Beispiel 5

Dieses Beispiel zeigt die Vorlage zum Verwalten von Ereignissen in einem Formular, das mit den Befehlen **DISPLAY SELECTION** oder **MODIFY SELECTION** angezeigt wird. Zur besseren Übersicht wird die Art des Ereignisses in der Titelleiste des Formularfensters angezeigt.

```
   ` Eine Formularmethode
Case of
   :(Form event=On Load)
      $vsTheEvent:="Formular wird angezeigt"
   :(Form event=On Unload)
      $vsTheEvent:="Ausgabeformular wurde verlassen, es verschwindet
      demnächst vom Bildschirm"
   :(Form event=On Display Detail)
      $vsTheEvent:="Datensatz # anzeigen"+String(Selected record number([TheTable]))
   :(Form event=On Menu Selected)
      $vsTheEvent:="Ein Menübefehl wurde ausgewählt"
   :(Form event=On Header")
      $vsTheEvent:="Kopfteil wird eingerichtet"
   :(Form event=On Clicked")
      $vsTheEvent:="Auf einen Datensatz wurde geklickt"
   :(Form event=On Double Clicked")
      $vsTheEvent:="Auf einen Datensatz wurde doppelgeklickt"
   :(Form event=On Open Detail)
      $vsTheEvent:="Datensatz #"+String(Selected record number([TheTable]))+" wurde doppelgeklickt"
   :(Form event=On Close Detail)
      $vsTheEvent:="Geht zurück in das Ausgabeformular"
   :(Form event=On Activate)
      $vsTheEvent:="Formularfenster wird zum vordersten Fenster"
   :(Form event=On Deactivate)
      $vsTheEvent:="Formularfenster ist nicht länger das vorderste Fenster"
   :(Form event=On Menu Selected)
      $vsTheEvent:="Ein Menübefehl wurde ausgewählt"
   :(Form event=On Outside Call)
      $vsTheEvent:="Aufruf von anderem Prozess wurde empfangen"
   Else
      $vsTheEvent:="Was passiert nun? Ereignis #"+String(Form event)
End case
SET WINDOW TITLE($vsTheEvent)
```

Beispiel 6

Beispiele zum Verwalten von Ereignissen On Before Keystroke und On After Keystroke finden Sie unter den Befehlen **Get edited text**, **Keystroke** und **FILTER KEYSTROKE**.

Beispiel 7

Dieses Beispiel zeigt, wie Sie Klicks und Doppelklicks genauso wie rollbare Bereiche verwenden können:

```
   ` Objektmethode asChoices für rollbaren Bereich
Case of
```

```
   :(Form event=On Load)
      ARRAY TEXT(...;asChoices;...)
   ` ...
      asChoices:=0
   :((Form event=On Clicked)|(Form event=On Double Clicked))
      If(asChoices#0)
   ` Ein Eintrag wurde angeklickt, führe hier etwas aus
   ` ...
      End if
   ` ...
End case
```

Beispiel 8

Dieses Beispiel zeigt, wie Sie Klicks und Doppelklicks unterschiedlich handhaben können. Mit dem Element Null behalten Sie das ausgewählte Element im Auge:

```
   ` Objektmethode asChoices für rollbaren Bereich
Case of
   :(Form event=On Load)
      ARRAY TEXT(...;asChoices;...)
   ` ...
      asChoices:=0
      asChoices{0}:="0"
   :(Form event=On Clicked)
      If(asChoices#0)
         If(asChoices#Num(asChoices))
   ` Ein neuer Eintrag wurde angeklickt, führe hier etwas aus
   ` ...
   ` Sichere das neu ausgewählte Element für das nächste Mal
               asChoices{0}:=String(asChoices)
         End if
      Else
         asChoices:=Num(asChoices{0})
      End if
   :(Form event=On Double Clicked)
      If(asChoices#0)
   ` Ein Eintrag wurde doppelgeklickt, führe hier etwas anderes aus
      End if
   ` ...
End case
```

Beispiel 9

Dieses Beispiel zeigt, wie Sie den Status eines Textbereichs mit den Ereignissen On Getting Focus und On losing Focus verwalten können:

```
   ` [Contacts];Formularmethode "Dateneingabe"
Case of
   :(Form event=On Load)
      C_TEXT(vtStatusArea)
      vtStatusArea:=""
   :(Form event=On Getting Focus)
      RESOLVE POINTER(Focus object;$vsVarName;$vlTableNum;$vlFieldNum)
      If(($vlTableNum#0) & ($vlFieldNum#0))
         Case of
            :($vlFieldNum=1) ` Datenfeld Nachname
               vtStatusArea:="Gib Nachnamen aus [Contacts] ein, er wird automatisch großgeschrieben"
   ` ...
            :($vlFieldNum=10) ` Datenfeld Postleitzahl
               vtStatusArea:="Gib fünfstellige Postleitzahl ein, sie wird  automatisch geprüft und bestätigt"
   ` ...
         End case
      End if
   :(Form event=On Losing Focus)
      vtStatusArea:=""
   ` ...
End case
```

Beispiel 10

Dieses Beispiel zeigt, wie Sie die Dateneingabe mit dem Ereignis On Close Box steuern können:

```
  ` Methode für Eingabeformular
$vpFormTable:=Current form table
Case of
  ` ...
  :(Form event=On Close Box)
    If(Modified record($vpFormTable->))
      CONFIRM("Dieser Datensatz wurde geändert. Änderungen sichern?")
      If(OK=1)
        ACCEPT
      Else
        CANCEL
      End if
    Else
      CANCEL
    End if
  ` ...
End case
```

Beispiel 11

Dieses Beispiel zeigt, wie Sie ein Datenfeld vom Typ Text oder alphanumerisch bei Änderung der Datenquelle in Großbuchstaben setzen:

```
  ` Objektmethode [Contacts]First Name
Case of
  ` ...
  :(Form event=On Data Change)
    [Contacts]First Name:=Uppercase(Substring([Contacts]First Name;1;1))]+Lowercase(Substring([Contacts]First Name;2))
  ` ...
End case
```

Beispiel 12

Dieses Beispiel zeigt, wie Sie ein Datenfeld vom Typ Text oder alphanumerisch bei Änderung der Datenquelle in Großbuchstaben setzen:

```
  ` Objektmethode [Contacts]First Name
Case of
  ` ...
  :(Form event=On Data Change)
    [Contacts]First Name:=Uppercase(Substring([Contacts]First Name;1;1))]+Lowercase(Substring([Contacts]First Name;2))
  ` ...
End case
```

Beispiel 13

Beispiel zum Verwalten eines Löschvorgangs in einer hierarchischen Liste:

```
  ... //Methode der hierarchischen Liste
:($Event=On Delete Action)
 ARRAY LONGINT($itemsArray;0)
 $Ref:=Selected list items(<>HL;$itemsArray;*)
 $n:=Size of array($itemsArray)

 Case of
   :($n=0)
     ALERT("Kein Eintrag ausgewählt")
     OK:=0
   :($n=1)
     CONFIRM("Wollen Sie diesen Eintrag löschen?")
   :($n>1)
     CONFIRM("Wollen Sie diese Einträge löschen?")
 End case
```

```
If(OK=1)
   For($i;1;$n)
      DELETE FROM LIST(<>HL;$itemsArray{$i};*)
   End for
End if
```

Beispiel 14

In diesem Beispiel synchronisert das Formularereignis On Scroll die Anzeige von zwei Bildern in einem Formular. Die Formularmethode enthält folgenden Code:

```
Case of
   :(Form event=On Scroll)
 //die Position des linken Bildes nehmen
      OBJECT GET SCROLL POSITION(*;"satellite";vPos;hPos)
 // und auf das rechte Bild anwenden
      OBJECT SET SCROLL POSITION(*;"plan";vPos;hPos;*)
End case
```

Ergebnis:

Beispiel 15

Die ausgewählte Zelle einer Listbox rot umrahmen und den Rahmen mitbewegen, wenn der Benutzer in der Listbox in vertikaler Richtung scrollt. In der Objektmethode zur Listbox schreiben Sie folgenden Code:

```
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)  //rotes Rechteck initialisieren
      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)
      $toAdd:=LISTBOX Get headers height(*;"LB1")  //Höhe des Kopfteils, damit es nicht überlappt
      If($ylb1+$toAdd<$y1) & ($ylb2>$y2)  //sind wir innerhalb der Listbox,
//verwalten wir der Einfachheit halber nur Kopfteile
//wir sollten auch horizontales Scrollen
//sowie Rollbalken verwalten
         OBJECT SET VISIBLE(*;"RedRect";True)
         OBJECT SET COORDINATES(*;"RedRect";$x1;$y1;$x2;$y2)
      Else
         OBJECT SET VISIBLE(*;"RedRect";False)
      End if

End case
```

Als Ergebnis folgt das rote Rechteck dem Scrollen der Listbox:

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 -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Gibt Wahr zurück, wenn die Ausführungsphase IN BREAK ist.

Beschreibung

In break gibt für die Ausführungsphase wahr zurück.

Damit die Ausführungsphase **In break** generiert wird, müssen Sie zuvor in der Designumgebung für das Formular und/oder die Objekte als Eigenschaft das Ereignis On Printing Break ausgewählt haben.

Hinweis: Diese Funktion entspricht der Verwendung von **Form event** und Testen, ob sie das Ereignis On Printing Break zurückgibt.

In footer

In footer -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Gibt Wahr zurück, wenn die Ausführungsphase In footer ist.

Beschreibung

In footer gibt Wahr für die Ausführungsphase In footer zurück.

Damit die Ausführungsphase **In footer** generiert wird, müssen Sie zuvor in der Designumgebung für das Formular und/oder die Objekte als Eigenschaft das Ereignis On Printing footer ausgewählt haben.

Hinweis: Diese Funktion entspricht der Verwendung von **Form event** und Testen, ob sie das Ereignis On Printing footer zurückgibt.

In header

In header -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Gibt Wahr zurück, wenn die Ausführungsphase IN HEADER ist.

Beschreibung

In header

Damit die Ausführungsphase **In header** generiert wird, müssen Sie zuvor in der Designumgebung für das Formular und/oder die Objekte als Eigenschaft das Ereignis On Header ausgewählt haben.

Hinweis: Diese Funktion entspricht der Verwendung von **Form event** und Testen, ob sie das Ereignis On Header zurückgibt.

Is waiting mouse up

Is waiting mouse up -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean	⮌	Wahr, wenn das Objekt auf ein Maus-Loslassen Ereignis wartet; sonst Falsch

Beschreibung

Die Funktion **Is waiting mouse up** gibt **Wahr** zurück, wenn das aktuelle Objekt angeklickt und die Maustaste nicht losgelassen wurde, während das Hauptfenster den Fokus hat. Sonst gibt sie **Falsch** zurück, insbesondere, wenn das Hauptfenster den Fokus verloren hat, bevor die Maustaste losgelassen wurde.

Die Funktion muss im Rahmen des aktuellen Objekts aufgerufen werden. Sie wurde eingerichtet zur Verwendung zusammen mit dem Formularereignis On Mouse Up, das für Felder oder Variablen vom Typ Bild verfügbar ist. Sie unterstützt im Code, wenn der Benutzer in ein Formularobjekt Bild geklickt hat, um etwas zu bewegen und diese Aktion durch ein externes Ereignis unterbrochen wird, wie z.B. ein Dialogfenster mit einer Meldung. In diesem Fall kann der interne Status des Objekts unendlich in Wartestellung bleiben, da er auf ein Maus-Loslassen Ereignis wartet, das nie passiert. Für solche Zwischenfälle müssen Sie Ihren Code zum Bewegen der Maus in einer Funktion **Is waiting mouse up** schützen, um sicher zu sein, dass die Aktion in einem gültigen Kontext ausgeführt wird.

Beispiel

Mit folgendem Code lässt sich das Nachverfolgen von Mausereignissen in einem Bildobjekt verwalten:

```
 //Objektmethode des Bildobjekts
C_LONGINT(vLtracking) //Flag für Modus zum Nachverfolgen
Case of
   :(Form event=On Clicked)
      If(Is waiting mouse up) //Die Maustaste wurde noch nicht losgelassen
         vLtracking:=1 //Wir sind im Modus zum Nachverfolgen
//... Hier Code zum Start für Nachverfolgung schreiben
      End if
   :(Form event=On Mouse Move)
      If(vLtracking=1) //Wir sind im Modus zum Nachverfolgen
         If(Not(Is waiting mouse up)) //Wir werden nie ein Maus-Loslassen haben
            vLtracking:=0 //Modus zum Nachverfolgen stoppen
//... Hier Code zum Verwalten oder Abbrechen von Nachverfolgen der Benutzeraktion schreiben
         Else //Das Objekt wartet noch auf ein Maus-Loslassen
//... Hier Code zum Nachverfolgen schreiben
         End if
      End if
   :(Form event=On Mouse Up) //Die Maustaste wurde losgelassen
//... Hier Code zum Abschließen der Nachverfolgung schreiben
      vLtracking:=0 //Ende des Modus zum Nachverfolgen
End case
```

Outside call

Outside call -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		Gibt Wahr zurück, wenn die Ausführungsphase OUTSIDE CALL ist.

Beschreibung

Damit die Ausführungsphase **Outside call** generiert wird, müssen Sie zuvor in der Designumgebung für das Formular und/oder die Objekte als Eigenschaft das Ereignis On Outside call ausgewählt haben.

Hinweis: Diese Funktion ist dasselbe wie **Form event** und Test, ob das Ereignis On Outside call zurückgegeben wird.

POST OUTSIDE CALL

POST OUTSIDE CALL (ProzessNr)

Parameter	Typ		Beschreibung
ProzessNr	Lange Ganzzahl	➡	Prozessnummer

Hinweis zur Kompatibilität

Dieser Befehl wurde umbenannt. In früheren 4D Versionen hieß er **CALL PROCESS**.

Beschreibung

Der Befehl **POST OUTSIDE CALL** ruft das Formular des vordersten Fensters von *ProzessNr* auf.

Wichtig: POST OUTSIDE CALL funktioniert nur zwischen Prozessen, die auf demselben Rechner laufen. Rufen Sie einen Prozess auf, der nicht existiert, wird nichts ausgeführt.

Zeigt *ProzessNr* (der gerufene Prozess) gerade kein Formular an, wird nichts ausgeführt. Das im gerufenen Prozess angezeigte Formular empfängt ein Ereignis On Outside call. Das funktioniert jedoch nur, wenn das Ereignis für dieses Formular in der Designumgebung im Fenster **Formulareigenschaften** aktiviert ist und das Ereignis in der zugeordneten Formularmethode verwaltet wird.

Hinweis: Das Ereignis On Outside call ändert den Eingabetext des empfangenden Eingabeformulars. Insbesondere bei Bearbeiten eines Feldes wird das Ereignis On Data change generiert.

Der rufende Prozess (der Prozess, in dem **POST OUTSIDE CALL** ausgeführt wird) "wartet" nicht. Er wirkt sich sofort aus. Bei Bedarf müssen Sie mit Interprozess- bzw. Prozessvariablen eine Warteschleife schreiben für die Antwort des aufgerufenen Prozesses. Die hierfür vorgesehenen Variablen können Sie mit den Befehlen **GET PROCESS VARIABLE** und **SET PROCESS VARIABLE** zwischen zwei Prozessen lesen und schreiben.
Mit den Befehlen **GET PROCESS VARIABLE** und **SET PROCESS VARIABLE** können Sie mit Prozessen kommunizieren, die keine Formulare anzeigen.

Tipp: POST OUTSIDE CALL hat die alternative Syntax **POST OUTSIDE CALL**(-1).
Um unnötigen Zeitaufwand zu vermeiden, zeichnet 4D die Interprozessvariablen bei einer Änderung nicht jedes Mal neu. Übergeben Sie im Parameter *ProzessNr* -1 anstelle einer Prozessnummer, führt 4D keinen "Outside call" aus. Stattdessen werden alle aktuell angezeigten Interprozessvariablen in allen Fenstern der Prozesse aktualisiert, die auf demselben Rechner laufen.

Beispiel

Siehe Beispiel zur **Semaphore**.

Right click

Right click -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		Wahr, wenn rechter Mausklick gefunden wurde, sonst Falsch

Beschreibung

Die Funktion **Right click** gibt *Wahr* zurück, wenn auf die rechte Maustaste geklickt wurde.

Diese Funktion sollte nur in Zusammenhang mit dem Formularereignis On clicked verwendet werden. Von daher muss im Strukturmodus geprüft werden, ob das Ereignis in den Formulareigenschaften bzw. im entsprechenden Objekt korrekt ausgewählt wurde.

SET TIMER

SET TIMER (ZähleTick)

Parameter	Typ		Beschreibung
ZähleTick	Lange Ganzzahl	➡	ZähleTick oder -1 = so bald wie möglich auslösen

Beschreibung

Der Befehl **SET TIMER** aktiviert das Formularereignis On Timer und setzt für den aktuellen Prozess die Anzahl Ticks zwischen jedem Formularereignis On Timer.

Hinweis: Weitere Informationen über dieses Formularereignis finden Sie in der Beschreibung zur Funktion **Form event**.

Rufen Sie diesen Befehl in einem Kontext auf, welcher kein Formular anzeigt, hat er keine Auswirkung.

Hinweis: Rufen Sie **SET TIMER** im Rahmen eines Unterformulars auf (Formularmethode des Unterformulars), wird das Formularereignis On Timer im Unterformular und nicht im Elternformular erzeugt.

Übergeben Sie in *ZähleTick* -1, aktiviert der Befehl das Formularereignis On Timer "so bald wie möglich", mit anderen Worten, sobald die 4D Anwendung die Steuerung an den Event-Manager übergeben hat. Das stellt sicher, dass ein Formular vollständig angezeigt wird, bevor es bearbeitet wird.

Wollen Sie das Auslösen des Formularereignisses On Timer per Programmierung deaktivieren, rufen Sie erneut **SET TIMER** auf und übergeben Sie in *ZähleTick* den Wert 0 (Null).

Beispiel

Sie möchten, dass der Rechner beim Anzeigen eines Formulars auf dem Bildschirm alle drei Sekunden ein Beep sendet. Dazu schreiben Sie folgende Formularmethode:

```
If(Form event=On Load)
   SET TIMER(60*3)
End if

If(Form event=On Timer)
   BEEP
End if
```

_o_During

_o_During -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		Gibt Wahr zurück, wenn die Ausführungsphase DURING ist.

Hinweis zur Kompatibilität

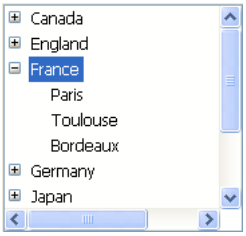
Diese Funktion ist veraltet (alter generischer Ausführungszyklus). Sie sollten stattdessen die Funktion **Form event** verwenden und prüfen, ob ein entsprechendes Ereignis, z.B. On Clicked zurückgegeben wird.

Hierarchische Listen

- Hierarchische Listen verwalten
- 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*

Hierarchische Listen verwalten

Hierarchische Listen sind Formularobjekte zum Anzeigen von Daten in Form einer Liste mit einer oder mehreren Ebenen, die sich auf- oder zuklappen lassen.



In Formularen dienen hierarchische Listen zur Anzeige oder Eingabe von Daten. Jeder Eintrag kann bis zu 2 Millionen Zeichen enthalten (maximale Größe eines Textfeldes) und mit einem Icon versehen werden. Sie unterstützen in der Regel Klicks, Doppelklicks, die Steuerung über Tastatur sowie Drag-and-Drop. Sie können auch nach dem Inhalt einer Liste suchen (Funktion **Find in list**).

Erstellen und Ändern

Hierarchische Listen lassen sich komplett per Programmierung erstellen (über die Funktionen **New list** oder **Copy list**) oder über Listen, die im Listeneditor des Designmodus definiert wurden (Funktion **Load list**).

Inhalt und Darstellung hierarchischer Listen werden über die Befehle und Funktionen dieses Kapitels per Programmierung gesteuert. Einige spezifische Darstellungsmerkmale lassen sich auch über die generischen Befehle im Kapitel **Objekte (Formulare)** steuern (siehe unten).

Über den Befehl **OBJECT SET LIST BY REFERENCE** oder **OBJECT SET LIST BY NAME** können Sie Referenzen auf hierarchische Listen dynamisch mit Formularobjekten aus Auswahllisten (Quellen, erforderliche und ausgeschlossene Werte) verbinden. Sie können auch Auswahllisten zuweisen, die über die Eigenschaftenliste im Listeneditor mit Formularobjekten definiert wurden.

ListRef und Objektname

Eine hierarchische Liste ist sowohl ein **Programmierspracheobjekt** im Speicher als auch ein **Formularobjekt**.
Auf das **Programmierspracheobjekt** wird mit der einmaligen internen ID *ListRef* vom Typ Lange Ganzzahl verwiesen. Diese ID wird von den Funktionen zurückgegeben, die zum Erstellen von Listen dienen: **New list**, **Copy list**, **Load list**, **BLOB to list**.

Es gibt im Speicher immer nur eine Instanz des Programmierspracheobjekts und jede Änderung im Objekt erscheint sofort an allen Stellen, wo es verwendet wird.
Das **Formularobjekt** ist hingegen nicht zwingend einmalig: Es kann mehrere Darstellungen derselben hierarchischen Liste in einem oder verschiedenen Formularen geben. In der Programmiersprache wird das Objekt, wie andere Formularobjekte, mit der Syntax (*;"ListName", etc.) angegeben.

Sie können die hierarchische Liste "Programmierspracheobjekt" mit der hierarchischen Liste "Formularobjekt" über eine Variable mit dem Wert *ListRef* verbinden. Schreiben Sie zum Beispiel:

```
mylist:=New list
```

... können Sie den Variablennamen *mylist* einfach mit dem Formularobjekt der hierarchischen Liste in der Eigenschaftenliste verbinden, so dass dieser das Programmierspracheobjekt steuert, dessen *ListRef* in *mylist* gespeichert ist.

Jede Darstellung der Liste hat Merkmale, die für alle Darstellungen gleich sind, und solche, die nur für eine bestimmte Darstellung gelten.

Folgende Merkmale sind für jede Darstellung der Liste spezifisch:

- Auswahl
- Status auf-/zugeklappt der einzelnen Zeilen
- Position der Rollbox

Andere Merkmale wie Schrift, Eingabekontrolle, Inhalt der Liste, Icons, etc. und die Attribute, die auf der Ebene des Programmierspracheobjekts (*ListRef*) definiert sind, sind für alle Darstellungen gleich und lassen sich nicht einzeln ändern.

Verwenden Sie Befehle, die mit auf-/zugeklappten Einträgen oder dem aktuellen Eintrag arbeiten, wie z.B. **Count list items** (ohne den Parameter *), muss gewährleistet sein, dass die zu verwendende Darstellung in eindeutiger Weise bestimmbar ist. Wollen Sie die hierarchische Liste im Speicher angeben, müssen Sie die ID *ListRef* mit Programmiersprachebefehlen verwenden. Wollen Sie die Darstellung einer hierarchischen Liste auf Formularebene angeben, müssen Sie im Befehl den Objektnamen vom Typ String mit der Syntax (*;"ListName", etc.) verwenden. Das ist dieselbe Syntax wie für die Befehle im Kapitel **Objekte (Formulare)**. Die meisten Befehle im Kapitel "hierarchische Listen", die mit den Listeneigenschaften arbeiten, verwenden diese Syntax. Ausführliche Informationen dazu finden Sie in der Beschreibung der einzelnen Befehle.

Warnung: Beachten Sie, dass bei Befehlen, die Eigenschaften setzen, die Syntax mit dem Objektnamen nicht bedeutet, dass der Befehl nur das angegebene Formularobjekt ändert. Die Aktion des Befehls richtet sich vielmehr nach dem Status des Objekts. So werden die Merkmale für hierarchische Listen immer in allen Darstellungen geändert.
Übergeben Sie z.B. die Anweisung **SET LIST ITEM FONT** (*;"mylist1";*;thefont), bedeutet das, dass Sie die Schrift eines Eintrags in der hierarchischen Liste ändern wollen, die dem Formularobjekt *mylist1* zugeordnet ist. Der Befehl berücksichtigt den aktuellen Eintrag in *mylist1* für die Änderung. Sie wird aber auch auf alle Darstellungen der Liste in allen Prozesse ausgeführt.

Unterstützung von @

Sie können, wie in Routinen zur Verwaltung von Objekteigenschaften, im Parameter *Objektname* das Zeichen "@" verwenden. Damit bestimmen Sie in der Regel einen Satz Objekte im Formular. Bei Befehlen für hierarchische Listen ist das jedoch nicht für alle Fälle geeignet. Diese Syntax hat je nach Befehlsart zwei unterschiedliche Auswirkungen:

- Bei Befehlen, die Eigenschaften setzen, bezeichnet diese Syntax standardmäßig alle Objekte mit dazu passendem Namen. So bestimmt der Parameter "LH@" alle Objekte des Typs hierarchische Liste, deren Name mit "LH" beginnt. Das gilt für folgende Befehle:
  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
- Bei Befehlen und Funktionen, die Eigenschaften liefern, bezeichnet diese Syntax das erste Objekt mit dem dazu passenden Namen. Das gilt für folgende Befehle und Funktionen:
  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

Generische Befehle für hierarchische Listen

Sie können die Darstellung einer hierarchischen Liste in einem Formular mit mehreren generischen 4D Befehlen verändern. Sie übergeben in diesen Befehlen entweder den Objektnamen der hierarchischen Liste (mit dem Parameter *) oder ihren Variablennamen (Standardsyntax).

Hinweis: Bei hierarchischen Listen enthält die Variable des Formulars den Wert *ListRef*. Führen Sie einen Befehl aus, der ein Attribut über die Variable der hierarchischen Liste verändert, lässt sich bei mehrfacher Darstellung keine Zielliste festlegen. Nur der Objektname ermöglicht die individuelle Unterscheidung zwischen den verschiedenen Darstellungen.

Folgende Befehle lassen sich mit hierarchischen Listen verwenden:
OBJECT SET FONT
OBJECT SET FONT STYLE
OBJECT SET FONT SIZE
OBJECT SET COLOR
OBJECT SET FILTER
OBJECT SET ENTERABLE
OBJECT SET SCROLLBAR
OBJECT SET SCROLL POSITION
OBJECT SET RGB COLORS

Hinweis: Diese Befehle verändern alle Darstellungen derselben Liste, auch wenn Sie nur eine Liste über ihren Objektnamen angeben. Davon ausgenommen ist der Befehl **OBJECT SET SCROLL POSITION**.

Priorität bei Befehlen für Eigenschaften

Bestimmte Eigenschaften einer hierarchischen Liste, wie z.B. das Attribut „Eingebbar" oder die Farbe, lassen sich auf drei Arten setzen: Über die Eigenschaftenliste im Designmodus, über einen Befehl aus dem Kapitel **Objekte (Formulare)** oder einen Befehl aus dem Kapitel "Hierarchische Listen".

Dabei gilt folgende Priorität in der Reihenfolge:

1. Befehle aus dem Kapitel "Hierarchische Listen"
2. Generische Befehle aus dem Kapitel **Objekte (Formulare)**
3. Optionen der Eigenschaftenliste

Diese Priorität gilt unabhängig von der Reihenfolge, in der die Befehle aufgerufen werden. Ändern Sie die Eigenschaft einer Zeile individuell mit einem Befehl für hierarchische Listen, hat der entsprechende Befehl für Objekteigenschaft keine Auswirkung auf diese Zeile, selbst wenn er danach aufgerufen wird. Ändern Sie z.B. die Farbe einer Zeile mit dem Befehl **SET LIST ITEM PROPERTIES**, hat der Befehl **OBJECT SET COLOR** keine Auswirkung auf diese Zeile.

Einträge nach Position oder Referenz verwalten

Es gibt zwei Möglichkeiten, um mit dem Inhalt hierarchischer Listen zu arbeiten: nach Position oder nach Referenz.

- Bei Position verwendet 4D zum Identifizieren die Position in Bezug auf die Einträge der Liste auf dem Bildschirm. Das Ergebnis ist unterschiedlich, je nachdem, ob Einträge in der Liste aufgeklappt oder zugeklappt sind. Beachten Sie, dass bei mehrfachen Darstellungen jedes Formularobjekt eine eigene Konfiguration auf- bzw. zugeklappter Einträge hat.
- Bei Referenz verwendet 4D die ID *EintragRef* der Listeneinträge. Jeder Eintrag lässt sich so individuell bestimmen, unabhängig von seiner Position oder der Darstellung in der hierarchischen Liste.

Referenznummern der Einträge verwenden (EintragRef)

Jeder Eintrag einer hierarchischen Liste hat eine Referenznummer (*EintragRef*) vom Typ Lange Ganzzahl. Dieser Wert dient nur für Ihren eigenen Gebrauch: 4D stellt diesen lediglich bereit.

Warnung: Sie können als Referenznummer einen beliebigen Wert vom Typ Lange Ganzzahl verwenden, außer 0 (Null). Die meisten Befehle dieses Kapitels verwenden den Wert 0 zur Angabe des zuletzt in der Liste hinzugefügten Eintrags.

Hier ein paar Tipps zur Verwendung von Referenznummern:

1. Sie müssen nicht jeden Eintrag mit einer einmaligen Nummer identifizieren (Anfängerebene).

- **Erstes Beispiel:** Sie erstellen per Programmierung ein System mit Registerkarten, z.B. ein Adressbuch. Da das System die Zahl der gewählten Registerkarte zurückgibt, benötigen Sie keine Referenznummern für die Einträge: Übergeben Sie im Parameter *EintragRef* einen beliebigen Wert außer 0. Für ein Adressbuch können sich auch im Designmodus eine Liste mit A, B, ... vordefinieren. Sie können diese auch per Programmierung erstellen, um die Buchstaben, für die es keine Datensätze gibt, zu entfernen.
- **Zweites Beispiel:** Beim Arbeiten mit einer Datenbank bauen Sie nach und nach eine Liste mit Kennwörtern auf. Sie können diese Liste am Ende jeder Sitzung über den Befehl **SAVE LIST** oder **LIST TO BLOB** speichern und zu Beginn der nächsten Sitzung über die Funktionen **Load list** oder **BLOB to list** erneut laden. Sie können diese Liste in einem Palettenfenster anzeigen. Klickt ein Benutzer dann auf ein Kennwort in der Liste, wird der gewählte Eintrag in den eingebbaren Bereich eingefügt, der im Prozess im Vordergrund ausgewählt ist. Sie können auch Drag&Drop verwenden. In jedem Fall ist es wichtig, dass Sie nur den ausgewählten Eintrag bearbeiten (per Klick oder Drag-and-Drop), da die Routinen **Selected list items** (bei Klick) und **DRAG AND DROP PROPERTIES** die Position des Eintrags zurückgeben, den Sie bearbeiten müssen. Verwenden Sie den Wert dieser Position, um den Titel des Eintrags über den Befehl **GET LIST ITEM** zu erhalten. Auch hier müssen Sie nicht jeden Eintrag individuell identifizieren, im Parameter *EintragRef* können Sie einen beliebigen Wert - außer 0 (Null) - übergeben.

2. Sie müssen die Einträge der Liste z.T. identifizieren (mittlere Ebene).
Verwenden Sie die Referenznummer der Einträge zum Speichern von Informationen, müssen Sie mit dem Eintrag arbeiten. Weitere Informationen dazu finden Sie im Beispiel zum Befehl **APPEND TO LIST**. In diesem Beispiel verwenden wir die Referenznummern zum Speichern von Datensatznummern. Es muss jedoch möglich sein, zwischen Einträgen in Datensätzen zur Tabelle [Abteilung] und [Angestellte] zu unterscheiden.

3. Sie müssen alle Einträge der Liste individuell identifizieren (fortgeschrittene Ebene).
Sie programmieren eine fortgeschrittene Verwaltung für hierarchische Listen, die ermöglicht, jeden Eintrag auf jeder Ebene der Liste individuell zu identifizieren. Der einfachste Weg ist ein eigener Zähler. Angenommen Sie erstellen eine Liste *hlList* mit der Funktion **APPEND TO LIST**. In diesem Stadium initialisieren Sie einen Zähler *vhlCounter* auf 1. Immer wenn Sie **APPEND TO LIST** oder **INSERT IN LIST** aufrufen, erhöhen Sie diesen Zähler um 1 (*vhlCounter:=vhlCounter+1*) und übergeben den Zählerwert als Referenznummer des Eintrags. Der Trick dabei ist, dass der Zähler nie zurückgestuft wird, wenn Einträge gelöscht werden — der Zähler kann nur erhöht werden. Da diese Nummern vom Typ Lange Ganzzahl sind, können Sie in einer initialisierten Liste bis zu 2 Millionen Einträge einfügen. (Bei solch einer hohen Anzahl von Einträgen empfehlen wir jedoch, nicht mehr mit einer Liste, sondern mit einer Tabelle zu arbeiten.)

Hinweis: Verwenden Sie **Bit Operatoren**, können Sie Referenznummern für Einträge auch zum Speichern von Information verwenden, die sich in eine Lange Ganzzahl setzen lässt, z.B. 2 Ganzzahlen, 4-byte Werte oder 32 Booleans.

Wann sind einmalige Referenznummern notwendig?

In den meisten Fällen, bei Verwendung hierarchischer Listen für Benutzerzwecke und wenn nur der aktuelle Eintrag bearbeitet wird (der angeklickte oder per Drag-and-Drop bewegte Eintrag), benötigen Sie überhaupt keine Referenznummern für Einträge. Hier reichen die Routinen **Selected list items** und **GET LIST ITEM** vollkommen aus. Außerdem ermöglichen die Befehle **INSERT IN LIST** und **DELETE FROM LIST** die Bearbeitung der Liste in Bezug auf den gewählten Eintrag.

Im allgemeinen benötigen Sie Referenznummern für Einträge nur, wenn Sie direkt auf einen beliebigen Eintrag per Programmierung zugreifen wollen, der nicht zwingenderweise der aktuell in der Liste ausgewählte Eintrag ist.

APPEND TO LIST

APPEND TO LIST (Liste ; EintragText ; EintragRef {; Unterliste ; Erweitert})

Parameter	Typ		Beschreibung
Liste	ListRef	➡	Referenznummer für Liste
EintragText	String	➡	Text für neuen Listeneintrag
EintragRef	Lange Ganzzahl	➡	Einmalige Referenznummer für neuen Eintrag in Liste
Unterliste	ListRef	➡	Optionale Unterliste für neuen Eintrag in Liste
Erweitert	Boolean	➡	Gibt an, ob die optionale Unterliste erweitert oder geschlossen ist

Beschreibung

Der Befehl **APPEND TO LIST** fügt in der hierarchischen Liste einen neuen Eintrag an mit der in *Liste* übergebenen Referenznummer.

In *EintragText* übergeben Sie den Text für den Eintrag. Sie können einen String oder Textausdruck bis zu 2 Millionen übergeben. Ab 4D v16 R4 können Sie in *EintragText* die Konstante ak standard action title übergeben, wenn dem Eintrag eine Standardaktion zugewiesen ist, um automatisch den lokalisierten Aktionsnamen zu verwenden. Weitere Informationen dazu finden Sie im Abschnitt **Standardaktionen**.

In *EintragRef* übergeben Sie die einmalige Referenznummer für den Eintrag (vom Typ Lange Ganzzahl). Auch wenn diese Referenznummer als einmalig definiert ist, können Sie einen beliebigen Wert übergeben. Weitere Informationen finden Sie im Abschnitt **Hierarchische Listen verwalten**.

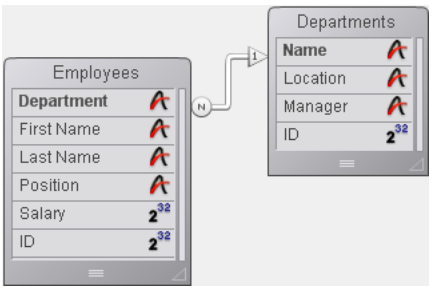
Soll ein Eintrag auch untergeordnete Einträge haben, übergeben Sie in *Unterliste* eine gültige Referenznummer. In diesem Fall müssen Sie auch den Parameter *Erweitert* setzen. Übergeben Sie TRUE oder FALSE, damit die Unterliste auf- bzw. zugeklappt erscheint.
Die in *Unterliste* angegebene Referenznummer muss sich auf eine bestehende Liste beziehen. Das kann eine Liste auf einer Ebene oder eine Liste mit Unterlisten sein. Wollen Sie dem neuen Eintrag keine Liste zuordnen, übergeben Sie keinen Parameter oder den Wert Null (0).

Tipps

- Mit dem Befehl **INSERT IN LIST** fügen Sie einen neuen Eintrag in die Liste ein. Mit dem Befehl **SET LIST ITEM** ändern Sie den Text eines vorhandenen Eintrags oder der zugeordneten Unterliste und deren Status auf- bzw. zugeklappt.
- Mit dem Befehl **SET LIST ITEM PROPERTIES** ändern Sie die Darstellung des neu hinzugefügten Eintrags.

Beispiel

Hier ist die Teilansicht einer Datenbankstruktur:



Die beiden Tabellen [Departments] und [Employees] enthalten folgenden Datensätze:

Name :	Location :	Manager :
Marine Biology	Aquarium B	Robert Masterson
Accounting	3rd floor	Anne Weston
Sales	First floor-West	George Jackson

Department :	First Name :	Last Name :
Marine Biology	Andrew	Parker
Accounting	Jessica	Anders
Sales	Peter	Parker
Marine Biology	Jeffrey	Dalton
Accounting	Maria	Peterson
Sales	Jordan	Solomon
Sales	Patrick	McDonald
Marine Biology	Henry	Paulson
Sales	Marlo	Robertson

Sie wollen eine hierarchische Liste mit Namen *hlList* anzeigen, die die Abteilungen auflistet und für jede Abteilung eine untergeordnete Liste mit den jeweiligen Angestellten. Die Objektmethode von *hlList* lautet:

```
` Objektmethode hierarchische Liste hlList

Case of

   :(Form event=On Load)
```

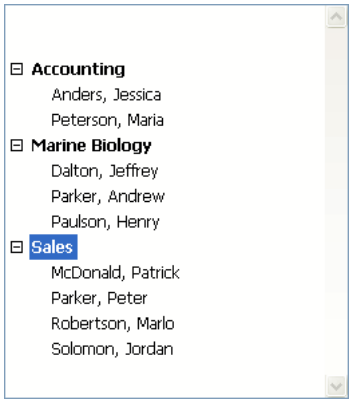
```
      C_LONGINT(hlList;$hSubList;$vlDepartment;$vlEmployee);$vlDepartmentID
   ` Erstelle neue leere hierarchische Liste
      hlList:=New list
   ` Wähle alle Datensätze aus der Tabelle [Departments]
      ALL RECORDS([Departments])
   ` Wähle für jede Abteilung
      For($vlDepartment;1;Records in selection([Departments]))
   ` die dazugehörigen Angestellten
         RELATE MANY([Departments]Name)
   ` Wie viele sind es?
         $vlNbEmployees:=Records in selection([Employees])
   ` Gibt es in dieser Abteilung mindestens einen Angestellten?
         If($vlNbEmployees>0)
   ` Erstelle untergeordnete Liste für den Eintrag Abteilung
            $hSubList:=New list
   ` Füge für jeden Angestellten
            For($vlEmployee;1;Records in selection([Employees]))
   ` den Eintrag Angestellter in die untergeordnete Liste ein
   ` Die Nummer des Datensatzes aus [Employees] wird als Referenznummer für den Eintrag übergeben
               APPEND TO LIST($hSubList;[Employees]Last Name+", "+[Employees]First Name;([Employees]ID)
   ` Gehe zum nächsten Datensatz [Employees]
               NEXT RECORD([Employees])
            End for
         Else
   ` Keine Angestellten, folglich keine untergeordnete Liste für den Eintrag Abteilung
            $hSubList:=0
         End if
   ` Füge Eintrag Abteilung in übergeordnete Liste ein
   ` Die Nummer des Datensatzes aus [Departments]
   ` wird als Referenznummer für den Eintrag übergeben. Das Bit #31
   ` der Referenznummer für den Eintrag wird auf Eins gesetzt,
   ` um die Einträge Abteilung und Angestellte voneinander zu unterscheiden.
   ` Siehe auch Fußnote, wie dieses Bit zusätzliche Informationen über den Eintrag liefern kann.
         APPEND TO LIST(hlList;[Departments]Name;[Departments]ID?+31;$hSublist;$hSubList#0)
   ` Setze den Eintrag Abteilung zur Hervorhebung der Hierarchie in Fettschrift
         SET LIST ITEM PROPERTIES(hlList;0;False;Bold;0)
   ` Gehe zur nächsten Abteilung
         NEXT RECORD([Departments])
      End for
   ` Sortiere die ganze Liste in aufsteigender Reihenfolge
      SORT LIST(hlList;>)
   ` Zeige die Liste in Windows-Darstellung an und lege als Mindesthöhe für Zeilen 14 Pts fest
      SET LIST PROPERTIES(hlList;Ala Windows;Windows node;14)

   :(Form event=On Unload)
   ` Die Liste wird nicht mehr benötigt; vergessen Sie nicht, sie zu entfernen!
      CLEAR LIST(hlList;*)

   :(Form event=On Double Clicked)
   ` Ein Doppelklick ist erfolgt
   ` Erhalte Position des gewählten Eintrags
      $vlItemPos:=Selected list items(hlList)
   ` Prüfe auf alle Fälle die Position
      If($vlItemPos#0)
   ` Hole Information über Eintrag der Liste
         GET LIST ITEM(hlList;$vlItemPos;$vlItemRef;$vsItemText;$vlItemSubList;$vbItemSubExpanded)
   ` Ist es ein Eintrag Abteilung?
         If($vlItemRef??31)
   ` Wenn ja, ist es ein Doppelklick auf einen Eintrag in Abteilung
            ALERT("Sie haben auf den Eintrag Abteilung doppelgeklickt"+Char(34)+$vsItemText+Char(34)+".")
         Else
   ` Wenn nicht, ist es Doppelklick auf einen Eintrag in Angestellte
   ` Finde den Datensatz aus [Departments] mit der übergeordneten Referenznummer
            $vlDepartmentID:=List item parent(hlList;$vlItemRef)?-31
            QUERY([Departments];[Departments]ID=$vlDepartmentID)
   ` Teile mit, wo der Angestellte arbeitet und wer der Vorgesetzte ist
            ALERT("Sie haben auf den Eintrag Angestellte doppelgeklickt"+Char(34)+$vsItemText+Char(34)+" wer arbeitet in der
Abteilung "+Char(34)+[Departments]Name+Char(34)+" mit dem Vorgesetzten "+Char(34)+[Departments]Manager+Char(34)+".")
         End if
      End if
End case
```

`Hinweis: 4D kann pro Tabelle bis zu 1 Milliarde Datensätze speichern. Wir verwenden hier das Bit #31 des unbenutzten hohen Bits zur Unterscheidung der Einträge Angestellte und Abteilung.

In diesem Beispiel gibt es nur einen Grund, die Einträge aus [Departments] von den Einträgen [Employees] zu unterscheiden:

1. Wir speichern Datensatznummern in den Referenznummern für Einträge. Das kann zur Folge haben, dass Einträge aus [Departments] dieselben Referenznummer wie Einträge aus [Employees] haben.
2. Mit der Funktion **List item parent** finden Sie den Eintrag, der dem ausgewählten Eintrag übergeordnet ist. Klicken wir auf einen Eintrag in [Employees] mit der Datensatznummer #10 und gibt es auch einen Eintrag in [Departments] mit #10, findet **List item parent** beim Durchlaufen der Listen zuerst den Eintrag [Departments], da nach der dem Eintrag übergebenen Referenznummer gesucht wird. Die Funktion gibt die Überordnung zum Eintrag in [Departments] und nicht die Überordnung zum Eintrag in [Employees] zurück.

Wir haben also einmalige Referenznummern für Einträge angelegt, um zwischen den Datensätzen [Departments] und [Employees] unterscheiden zu können. Nach Ausführung des Formulars sieht die Liste folgendermaßen aus:



Hinweis: Dieses Beispiel eignet sich für die Benutzeroberfläche bei kleiner Anzahl an Datensätzen. Beachten Sie, dass Listen im Speicher gehalten werden — erstellen Sie deshalb keine hierarchischen Listen mit Tausenden von Einträgen.

CLEAR LIST

CLEAR LIST (Liste {; *})

Parameter	Typ		Beschreibung
Liste	ListRef	➡	Referenznummer der Liste
*		➡	Löscht evtl. vorhandene Unterlisten aus dem Speicher. Ohne Angabe werden evtl. vorhandene Unterlisten nicht gelöscht

Beschreibung

Der Befehl **CLEAR LIST** gibt hierarchische Listen mit der in *Liste* übergebenen Referenznummer frei.

Im Normalfall übergeben Sie den optionalen * Parameter, damit alle evtl. vorhandenen Unterlisten zu Einträgen oder Untereinträgen der Liste ebenfalls freigegeben werden.

Eine Liste, die einem Formularobjekt über das Fenster Objekteigenschaften zugeordnet wurde, lädt und löscht 4D automatisch. Sie benötigen dafür nicht den Befehl **CLEAR LIST**. Rufen Sie **CLEAR LIST** jedoch immer auf, wenn Sie eine Liste laden, kopieren, aus einem BLOB entnehmen oder per Programmierung erstellen und bearbeitet haben.

Wollen Sie eine Unterliste löschen, die einem Eintrag (auf beliebiger Ebene) einer anderen Liste im aktuell angezeigten Formular zugeordnet ist, gehen Sie folgendermaßen vor:

1. Wählen Sie **GET LIST ITEM** für den entsprechenden Eintrag, um die Referenz der Unterliste zu erhalten.
2. Wählen Sie **SET LIST ITEM** für den entsprechenden Eintrag, um die Unterliste vor dem Löschen vom Eintrag der Liste zu trennen.
3. Wählen Sie **CLEAR LIST**, um die Unterliste mit der über **GET LIST ITEM** erhaltenen Referenznummer zu löschen.

Beispiel 1

Sie haben eine sog. Aufräumroutine, die alle nicht länger benötigten Objekte und Daten löscht, z.B. wenn ein Fenster geschlossen und ein Formular aus dem Speicher entfernt wird. Unter Umständen löschen Sie erneut eine hierarchische Liste, die je nach Aktion des Benutzers im Formular bereits gelöscht wurde. Mit **Is a list** wird die Liste nur wenn erforderlich gelöscht:

```
` Auszug aus der Aufräumroutine
If(Is a list(hlList))
   CLEAR LIST(hlList;*)
End if
```

Beispiel 2

Siehe Beispiel zur Funktion **Load list**.

Beispiel 3

Siehe Beispiel zur Funktion **BLOB to list**.

Copy list

Copy list (Liste) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Liste	ListRef	→	Referenz der zu kopierenden Liste
Funktionsergebnis	ListRef	↩	Referenznummer der duplizierten Liste

Beschreibung

Die Funktion **Copy list** dupliziert die Liste mit der in *Liste* übergebenen Referenznummer und gibt die Referenznummer der neuen Liste zurück.

Rufen Sie nach dem Bearbeiten der neuen Liste den Befehl **CLEAR LIST** auf, um die Liste freizugeben.

Count list items

Count list items ({* ;} Liste {; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String). Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Referenznummer der Liste (ohne *) oder Objektname der Liste (mit *)
*	Operator	➡	Ohne * (Standard): Gibt sichtbare Einträge zurück (erweitert). Mit *: Gibt alle Listeneinträge zurück
Funktionsergebnis	Lange Ganzzahl	⮌	Ohne 2ten *: Anzahl der sichtbaren (erweiterten) Listeneinträge. Mit 2tem *: Gesamtzahl der Listeneinträge

Beschreibung

Die Funktion **Count list items** gibt entweder die Anzahl der aktuell "sichtbaren" oder alle Einträge in der Liste mit der in *Liste* übergebenen Referenznummer oder dem Objektnamen zurück.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung einer Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (ListRef). Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Arten verwenden. Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

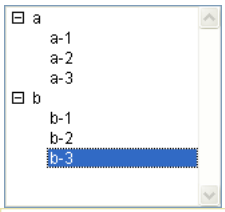
Hinweis: Verwenden Sie das Zeichen @ im Namen der Liste, erhalten Sie einen Satz Objekte im Formular, zu denen der Name passt. **Count list items** wird aber nur auf das erste Objekt mit dem passenden Namen angewandt.

Mit dem zweiten Parameter * gibt die Funktion alle Einträge in der Liste zurück, egal ob diese auf- oder zugeklappt ist.
Ohne den Parameter * gibt **Count list items** nur die Anzahl der Einträge zurück, die je nach Erweiterung der Liste und der dazugehörigen Unterlisten sichtbar sind.

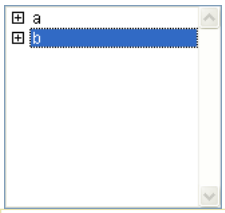
Wenden Sie diese Funktion auf eine Liste in einem Formular an.

Beispiele

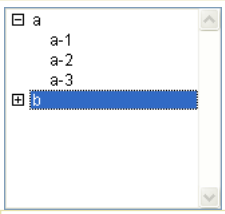
In der Anwendungsumgebung gibt es die Liste *hList*:




```
$vlNbItems:=Count list items(hList)  ` $vlNbItems erhält 8
$vlNbItems:=Count list items(hList;*)  ` $vlNbItems erhält ebenfalls 8
```




```
$vlNbItems:=Count list items(hList)  ` $vlNbItems erhält hier 2
$vlNbItems:=Count list items(hList;*)  ` $vlNbItems erhält weiterhin 8
```




```
$vlNbItems:=Count list items(hList)  ` $vlNbItems erhält hier 5
$vlNbItems:=Count list items(hList;*)  ` $vlNbItems erhält weiterhin 8
```

DELETE FROM LIST

DELETE FROM LIST ({* ;} Liste ; EintragRef | * {; *})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String). Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Referenznummer der Liste (ohne *) oder Objektname der Liste (mit * Stern)
EintragRef | *	Lange Ganzzahl, Operator	➡	Referenznummer des Eintrags oder 0 für den zuletzt hinzugefügten Eintrag oder * für den aktuellen Eintrag der Liste
*		➡	Mit *: entfernt Unterlisten aus dem Speicher. Ohne *: entfernt Unterlisten nicht

Beschreibung

Der Befehl **DELETE FROM LIST** löscht einen Eintrag aus der Liste mit der in *Liste* übergebenen Referenznummer oder Objektname.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung einer Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (*ListRef*). Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Arten verwenden. Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktuellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

Übergeben Sie in *EintragRef* einen Stern (*), löschen Sie den aktuell ausgewählten Eintrag in der Liste. Sie können auch 0 übergeben, damit der zuletzt hinzugefügte Eintrag gelöscht wird.
Sonst übergeben Sie die Referenznummer des zu löschenden Eintrags. Gibt es keinen Eintrag für diese Referenznummer, hat der Befehl keine Auswirkung.

Arbeiten Sie mit Referenznummern, sollten diese einmalig sein, damit Sie die Einträge voneinander unterscheiden können. Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **APPEND TO LIST**.

Übergeben Sie – unabhängig vom zu löschenden Eintrag – immer den optionalen Parameter *, damit 4D automatisch eine evtl. zugeordnete Unterliste löscht. Andernfalls sollten Sie zuvor die Referenznummer der evtl. vorhandenen Unterliste zu diesem Eintrag erhalten haben, damit Sie diese bei Bedarf mit dem Befehl **CLEAR LIST** löschen können.

Beispiel

Folgender Code löscht den aktuell ausgewählten Eintrag der Liste *hList*. Eine evtl. zugeordnete Unterliste wird ebenfalls gelöscht:

```
DELETE FROM LIST(hList;*;*)
```

Find in list

Find in list ({* ;} Liste ; Wert ; Reichweite {; ZeilenArray {; *}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String). Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Mit *: Objektname der Liste. Ohne *: Referenznummer der Liste
Wert	String	➡	Zu suchender Wert
Reichweite	Ganzzahl	➡	0=Hauptliste, 1=Unterliste
ZeilenArray	Array Lange Ganzzahl	⬅	Ohne 2. *: Array mit Positionen der gefundenen Zeilen Mit 2. *: Array mit Referenznummer der gefundenen Zeilen
*	Operator	➡	Ohne *: Verwende Position der Zeilen Mit *: Verwende Referenznummer der Zeilen
Funktionsergebnis	Lange Ganzzahl	↩	Ohne 2. *: Position der gefundenen Zeilen Mit 2. *: Referenznummer der gefundenen Zeilen

Beschreibung

Die Funktion **Find in list** gibt die Position oder Referenz der ersten Zeile in der Liste zurück, die dem in *Wert* übergebenen String entspricht. Werden mehrere Zeilen gefunden, kann die Funktion auch ein Array *ZeilenArray* mit der Position oder Referenz jeder Zeile füllen.

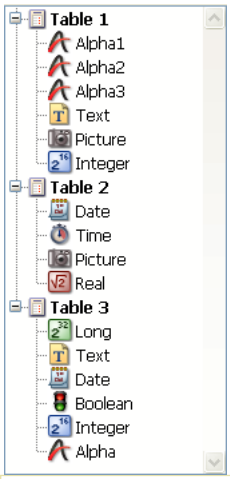
Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung der Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (ListRef). Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Referenznummern für Einträge (der zweite * ist nicht übergeben), können Sie beide Arten verwenden.
Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit Positionen von Einträgen (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da die Position von Einträgen von einer Darstellung zur nächsten variieren kann.

Hinweis: Verwenden Sie das Zeichen @ im Objektnamen der Liste und enthält das Formular mehrere Listen, zu denen dieser Name passt, wird **Find in list** auf das erste Objekt mit dem passenden Namen angewandt.

Mit dem zweiten * definieren Sie, ob Sie mit den aktuellen Positionen der Zeilen (* nicht übergeben) oder mit den absoluten Referenzen der Zeilen (* übergeben) arbeiten wollen.
Im Parameter *Wert* übergeben Sie die zu suchenden Zeichenketten. Die Suche ist vom Typ „ist genau", d.h. die Suche nach „Holz" findet nicht das Wort „Holzhaus". Sie können jedoch das Jokerzeichen (@) für Suchläufe vom Typ "beginnt mit", "endet mit" oder "enthält" verwenden.
Der Parameter *Reichweite* legt fest, ob die Suche nur auf der ersten Ebene der Liste oder auch mit allen Unterlisten ausgeführt wird. Mit 0 (Null) begrenzen Sie die Suche auf die erste Ebene der Liste, mit 1 wird sie auch auf alle Unterlisten ausgeweitet.
Um die Position oder Zahl aller Zeilen zu suchen, die Wert entsprechen, übergeben Sie im optionalen Parameter *ZeilenArray* ein Array vom Typ Lange Ganzzahl. Die Funktion erstellt bei Bedarf das Array und passt es an. Sie füllt das Array mit den Positionen (der zweite * ist nicht übergeben) oder den Referenznummern (der zweite * ist übergeben) der gefundenen Einträge.
Die Positionen werden in Bezug auf den obersten Eintrag in der Hauptliste angegeben. Dabei wird auch der aktuelle Status auf-/zugeklappt der Liste und Unterlisten berücksichtigt.
Entspricht keine Zeile dem gesuchten Wert, gibt die Funktion 0 (Null) zurück und das Array *ZeilenArray* wird leer zurückgegeben.

Beispiel

Wir gehen von folgender hierarchischen Liste aus:




```
$vlItemPos:=Find in list(hList;"P@";1;$arrPos)
  `$vlItemPos ist gleich 6
  `$arrPos{1} ist gleich 6 und $arrPos{2} ist gleich 11
$vlItemRef:=Find in list(hList;"P@";1;$arrRefs;*)
  `$vlItemRef ist gleich 7
  `$arrRefs{1} ist gleich 7 und $arrRefs{2} ist gleich 18
```

```
$vlItemPos:=Find in list(hList;"Date";1;$arrPos)
  `$vlItemPos ist gleich 9
  `$arrPos{1} ist gleich 9 und $arrPos{2} ist gleich 16
$vlItemRefFind in list(hList;"Date";1;$arrRefs;*)
  `$vlItemRef ist gleich 11
  `$arrRefs{1} ist gleich 11 und $arrRefs{2} ist gleich 23
$vlItemPos:=Find in list(hList;"Date";0;*)
  `$vlItemPos ist gleich 0
```

GET LIST ITEM

GET LIST ITEM ({* ;} Liste ; EintragPos | * ; EintragRef ; NeuerText {; Unterliste ; Erweitert})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (string). Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Ohne *: Referenznummer der Liste Mit *: Objektname der Liste
EintragPos | *	Operator, Lange Ganzzahl	➡	Stelle des Eintrags in auf-/zugeklappten Listen oder * für aktuellen Eintrag der Liste
EintragRef	Lange Ganzzahl	⬅	Referenznummer des Eintrags
NeuerText	String	⬅	Text des Eintrags
Unterliste	ListRef	⬅	Referenznummer der Unterliste
Erweitert	Boolean	⬅	Ist eine Unterliste zugeordnet: TRUE = Unterliste ist gerade erweitert, FALSE = Unterliste ist gerade geschlossen

Beschreibung

Der Befehl **GET LIST ITEM** gibt Informationen zum Eintrag, angegeben in *EintragPos* in der Liste mit Referenznummer oder Objektname, übergeben in *Liste*.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung einer Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (ListRef). Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Arten verwenden. Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

Hinweis: Verwenden Sie das Zeichen @ im Namen der Liste, erhalten Sie einen Satz Objekte im Formular, zu denen der Name passt. **GET LIST ITEM** wird aber nur auf das erste Objekt mit dem passenden Namen angewandt.

Die Position richtet sich nach dem aktuellen Status erweitert/geschlossen der Liste und den dazugehörigen Unterlisten. Die Position muss zwischen 1 und dem von **Count list items** zurückgegebenen Wert liegen. Liegt er außerhalb dieses Bereichs, gibt **GET LIST ITEM** leere Werte zurück, also 0 (Null), "", o.ä..

Nach dem Aufruf finden Sie:

- In *EintragRef* die Referenznummer des Eintrags.
- In *NeuerText* den Text des Eintrags.

Mit den optionalen Parametern *Unterliste* und *Erweitert*:

- Gibt *Unterliste* die Referenznummer der zugeordneten Unterliste zurück. Hat der Eintrag keine Unterliste, gibt *Unterliste* Null (0) zurück.
- Hat der Eintrag eine Unterliste, gibt *Erweitert* TRUE zurück, wenn sie erweitert ist, FALSE, wenn sie geschlossen ist.

Beispiel 1

hList hat Einträge mit einmaligen Referenznummern. Folgender Code wechselt per Programmierung den Status erweitert/geschlossen der Unterliste, sofern sie vorhanden ist und dem aktuell ausgewählten Eintrag zugeordnet wurde:

```
$vlItemPos:=Selected list items(hList)
If($vlItemPos>0)
   GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText;$hSublist;$vbExpanded)
   If(Is a list($hSublist))
      SET LIST ITEM(hList;$vlItemRef;$vsItemText;$vlItemRef;$hSublist;Not($vbExpanded))
   End if
End if
```

Beispiel 2

Siehe Beispiel zum Befehl **APPEND TO LIST**.

Get list item font

Get list item font ({* ;} Liste ; EintragRef | *) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Mit *: Objektname der Liste. Ohne *: Referenznummer der Liste
EintragRef | *	Lange Ganzzahl, Operator	➡	Referenznummer des Eintrags oder 0 für den zuletzt hinzugefügten Eintrag oder * für den aktuellen Eintrag in der Liste
Funktionsergebnis	String	↩	Schriftname

Beschreibung

Die Funktion **Get list item font** gibt den Namen der aktuellen Schrift der Zeile zurück, definiert durch den Parameter *EintragRef* der Liste, definiert durch die Referenznummer oder den Objektnamen im Parameter *Liste*.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung der Liste im Formular. Geben Sie diesen Parameter nicht an, ist Liste die Referenznummer einer hierarchischen Liste (ListRef).
Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Arten verwenden.
Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktuellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

Hinweis: Verwenden Sie das Zeichen @ im Objektnamen der Liste und enthält das Formular mehrere Listen, zu denen dieser Name passt, wird **Get list item font** auf das erste Objekt mit dem passenden Namen angewandt.

Sie können in *EintragRef* eine Referenznummer übergeben. Entspricht diese Nummer keiner Zeile in der Liste, führt der Befehl nichts aus.
Sie können auch 0 (Null) in *EintragRef* übergeben, um die Schrift der zuletzt hinzugefügten Zeile (über **APPEND TO LIST**) in der Liste zu erhalten.
Übergeben Sie * in *EintragRef*, erhält die Funktion die Schrift für die aktuelle Zeile der Liste. Sind mehrere Zeilen manuell ausgewählt, ist die aktuelle Zeile die zuletzt ausgewählte Zeile. Ist keine Zeile ausgewählt, führt der Befehl nichts aus.

GET LIST ITEM ICON

GET LIST ITEM ICON ({* ;} Liste ; EintragRef | * ; Icon)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Mit *: Objektname der Liste Ohne *: Referenznummer der Liste
EintragRef | *	Operator, Lange Ganzzahl	➡	Referenznummer des Eintrags oder 0 für den zuletzt hinzugefügten Eintrag oder * für den aktuellen Eintrag in der Liste
Icon	Bildvariable	⬅	Der Zeile zugeordnetes Icon

Beschreibung

Der Befehl **GET LIST ITEM ICON** gibt im Parameter *Icon* das Icon zurück, das der Zeile zugeordnet ist, definiert durch den Parameter *EintragRef* der Liste, definiert durch die Referenznummer oder den Objektnamen im Parameter *Liste*.
Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung der Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (ListRef).
Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Syntaxarten verwenden.
Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktuellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

Hinweis: Verwenden Sie das Zeichen @ im Objektnamen der Liste und enthält das Formular mehrere Listen, zu denen dieser Name passt, wird **GET LIST ITEM ICON** auf das erste Objekt mit dem passenden Namen angewandt.

Sie können in *EintragRef* eine Referenznummer übergeben. Entspricht diese Nummer keiner Zeile in der Liste, führt der Befehl nichts aus.
Sie können auch 0 (Null) in *EintragRef* übergeben, um die zuletzt hinzugefügte Zeile (über **APPEND TO LIST**) in der Liste anzugeben.
Übergeben Sie * in *EintragRef*, um die aktuelle Zeile der Liste anzugeben. Sind mehrere Zeilen manuell ausgewählt, ist die aktuelle Zeile die zuletzt ausgewählte Zeile. Ist keine Zeile ausgewählt, führt der Befehl nichts aus.
In *Icon* übergeben Sie eine Bildvariable. Sie enthält nach Ausführen des Befehls das der Zeile zugeordnete Icon, unabhängig von der zugrundeliegenden Quelle des Icons (statisches Bild, Ressource oder Bildausdruck).
Ist der Zeile kein Icon zugeordnet, wird die Variable Icon leer zurückgegeben.

Hinweis: Wurde das einem Eintrag zugewiesene Icon über eine statische Referenz definiert (Ressourcenreferenz oder Bild aus der Bildbibliothek), lässt sich die Nummer über den Befehl **GET LIST ITEM PROPERTIES** herausfinden.

GET LIST ITEM PARAMETER

GET LIST ITEM PARAMETER ({* ;} Liste ; EintragRef | * ; Selector ; Wert)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Mit *: Objektname der Liste Ohne *: Referenznummer der Liste
EintragRef | *	Lange Ganzzahl, Operator	➡	Referenznummer des Eintrags oder 0 für den zuletzt hinzugefügten Eintrag oder * für den aktuellen Eintrag in der Liste
Selector	String	➡	Konstante des Parameters
Wert	String, Boolean, Zahl	⬅	Aktueller Wert des Parameters

Beschreibung

Der Befehl **GET LIST ITEM PARAMETER** findet den aktuellen Wert des Parameters *Selector* für *EintragRef* der hierarchischen Liste, definiert durch die Referenznummer oder den Objektnamen im Parameter *Liste*.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung der Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (ListRef). Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Syntaxarten verwenden.Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktuellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

Hinweis: Verwenden Sie das Zeichen @ im Objektnamen der Liste und enthält das Formular mehrere Listen, zu denen dieser Name passt, wird **GET LIST ITEM PARAMETER** auf das erste Objekt mit dem passenden Namen angewandt.

Sie können in *EintragRef* eine Referenznummer übergeben. Entspricht diese Nummer keiner Zeile in der Liste, führt der Befehl nichts aus. Sie können auch 0 (Null) übergeben, um die Änderung der zuletzt hinzugefügten Zeile (über **APPEND TO LIST**) in der Liste anzufordern.
Übergeben Sie * in *EintragRef*, gilt der Befehl für die aktuelle Zeile der Liste. Sind mehrere Zeilen manuell ausgewählt, ist die aktuelle Zeile die zuletzt ausgewählte Zeile. Ist keine Zeile ausgewählt, führt der Befehl nichts aus.
In *Selector* können Sie die Konstante Additional text oder Standard action aus dem Thema **Hierarchische Listen** oder einen beliebigen eigenen Wert übergeben. Weitere Informationen dazu finden Sie unter dem Befehl **SET LIST ITEM PARAMETER**.

GET LIST ITEM PARAMETER ARRAYS

GET LIST ITEM PARAMETER ARRAYS ({* ;} Liste ; EintragRef ; arrAuswahl {; arrWerte})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Liste ist ein Objektname (String) Ohne *: Liste ist eine Listenreferenznummer
Liste	ListRef, String	➡	Mit *: Name des Listentyp Objekts ohne *: Listenreferenznummer
EintragRef	Lange Ganzzahl, Operator	➡	Referenznummer des Eintrags oder 0 für den letzten in der Liste angefügten Eintrag oder * für den aktuellen Listeneintrag
arrAuswahl	Array Text	⬅	Array der Parameternamen
arrWerte	Array Text	⬅	Array der Parameterwerte

Beschreibung

Der Befehl **GET LIST ITEM PARAMETER ARRAYS** findet alle Parameter in einer einzelnen Zelle - optional auch die Werte, die dem Eintrag *EintragRef* in der hierarchischen Liste zugeordnet sind, dessen Referenz bzw. Objektname im Parameter *Liste* übergeben ist.

Den Einträgen zugewiesene Parameter speichern zusätzliche Information zu jedem Eintrag. Sie werden mit dem Befehl **SET LIST ITEM PARAMETER** gesetzt.
Übergeben Sie den ersten optionalen Parameter *, ist *Liste* ein Objektname (String), der der Listendarstellung im Formular entspricht. Ohne diesen Parameter ist Liste eine Referenz auf die hierarchische Liste (ListRef). Benutzen Sie eine einzelne Listendarstellung oder arbeiten mit Struktureinträgen (zweiter * wird weggelassen), können Sie beide Syntaxarten verwenden. Benutzen Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit dem aktuellen Eintrag (zweiter * ist übergeben), müssen Sie die Syntax für den Objektnamen verwenden, da jede Darstellung einen anderen aktuellen Eintrag haben kann.

GET LIST ITEM PARAMETER ARRAYS gibt Parameter für den Eintrag *EintragRef* im Array Text *arrAuswahl* zurück. Ist das Array Text *arrWerte* übergeben, gibt der Befehl die diesen Parametern zugewiesenen Werte zurück.
arrWerte muss ein Array vom Typ Text sein. Sind andere Werte zugeordnet (Zahl oder Boolean), werden sie in Strings umgewandelt (Wahr="1", Falsch="0").

Beispiel

Nehmen wir folgende hierarchische Liste:

```
<>HL:=New list
$ID:=30
APPEND TO LIST(<>HL;"Martin";$ID)
 //5 parameters
SET LIST ITEM PARAMETER(<>HL;$ID;"Vorname";"Phil")
SET LIST ITEM PARAMETER(<>HL;$ID;"Geburtstag";"01/02/1978")
SET LIST ITEM PARAMETER(<>HL;$ID;"Männlich";True) //Boolean
SET LIST ITEM PARAMETER(<>HL;$ID;"Alter";33) //number
SET LIST ITEM PARAMETER(<>HL;$ID;"Stadt";"Dallas")
```

Zur Vereinfachung wurde die Liste einem Listenobjekt mit demselben Namen (<>HL) zugewiesen.
Wird der Eintrag "Martin" in der Liste ausgewählt, können Sie die dazugehörigen Parameter mit folgendem Code herausfinden:

```
ARRAY TEXT(arrParamNames;0)
GET LIST ITEM PARAMETER ARRAYS(*;"<>HL";*;arrParamNames)
 // arrParamNames{1} enthält "Vorname"
 // arrParamNames{2} enthält "Geburtstag"
 // arrParamNames{3} enthält "Männlich"
 // arrParamNames{4} enthält "Alter"
 // arrParamNames{5} enthält "Stadt"
```

Wollen Sie auch die Parameter Werte erhalten, schreiben Sie:

```
ARRAY TEXT(arrParamNames;0)
ARRAY TEXT(arrParamValues;0)
GET LIST ITEM PARAMETER ARRAYS(*;"<>HL";*;arrParamNames;arrParamValues)
 // arrParamValues{1} enthält "Phil"
 // arrParamValues{2} enthält "01/02/1978"
 // arrParamValues{3} enthält "1"
 // arrParamValues{4} enthält "33"
 // arrParamValues{5} enthält "Dallas"
```

GET LIST ITEM PROPERTIES

GET LIST ITEM PROPERTIES ({* ;} Liste ; EintragRef | * ; Eingebbar {; NeuerStil {; Icon {; Farbe}}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Referenznummer der Liste (ohne *) oder Objektname der Liste (mit *)
EintragRef | *	Operator, Lange Ganzzahl	➡	Referenznummer des Eintrags oder 0 für zuletzt hinzugefügten Eintrag oder * für den aktuellen Eintrag der Liste
Eingebbar	Boolean	⬅	TRUE = Eingebbar, FALSE = Nicht-eingebbar
NeuerStil	Lange Ganzzahl	⬅	Schriftstil für den Eintrag
Icon	Lange Ganzzahl	⬅	131072 + Bildreferenznummer
Farbe	Lange Ganzzahl	⬅	Wert für RGB-Farbe

Beschreibung

Der Befehl **GET LIST ITEM PROPERTIES** gibt die Eigenschaften des Eintrags zurück mit der in *EintragRef* übergebenen Referenznummer in der Liste mit der in *Liste* übergebenen Referenznummer.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung einer Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (*ListRef*). Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Arten verwenden. Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

Hinweis: Verwenden Sie das Zeichen @ im Namen der Liste, erhalten Sie einen Satz Objekte im Formular, zu denen der Name passt. **GET LIST ITEM PROPERTIES** wird aber nur auf das erste Objekt mit dem passenden Namen angewandt.

In *EintragRef* können Sie entweder eine Referenznummer zuweisen, den Wert 0 für den zuletzt hinzugefügten Eintrag in die Liste oder * für den aktuellen Wert in die Liste. Wurden mehrere Einträge ausgewählt, ist der zuletzt ausgewählte Eintrag der aktuelle.

Übergeben Sie * und ist kein Eintrag ausgewählt oder gibt es keinen Eintrag mit der zugewiesenen Referenznummer, behält der Befehl die Parameter unverändert bei.

Arbeiten Sie mit Referenznummern, sollten diese einmalig sein, damit Sie die Einträge voneinander unterscheiden können. Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **APPEND TO LIST**.

Nach dem Aufruf gilt folgendes:

- *Eingebbar* gibt TRUE zurück, wenn der Eintrag eingebbar ist.
- *Stil* gibt den Schriftstil des Eintrags zurück.
- *Icon* gibt das zugewiesene Icon bzw. das Bild zurück, bei *0* wurde nichts zugewiesen.
- *Farbe* gibt die Farbe des Textes für den angegebenen Eintrag zurück.

Hinweis: Mit dem Befehl **GET LIST ITEM ICON** können Sie in einer Bildvariablen den Icon herausfinden, der einem Eintrag zugeordnet ist.

Weitere Informationen zu diesen Eigenschaften finden Sie in der Beschreibung zum Befehl **SET LIST ITEM PROPERTIES**.

GET LIST PROPERTIES

GET LIST PROPERTIES (Liste ; Darstellung {; Icon {; Zeilenhöhe {; Doppelklick {; MultiAuswahl {; Editierbar}}}}})

Parameter	Typ		Beschreibung
Liste	ListRef	⇒	Referenznummer der Liste
Darstellung	Lange Ganzzahl	⇐	Grafische Darstellung der Liste: 1=Hierarchische Liste a la Macintosh, 2=Hierarchische Liste a la Windows
Icon	Lange Ganzzahl	⇐	*** Überholt, immer 0 ***
Zeilenhöhe	Lange Ganzzahl	⇐	Mindesthöhe der Zeilen in Pixel
Doppelklick	Lange Ganzzahl	⇐	Unterliste mit Doppelklick Ein-/ausblenden, 0 = Ja, 1= Nein
MultiAuswahl	Lange Ganzzahl	⇐	Mehrfache Auswahl: 0 = Nein, 1 = Ja
Editierbar	Lange Ganzzahl	⇐	Liste von Benutzer editierbar: 0 = Nein, 1 = Ja

Beschreibung

Der Befehl **GET LIST PROPERTIES** gibt Information über die hierarchische Liste mit der in *Liste* übergebenen Referenznummer zurück.

Der Parameter *Darstellung* gibt die grafische Darstellung der Liste zurück.
Der Parameter *Icon* ist überholt, er gibt immer 0 zurück.
Der Parameter *Zeilenhöhe* gibt die Mindesthöhe für die Zeilen an.

Ist *Doppelklick* auf 0 (Null) gesetzt, blendet ein Doppelklick auf einen Listeneintrag die Unterliste ein bzw. aus (Standardeinstellung). Ist *Doppelklick* auf 1 gesetzt, ist das Ein-/Ausblenden inaktiv.

Ist *MultiAuswahl* auf 0 (Null) gesetzt, ist keine mehrfache Auswahl der Elemente in der Liste möglich, weder manuell, noch per Programmierung. Ist *MultiAuswahl* auf 1 gesetzt, ist mehrfache Auswahl erlaubt.
Ist *Editierbar* auf 1 gesetzt, ist die Liste eingebbar, wenn sie in der Anwendungsumgebung erscheint. Ist *Editierbar* auf 0 (Null) gesetzt, ist sie nicht eingebbar.
Diese Eigenschaften können Sie mit dem Befehl **SET LIST PROPERTIES** und/oder in der Designumgebung im Listeneditor festlegen, wenn die Liste dort erstellt oder mit dem Befehl **SAVE LIST** gesichert wurde.

Ausführliche Informationen zu der Darstellung und dem Verhalten finden Sie unter dem Befehl **SET LIST PROPERTIES**.

INSERT IN LIST

INSERT IN LIST ({* ;} Liste ; VorEintragRef | * ; EintragText ; EintragRef {; Unterliste ; Erweitert})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Referenznummer der Liste (ohne *) oder Objektname der Liste (mit *)
VorEintragRef | *	Lange Ganzzahl, Operator	➡	Referenznummer oder 0 für den zuletzt hinzugefügten Eintrag oder * für den aktuellen Eintrag der Liste
EintragText	String	➡	Text für neuen Listeneintrag
EintragRef	Lange Ganzzahl	➡	Einmalige Referenznummer für neuen Listeneintrag
Unterliste	ListRef	➡	Optionale Unterliste für neuen Listeneintrag
Erweitert	Boolean	➡	Gibt an, ob die Unterliste auf- oder zugeklappt ist.

Beschreibung

Der Befehl **INSERT IN LIST** fügt einen neuen Eintrag in die Liste mit der in *Liste* übergebenen Referenznummer oder dem Objektnamen ein.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung einer Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (*ListRef*). Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Arten verwenden. Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

Mit dem Parameter *VorEintragRef* können Sie den Eintrag bestimmen, vor dem der neue Eintrag eingefügt werden soll:

- Übergeben Sie den Wert 0 (Null), um den zuletzt in der Liste hinzugefügten Eintrag zu bestimmen. Der neu eingefügte Eintrag wird dann zum ausgewählten Eintrag.
- Übergeben Sie *, wird der Eintrag vor dem aktuell ausgewählen Eintrag in der Liste eingefügt. In diesem Fall wird der neu eingefügte Eintrag auch zum ausgewählten Eintrag.
- Wollen Sie hingegen einen Eintrag vor einem bestimmten Eintrag einfügen, übergeben Sie dessen Referenznummer. In diesem Fall wird der neu eingefügte Eintrag nicht automatisch ausgewählt. Gibt es keinen Eintrag mit dieser Referenznummer, hat der Befehl keine Auswirkung.

In *EintragText* übergeben Sie den Text des neuen Eintrags. Ab 4D v16 R4 können Sie in *EintragText* die Konstante ak standard action title übergeben, wenn dem Eintrag eine Standardaktion zugewiesen ist, um automatisch den lokalisierten Aktionsnamen zu verwenden. Weitere Informationen dazu finden Sie im Abschnitt **Standardaktionen**.

In *EintragRef* übergeben Sie die Referenznummer des neuen Eintrags. Auch wenn diese Refererenznummer als einmalig definiert ist, können Sie einen beliebigen Wert übergeben. Weitere Informationen dazu finden Sie im Abschnitt **Referenznummern der Einträge verwenden (EintragRef)**.

Wollen Sie dem Eintrag Untereinträge zuordnen, müssen Sie im Parameter *Unterliste* eine gültige Listenreferenz übergeben. In diesem Fall müssen Sie auch den Parameter *Erweitert* übergeben. Übergeben Sie hier *Wahr* oder *Falsch*, um die Unterliste entweder auf- oder zugeklappt anzuzeigen.

Beispiel

Folgender Code fügt einen Eintrag (ohne angehängte Unterliste) vor dem aktuell ausgewählten Eintrag in die Liste *hList* ein:

```
vlUniqueRef:=vlUniqueRef+1
INSERT IN LIST(hList;*;"New Item";vlUniqueRef)
```

Is a list

Is a list (Liste) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Liste	ListRef	⇒	zu testender Wert von ListRef
Funktionsergebnis	Boolean	⇐	TRUE, wenn Liste hierarchisch ist FALSE, wenn Liste nicht hierarchisch ist

Beschreibung

Die Funktion **Is a list** gibt TRUE zurück, wenn der in Liste übergebene Wert eine gültige Referenz für eine hierarchische Liste ist. Bei ungültiger Referenz gibt er FALSE zurück.

Beispiel 1

Siehe Beispiel für den Befehl **CLEAR LIST**.

Beispiel 2

Siehe Beispiele für den Befehl **DRAG AND DROP PROPERTIES**.

List item parent

List item parent ({* ;} Liste ; EintragRef | *) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Referenznummer der Liste (ohne *) oder Objektname der Liste (mit *)
EintragRef | *	Operator, Lange Ganzzahl	➡	Referenznummer des Eintrags oder 0 für zuletzt hinzugefügten Eintrag oder * für aktuellen Eintrag in der Liste
Funktionsergebnis	Lange Ganzzahl	⮌	Referenznummer des übergeordneten Eintrags 0, wenn kein Eintrag existiert

Beschreibung

Die Funktion **List item parent** gibt die Referenznummer des übergeordneten Eintrags zurück.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung einer Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (*ListRef*). Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Arten verwenden. Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

Hinweis: Verwenden Sie das Zeichen @ im Namen der Liste, erhalten Sie einen Satz Objekte im Formular, zu denen der Name passt. **List item parent** wird aber nur auf das erste Objekt mit dem passenden Namen angewandt.

In *Liste* übergeben Sie die Referenznummer der Liste; in *EintragRef* die Referenznummer eines Eintrags der Liste oder *. Übergeben Sie den Wert 0 (Null), gilt die Funktion für den zuletzt in der Liste hinzugefügten Eintrag. Mit * bezieht sich die Funktion auf den aktuellen Eintrag der Liste. Wurden mehrere Einträge manuell ausgewählt, ist der zuletzt ausgewählte Eintrag der aktuelle.

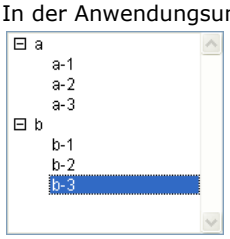
Gehört dagegen der entsprechenden Eintrag in der Liste zu einer Unterliste, erhalten Sie die Referenznummer des übergeordneten Eintrags.

Existiert kein Eintrag mit der übergebenen Referenznummer oder Sie haben * übergeben und es ist kein Eintrag gewählt, oder es gibt keinen übergeordneten Eintrag, gibt **List item parent** den Wert 0 (Null) zurück.

Arbeiten Sie mit Referenznummern, sollten diese einmalig sein, damit Sie die Einträge unterscheiden können. Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **APPEND TO LIST**.

Beispiel

In der Anwendungsumgebung gibt es die Liste *hList*:



Die Referenznummern der Einträge werden folgendermaßen gesetzt:

Eintrag	Referenznummer
a	100
a - 1	101
a - 2	102
b	200
b - 1	201
b - 2	202
b - 3	203

- Wählen Sie in folgendem Code den Eintrag "b - 3", erhält die Variable *$vlParentItemRef* die Nummer 200, die Nummer für den übergeordneten Eintrag "b":

```
$vlItemPos:=Selected list items(hList)
GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText)
$vlParentItemRef:=List item parent(hList;$vlItemRef)  ` $vlParentItemRef erhält 200
```

- Wählen Sie "a - 1", erhält die Variable *$vlParentItemRef* die Nummer 100, die Nummer für den übergeordneten Eintrag "a".
- Wählen Sie "a" oder "b", erhält die Variable *$vlParentItemRef* die 0, da es für diese Einträge keine übergeordneten Einträge gibt.

List item position

List item position ({* ;} Liste ; EintragRef) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Referenznummer
Liste	ListRef, String	➡	Ohne *: Referenznummer der Liste Mit *: Objektname der Liste
EintragRef	Lange Ganzzahl	➡	Referenznummer des Eintrags
Funktionsergebnis	Lange Ganzzahl	⮌	Stelle des Eintrags in auf-/zugeklappten Listen

Beschreibung

Die Funktion **List item position** gibt die Position des Eintrags mit der in *EintragRef* übergebenen Referenznummer in der Liste mit der in *Liste* übergebenen Referenznummer oder dem Objektnamen zurück.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung einer Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (*ListRef)*. Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Arten verwenden. Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

Hinweis: Verwenden Sie das Zeichen @ im Namen der Liste, erhalten Sie einen Satz Objekte im Formular, zu denen der Name passt. **List item position** wird aber nur auf das erste Objekt mit dem passenden Namen angewandt.

Hinweis: Im Gegensatz zu den anderen Funktionen dieses Kapitels können Sie dieser Funktion in *EintragRef* nicht den Wert 0 zuordnen, um den letzten Eintrag anzugeben.

Die Position richtet sich nach dem obersten Eintrag der Hauptliste und dem aktuellen Status erweitert bzw. geschlossen der Liste und deren Unterlisten. Demnach ist das Ergebnis eine Zahl zwischen 1 und dem in **Count list items** zurückgegebenen Wert. Liegt der Eintrag in einer geschlossenen Liste, erweitert **List item position** die entsprechende Liste und macht den Eintrag sichtbar. Existiert der Eintrag nicht, gibt **List item position** den Wert 0 zurück.

LIST OF CHOICE LISTS

LIST OF CHOICE LISTS (NumArray ; NamenArray)

Parameter	Typ		Beschreibung
NumArray	Array Lange Ganzzahl	⇐	Nummern der Auswahllisten
NamenArray	Array Text	⇐	Namen der Asuwahllisten

Beschreibung

Der Befehl **LIST OF CHOICE LISTS** gibt in den aufeinander abgestimmten Arrays *NumArray* und *NamenArray* die Nummern und Namen der Auswahllisten zurück, die im Listeneditor des Designmodus definiert wurden.

Die Nummern der Auswahllisten werden in der Reihenfolge ihrer Erstellung angelegt. Sie werden im Listeneditor in alphabetischer Reihenfolge angezeigt.

Load list

Load list (Listenname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Listenname	String	➡	Name der Liste, die im Listeneditor der Designumgebung erstellt wurde
Funktionsergebnis	ListRef	⮌	Referenznummer der neu erstellten Liste

Beschreibung

Die Funktion **Load list** erstellt eine neue hierarchische Liste mit dem Inhalt, den Sie aus der Liste kopieren und mit dem Namen, den Sie in *Listenname* übergeben. Sie gibt dann die Referenznummer der neu erstellten Liste zurück.

Um die in der Datenbank angegebenen Listen zu finden, rufen Sie den Befehl **LIST OF CHOICE LISTS** auf.
Um sicherzustellen, dass die mit *Listenname* definierte Liste existiert, rufen Sie die Funktion **Is a list** auf.

Beachten Sie, dass die neue Liste eine Kopie der Liste aus der Designumgebung ist. Von daher werden Änderungen in der neuen Liste nicht in die Liste der Designumgebung übernommen und umgekehrt.

Mit dem Befehl **SAVE LIST** können Sie Änderungen in der neu erstellten Liste sichern.

Rufen Sie nach Bearbeiten der neu erstellten Liste den Befehl **CLEAR LIST** auf, damit die Liste freigegeben wird. Andernfalls bleibt sie im Speicher, bis die Arbeitssitzung endet oder der Prozess, der die Liste erstellt, endet oder abbricht.

Tipp: Ordnen Sie eine Liste einem Formularobjekt zu (hierarchische Liste, Registerkarte oder hierarchisches PopUp-Menü) und wählen dafür als Objekteigenschaft Auswahlliste, lädt und löscht 4D die Liste automatisch. Sie müssen hier nicht **Load list** oder **CLEAR LIST** in der Objektmethode aufrufen.

Beispiel

Sie erstellen eine internationale Datenbank, in der Sie zwischen mehreren Sprachen wechseln können. Legen Sie in einem Formular eine hierarchische Liste mit Namen *hlList* mit mehreren Standardoptionen an. In der Designumgebung haben Sie mehrere Listen erstellt, wie z.B. "Std Options US" für die englische Version, "Std Options FR" für die französische Version, "Std Options SP" für die spanische Version, usw.. In einer Interprozessvariablen mit Namen *◇gsCurrentLanguage* speichern Sie einen Programmiercode mit zwei Zeichen, also "US" für die englische Version, "FR" für die französische Version, "SP" für die spanische Version, usw.. Mit der folgenden Methode stellen Sie sicher, dass Ihre Liste immer in der aktuell ausgewählten Sprache geladen wird:

```
` Objektmethode hierarchische Liste hlList
Case of
   :(Form event=On Load)
      C_LONGINT(hlList)
      hlList:=Load list("Std Options"+◇gsCurrentLanguage)
   :(Form event=On Unload)
      CLEAR LIST(hlList;*)
End case
```

New list

New list -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	ListRef	Referenznummer der Liste

Beschreibung

Die Funktion **New list** erstellt eine neue leere hierarchische Liste im Speicher und gibt ihre einmalige Referenznummer zurück.
WARNUNG: Hierarchische Listen werden im Speicher gehalten. Deshalb ist es wichtig, dass Sie die Liste nach dem Bearbeiten mit dem Befehl **CLEAR LIST** aus dem Speicher entfernen, damit sie verfügbar ist.
Sie können hierarchische Listen auch mit folgenden Funktionen erstellen:

- **Copy list** dupliziert eine Liste aus einer bestehenen Liste.
- **Load list** erstellt eine Liste aus einer Auswahlliste, die im Listeneditor der Designumgebung manuell oder über Programmierung erzeugt wurde.
- **BLOB to list** erstellt eine Liste aus dem Inhalt eines BLOB, in welchem eine Liste vorher gespeichert wurde.

Haben Sie mit **New list** eine hierarchische Liste erstellt, können Sie:

- Mit den Befehlen **APPEND TO LIST** oder **INSERT IN LIST** Einträge hinzufügen.
- Mit dem Befehl **DELETE FROM LIST** Einträge löschen.

Beispiel

Siehe Beispiel für den Befehl **APPEND TO LIST**.

SAVE LIST

SAVE LIST (Liste ; Listenname)

Parameter	Typ		Beschreibung
Liste	ListRef	⇒	Referenznummer der Liste
Listenname	String	⇒	Listenname, wie er im Listeneditor der Designumgebung erscheint

Beschreibung

Der Befehl **SAVE LIST** sichert die Liste mit der in *Liste* übergebenen Referenznummer und dem in *Listenname* übergebenen Namen, wie er im Listeneditor der Designumgebung erscheint.

Gibt es bereits eine Liste mit diesem Namen, wird der Inhalt überschrieben.

SELECT LIST ITEMS BY POSITION

SELECT LIST ITEMS BY POSITION ({* ;} Liste ; EintragPos {; PosArray})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Ohne *: Referenznummer der Liste Mit *: Objektname der Liste
EintragPos	Lange Ganzzahl	➡	Stelle des Eintrags in auf-/zugeklappter Liste
PosArray	Array Lange Ganzzahl	➡	Array der Positionen in auf-/zugeklappter Liste

Beschreibung

Der Befehl **SELECT LIST ITEMS BY POSITION** wählt den Eintrag mit der in *EintragPos* übergebenen Position und optional in *PosArray* in der Liste mit der in *Liste* übergebenen Referenznummer oder dem Objektnamen.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung einer Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (*ListRef*). Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Arten verwenden. Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

Hinweis: Verwenden Sie das Zeichen @ im Namen der Liste, erhalten Sie einen Satz Objekte im Formular, zu denen der Name passt. **SELECT LIST ITEMS BY POSITION** wird aber nur auf das erste Objekt mit dem passenden Namen angewandt.

Die Position der Einträge richtet sich immer nach dem aktuellen Status geschlossen/erweitert der Liste und der dazugehörigen Unterlisten. Sie übergeben einen Wert zwischen 1 und dem von **Count list items** zurückgegebenen Wert. Liegt der Wert außerhalb dieses Bereichs, wird kein Eintrag ausgewählt.

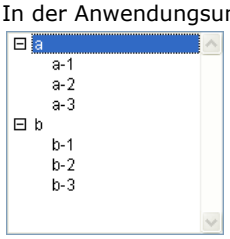
Übergeben Sie den Parameter *PosArray* nicht, enthält *EintragPos* die Position des auszuwählenden Eintrags.

Mit dem optionalen Parameter *PosArray* können Sie in *Liste* mehrere Einträge gleichzeitig auswählen. In *PosArray* müssen Sie ein Array übergeben, in dem jede Zeile die Position eines auszuwählenden Eintrags angibt.
Mit diesem Parameter setzt der in *EintragPos* definierte Eintrag den neuen aktuellen Eintrag der Liste in die Auswahl im Ergebnis. Er gehört nicht zwingend zum Satz Einträge, definiert durch das Array. Der aktuelle Eintrag ist vielmehr der, welcher bei Verwendung des Befehls **EDIT ITEM** bearbeitet wird.

Hinweis: Damit Sie mehrere Einträge gleichzeitig – manuell oder per Programmierung – in einer hierarchischen Liste auswählen können, muss für die Liste die Eigenschaft MultiSelections aktiviert sein. Sie wird über den Befehl **SET LIST PROPERTIES** gesetzt.

Beispiel

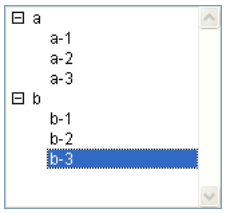
In der Anwendungsumgebung gibt es die hierarchische Liste *hList*:



Nach Ausführung des Code:

```
SELECT LIST ITEMS BY POSITION(hList;Count list items(hList))
```

wird der letzte sichtbare Eintrag aufgerufen:

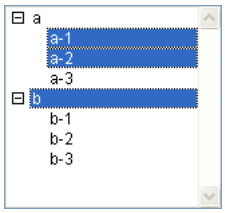


Nach Ausführung des Code:

```
SET LIST PROPERTIES(hList,0;0;18;0;1)
  ` Sie müssen 1 als letzten Parameter übergeben, damit mehrfache Auswahlen möglich sind.
ARRAY LONGINT($arr;3)
$arr{1}:=2
$arr{2}:=3
$arr{3}:=5
```

```
SELECT LIST ITEMS BY POSITION(hList;3;$arr) ` Der 3. Eintrag wird als aktueller Eintrag bestimmt
```

.. werden der 2., 3. und 5. Eintrag der hierarchischen Liste aufgerufen.



SELECT LIST ITEMS BY REFERENCE

SELECT LIST ITEMS BY REFERENCE (Liste ; EintragRef {; RefArray})

Parameter	Typ		Beschreibung
Liste	ListRef	➡	Referenznummer der Liste
EintragRef	Lange Ganzzahl	➡	Referenznummer des Eintrags
RefArray	Array Lange Ganzzahl	➡	Array mit Referenznummern der Einträge

Beschreibung

Der Befehl **SELECT LIST ITEMS BY REFERENCE** wählt die Einträge mit der in *EintragRef* übergebenen Referenznummer, sowie optional in *RefArray* in der Liste mit der in *Liste* übergebenen Referenznummer.
Gibt es keinen Eintrag mit der übergebenen Referenznummer, hat der Befehl keine Auswirkung. Übergeben Sie in *EintragRef* den Wert 0 (Null), wird der zuletzt in die Liste eingefügte Eintrag angegeben.
Ist ein Eintrag gerade nicht sichtbar, weil er in einer geschlossenen Liste liegt, erweitert **SELECT LIST ITEMS BY REFERENCE** die entsprechende Liste, damit der Eintrag sichtbar wird.

Mit dem optionalen Parameter *RefArray* können Sie in *Liste* mehrere Einträge gleichzeitig auswählen. In *RefArray* müssen Sie ein Array übergeben, in dem jede Zeile die Referenz eines auszuwählenden Eintrags angibt.
Mit diesem Parameter setzt der in *EintragRef* definierte Eintrag den neuen aktuellen Eintrag der Liste in die resultierende Auswahl. Er muss nicht zwingend zum Satz Einträge gehören, definiert durch das Array. Der aktuelle Eintrag ist vielmehr der, welcher bei Verwendung des Befehls **EDIT ITEM** bearbeitet wird.

Hinweis: Damit Sie mehrere Einträge gleichzeitig – manuell oder per Programmierung – in einer hierarchischen Liste auswählen können, muss für die Liste die Eigenschaft **Mehrfache Auswahl** aktiviert sein. Sie wird über den Befehl **SET LIST PROPERTIES** gesetzt.

Arbeiten Sie mit Referenznummern, sollten diese einmalig sein, damit Sie die Einträge voneinander unterscheiden können. Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **APPEND TO LIST**.

Beispiel

hList ist eine Liste mit einmaligen Referenznummern. Folgende Objektmethode für eine Schaltfläche wählt den übergeordneten Eintrag (sofern vorhanden) des aktuell ausgewählten Eintrags:

```
$vlItemPos:=Selected list items(hList)
  ` Hole Position des gewählten Eintrags
GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText)
  ` Hole Referenznummer des ausgewählten Eintrags
$vlParentItemRef:=List item parent(hList;$vlItemRef)
  ` Hole Referenznummer des übergeordneten Eintrags (sofern vorhanden)
If($vlParentItemRef>0)
   SELECT LIST ITEMS BY REFERENCE(hList;List item parent(hList;$vlItemRef))
  ` Wähle übergeordneten Eintrag
End if
```

Selected list items

Selected list items ({* ;} Liste {; ZeilenArray {; *}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Ohne *: Referenznummer der Liste Mit *: Objektname der Liste
ZeilenArray	Array Lange Ganzzahl	⬅	Ohne 2. *: Array enthält Positionen der gewählten Einträge in erweiterter Liste. Mit 2tem *: Array enthält Referenzen der gewählten Einträge
*	Operator	➡	Ohne *: Stelle der Einträge Mit *: Referenz der Einträge
Funktionsergebnis	Lange Ganzzahl	➲	Ohne 2ten *: Stelle des aktuellen Eintrags in auf-/zugeklappter Liste Mit 2tem *: Referenz des gewählten Eintrags

Beschreibung

Die Funktion **Selected list items** gibt die Position bzw. die Referenz des ausgewählten Eintrags in der Liste mit der in *Liste* übergebenen Referenznummer oder dem Objektnamen an.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung einer Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (*ListRef*). Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Arten verwenden. Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

Hinweis: Verwenden Sie das Zeichen @ im Namen der Liste, erhalten Sie einen Satz Objekte im Formular, zu denen der Name passt. **Selected list items** wird aber nur auf das erste Objekt mit dem passenden Namen angewandt.

Bei mehrfacher Auswahl gibt die Funktion in *ZeilenArray* die Position bzw. die Referenz jedes gewählten Eintrags zurück. Sie wenden diese Funktion auf eine Liste in einem Formular an. Damit finden Sie heraus, welchen Eintrag der Benutzer ausgewählt hat.

Mit dem Parameter * bestimmen Sie, ob Sie mit den Positionen oder den Referenzen des aktuellen Eintrags arbeiten wollen. Für Referenzen geben Sie den optionalen Parameter * an, für Positionen nicht.

In *ZeilenArray* übergeben Sie ein Array vom Typ Lange Ganzzahl. Die Funktion erstellt und passt das Array bei Bedarf an. Nach Ausführung der Funktion enthält *ZeilenArray*:

- Ohne * die Position jedes gewählten Eintrags relativ zum Status auf-/zugeklappt der Liste.
- Mit * die feste Referenz jedes gewählten Eintrags.
  Wurden keine Einträge ausgewählt, wird das Array leer zurückgegeben.

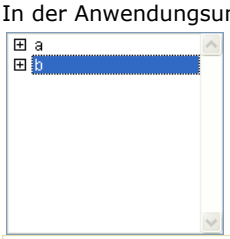
Hinweis: Bei mehrfacher Auswahl gibt die Funktion die Position oder Referenz des aktuellen Eintrags in *Liste* zurück. Das ist der Eintrag, den der Benutzer zuletzt angeklickt hat (manuelle Auswahl) oder gesetzt über die Befehle **SELECT LIST ITEMS BY POSITION** bzw. **Selected list items** (programmierte Auswahl).

Sind der Liste Unterlisten zugeordnet, verwenden Sie diese Funktion für die Hauptliste, d.h. die im Formular geöffnete Liste und nicht für eine der Unterlisten. Die Position richtet sich nach dem ersten Eintrag der Hauptliste und dem aktuellen Status auf- bzw. zugeklappt der Liste und ihrer Unterlisten.

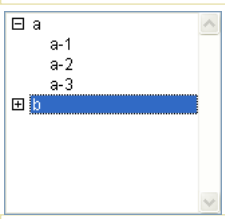
Wurden keine Einträge gewählt, gibt die Funktion 0 (Null) zurück.

Beispiel

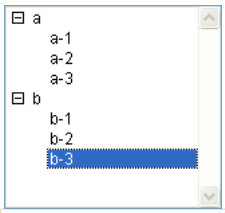
In der Anwendungsumgebung gibt es eine Liste *hList*:



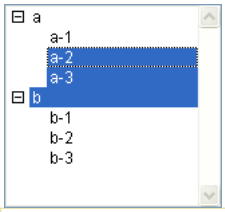

```
$vlItemPos:=Selected list items(hList)  ` $vlItemPos erhält hier 2
```



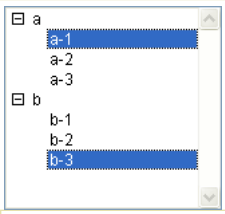

```
$vlItemPos:=Selected list items(hList)  ` $vlItemPos erhält hier 5
$vlItemPos:=Selected list items(hList;*)  ` $vlItemPos kann z.B. 200 enthalten
```




```
$vlItemPos:=Selected list items(hList) ` $vlItemPos erhält hier 8
$vlItemPos:=Selected list items(hList;*) ` $vlItemPos erhält hier wieder 200
```




```
$vlItemPos:=Selected list items(hList;$arrPos) ` $vlItemPos erhält hier 3
 ` $arrPos{1} erhält 3, $arrPos{2} erhält 4 und $arrPos{3} erhält 5+
```




```
$vlItemRef:=Selected list items(hList;$arrRefs;*) ` $vlItemRef kann z.B. 203 enthalten
 ` $arrRefs{1} enthält 101, $arrRefs{2} enthält 203.
```

SET LIST ITEM

SET LIST ITEM ({* ;} Liste ; EintragRef | * ; NeuerEintragText ; NeuerEintragRef {; Unterliste ; Erweitert})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste eine Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Referenznummer der Liste (ohne *) oder Objektname der Liste (mit *)
EintragRef | *	Operator, Lange Ganzzahl	➡	Referenznummer des Eintrags oder 0 für zuletzt hinzugefügten Eintrag oder * für aktuellen Eintrag in der Liste
NeuerEintragText	String	➡	Text für neuen Eintrag
NeuerEintragRef	Lange Ganzzahl	➡	Referenznummer für neuen Eintrag
Unterliste	ListRef	➡	Neue Unterliste zum Eintrag oder 0, wenn keine Unterliste (evtl. vorhandene wird entfernt) oder -1 ohne Änderung
Erweitert	Boolean	➡	Gibt an, ob Unterliste (sofern vorhanden) auf- oder zugeklappt ist

Beschreibung

Der Befehl **SET LIST ITEM** ändert den Eintrag, definiert in *EintragRef*, in der Liste mit der in *Liste* übergebenen Referenznummer.

Gibt es keinen Eintrag mit der übergebenen Referenznummer, hat der Befehl keine Auswirkung. Übergeben Sie in *EintragRef* den optionalen Parameter 0, können Sie mit **APPEND TO LIST** den zuletzt in der Liste angefügten Eintrag bestimmen. Übergeben Sie in *EintragRef* den optionalen Parameter *, gilt der Befehl für den aktuellen Eintrag in der Liste. Wurden mehrere Einträge manuell ausgewählt, ist der zuletzt ausgewählte Eintrag der aktuelle. Wurde kein Eintrag ausgewählt, führt der Befehl nichts aus.

Arbeiten Sie mit Referenznummern, sollten diese einmalig sein, damit Sie die Einträge voneinander unterscheiden können. Weitere Informationen dazu finden Sie im Abschnitt **Hierarchische Listen verwalten**.

In *NeuerEintragText* übergeben Sie den neuen Text für den Eintrag. In *NeuerEintragRef* übergeben Sie den neuen Wert, wenn Sie die Referenznummer ändern wollen, ansonsten denselben Wert wie in *EintragRef*.

Wollen Sie dem Eintrag eine Liste zuordnen, übergeben Sie eine Referenznummer in *Unterliste*. In diesem Fall müssen Sie für die neue Unterliste auch den Parameter *Erweitert* angeben: WAHR, wenn die Unterliste erweitert, FALSCH, wenn sie geschlossen ist.

Wollen Sie eine bereits angehängte Unterliste wieder entfernen, übergeben Sie in Unterliste 0 (Null). In diesem Fall empfiehlt es sich auch, zuvor mit dem Befehl **APPEND TO LIST** die Referenznummer dieser Liste abzufragen. So können Sie die Unterliste später mit **CLEAR LIST** wieder löschen, wenn sie nicht mehr benötigt wird.

Wollen Sie die Unterliste des Eintrags nicht ändern, übergeben Sie in *Unterliste* -1.

Hinweis: Die optionalen Parameter *Unterliste* und *Erweitert* müssen immer zusammen übergeben werden.

Beispiel 1

hList ist eine Liste mit einmaligen Referenznummern. Folgende Objektmethode für eine Schaltfläche fügt dem aktuell ausgewählten Eintrag einen untergeordneten Eintrag hinzu.

```
$vlItemPos:=Selected list items(hList)
If($vlItemPos>0)
   GET LIST ITEM(hList;$vlItemPos;$vlItemRef;$vsItemText;$hSublist;$vbExpanded)
   $vbNewSubList:=Not(Is a list($hSublist))
   If($vbNewSubList)
      $hSublist:=New list
   End if
   vlUniqueRef:=vlUniqueRef+1
   APPEND TO LIST($hSubList;"New Item";vlUniqueRef)
   If($vbNewSubList)
      SET LIST ITEM(hList;$vlItemRef;$vsItemText;$vlItemRef;$hSublist;True)
   End if
   SELECT LIST ITEMS BY REFERENCE(hList;vlUniqueRef)
End if
```

Beispiel 2

Siehe Beispiel zum Befehl **GET LIST ITEM**.

Beispiel 3

Siehe Beispiel zum Befehl **APPEND TO LIST**.

SET LIST ITEM FONT

SET LIST ITEM FONT ({* ;} Liste ; EintragRef | * ; Schrift)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Mit *: Objektname der Liste Ohne *: Referenznummer der Liste
EintragRef | *	Lange Ganzzahl, Operator	➡	Referenznummer des Eintrags oder 0 für den zuletzt hinzugefügten Eintrag oder * für den aktuellen Eintrag der Liste
Schrift	String, Lange Ganzzahl	➡	Schriftname oder -nummer

Beschreibung

Der Befehl **SET LIST ITEM FONT** verändert die Schrift der Zeile, definiert durch den Parameter *EintragRef* der Liste, definiert durch die Referenznummer oder den Objektnamen im Parameter *Liste*.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung einer Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (*ListRef*).
Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Arten verwenden.
Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.
Sie können in *EintragRef* eine Referenznummer übergeben. Entspricht diese Nummer keiner Zeile in der Liste, führt der Befehl nichts aus.
Sie können auch 0 (Null) übergeben, um die Änderung der zuletzt hinzugefügten Zeile (über **APPEND TO LIST**) in der Liste anzufordern.
Übergeben Sie * in *EintragRef*, gilt der Befehl für die aktuelle Zeile der Liste. Sind mehrere Zeilen manuell ausgewählt, ist die aktuelle Zeile die zuletzt ausgewählte Zeile. Ist keine Zeile ausgewählt, führt der Befehl nichts aus.
Im Parameter *Schrift* übergeben Sie Name oder Nummer der zu verwendenden Schrift. Um wieder die Standardschrift der hierarchische Liste anzuwenden, übergeben Sie einen leeren String in *Schrift*.

Beispiel

Die Schrift Times für die aktuelle Zeile der Liste übergeben:

```
SET LIST ITEM FONT(*;"Mylist";*;"Times")
```

SET LIST ITEM ICON

SET LIST ITEM ICON ({* ;} Liste ; EintragRef | * ; Icon)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Mit *: Objektname der Liste Ohne *: Referenznummer der Liste
EintragRef | *	Lange Ganzzahl, Operator	➡	Referenznummer des Eintrags oder 0 für den zuletzt hinzugefügten Eintrag oder * für den aktuellen Eintrag in der Liste
Icon	Bild	➡	Der Zeile zuzuordnendes Icon

Beschreibung

Der Befehl **SET LIST ITEM ICON** ändert das Icon, das der Zeile zugeordnet ist, definiert durch den Parameter *EintragRef* der Liste, definiert durch die Referenznummer oder den Objektnamen im Parameter *Liste*.

Hinweis: Sie können das einer Zeile zugeordnete Icon zwar auch mit dem Befehl **SET LIST ITEM PROPERTIES** verändern. Er erlaubt aber, im Gegensatz zu diesem Befehl, nur statische Bildreferenzen. Das sind Referenzen auf Ressourcen oder Bilder aus der Bildbibliothek.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung der Liste im Formular. Geben Sie diesen Parameter nicht an, ist Liste die Referenznummer einer hierarchischen Liste (ListRef).
Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Syntaxarten verwenden.
Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.
Sie können in *EintragRef* eine Referenznummer übergeben. Entspricht diese Nummer keiner Zeile in der Liste, führt der Befehl nichts aus.
Sie können auch 0 (Null) übergeben, um die Schrift der zuletzt hinzugefügten Zeile (über **APPEND TO LIST**) in der Liste zu erhalten.
Übergeben Sie * in *EintragRef*, um die aktuelle Zeile der Liste anzugeben. Sind mehrere Zeilen manuell ausgewählt, ist die aktuelle Zeile die zuletzt ausgewählte Zeile. Ist keine Zeile ausgewählt, führt der Befehl nichts aus.
Im Parameter *Icon* übergeben Sie einen gültigen 4D Bildausdruck (Feld, Variable, Zeiger, etc.). Das Bild wird links vor der Zeile gesetzt.

Beispiel

Wir wollen dasselbe Bild zwei verschiedenen Einträgen zuweisen. Nachfolgender Code ist optimiert, da das Bild nur einmal in den Speicher geladen wird:

```
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 ({* ;} Liste ; EintragRef | * ; Selector ; Wert)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Mit *: Objektname der Liste Ohne *: Referenznummer der Liste
EintragRef | *	Operator, Lange Ganzzahl	➡	Referenznummer des Eintrags oder 0 für zuletzt hinzugefügten Eintrag oder * für den aktuellen Eintrag in der Liste
Selector	String	➡	Konstante des Parameters
Wert	String, Boolean, Zahl	➡	Wert des Parameters

Beschreibung

Der Befehl **SET LIST ITEM PARAMETER** ändert den Parameter *Selector* für die Zeile, definiert durch *EintragRef*, in der hierarchischen Liste, definiert durch die Referenznummer oder den Objektnamen im Parameter *Liste*.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung der Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (ListRef).
Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Syntaxarten verwenden.
Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

- In *Selector* können Sie eine der folgenden Konstanten unter dem Thema **Hierarchische Listen** übergeben:

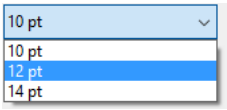
Konstante	Typ	Wert	Kommentar
Additional text	Zeichenkette	4D_additional_text	Diese Konstante fügt auf der rechten Seite von *EintragRef* Text hinzu. Dieser Zusatztext erscheint immer im rechten Teil der Liste, selbst wenn der Benutzer den horizontal scrollenden Cursor bewegt. In *Wert* übergeben Sie den anzuzeigenden Text für diese Konstante.
Associated standard action	Zeichenkette	4D_standard_action_name	Diese Konstante weist *EintragRef* eine Standardaktion zu. In diesem Fall müssen Sie im Parameter *Wert* den Namen einer Standardaktion mit einem Parameter übergeben, z.B. "fontSize?value=10pt". Weitere Informationen dazu finden Sie im Abschnitt **Standardaktionen** des Handbuchs *4D Designmodus*.

- **Eigener Selektor:** Sie können auch einen eigenen Text übergeben und ihn mit einem Wert vom Typ Text, Zahl oder Boolean in *Selector* zuweisen. Dieser Wert wird mit der Zeile in der Liste gespeichert. Über den Befehl **GET LIST ITEM PARAMETER** können Sie Wert finden. Auf diese Weise können Sie jede Art von Oberfläche setzen, die hierarchischen Listen zugeordnet ist. Sie können z.B. in einer Liste mit Kundennamen das Alter jeder Person speichern, es jedoch nur anzeigen, wenn der entsprechende Eintrag ausgewählt ist.

Beispiel

Über das Feature Standardaktionen als Auswahlliste eines hierarchischen PopUp-Menüs eine eigene Liste mit verschiedenen Schriftgrößen setzen:

```
$myList:=New list
APPEND TO LIST($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 ({* ;} Liste ; EintragRef | * ; Eingebbar ; NeuerStil ; Icon {; Farbe})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Liste ein Objektname (String) Ohne * ist Liste eine Listenreferenznummer
Liste	ListRef, String	➡	Referenznummer der Liste (ohne *) oder Objektname der Liste (ohne *)
EintragRef | *	Operator, Lange Ganzzahl	➡	Referenznummer des Eintrags oder 0 für zuletzt hinzugefügten Eintrag oder * für aktuellen Eintrag der Liste
Eingebbar	Boolean	➡	WAHR = Eingebbar; FALSCH = Nicht eingebbar
NeuerStil	Lange Ganzzahl	➡	Schriftstil des Eintrags
Icon	Lange Ganzzahl	➡	131072 + Referenznummer des Bildes oder 0
Farbe	Lange Ganzzahl	➡	RGB Farbwert oder -1 = Rücksetzen auf Originalfarbe

Beschreibung

Der Befehl **SET LIST ITEM PROPERTIES** ändert den Eintrag mit der in *EintragRef* übergebenen Referenznummer in der Liste mit der in *Liste* übergebenen Referenznummer oder Objektname.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Liste* der Objektname (String) der Darstellung einer Liste im Formular. Geben Sie diesen Parameter nicht an, ist *Liste* die Referenznummer einer hierarchischen Liste (*ListRef*).
Verwenden Sie nur eine einzige Darstellung der Liste oder arbeiten mit Strukturzeilen (der zweite * ist nicht übergeben), können Sie beide Arten verwenden.
Verwenden Sie dagegen mehrere Darstellungen derselben Liste und arbeiten mit der aktellen Zeile (der zweite * ist übergeben), müssen Sie die Syntax mit dem Objektnamen verwenden, da jede Darstellung eine eigene aktuelle Zeile haben kann.

Gibt es keinen Eintrag mit der übergebenen Referenznummer, hat der Befehl keine Auswirkung. Übergeben Sie in *EintragRef* den optionalen Parameter 0, können Sie mit **APPEND TO LIST** den zuletzt in der Liste angefügten Eintrag ändern.
Übergeben Sie in *EintragRef* den optionalen Parameter *, gilt der Befehl für den aktuellen Eintrag in der Liste. Wurden mehrere Einträge manuell ausgewählt, ist der zuletzt ausgewählte Eintrag der aktuelle. Wurde kein Eintrag ausgewählt, führt der Befehl nichts aus.

Arbeiten Sie mit Referenznummern, sollten diese einmalig sein, damit Sie die Einträge unterscheiden können. Weitere Informationen dazu finden Sie im Abschnitt **Hierarchische Listen verwalten**.

Hinweis: Mit dem Befehl **SET LIST ITEM** können Sie den Text des Eintrags oder der dazugehörigen Unterlisten ändern.

Übergeben Sie in *Eingebbar* WAHR, ist der Eintrag eingebbar. Übergeben Sie FALSCH, ist er nicht eingebbar.
Wichtig: Eingebbare Einträge müssen zu einer eingebbaren Liste gehören. Der Befehl **OBJECT SET ENTERABLE** macht eine ganze Liste eingebbar. Der Befehl **SET LIST ITEM PROPERTIES** macht einen einzelnen Eintrag der Liste eingebbar. Ändern Sie die Eigenschaft eingebbar für die Liste, hat das keine Auswirkung auf die Eigenschaft der einzelnen Einträge. Ein Eintrag kann jedoch nur eingebbar sein, wenn die dazugehörige Liste eingebbar ist.

Mit dem Parameter *Stil* legen Sie den Schriftstil des Eintrags fest. Übergeben Sie eine oder mehrere der folgenden vordefinierten Konstanten:

Konstante	Typ	Wert
Bold	Lange Ganzzahl	1
Italic	Lange Ganzzahl	2
Plain	Lange Ganzzahl	0
Underline	Lange Ganzzahl	4

Wollen Sie dem Eintrag ein Icon zuordnen, übergeben Sie Use PicRef+N im Parameter *Icon*, wobei N die Referenznummer eines Bildes aus der Bildbibliothek der Designumgebung ist. Übergeben Sie Null (0), wenn der Eintrag keine Grafik enthalten soll.

Hinweise:

- Use PicRef ist eine vordefinierte Konstante unter dem Thema **Hierarchische Listen**.
- Wollen Sie 4D Bilder für Ausdrücke, wie Felder, Variablen o.ä. verwenden, um Icons für Einträge festzulegen, verwenden Sie den Befehl **SET LIST ITEM ICON**.

Mit dem optionalen Parameter *Farbe* können Sie die Farbe eines Eintragstexts verändern. Sie muss als RGB-Farbe angegeben werden, z.B. eine Lange Ganzzahl mit 4-Byte im Format 0x00RRGGBB. Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **OBJECT SET RGB COLORS**. Übergeben Sie -1 in *Farbe*, erscheint wieder die Orginalfarbe des Eintrags.

Beispiel 1

Siehe Beispiel zum Befehl **APPEND TO LIST**.

Beispiel 2

Das folgende Beispiel setzt den Schriftstil des aktuellen Eintrags von *List* in Fett und Hellrot.

```
SET LIST ITEM PROPERTIES(List;*;True;Bold;0;0x00FF0000)
```

SET LIST PROPERTIES

SET LIST PROPERTIES (Liste ; Darstellung {; Icon {; Zeilenhöhe {; Doppelklick {; MultiAuswahl {; Editierbar}}}}})

Parameter	Typ		Beschreibung
Liste	ListRef	➡	Referenznummer der Liste
Darstellung	Lange Ganzzahl	➡	*** Überholt, muss immer 0 sein ***
Icon	Lange Ganzzahl	➡	*** Überholt, muss immer 0 sein ***
Zeilenhöhe	Lange Ganzzahl	➡	Mindesthöhe der Zeilen in Pixel
Doppelklick	Lange Ganzzahl	➡	Unterliste mit Doppelklick ein-/ ausblenden, 0 = Ja, 1 = Nein
MultiAuswahl	Lange Ganzzahl	➡	Mehrfache Auswahlen: 0 = Nein (Standard), 1 = Ja
Editierbar	Lange Ganzzahl	➡	0 = Liste nicht vom Benutzer editierbar 1 = Liste vom Benutzer editierbar (Standard)

Beschreibung

Der Befehl **SET LIST PROPERTIES** setzt die Eigenschaften für Zeilenhöhe und Auswahl in der hierarchischen Liste mit der in *Liste* übergebenen Referenznummer.

Hinweis zur Kompatibilität: Die Parameter *Darstellung* und *Icon* sind überholt, sie müssen immer 0 erhalten.

Hinweis: Wollen Sie die Symbole für einzelne Elemente in der Liste individuell gestalten, verwenden Sie den Befehl **SET LIST ITEM PROPERTIES**.

Übergeben Sie keinen Parameter *Zeilenhöhe*, richtet sich die Zeilenhöhe in der hierarchischen Liste nach Schriftstil und Schriftgröße des Objekts. Sie können im Parameter *Zeilenhöhe* auch eine Mindesthöhe für Zeilen in der hierarchischen Liste übergeben. Der hier übergebene Wert hat dann für hierarchische Listen Vorrang vor der durch Schriftstil und -größe vorgegebene Zeilenhöhe. Übergeben Sie 0 für die standardmäßige Höhe.

Standardmäßig wird die Unterliste bei Doppelklick ein- bzw. ausgeblendet. Ist der optionale Parameter *Doppelklick* nicht übergeben oder auf 0 (Null) gesetzt, blendet ein Doppelklick auf einen Listeneintrag die Unterliste ein bzw. aus (Standardeinstellung). Ist *Doppelklick* auf 1 gesetzt, ist das Ein-/Ausblenden inaktiv. Nur der Doppelklick ist inaktiv. Der Benutzer kann weiterhin Unterlisten ein- bzw. ausblenden, wenn er auf das Symbol vor dem Listeneintrag klickt.

Der optionale Parameter *MultiAuswahl* gibt an, ob *Liste* mehrfache Auswahlen akzeptiert. Standardmäßig können Sie nicht gleichzeitig mehrere Einträge einer hierarchischen Liste auswählen. Um diese Funktion für die Liste zu aktivieren, übergeben Sie in *MultiAuswahl* den Wert 1. Dann gibt es zwei Möglichkeiten:

- Manuell mit der Kombination **Umschalttaste+Klick** für eine zusammenhängende Auswahl oder unter Windows mit der Kombination **ctrl-Taste+Klick**, auf macOS **Befehlstaste+Klick** für eine unterbrochene Auswahl.
- Per Programmierung über die Befehle **SELECT LIST ITEMS BY POSITION** und **SELECT LIST ITEMS BY REFERENCE**. Lassen Sie den Parameter *MultiAuswahl* weg oder übergeben den Wert 0 (Null), gilt die Standardeinstellung, d.h. es kann nur ein Eintrag ausgewählt werden.

Der optionale Parameter *Editierbar* gibt an, ob *Liste* vom Benutzer editierbar sein muss, wenn sie als Auswahlliste angezeigt wird, die einem Feld bzw. einer Variablen bei der Dateneingabe zugeordnet ist. Ist *Liste* editierbar, enthält das Fenster mit der Auswahlliste die Schaltfläche **Ändern**, so dass der Benutzer über einen spezifischen Editor Werte hinzufügen, löschen und sortieren kann.
Lassen Sie den Parameter *Editierbar* weg oder übergeben den Wert 1, ist die Liste editierbar. Übergeben Sie 0 (Null), ist die Liste nicht editierbar.

Beispiel

Doppelklick zum Auf-/Zuklappen der Unterliste nicht zulassen. In der Formularmethode schreiben Sie:

```
Case of
   :(Form event=On Load)
      hlCities:=Load list("Cities")  //Die Auswahlliste "Cities" in das Formularobjekt hlCities laden
      SET LIST PROPERTIES(hlCities;0;0;0;1)  //Doppelklick zum Auf-/Zuklappen nicht zulassen
End case
```

SORT LIST

SORT LIST (Liste {; > oder <})

Parameter	Typ		Beschreibung
Liste	ListRef	➡	Referenznummer der Liste
> oder <	Operator	➡	Sortierreihenfolge: > sortiert in aufsteigender Reihenfolge < sortiert in absteigender Reihenfolge

Beschreibung

Der Befehl **SORT LIST** sortiert die Liste mit der in *Liste* übergebenen Referenznummer.

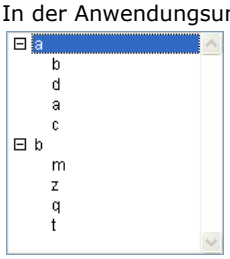
Für Sortieren in aufsteigender Reihenfolge übergeben Sie >. Für Sortieren in absteigender Reihenfolge übergeben Sie <. Übergeben Sie keinen Parameter, sortiert **SORT LIST** standardmäßig in aufsteigender Reihenfolge.

SORT LIST sortiert alle Ebenen der Liste; und zwar zuerst auf der ersten Ebene die Einträge der Liste, dann die Einträge jeder evtl. vorhandenen Unterliste, usw. bis alle Ebenen der Liste durchlaufen sind. Deshalb wenden Sie **SORT LIST** normalerweise auf eine Liste in einem Formular an. Das Sortieren einer Unterliste hat wenig Gewicht, da sich die Reihenfolge durch Aufrufen einer höheren Ebene wieder ändert.

SORT LIST ändert nicht die ausgewählten Listeneinträge oder den aktuellen Status erweitert/geschlossen der Liste und Unterlisten. Da der ausgewählte Eintrag jedoch durch den Sortiervorgang verschoben werden kann, kann **Selected list items** vor und nach der Sortierung eine andere Position zurückgeben.

Beispiel

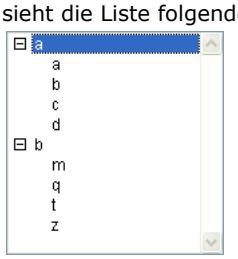
In der Anwendungsumgebung gibt es die Liste *hList*:



Nach Ausführung des Code:

```
` Sortiere Liste und Unterlisten in aufsteigender Reihenfolge
SORT LIST(hList;>)
REDRAW LIST(hList)
` Vergessen Sie nicht, REDRAW LIST aufzurufen, damit die Liste aktualisiert wird.
```

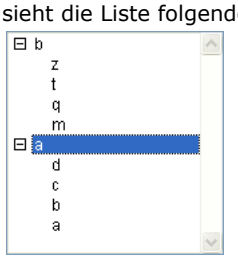
sieht die Liste folgendermaßen aus:



Nach Ausführung des Code:

```
` Sortiere Liste und Unterlisten in absteigender Reihenfolge
SORT LIST(hList;<)
REDRAW LIST(hList)
` Vergessen Sie nicht, REDRAW LIST aufzurufen, damit die Liste aktualisiert wird.
```

sieht die Liste folgendermaßen aus:



_o_REDRAW LIST

_o_REDRAW LIST (Liste)

Parameter	Typ		Beschreibung
Liste	ListRef	➡	Referenznummer der Liste

Beschreibung

Dieser Befehl ist ab 4D v11 nicht mehr erforderlich, da alle Darstellungen hierarchischer Listen jetzt automatisch aktualisiert werden. Der Befehl führt nichts aus, wenn er aufgerufen wird.

HTTP Client

- HTTP AUTHENTICATE
- HTTP Get
- HTTP Get certificates folder
- HTTP GET OPTION
- HTTP Request
- HTTP SET CERTIFICATES FOLDER
- HTTP SET OPTION

HTTP AUTHENTICATE

HTTP AUTHENTICATE (Name ; Kennwort {; authMethode} {; *})

Parameter	Typ		Beschreibung
Name	Text	➡	Benutzername
Kennwort	Text	➡	Benutzerkennwort
authMethode	Lange Ganzzahl	➡	Authentifizierungsmethode: 0 oder weggelassen=nicht spezifiziert, 1=BASIC, 2=DIGEST
*	Operator	➡	Mit Stern: Authentifizierung nach Proxy

Beschreibung

Der Befehl **HTTP AUTHENTICATE** ermöglicht HTTP-Anfragen an Server, die eine Authentifizierung der Client-Anwendung verlangen. Die Methoden BASIC und DIGEST werden unterstützt, sowie das Vorhandensein von Proxy.

In den Parametern *Name* und *Kennwort* übergeben Sie die erforderlichen Identifizierungsdaten (Benutzername und Kennwort). Diese Daten werden verschlüsselt und in der nächsten HTTP Anfrage über die Funktionen **HTTP Request** oder **HTTP Get** hinzugefügt, so dass sie vor jeder HTTP Anfrage den Befehl **HTTP AUTHENTICATE** aufrufen müssen.
Der optionale Parameter *authMethode* gibt die zu verwendende Authentifizierungsmethode an.

Sie können eine der folgenden Konstanten unter dem Thema **HTTP Client** verwenden:

Konstante	Typ	Wert	Kommentar
HTTP basic	Lange Ganzzahl	1	Die Authentifizierungsmethode BASIC verwenden
HTTP digest	Lange Ganzzahl	2	Die Authentifizierungsmethode DIGEST verwenden

Lassen Sie den Parameter *authMethode* weg oder übergeben Sie 0, wählt das Programm die geeignete Methode. In diesem Fall sendet 4D eine zusätzliche Anfrage, um die Authentifizierungsmethode zu bestimmen.

Mit dem Parameter * geben Sie an, dass die Information zur Authentifizierung für ein HTTP Proxy dienen soll. Diese Einstellung muss implementiert werden, wenn es ein Proxy gibt, das die Authentifizierung zwischen Client und dem HTTP Server erfordert. Ist der Server selbst authentifiziert, ist eine doppelte Authentifizierung notwendig.

Die Information zur Authentifizierung wird standardmäßig für jede Anfrage im aktuellen Prozess gespeichert und wiederverwendet. Über eine Option des Befehls **HTTP SET OPTION** lässt sich diese Information aber auch nach jeder Anfrage zurücksetzen. In diesem Fall müssen Sie den Befehl **HTTP AUTHENTICATE** vor jedem Aufrufen von **HTTP Request** oder **HTTP Get** ausführen.

Beispiel

Beispiele für Anfragen mit Authentifizierung:

```
// Authentifizierung auf HTTP Server im DIGEST Modus
HTTP AUTHENTICATE("httpUser";"123";2)
// Authentifizierung auf Proxy im Standardmodus
HTTP AUTHENTICATE("ProxyUser";"456";*)
$httpStatus:=HTTP Get(...)
```

HTTP Get

HTTP Get (url ; Antwort {; KopfteilNamen ; KopfteilWerte}{; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
url	Text	➡	URL, an die die Anfrage gesendet wird
Antwort	Text, BLOB, Bild, Objekt	⬅	Ergebnis der Anfrage
KopfteilNamen	Array Text	➡	Kopfteilnamen der Anfrage
		⬅	zurückgegebene Kopfteilnamen
KopfteilWerte	Array Text	➡	Kopfteilwerte der Anfrage
		⬅	zurückgegebene Kopfteilwerte
*	Operator	➡	Mit *: Verbindung wird beibehalten (keep-alive) Ohne *: Verbindung wird automatisch geschlossen
Funktionsergebnis	Lange Ganzzahl	↩	HTTP Status Code

Beschreibung

Die Funktion **HTTP Get** sendet eine HTTP GET Anfrage direkt an eine bestimmte URL und bearbeitet die HTTP Server Antwort.
Im Parameter *url* übergeben Sie die URL, an die die Anfrage gesendet werden soll. Dafür verwenden Sie folgende Syntax:

```
http://[{user}:[{password}]@]host[:{port}][/{path}][?{queryString}]
```

Sie können z.B. folgende Strings übergeben:

```
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 (**)
```

(*) Während HTTPS Anfragen wird die Zertifizierungsstelle nicht überprüft.
(**) Weitere Informationen zu IPv6 Adressen in urls finden Sie unter RFC 2732

Nach Ausführen des Befehls empfängt der Parameter *Antwort* das vom Server zurückgegebene Ergebnis der Anfrage. Es entspricht dem Hauptteil der Antwort ohne Kopfteile. In *Antwort* können Sie verschiedene Variablentypen übergeben:

- Text: Wenn als Ergebnis ein Text erwartet wird (siehe nachfolgender Hinweis)
- BLOB: Wenn ein Ergebnis in binärer Form erwartet wird
- Bild: Wenn als Ergebnis ein Bild erwartet wird
- Objekt: When als Ergebnis ein **C_OBJECT** Objekt erwartet wird

Hinweis: Wird in *Anwort* eine Textvariable übergeben, versucht 4D die vom Server zurückgegebenen Daten zu entschlüsseln. 4D versucht zuerst, den Zeichensatz vom *content-type* Kopfteil zu finden, dann den Inhalt vom Typ BOM, und zuletzt jeden *http-equiv Zeichensatz* (in html Inhalt) bzw. *encoding* Attribut (für xml). Wird kein Zeichensatz gefunden, versucht 4D die Antwort in ANSI zu entschlüsseln. Schlägt die Konvertierung fehl, ist der Ergebnistext leer. Sind Sie unsicher, ob der Server eine Information in einem Zeichensatz oder ein BOM zurückgibt, kennen jedoch die Codierung, ist es passender, *Antwort* in BLOB zu übergeben und die Funktion **Convert to text** aufzurufen.

Übergeben Sie den Typ BLOB, enthält er vom Server zurückgegebenen Text, ein Bild oder anderen Inhalt (.wav, .zip, etc.). Sie müssen dann die Wiederherstellung dieses Inhalts selbst steuern (das BLOB enthält keine Kopfteile). Übergeben Sie ein Objekt vom Typ **C_OBJECT**,und gibt die Anfrage ein Ergebnis mit Inhalt vom Typ "application/json" (oder "irgendwas/json"), versucht 4D den JSON Inhalt zu analysieren (parsen), um das Objekt zu definieren.

In *KopfteilNamen* und *KopfteilWerte* übergeben Sie Arrays mit den Namen und Werten aus den Kopfteilen der Anfrage.
Diese Arrays enthalten nach Ausführen der Methode die Namen und Werte der vom HTTP Server zurückgegebenen Kopfteile. So können Sie vorallem Cookies verwalten.

Mit dem Parameter * aktivieren Sie den keep-alive Mechanismus für die Server Verbindung. Dieser Parameter ist standardmäßig nicht gewählt, d.h. keep-alive ist nicht aktiviert.

Die Funktion gibt einen standardmäßigen HTTP Status Code (200=OK usw.) zurück, wie vom Server zurückgegeben. Die Liste der HTTP Status Codes finden Sie unter RFC 2616.
Kommt die Verbindung zum Server wegen Netzwerkproblemen (DNS ist fehlgeschlagen, Server ist nicht erreichbar, ...) nicht zustande, gibt die Funktion 0 zurück und ein Fehler wird erzeugt. Sie können Fehler über eine Fehlerverwaltungsmethode abfangen, die über den Befehl **ON ERR CALL** installiert wird.

Beispiel 1

Das 4D Logo auf der 4D Web Site laden:

```
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)
```

Beispiel 2

Ein RFC laden:

```
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)
```

Beispiel 3

Ein Video laden:

```
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 -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Text	Vollständiger Pfadname des aktiven Ordners Zertifikate

Beschreibung

Die Funktion **HTTP Get certificates folder** gibt den kompletten Pfadnamen des aktiven Ordners Client Zertifikate zurück.
4D verwendet standardmäßig den Ordner "ClientCertificatesFolder", der neben der Strukturdatei liegt (wird nur bei Bedarf angelegt). Über den neuen Befehl **HTTP SET CERTIFICATES FOLDER** können Sie aber auch einen eigenen Ordner für den aktuellen Prozess anlegen.

Beispiel

Den Ordner Zertifikate vorübergehend ändern:

```
C_TEXT($certifFolder)
$certifFolder :=HTTP Get certificates folder  //aktuellen Ordner sichern
HTTP SET CERTIFICATES FOLDER("C:/temp/certifTempo/")
... //spezifische Anfragen ausführen
HTTP SET CERTIFICATES FOLDER($certifFolder)  //vorigen Ordner wiederherstellen
```

HTTP GET OPTION

HTTP GET OPTION (Option ; Wert)

Parameter	Typ		Beschreibung
Option	Lange Ganzzahl	➡	Code der zu erhaltenden Option
Wert	Lange Ganzzahl	⬅	Aktueller Wert der Option

Beschreibung

Die Funktion **HTTP GET OPTION** gibt den aktuellen Wert der HTTP Optionen zurück. Das sind vom Client gesetzte Optionen für die nächste Anfrage, ausgelöst über die Funktionen **HTTP Get** oder **HTTP Request**. Der aktuelle Wert einer Option kann der Standardwert sein oder ein Wert, der über den Befehl **HTTP SET OPTION** geändert wurde.

Hinweis: Die gesetzten Optionen gelten lokal für den aktuellen Prozess, bzw. lokal für die ausführende Komponente.

Im Parameter *Option* übergeben Sie die Nummer der zu setzenden Option. Sie können Sie eine der folgenden Konstanten unter dem Thema **HTTP Client** verwenden:

Konstante	Typ	Wert	Kommentar
HTTP compression	Lange Ganzzahl	6	Wert = 0 (nicht komprimieren) oder 1 (komprimieren), Standardwert: 0 Diese Option aktiviert oder deaktiviert das Komprimieren, um den Austausch für Anfragen zwischen Client und Server zu beschleunigen. Ist es aktiviert, verwendet der HTTP Client je nach der Server Antwort die Deflate oder gzip Komprimierung.
HTTP display auth dial	Lange Ganzzahl	4	Wert = 0 (Dialogfenster nicht anzeigen) oder 1 (Dialogfenster anzeigen). Standardwert: 0 Diese Option zeigt den Authentifizierungsdialog beim Ausführen der Funktion **HTTP Get** oder **HTTP Request** an. Standardmäßig wird dieses Dialogfenster nie angezeigt und Sie müssen den Befehl **HTTP AUTHENTICATE** verwenden. Soll der Authentifizierungsdialog jedoch erscheinen, damit Benutzer ihre Daten zur Identifizierung eingeben können, übergeben Sie 1 in Wert. Das Dialogfenster erscheint nur, wenn für die Anfrage eine Authentifizierung erforderlich ist
HTTP follow redirect	Lange Ganzzahl	2	Wert = 0 (keine Redirektion zulassen) oder 1 (Redirektion zulassen). Standardwert = 1
HTTP max redirect	Lange Ganzzahl	3	Wert = Maximale Anzahl der zugelassenen Redirektionen. Standardwert = 2
HTTP reset auth settings	Lange Ganzzahl	5	Wert = 0 (Information nicht löschen) oder 1 (Information löschen). Standardwert: 0 Diese Option weist 4D an, die Authentifizierungsdaten des Benutzers (Benutzername, Kennwort, Methode, etc.) nach jeder Ausführung der Funktion **HTTP Get** oder **HTTP Request** im gleichen Prozess wiederherzustellen. Diese Information wird standardmäßig beibehalten und für jede Anfrage wiederverwendet. Übergeben Sie 1 in *Wert,* um diese Information nach jedem Aufruf zu löschen. Beachten Sie, dass die Information unabhängig von der Einstellung gelöscht wird, wenn der Prozess gestoppt wird.
HTTP timeout	Lange Ganzzahl	1	Wert = Timeout der Client Anfrage in Sekunden. Dieses Timeout bestimmt, wie lange der HTTP Client auf eine Antwort des Server wartet. Ist die Zeit überschritten, schließt der Client die Sitzung und die Anfrage geht verloren. Standardmäßig sind 120 Sekunden eingestellt. Das lässt sich bei bestimmten Bedingungen, wie Netzwerkstatus, Merkmale der Anfrage, etc. ändern.

Im Parameter *Wert* übergeben Sie eine Variable, um den aktuellen Wert von *Option* zu empfangen.

HTTP Request

HTTP Request (httpMethode ; url ; Inhalt ; Antwort {; KopfteilNamen ; KopfteilWerte}{; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
httpMethode	Text	➡	HTTP Methode für Anfrage
url	Text	➡	URL, an die die Anfrage gesendet wird
Inhalt	Text, BLOB, Bild, Objekt	➡	Inhalt des Hauptteils der Anfrage
Antwort	Text, BLOB, Bild, Objekt	⬅	Ergebnis der Anfrage
KopfteilNamen	Array Text	➡	Kopfteilnamen der Anfrage
		⬅	zurückgegebene Kopfteilnamen
KopfteilWerte	Array Text	➡	Kopfteilwerte der Anfrage
		⬅	zurückgegebene Kopfteilwerte
*	Operator	➡	Mit Stern: Verbindung wird beibehalten (keep-alive) Ohne Stern: Verbindung wird automatisch geschlossen
Funktionsergebnis	Lange Ganzzahl	➲	HTTP Status Code

Beschreibung

Die Funktion **HTTP Request** erlaubt, alle Arten von HTTP Anfragen an eine spezifische URL zu senden und die HTTP Server Antwort zu empfangen.

Im Parameter *httpMethode* übergeben Sie die HTTP Methode der Anfrage. Sie können eine der folgenden Konstanten unter dem Thema **HTTP Client** verwenden:

Konstante	Typ	Wert	Kommentar
HTTP DELETE method	Zeichenkette	DELETE	Siehe unter RFC 2616
HTTP GET method	Zeichenkette	GET	Siehe unter RFC 2616. Entspricht der Verwendung der Funktion **HTTP Get**.
HTTP HEAD method	Zeichenkette	HEAD	Siehe unter RFC 2616
HTTP OPTIONS method	Zeichenkette	OPTIONS	Siehe unter RFC 2616
HTTP POST method	Zeichenkette	POST	Siehe unter RFC 2616
HTTP PUT method	Zeichenkette	PUT	Siehe unter RFC 2616
HTTP TRACE method	Zeichenkette	TRACE	Siehe unter RFC 2616

Im Parameter *url* übergeben Sie die URL, an welche die Anfrage gesendet werden soll. Die Syntax lautet:

```
http://[{user}:[{password}]@]host[:{port}][/{path}][?{queryString}]
```

Sie können z.B. folgende Strings übergeben:

```
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 (**)
```

(*) Während HTTPS Anfragen wird die Gültigkeit des Zertifikats nicht geprüft.
(**) Weitere Informationen zu IPv6 Adressen in urls finden Sie unter RFC 2732.

Im Parameter *Inhalt* übergeben Sie den Hauptteil der Anfrage. Die hier übergebenen Daten richten sich nach der HTTP Methode der Anfrage.
Sie können Daten vom Typ Text, BLOB, Bild oder Objekt senden. Ist kein Typ angegeben, werden folgende Typen verwendet:

- Für Text: text/plain - UTF8
- Für BLOBs: application/byte-stream
- Für Bilder: bekannter MIME Typ (der Beste für Web)
- Für Objekte **C_OBJECT**: application/json

Nach Ausführen des Befehls empfängt der Parameter *Antwort* das Ergebnis der vom Server zurückgegebenen Anfrage. Dieses Ergebnis entspricht dem Hauptteil der Antwort, d.h. ohne Kopfteile. In *Antwort* können Sie verschiedene Variablentypen übergeben:

- Text: Wenn als Ergebnis ein Text erwartet wird (siehe nachfolgender Hinweis)
- BLOB: Wenn ein Ergebnis in binärer Form erwartet wird
- Bild: Wenn als Ergebnis ein Bild erwartet wird
- Objekt **C_OBJECT**: Wenn als Ergebnis ein Objekt erwartet wird

Hinweis: Wird in *Anwort* eine Textvariable übergeben, versucht 4D die vom Server zurückgegebenen Daten zu entschlüsseln. 4D versucht zuerst, den Zeichensatz vom *content-type* Kopfteil zu finden, dann den Inhalt vom Typ BOM, und zuletzt jeden *http-equiv Zeichensatz* (in html Inhalt) bzw. *encoding* Attribut (für xml). Wird kein Zeichensatz gefunden, versucht 4D die Antwort in ANSI zu entschlüsseln. Schlägt die Konvertierung fehl, ist der Ergebnistext leer. Sind Sie unsicher, ob der Server eine

Information in einem Zeichensatz oder ein BOM zurückgibt, kennen dagegen die Codierung, ist es passender, *Antwort* in BLOB zu übergeben und die Funktion **Convert to text** aufzurufen.

Entspricht das vom Server zurückgegebene Ergebnis nicht dem Variablentyp von *Antwort*, wird es leer gelassen und die Systemvariable OK wird auf 0 gesetzt.

In *KopfteilNamen* und *KopfteilWerte* übergeben Sie Arrays mit den Namen und Werten der Kopfteile der Anfrage. Diese Arrays enthalten nach Ausführen der Methode die Namen und Werte der vom HTTP Server zurückgegebenen Kopfteile. So können Sie vorallem Cookies verwalten.

Mit dem Parameter * aktivieren Sie den keep-alive Mechanismus für die Server Verbindung. Dieser Parameter wird standardmäßig weggelassen, d.h. keep-alive ist nicht aktiviert.
Die Funktion gibt einen standardmäßigen HTTP Status Code (200=OK usw.) zurück, wie vom Server zurückgegeben. Die Liste der HTTP Status Codes finden Sie unter RFC 2616.
Kommt die Verbindung zum Server wegen Netzwerkproblemen (DNS ist fehlgeschlagen, Server ist nicht erreichbar, ...) nicht zustande, gibt die Funktion 0 zurück und ein Fehler wird erzeugt. Sie können Fehler über eine Methode mit dem Befehl **ON ERR CALL** abfangen.

Beispiel 1

Datensatz aus einer remote Anwendung löschen:

```
C_TEXT($response)
$body_t:="{record_id:25}"
$httpStatus_l:=HTTP Request(HTTP DELETE method;"Beispielanwendung.com";$body_t;$response)
```

Hinweis: Sie müssen die Anfrage auf dem Remote Server in passender Weise bearbeiten, **HTTP Request** verwaltet nur die Anfrage und das zurückgegebene Ergebnis.

Beispiel 2

Datensatz in einer remote Anwendung hinzufügen:

```
C_TEXT($response)
$body_t:="{fName:'Jonas',fName:'Danner'}"
$httpStatus_l:=HTTP Request(HTTP PUT method;"Beispielanwendung.com";$body_t;$response)
```

Hinweis: Sie müssen die Anfrage auf dem Remote Server in passender Weise bearbeiten, **HTTP Request** verwaltet nur die Anfrage und das zurückgegebene Ergebnis.

Beispiel 3

Datensatz in JSON in einer remote Datenbank hinzufügen:

```
C_OBJECT($content)
OB SET($content;"Nachname";"Danner";"Vorname";"Jonas")
$result:=HTTP Request(HTTP PUT method;"Beispielanwendung.com";$content;$response)
```

HTTP SET CERTIFICATES FOLDER

HTTP SET CERTIFICATES FOLDER (OrdnerZertifikate)

Parameter	Typ		Beschreibung
OrdnerZertifikate	Text	➡	Pfadname und Name des Client Ordner Zertifikate

Beschreibung

Der Befehl **HTTP SET CERTIFICATES FOLDER** ändert den aktiven Ordner Client Zertifikate für alle Prozesse der aktuellen Sitzung.

In diesem Ordner sucht 4D nach den Dateien mit Client Zertifikaten, die für Web Server erforderlich sind. Standardmäßig, also ohne Ausführen des Befehls **HTTP SET CERTIFICATES FOLDER** verwendet 4D einen Ordner mit Namen "ClientCertificatesFolder", der neben der Strukturdatei liegt. Dieser Ordner wird nur bei Bedarf angelegt.

In 4D v14 lassen sich jetzt mehrere Client Zertifikate verwenden.

In *OrdnerZertifikate* übergeben Sie den Pfadnamen des eigenen Ordners mit den Client Zertifikaten. Sie können entweder einen Pfadnamen in Bezug auf die Strukturdatei der Anwendung oder einen absoluten Pfadnamen übergeben. Der Pfad muss die Schreibweise des Betriebssystems berücksichtigen. Zum Beispiel:

- (OS X): Disk:Applications:myserv:folder
- (Windows): C:\Applications\myserv\folder

Der neue Pfad wird sofort nach Ausführen dieses Befehls von später ausgeführten Befehlen wie **HTTP Request** verwendet, d.h. Sie müssen die Anwendung nicht neu starten. Er wird in allen Prozessen der Datenbank verwendet.

Existiert der angegebene Ordner nicht an der definierten Stelle oder ist der in *OrdnerZertifikate* übergebene Pfad ungültig, wird ein Fehler generiert. Sie können ihn mit einer Fehlerverwaltungsmethode abfangen, die über den Befehl **ON ERR CALL** installiert wird.

Über SSL Zertifikate

Wie im Abschnitt **TLS Protokoll verwenden (HTTPS)** beschrieben, müssen von 4D gesteuerte SSL Zertifikate im **PEM Format** sein. Sendet Ihr Provider für Zertifikate, z.B. startssl, ein Zertifikat in binärem Format, wie .crt, .pfx oder .p12 (das Format hängt auch von Ihrem Browser ab), müssen Sie es in PEM Format konvertieren, damit es verwendbar ist. Über Web Sites wie sslshopper können Sie diese Konvertierung online ausführen.

Beispiel

Den Ordner Zertifikate vorübergehend ändern:

```
C_TEXT($certifFolder)
$certifFolder :=HTTP Get certificates folder  //aktuellen Ordner sichern
HTTP SET CERTIFICATES FOLDER("C:/temp/certifTempo/")
... //spezifische Anfragen ausführen
HTTP SET CERTIFICATES FOLDER($certifFolder)  //vorigen Ordner wiederherstellen
```

HTTP SET OPTION

HTTP SET OPTION (Option ; Wert)

Parameter	Typ		Beschreibung
Option	Lange Ganzzahl	➡	Code der zu setzenden Option
Wert	Lange Ganzzahl	➡	Wert der Option

Beschreibung

Der Befehl **HTTP SET OPTION** setzt verschiedene Optionen, die von der nächsten HTTP Anfrage verwendet werden, ausgelöst über die Funktionen **HTTP Get** oder **HTTP Request**. Sie rufen diesen Befehl so oft auf, wie es Optionen zum Setzen gibt.

Hinweis: Die gesetzten Optionen gelten lokal für den aktuellen Prozess, bzw. lokal für die ausführende Komponente.

Im Parameter *Option* übergeben Sie die Nummer der zu setzenden Option, im Parameter *Wert* den neuen Wert dieser Option. Für *Option* können Sie eine der folgenden Konstanten unter dem Thema **HTTP Client** verwenden:

Konstante	Typ	Wert	Kommentar
HTTP compression	Lange Ganzzahl	6	Wert = 0 (nicht komprimieren) oder 1 (komprimieren), Standardwert: 0 Diese Option aktiviert oder deaktiviert das Komprimieren, um den Austausch für Anfragen zwischen Client und Server zu beschleunigen. Ist es aktiviert, verwendet der HTTP Client je nach der Server Antwort die Deflate oder gzip Komprimierung.
HTTP display auth dial	Lange Ganzzahl	4	Wert = 0 (Dialogfenster nicht anzeigen) oder 1 (Dialogfenster anzeigen). Standardwert: 0 Diese Option zeigt den Authentifizierungsdialog beim Ausführen der Funktion **HTTP Get** oder **HTTP Request** an. Standardmäßig wird dieses Dialogfenster nie angezeigt und Sie müssen den Befehl **HTTP AUTHENTICATE** verwenden. Soll der Authentifizierungsdialog jedoch erscheinen, damit Benutzer ihre Daten zur Identifizierung eingeben können, übergeben Sie 1 in Wert. Das Dialogfenster erscheint nur, wenn für die Anfrage eine Authentifizierung erforderlich ist
HTTP follow redirect	Lange Ganzzahl	2	Wert = 0 (keine Redirektion zulassen) oder 1 (Redirektion zulassen). Standardwert = 1
HTTP max redirect	Lange Ganzzahl	3	Wert = Maximale Anzahl der zugelassenen Redirektionen. Standardwert = 2
HTTP reset auth settings	Lange Ganzzahl	5	Wert = 0 (Information nicht löschen) oder 1 (Information löschen). Standardwert: 0 Diese Option weist 4D an, die Authentifizierungsdaten des Benutzers (Benutzername, Kennwort, Methode, etc.) nach jeder Ausführung der Funktion **HTTP Get** oder **HTTP Request** im gleichen Prozess wiederherzustellen. Diese Information wird standardmäßig beibehalten und für jede Anfrage wiederverwendet. Übergeben Sie 1 in *Wert,* um diese Information nach jedem Aufruf zu löschen. Beachten Sie, dass die Information unabhängig von der Einstellung gelöscht wird, wenn der Prozess gestoppt wird.
HTTP timeout	Lange Ganzzahl	1	Wert = Timeout der Client Anfrage in Sekunden. Dieses Timeout bestimmt, wie lange der HTTP Client auf eine Antwort des Server wartet. Ist die Zeit überschritten, schließt der Client die Sitzung und die Anfrage geht verloren. Standardmäßig sind 120 Sekunden eingestellt. Das lässt sich bei bestimmten Bedingungen, wie Netzwerkstatus, Merkmale der Anfrage, etc. ändern.

Sie können Optionen in beliebiger Reihenfolge aufrufen. Wird dieselbe Option mehrmals gesetzt, wird nur der Wert des letzten Aufrufs berücksichtigt.

Import und Export

- EXPORT DATA
- EXPORT DIF
- EXPORT ODBC
- EXPORT SYLK
- EXPORT TEXT
- IMPORT DATA
- IMPORT DIF
- IMPORT ODBC
- IMPORT SYLK
- IMPORT TEXT

EXPORT DATA

EXPORT DATA (DateiName {; Projekt {; *}})

Parameter	Typ		Beschreibung
DateiName	String	➡	Kompletter Pfadname der Exportdatei
Projekt	Textvariable, BLOB Variable	➡	Inhalt des Exportprojekts
		⬅	Neuer Inhalt des Exportprojekts (wenn Parameter * übergeben wurde)
*	Operator	➡	Zeigt Exportdialog und aktualisiert das Projekt

Beschreibung

Der Befehl **EXPORT DATA** exportiert Daten in die Datei *DateiName*. 4D kann Daten in folgenden Formaten exportieren: Text, Text mit fester Länge, XML, SYLK, DIF, DBF (dBase) und 4D.

Übergeben Sie in *DateiName* einen leeren String, zeigt **EXPORT DATA** den Standarddialog zum Sichern an. Hier kann der Benutzer Name, Typ und Ort der Exportdatei festlegen. Nach Bestätigen dieses Dialogs enthält die Systemvariable *Document* den Zugriffspfad und den Namen der Datei. Klickt der Benutzer auf die Schaltfläche **Abbrechen**, wird die Ausführung des Befehls gestoppt. Die Systemvariable *OK* hat dann den Wert 0 (Null).

Mit dem optionalen Parameter *Projekt* können Sie ein Projekt zum Exportieren der Daten verwenden. Damit wird der Export direkt ausgeführt, d.h. ohne Eingreifen des Benutzers (außer Sie verwenden die Option *, siehe unten). Übergeben Sie den optionalen Parameter *Projekt* nicht, wird das Dialogfenster für Export angezeigt. Der Benutzer kann die Exportparameter selbst festlegen oder ein bestehendes Exportprojekt laden.

Ein Exportprojekt enthält alle Exportparameter wie die Tabellen und Felder zum Exportieren, die Begrenzer, etc. Im Parameter *Projekt* können Sie entweder eine Textvariable mit XML oder mit einer Referenz auf ein zuvor vorhandenes DOM Element, bzw. ein BLOB übergeben. Projekte lassen sich per Programmierung (nur im XML Format) erstellen oder durch Laden der Parameter, die zuvor im Exportdialog definiert wurden. Im 2. Fall haben Sie dafür zwei Möglichkeiten:

- Sie verwenden den Befehl **EXPORT DATA** mit einem leeren Parameter *Projekt* und dem optionalen Parameter *. Dann speichern Sie den Parameter *Projekt* in ein Feld vom Typ Text oder BLOB (siehe unten). Auf diese Weise können Sie das Projekt mit der Datendatei sichern.
- Sie sichern das Projekt auf die Festplatte, laden es dann z.B. über die Funktion **DOM Parse XML source** und übergeben seine Referenz im Parameter *Projekt*.

Hinweis zur Kompatibilität: Ab 4D Version 12 werden Exportprojekte in XML codiert. 4D kann Exportprojekte, die mit einer früheren 4D Version im BLOB Format erstellt wurden, öffnen. Mit v12 erstellte Projekte lassen sich dagegen nicht mit Version 11 oder älter öffnen. Wir empfehlen deshalb zum Verwalten der Exportdateien Variablen vom Typ Text zu verwenden.

Ist der optionale Parameter * angegeben, wird das Dialogfenster für den Export mit den Parametern des Projekts angezeigt. So können Sie ein vordefiniertes Projekt einsetzen, und trotzdem einen bzw. mehrere Parameter ändern. Außerdem enthält der Parameter *Projekt* nach Schließen des Dialogfensters für den Export die Parameter des "neuen" Projekts. Sie können dann das neue Projekt in einem BLOB Feld, auf der Festplatte, etc. speichern.

War der Export erfolgreich, hat die Systemvariable *OK* den Wert 1.

Beispiel 1

Dieses Beispiel zeigt die Verwendung des Befehls **EXPORT DATA**, um Daten im binären Format zu exportieren.

- Diese Methode macht eine Schleife in alle Tabellen der Datenbank und ruft die Methode ***ExportBinary*** auf:

```
C_TEXT($ExportPath)
C_LONGINT($i)
$ExportPath:=Select folder("Wählen Sie den Export Ordner:")
If(Ok=1)
   For($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
```

- Hier ist der Code für die Methode ***ExportBinary***:

```
C_POINTER($1) //Tabelle
C_TEXT($2) //Pfad der Zieldatei
C_BOOLEAN($3) //exportiere alle Datensätze
C_LONGINT($i)
C_TEXT($ref)
$ref:=DOM Create XML Ref("Einstellungen-Import-Export")
 // Exportiere die Tabelle "$1" in '4D' binäres Format, alle Datensätze oder nur die aktuelle Auswahl
DOM SET XML ATTRIBUTE($ref;"table_no";Table($1);"format";"4D";"all_records";$3)
```

```
  // Definition der Felder zum Exportieren
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("Fehler während Export der Tabelle"+Table name($1))
End if
DOM CLOSE XML($ref)
```

Beispiel 2

Dieses Beispiel erstellt ein leeres Projekt und speichert die vom Benutzer gesetzten Parameter im Dialogfenster für den Export:

```
C_TEXT($exportParams)
EXPORT DATA("DocExport.txt";$exportParams;*)  //Anzeige des Dialogfensters für den Export
```

Systemvariablen und Mengen

Klickt der Benutzer im Standarddialog für Öffnen oder im Dialog für Export auf die Schaltfläche **Abbrechen**, hat die Systemvariable OK den Wert 0 (Null). War der Export erfolgreich, hat die Systemvariable OK den Wert 1.

EXPORT DIF

EXPORT DIF ({Tabellenname ;} Dokumentname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Auswahl exportiert werden soll Ohne Angabe Haupttabelle
Dokumentname	String	➡	Name des Exportdokuments

Beschreibung

Der Befehl **EXPORT DIF** exportiert die aktuelle Auswahl der Tabelle *Tabellenname* im aktuellen Ausgabeformular. Die Daten werden im Format DIF in das Dokument *Dokumentname* geschrieben.
Der Parameter *Tabellenname* ist optional. Geben Sie ihn nicht an, wird die aktuelle Auswahl der Haupttabelle exportiert.

EXPORT DIF exportiert die Felder und Variablen des aktuellen Ausgabeformulars gemäß ihrer Eingabereihenfolge. Die Datenfelder des aktuellen Ausgabeformulars werden in der Reihenfolge gefüllt, in der sie im Formular auftreten. Verwenden Sie ein Formular, das nur die Datenfelder oder eingebbaren Objekte für den Export enthält. Setzen Sie keine Schaltflächen oder andere Objekttypen in das Exportdokument. Objekte aus Unterformularen werden nicht exportiert.

Für jeden exportierten Datensatz wird ein Ereignis On Load zur Formularmethode gesendet. Verwenden Sie dieses Ereignis für Variablen im Exportdokument.

Dokumentname kann den Pfadnamen für das zu exportierende Dokument enthalten. Gibt es bereits ein Dokument mit demselben Namen, wird sein Inhalt gelöscht und mit dem neuen Inhalt überschrieben. Ist *Dokumentname* ein leerer Text, erscheint der Standarddialog zum Sichern von Dokumenten. Annulliert der Benutzer diesen Dialog, wird der Export abgebrochen und die Systemvariable auf OK auf 0 gesetzt.

Während dem Export erscheint ein Ablaufbalken. Der Benutzer kann auf die Schaltfläche **Stop** klicken, um den Export abzubrechen. Bereits importierte Datensätze werden jedoch nicht entfernt. Wurde der Export erfolgreich ausgeführt, nimmt die Systemvariable OK den Wert 1 an. Trat ein Fehler auf oder wurde die Operation unterbrochen, hat sie den Wert 0. Über den Befehl **MESSAGES OFF** lässt sich die Ablaufanzeige ausblenden.

Der Befehl verwendet standardmäßig den UTF-8 Zeichensatz. Da Dokumente im DIF Format generell den Zeichensatz IBM437 verwenden, müssen Sie evtl. über den Befehl **USE CHARACTER SET** den passenden Zeichensatz angeben.

EXPORT DIF verwendet standardmäßig den Tabulator (Code 9) als Trennung für Felder und die Zeilenschaltung (Code 13) als Trennung für Datensätze. Sie können diese beiden Trennzeichen ändern, indem Sie den zwei **Systemvariablen** *FldDelimit* (Trennung für Felder) und *RecDelimit* (Trennung für Datensätze) andere Werte zuweisen. Der Benutzer kann die Standardeinstellung im Exportdialog ändern. Da Textfelder Zeilenschaltungen enthalten können, verwenden Sie die Zeilenschaltung als Trennzeichen mit der nötigen Vorsicht.

Beispiel

Folgendes Beispiel exportiert Daten in ein DIF Dokument. Die Methode legt zuerst das Ausgabeformular an, so dass die Daten in korrekter Form exportiert werden und zeigt dann den Export an:

```
FORM SET OUTPUT([People];"Export")
EXPORT DIF([People];"NewPeople")  ` Export in das Dokument "NewPeople"
```

Systemvariablen und Mengen

OK hat den Wert 1, wenn der Export erfolgreich abgeschlossen wurde; trat ein Fehler auf, hat OK den Wert 0.

EXPORT ODBC

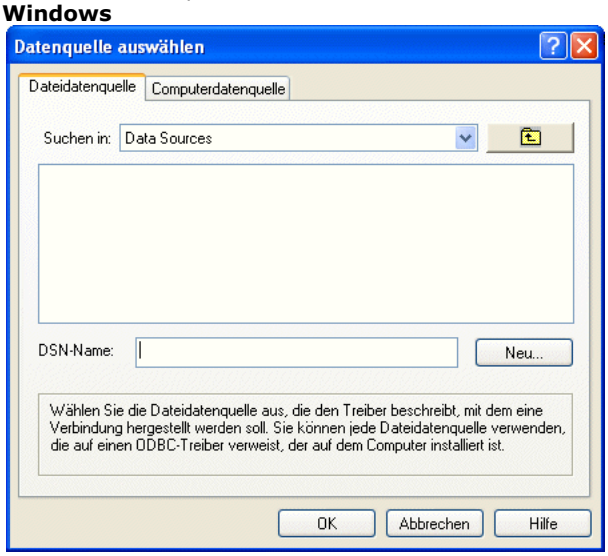
EXPORT ODBC (QuellTabelle {; Projekt {; *}})

Parameter	Typ		Beschreibung
QuellTabelle	String	➡	Name der Tabelle in ODBC Datenquelle
Projekt	BLOB	➡	Inhalt des Exportprojekts
		⬅	Mit *: Neuer Inhalt des Exportprojekts
*	Operator	➡	Anzeige des Exportdialogs und Projekt Update

Beschreibung

Der Befehl **EXPORT ODBC** exportiert Daten aus der Tabelle *QuellTabelle* einer externen ODBC Quelle. Rufen Sie den Befehl außerhalb einer Verbindung auf, die zuvor mit dem Befehl **SQL LOGIN** geöffnet wurde, erscheint das Dialogfenster zum Wählen der ODBC Datenquelle:

Windows



Mac OS

Klickt der Benutzer im Dialogfenster auf die Schaltfläche **Abbrechen**, wird die Ausführung gestoppt und die Systemvariable OK wird auf 0 (Null) gesetzt.

Übergeben Sie nicht den optionalen Parameter *Projekt*, zeigt 4D den ODBC Exportdialog, in dem der Benutzer die Operation konfigurieren kann. Weitere Informationen dazu finden Sie im Abschnitt **Import und Export über ODBC Datenquelle** des Handbuchs *4D Designmodus*.

Übergeben Sie in *Projekt* ein BLOB mit einem gültigen ODBC Exportprojekt, wird der Export direkt ausgeführt, ohne Zutun des Benutzers. Dazu genügt es, ein Projekt zu laden, dass zuvor auf der Festplatte in das Feld oder die BLOB Variable gesichert wurde, welche Sie über den Befehl **DOCUMENT TO BLOB** im Parameter *Projekt* übergeben haben. ODBC Exportprojekte werden über das Dialogfenster für ODBC Export gesichert.

Sie können **EXPORT ODBC** auch mit einem leeren Parameter *Projekt* und dem optionalen Parameter * verwenden, dann *Projekt* in einem BLOB Feld speichern (siehe unten). So können Sie einerseits das Projekt mit der Datendatei speichern und andererseits das Laden von der Festplatte in ein BLOB umgehen.

Hinweis: Unter dem Befehl **EXPORT DATA** finden Sie ein Beispiel mit der Definition eines leeren Projekts. Beachten Sie, dass Projekte, die im Dialogfenster für ODBC Export generiert wurden, weder mit den Befehlen noch mit dem Standard Exportdialog von 4D kompatibel sind.

Wird der optionale Parameter * angegeben, erscheint das Dialogfenster für ODBC Datenexport mit den in *Projekt* definierten Einstellungen – sofern vorhanden. Auf diese Weise können Sie ein vordefiniertes Projekt verwenden, dessen Parameter Sie bei Bedarf verändern können. *Projekt* enthält dann nach Schließen des Dialogfensters die neu definierten Parameter. Sie können es in einem Feld vom Typ BLOB, in einer Datei auf der Festplatte, o.ä. speichern.

Systemvariablen und Mengen

Klickt der Benutzer in einem der Dialogfenster (zum Auswählen der Datenquelle oder Exporteinstellungen) auf **Abbrechen**, wird die Systemvariable OK auf 0 gesetzt. Bei korrekt ausgeführtem Export wird die Systemvariable OK auf 1 gesetzt.

EXPORT SYLK

EXPORT SYLK ({Tabellenname ;} Dokumentname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Auswahl exportiert werden soll Ohne Angabe Haupttabelle
Dokumentname	String	➡	Name des Exportdokuments

Beschreibung

Der Befehl **EXPORT SYLK** exportiert die aktuelle Auswahl der Tabelle *Tabellenname* im aktuellen Prozess. Die Daten werden in das Dokument *Dokumentname* geschrieben, einem Windows oder Macintosh Dokument im Format SYLK auf der Festplatte. Der Parameter *Tabellenname* ist optional. Geben Sie ihn nicht an, wird die aktuelle Auswahl der Haupttabelle exportiert.

EXPORT SYLK exportiert die Felder und Variablen des aktuellen Ausgabeformulars gemäß ihrer Eingabereihenfolge. Die Datenfelder des aktuellen Ausgabeformulars werden in der Reihenfolge gefüllt, in der sie im Formular auftreten. Verwenden Sie ein Formular, das nur die Datenfelder oder eingebbaren Objekte für den Export enthält. Setzen Sie keine Schaltflächen oder andere Objekttypen in das Exportdokument. Objekte aus Unterformularen werden nicht exportiert.

Für jeden exportierten Datensatz wird ein Ereignis On Load zur Formularmethode gesendet. Verwenden Sie dieses Ereignis für Variablen im Exportdokument.

Dokumentname kann den Pfadnamen für das zu exportierende Dokument enthalten. Gibt es bereits ein Dokument mit demselben Namen, wird sein Inhalt gelöscht und mit dem neuen Inhalt überschrieben. Ist *Dokumentname* ein leerer Text, erscheint der Standarddialog zum Öffnen von Dokumenten. Annulliert der Benutzer diesen Dialog, wird der Export abgebrochen und die Systemvariable auf OK auf 0 gesetzt.

Während dem Export erscheint ein Ablaufbalken. Der Benutzer kann auf die Schaltfläche **Stop** klicken, um den Export abzubrechen. Bereits importierte Datensätze werden jedoch nicht entfernt. Wurde der Export erfolgreich ausgeführt, nimmt die Systemvariable OK den Wert 1 an. Trat ein Fehler auf oder wurde die Operation unterbrochen, hat sie den Wert 0. Über den Befehl **MESSAGES OFF** lässt sich die Ablaufanzeige ausblenden.

Der Befehl verwendet standardmäßig den UTF-8 Zeichensatz. Da Dokumente im SYLK Format generell den Zeichensatz ISO-8859-1 verwenden, müssen Sie evtl. über den Befehl **USE CHARACTER SET** den passenden Zeichensatz angeben.

EXPORT SYLK verwendet standardmäßig den Tabulator (Code 9) als Trennung für Felder, als Trennung für Datensätze auf OS X die Zeilenschaltung (Code 13), unter Windows Zeilenschaltung + Zeilenvorschub (Code 13 + Code 10). Sie können diese beiden Trennzeichen ändern, indem Sie die zwei **Systemvariablen** *FldDelimit* (Trennung für Felder) und *RecDelimit* (Trennung für Datensätze) ändern. Der Benutzer kann die Standardeinstellung im Exportdialog des Designmodus ändern. Beachten Sie, dass vorhandene Zeichen für Datenfeld- oder Datensatztrenner in der exportierten Datei automatisch durch Leerzeichen ersetzt werden, damit der Importprozess nicht unterbrochen wird.

Beispiel

Folgendes Beispiel exportiert Daten in ein SYLK Dokument. Die Methode legt zuerst das Ausgabeformular an, so dass die Daten in korrekter Form exportiert werden und zeigt dann den Export an:

```
FORM SET OUTPUT([People];"Export")
EXPORT SYLK([People];"NewPeople,slk") ` Exportiere in Dokument "NewPeople.slk"
```

Systemvariablen und Mengen

OK hat den Wert 1, wenn der Export erfolgreich abgeschlossen wurde; trat ein Fehler auf, hat OK den Wert 0.

EXPORT TEXT

EXPORT TEXT ({Tabellenname ;} Dokumentname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, aus der Daten exportiert werden sollen Ohne Angabe Haupttabelle
Dokumentname	String	➡	Textdokument für Empfangen der Daten

Beschreibung

Der Befehl **EXPORT TEXT** exportiert die aktuelle Auswahl der Tabelle *Tabellenname* im aktuellen Prozess. Die Daten werden in das Dokument *Dokumentname* geschrieben. *Tabellenname* ist optional. Geben Sie ihn nicht an, wird die aktuelle Auswahl der Haupttabelle exportiert.

EXPORT TEXT exportiert die Felder und Variablen des aktuellen Ausgabeformulars gemäß ihrer Eingabereihenfolge. Die Datenfelder des aktuellen Ausgabeformulars werden in der Reihenfolge gefüllt, in der sie im Formular auftreten. Verwenden Sie ein Formular, das nur die Datenfelder oder eingebbaren Objekte für den Export enthält. Legen Sie keine Schaltflächen oder andere Objekttypen in das Exportdokument. Objekte aus Unterformularen werden nicht exportiert.

Für jeden exportierten Datensatz wird ein Ereignis On Load zur Formularmethode gesendet. Verwenden Sie dieses Ereignis für Variablen im Exportdokument.

Dokumentname kann den Pfadnamen für das zu exportierende Dokument enthalten. Gibt es bereits ein Dokument mit demselben Namen, wird sein Inhalt gelöscht und mit dem neuen Inhalt überschrieben. Ist *Dokumentname* ein leerer Text, erscheint der Standarddialog zum Sichern von Dokumenten. Annulliert der Benutzer diesen Dialog, wird der Export abgebrochen und die Systemvariable auf OK auf 0 gesetzt.

Während dem Export erscheint ein Ablaufbalken. Der Benutzer kann auf die Schaltfläche **Stop** klicken, um den Export abzubrechen. Bereits importierte Datensätze werden jedoch nicht entfernt. Wurde der Export erfolgreich ausgeführt, nimmt die Systemvariable OK den Wert 1 an. Trat ein Fehler auf oder wurde die Operation unterbrochen, hat sie den Wert 0. Über den Befehl **MESSAGES OFF** lässt sich die Ablaufanzeige ausblenden.
Der Befehl verwendet standardmäßig den UTF-8 Zeichensatz. Um diesen zu ändern, verwenden Sie den Befehl **USE CHARACTER SET**.

EXPORT TEXT verwendet standardmäßig den Tabulator (Code 9) als Trennung für Felder, als Trennung für Datensätze auf OS X die Zeilenschaltung (Code 13), unter Windows Zeilenschaltung + Zeilenvorschub (Code 13 + Code 10). Sie können diese beiden Trennzeichen ändern, indem Sie die zwei **Systemvariablen** *FldDelimit* (Trennung für Felder) und *RecDelimit* (Trennung für Datensätze) ändern. Der Benutzer kann die Standardeinstellung im Exportdialog des Designmodus ändern. Beachten Sie, dass vorhandene Zeichen für Datenfeld- oder Datensatztrenner in der exportierten Datei automatisch durch Leerzeichen ersetzt werden, damit der Importprozess nicht unterbrochen wird,

Beispiel

Folgendes Beispiel exportiert Daten in ein Textdokument. Die Methode legt zuerst das Ausgabeformular an, so dass die Daten in korrekter Form exportiert werden, ändert die 4D Variablen für Trennzeichen und zeigt dann den Export an:

```
FORM SET OUTPUT([People];"Export")
FldDelimit:=27  ` Setze Trennzeichen für Datenfeld auf Escape
RecDelimit:=10  ` Setze Trennzeichen für Datensatz auf Line Feed
EXPORT TEXT([People];"NewPeople.txt")  ` Exportiere in Dokument "NewPeople.txt"
```

Systemvariablen und Mengen

OK hat den Wert 1, wenn der Export erfolgreich abgeschlossen wurde; tritt ein Fehler auf, hat OK den Wert 0.

IMPORT DATA

IMPORT DATA (DateiName {; Projekt {; *}})

Parameter	Typ		Beschreibung
DateiName	String	➡	Zugriffspfad und Name der Importdatei
Projekt	Textvariable, BLOB Variable	➡	Inhalt des Importprojekts
		⬅	Neuer Inhalt des Importprojekts (wenn der Parameter * übergeben wurde)
*	Operator	➡	Zeigt das Dialogfenster für den Import und aktualisiert das Projekt

Beschreibung

Der Befehl **IMPORT DATA** importiert die Daten in die Datei *DateiName*. 4D kann die Daten in folgenden Formaten importieren: Text, Text mit fester Länge, XML, SYLK, DIF, DBF (dBase) und 4D.

Übergeben Sie in *DateiName* einen leeren String, zeigt **IMPORT DATA** den Standarddialog zum Öffnen an. Hier kann der Benutzer Name, Typ und Ort der Importdatei festlegen. Nach Bestätigen dieses Dialogs enthält die Systemvariable *Document* den Zugriffspfad und den Namen der Datei. Klickt der Benutzer auf die Schaltfläche **Abbrechen**, wird die Ausführung des Befehls gestoppt. Die Systemvariable *OK* hat dann den Wert 0 (Null).

Mit dem optionalen Parameter *Projekt* können Sie ein Projekt zum Importieren der Daten verwenden. Damit wird der Import direkt ausgeführt, ohne Eingreifen des Benutzers (außer mit der Option *, siehe unten) Übergeben Sie den optionalen Parameter *Projekt* nicht, wird das Dialogfenster für Import angezeigt. Der Benutzer kann die Importparameter selbst festlegen oder ein bestehendes Importprojekt laden.

Ein Importprojekt enthält alle Importparameter wie die Tabellen und Felder, in die importiert wird, die Begrenzer, etc. Im Parameter *Projekt* können Sie entweder eine Textvariable mit XML oder mit einer Referenz auf ein zuvor vorhandenes DOM Element, bzw. ein BLOB übergeben. Projekte lassen sich per Programmierung (nur im XML Format) erstellen oder durch Laden der Parameter, die zuvor im Importdialog definiert wurden. Im 2. Fall haben Sie dafür zwei Möglichkeiten:

- Sie verwenden den Befehl **IMPORT DATA** mit einem leeren Parameter *Projekt* und dem optionalen Parameter *. Dann speichern Sie den Parameter *Projekt* in ein Feld vom Typ Text oder BLOB (siehe unten). Auf diese Weise können Sie das Projekt mit der Datendatei sichern.
- Sie sichern das Projekt auf die Festplatte, laden es dann z.B. über die Funktion **DOM Parse XML source** und übergeben seine Referenz im Parameter *Projekt*.

Hinweis zur Kompatibilität: Ab 4D Version 12 werden Importprojekte in XML codiert. 4D kann Importprojekte, die mit einer früheren 4D Version im BLOB Format erstellt wurden, öffnen. Mit v12 erstellte Projekte lassen sich dagegen nicht mit Version 11 oder älter öffnen. Wir empfehlen deshalb zum Verwalten der Importdateien Variablen vom Typ Text zu verwenden.

Ist der optionale Parameter * angegeben, wird das Dialogfenster für den Import mit den Parametern des Projekts angezeigt. So können Sie ein vordefiniertes Projekt einsetzen, und trotzdem einen bzw. mehrere Parameter ändern. Außerdem enthält der Parameter *Projekt* nach Schließen des Dialogfensters für den Import die Parameter des "neuen" Projekts. Sie können dann das neue Projekt in einem BLOB Feld, auf der Festplatte, etc. speichern.

War der Import erfolgreich, hat die Systemvariable *OK* den Wert 1.

Hinweis: Unter dem Befehl **EXPORT DATA** finden Sie ein Beispiel zum Definieren eines leeren Projekts.

Systemvariablen und Mengen

Klickt der Benutzer im Standarddialog für Sichern bzw. im Dialogfenster für den Import auf die Schaltfläche **Abbrechen**, hat die Systemvariable OK den Wert 0 (Null). War der Import erfolgreich, hat die Systemvariable OK den Wert 1.

IMPORT DIF

IMPORT DIF ({Tabellenname ;} Dokumentname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, in die importiert werden soll Ohne Angabe Haupttabelle
Dokumentname	String	➡	Name des Importdokuments

Beschreibung

Der Befehl **IMPORT DIF** importiert die Daten des Dokumentes *Dokumentname* im Format DIF. Die so eingelesenen Daten werden in Datensätzen der Tabelle *Tabellenname* gespeichert.
Der Parameter *Tabellenname* ist optional. Geben Sie ihn nicht an, wird die aktuelle Auswahl der Haupttabelle importiert.

IMPORT DIF importiert die Felder und Variablen des aktuellen Eingabeformulars gemäß ihrer Eingabereihenfolge. Die Datenfelder des aktuellen Eingabeformulars werden in der Reihenfolge gefüllt, in der sie im Formular auftreten. Verwenden Sie ein Formular, das nur die Datenfelder oder eingebbaren Objekte für den Import enthält. Setzen Sie keine Schaltflächen oder andere Objekttypen im Importdokument. Objekte aus Unterformularen werden nicht importiert. Stimmt die Zahl der Felder oder Variablen nicht mit der Zahl der Datenfelder im Importdokument überein, ignoriert 4D alle zusätzlichen Werte.

Hinweis: Um sicherzustellen, dass die Daten in die korrekten Objekte importiert werden, wählen Sie das Objekt aus, in welches das erste Feld importiert werden soll, und setzen es nach vorne. Setzen Sie dann der Reihe nach Felder und Variablen nach vorne, um sicherzustellen, das Sie für jedes importierte Feld ein Feld oder eine Variable haben.

Für jeden importierten Datensatz wird ein Ereignis On Validate zur Formularmethode gesendet. Verwenden Sie im Eingabeformular Variablen, kopieren Sie über dieses Ereignis Daten von Variablen in Datenfelder.

Dokumentname kann den Pfadnamen für das zu importierende Dokument enthalten. Ist *Dokumentname* ein leerer Text, erscheint der Standarddialog zum Öffnen von Dokumenten.

Sie können nun das zu importierende Dokument auswählen. In diesem Fall können Sie durch die Systemvariable *Document* den Namen des importierten Dokumentes erhalten. Klicken Sie auf die Schaltfläche **Öffnen**, nimmt die Systemvariable OK den Wert 1 an. Trat ein Fehler auf, hat sie den Wert 0. Haben Sie den Befehl **MESSAGES OFF** nicht aufgerufen, erscheint die Ablaufanzeige auf dem Bildschirm. Mit der Schaltfläche **Stop** brechen Sie den Import ab. Die Systemvariable OK erhält ebenfalls den Wert 0. Bereits importierte Datensätze werden jedoch nicht entfernt.

Der Befehl verwendet standardmäßig den UTF-8 Zeichensatz. Da Dokumente im DIF Format generell den Zeichensatz IBM437 verwenden, müssen Sie evtl. über den Befehl **USE CHARACTER SET** den passenden Zeichensatz angeben.

IMPORT DIF verwendet standardmäßig den Tabulator (Code 9) als Trennung für Felder und die Zeilenschaltung (Code 13) als Trennung für Datensätze. Sie können diese beiden Trennzeichen ändern, indem Sie die zwei **Systemvariablen** *FldDelimit* (Trennung für Felder) und *RecDelimit* (Trennung für Datensätze) ändern. Der Benutzer kann die Standardeinstellung im Importdialog ändern. Da Textfelder Zeilenschaltungen enthalten können, sollten Sie die Zeilenschaltung als Trennzeichen mit der nötigen Vorsicht verwenden.

Beispiel

Folgendes Beispiel importiert Daten aus einem DIF Dokument. Die Methode legt zuerst das Eingabeformular an, damit die Daten in korrekter Form importiert werden und zeigt dann den Import an:

```
FORM SET INPUT([People];"Import")
IMPORT DIF([People];"NewPeople.dif")  ` Import aus Dokument "NewPeople.dif"
```

Systemvariablen und Mengen

OK hat den Wert 1, wenn der Import erfolgreich war. Trat ein Fehler auf, hat OK den Wert 0.

IMPORT ODBC

IMPORT ODBC (QuellTabelle {; Projekt {; *}})

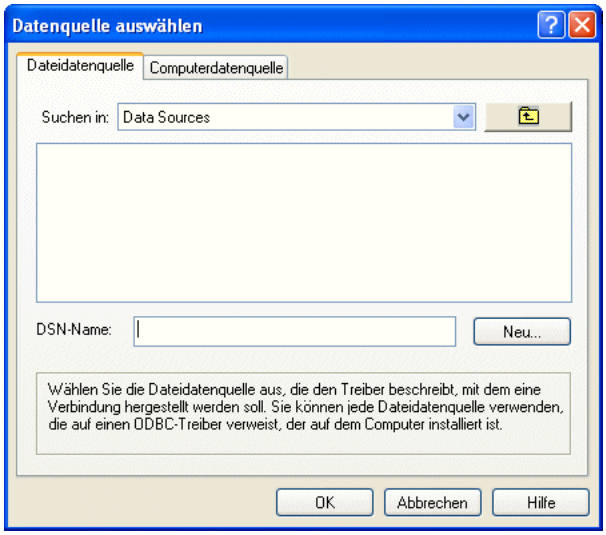
Parameter	Typ		Beschreibung
QuellTabelle	String	➡	Name der Tabelle in der Datenquelle
Projekt	BLOB	➡	Inhalt des Import-Projekts
		⬅	Mit *: Neuer Inhalt des Import-Projekts
*	Operator	➡	Anzeige des Importdialogs und Projekt Update

Beschreibung

Der Befehl **IMPORT ODBC** importiert Daten aus der Tabelle *QuellTabelle* einer externen ODBC Quelle.

Rufen Sie **IMPORT ODBC** außerhalb einer Verbindung auf, die zuvor mit **SQL LOGIN** geöffnet wurde, erscheint das Dialogfenster zum Wählen der ODBC Datenquelle:

Windows



Mac OS

Klickt der Benutzer im Dialogfenster auf die Schaltfläche **Abbrechen**, wird die Ausführung gestoppt und die Systemvariable OK auf 0 (Null) gesetzt.

Hinweis: Dieser Befehl ist bei Verbindungen mit dem internen SQL Kernel von 4D nicht verwendbar.

Ohne den optionalen Parameter *Projekt* zeigt 4D das Dialogfenster für ODBC Import, über das der Benutzer die Operation konfigurieren kann. Weitere Informationen dazu finden Sie im im Abschnitt **Import und Export über ODBC Datenquelle** des Handbuchs *4D Designmodus*.

Übergeben Sie ein BLOB mit einem gültigen ODBC Importprojekt in *Projekt*, wird der Import direkt ausgeführt, ohne Zutun des Benutzers. Dazu genügt es, ein Projekt zu laden, das zuvor auf der Festplatte im Feld oder der BLOB Variablen gesichert wurde, welche Sie über den Befehl **DOCUMENT TO BLOB** im Parameter *Projekt* übergeben haben. ODBC Importprojekte werden über das Dialogfenster für ODBC Import gesichert.

Sie können **IMPORT ODBC** auch mit einem leeren Parameter *Projekt* und dem optionalen Parameter * verwenden, dann *Projekt* in einem BLOB Feld speichern (siehe unten). So können Sie einerseits das Projekt mit der Datendatei speichern und andererseits das Laden von der Festplatte in ein BLOB umgehen.

Hinweis: Unter dem Befehl **EXPORT DATA** finden Sie ein Beispiel mit der Definition eines leeren Projekts. Beachten Sie, dass Projekte, die über den ODBC Importdialog erstellt werden, weder mit den Befehlen noch mit dem Standard Importdialog von 4D kompatibel sind.

Wird der optionale Parameter * angegeben, erscheint das Dialogfenster für ODBC Datenimport mit den in *Projekt* definierten Einstellungen – sofern vorhanden. Auf diese Weise können Sie ein vordefiniertes Projekt verwenden, dessen Parameter Sie bei Bedarf verändern können. *Projekt* enthält dann nach Schließen des Dialogfensters die neu definierten Parameter. Sie können es in einem Feld vom Typ BLOB, in einer Datei auf der Festplatte, o.ä. speichern.

Systemvariablen und Mengen

Klickt der Benutzer in einem der Dialogfenster (zum Auswählen der Datenquelle oder Importeinstellungen) auf die Schaltfläche **Abbrechen**, wird die Systemvariable OK auf 0 (Null) gesetzt. Bei korrekt ausgeführtem Import wird die Systemvariable OK auf 1 gesetzt.

IMPORT SYLK

IMPORT SYLK ({Tabellenname ;} Dokumentname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, in die importiert werden soll Ohne Angabe Haupttabelle
Dokumentname	String	➡	Name des Importdokuments

Beschreibung

Der Befehl **IMPORT SYLK** importiert die Daten des Dokumentes *Dokumentname* im Format SYLK. Die so eingelesenen Daten werden in neuen Datensätzen der Tabelle *Tabellenname* gespeichert.
Der Parameter *Tabellenname* ist optional. Geben Sie ihn nicht an, wird die aktuelle Auswahl der Haupttabelle importiert.
IMPORT SYLK importiert die Felder und Variablen des aktuellen Eingabeformulars gemäß ihrer Eingabereihenfolge. Die Datenfelder des aktuellen Eingabeformulars werden in der Reihenfolge gefüllt, in der sie im Formular auftreten. Verwenden Sie ein Formular, das nur die Datenfelder oder eingebbaren Objekte für den Import enthält. Setzen Sie keine Schaltflächen oder andere Objekttypen im Importdokument. Objekte aus Unterformularen werden nicht importiert. Stimmt die Zahl der Felder oder Variablen nicht mit der Zahl der Datenfelder im Importdokument überein, ignoriert 4D alle zusätzlichen Werte.

Hinweis: Um sicherzustellen, dass die Daten in die korrekten Objekte importiert werden, wählen Sie das Objekt aus, in welches das erste Feld importiert werden soll, und setzen es nach vorne. Setzen Sie dann der Reihe nach Felder und Variablen nach vorne, um sicherzustellen, das Sie für jedes importierte Feld ein Feld oder eine Variable haben.

Für jeden importierten Datensatz wird ein Ereignis On Validate zur Formularmethode gesendet. Verwenden Sie im Eingabeformular Variablen, kopieren Sie über dieses Ereignis Daten von Variablen in Datenfelder.

Dokumentname kann den Pfadnamen für das zu importierende Dokument enthalten. Ist Dokumentname ein leerer Text, erscheint der Standarddialog zum Öffnen von Dokumenten. Annulliert der Benutzer diesen Dialog, wird der Import abgebrochen und die Systemvariable auf OK auf 0 gesetzt.

Während dem Import erscheint ein Ablaufbalken. Der Benutzer kann auf die Schaltfläche **Stop** klicken, um den Import abzubrechen. Bereits importierte Datensätze werden jedoch nicht entfernt. Wurde der Import erfolgreich ausgeführt, nimmt die Systemvariable OK den Wert 1 an. Trat ein Fehler auf oder wurde die Operation unterbrochen, hat sie den Wert 0. Über den Befehl **MESSAGES OFF** lässt sich die Ablaufanzeige ausblenden.

Der Befehl verwendet standardmäßig den UTF-8 Zeichensatz. Da Dokumente im SYLK Format generell den Zeichensatz ISO-8859-1 nutzen, müssen Sie evtl. über den Befehl **USE CHARACTER SET** den passenden Zeichensatz angeben.

IMPORT SYLK verwendet standardmäßig den Tabulator (Code 9) als Trennung für Felder und die Zeilenschaltung (Code 13) als Trennung für Datensätze. Sie können diese beiden Trennzeichen ändern, wenn Sie den zwei **Systemvariablen** *FldDelimit* (Trennung für Felder) und *RecDelimit* (Trennung für Datensätze) andere Werte zuweisen. Der Benutzer kann die Standardeinstellung im Exportdialog ändern. Da Textfelder Zeilenschaltungen enthalten können, sollten Sie die Zeilenschaltung als Trennzeichen mit der nötigen Vorsicht verwenden.

Beispiel

Folgendes Beispiel importiert Daten aus einem SYLK Dokument. Die Methode legt zuerst das Eingabeformular an, so dass die Daten in korrekter Form importiert werden und zeigt dann den Import an:

```
FORM SET INPUT([People];"Import")
IMPORT SYLK([People];"NewPeople.slk")  ` Importiere aus Dokument “NewPeople.slk
```

Systemvariablen und Mengen

OK hat den Wert 1, wenn der Import erfolgreich war. Trat ein Fehler auf, hat OK den Wert 0.

IMPORT TEXT

IMPORT TEXT ({Tabellenname ;} Dokumentname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, in die importiert werden soll Ohne Angabe Haupttabelle
Dokumentname	String	➡	Textdokument, aus dem Daten importiert werden

Beschreibung

Der Befehl **IMPORT TEXT** importiert die Daten des Dokumentes *Dokumentname*, ein Windows oder Macintosh Textdokument und liest sie in neu erstellte Datensätze der Tabelle *Tabellenname* ein.
Der Parameter *Tabellenname* ist optional. Geben Sie ihn nicht an, wird die aktuelle Auswahl der Haupttabelle importiert.

IMPORT TEXT importiert die Felder und Variablen des aktuellen Eingabeformulars gemäß ihrer Eingabereihenfolge. Die Datenfelder des aktuellen Eingabeformulars werden in der Reihenfolge gefüllt, in der sie im Formular auftreten. Verwenden Sie ein Formular, das nur die Datenfelder oder eingebbaren Objekte für den Import enthält. Eingabeformulare für den Import können keine Schaltflächen enthalten. Objekte aus Unterformularen werden nicht importiert. Stimmt die Zahl der Felder oder Variablen nicht mit der Zahl der Datenfelder im Importdokument überein, ignoriert 4D alle zusätzlichen Werte.

Hinweis: Um sicherzustellen, dass die Daten in die korrekten Objekte importiert werden, wählen Sie das Objekt aus, in welches das erste Feld importiert werden soll, und setzen es nach vorne. Setzen Sie dann der Reihe nach Felder und Variablen nach vorne, um sicherzustellen, das Sie für jedes importierte Feld ein Feld oder eine Variable haben.

Für jeden importierten Datensatz wird ein Ereignis On Validate zur Formularmethode gesendet. Verwenden Sie im Eingabeformular Variablen, kopieren Sie über dieses Ereignis Daten von Variablen in Datenfelder.

Dokumentname kann den Pfadnamen für das zu importierende Dokument enthalten. Ist *Dokumentname* ein leerer Text, erscheint der Standarddialog zum Öffnen von Dokumenten. Annulliert der Benutzer diesen Dialog, wird der Import abgebrochen und die Systemvariable auf OK auf 0 gesetzt.

Während dem Import erscheint ein Ablaufbalken. Der Benutzer kann auf die Schaltfläche **Stop** klicken, um den Import abzubrechen. Bereits importierte Datensätze werden jedoch nicht entfernt. Wurde der Import erfolgreich ausgeführt, nimmt die Systemvariable OK den Wert 1 an. Trat ein Fehler auf oder wurde die Operation unterbrochen, hat sie den Wert 0. Über den Befehl **MESSAGES OFF** lässt sich die Ablaufanzeige ausblenden.

Der Befehl verwendet standardmäßig den Zeichensatz UTF-8. Um diesen zu ändern, verwenden Sie den Befehl **USE CHARACTER SET**.

IMPORT TEXT importiert die Daten mit 2 Trennzeichen:

- Mit dem Tabulator (Code 9) als Trennung für Felder.
- Mit der Zeilenschaltung (Code 13) als Trennung für Datensätze.

Sie können diese beiden Trennzeichen ändern, indem Sie den zwei **Systemvariablen** *FldDelimit* (Trennung für Felder) und *RecDelimit* (Trennung für Datensätze) andere Werte zuweisen. Der Benutzer kann die Standardeinstellung im Importdialog ändern. Da Textfelder Zeilenschaltungen enthalten können, verwenden Sie die Zeilenschaltung als Trennzeichen mit der nötigen Vorsicht.

Beispiel

Folgendes Beispiel importiert Daten aus einem Textdokument. Die Methode legt zuerst das Eingabeformular an, so dass die Daten in korrekter Form importiert werden, ändert die 4D Variablen für Trennzeichen und zeigt dann den Import an:

```
FORM SET INPUT([People];"Import")
FldDelimit:=27  ` Setze Trennung für Felder auf Escape
RecDelimit:=10  ` Setze Trennung für Datensatz auf Line Feed
IMPORT TEXT([People];"NewPeople.txt")  ` Importiere aus Dokument "NewPeople.txt"
```

Systemvariablen und Mengen

OK hat den Wert 1, wenn der Import erfolgreich war. Tritt ein Fehler auf, hat OK den Wert 0.

JSON

- Überblick über JSON Befehle
- JSON Parse
- JSON PARSE ARRAY
- JSON Resolve pointers
- JSON Stringify
- JSON Stringify array
- JSON TO SELECTION
- JSON Validate
- Selection to JSON

Überblick über JSON Befehle

JSON Befehle generieren und analysieren Objekte der Programmiersprache im Format JSON. Das JSON Format ermöglicht insbesondere, über einen Web Browser auf 4D Anwendungen zuzugreifen (Daten und Struktur).

Die Programmiersprache von 4D v14 bietet als neues Haupt-Feature die Unterstützung von strukturierten Objekten, um den Austausch von strukturierten Daten zu erleichtern. Über die Befehle im Kapitel "JSON" kann 4D direkt mit JSON Objekten arbeiten, aber genauso mit "native" Objekten, d.h. mit von JSON inspirierter Struktur, die den Austausch mit vielen anderen Programmiersprachen möglich machen. Weitere Informationen dazu finden Sie im Kapitel **Objekte (Sprache)**.

Einführung in JSON

"JSON oder JavaScript Object Notation ist ein generisches text-basiertes Datenformat, das sich ableitet von der Objektnotation der ECMAScript Sprache" (Quelle: Wikipedia). JSON ist komplett unabhängig von Programmiersprachen, folgt aber vielen Konventionen, die Programmierer von C-basierten Sprachen (C++, JavaScript, Perl, u.v.a.) kennen. Dieses Format eignet sich besonders zum Datenaustausch.

Dieser Abschnitt gibt einen Überblick über die in JSON integrierten Merkmale für Notation. Eine ausführliche Beschreibung dazu finden Sie unter: www.json.org/index.html.

JSON Syntax

Die Syntax von JSON basiert auf folgenden Grundlagen:

- Daten bestehen aus Name/Wert Paaren
- Daten werden durch Komma getrennt
- Objekte werden durch geschweifte Klammern { } definiert
- Arrays werden durch eckige Klammern [ ] definiert

JSON Eigenschaften

JSON Daten werden in Paaren Name/Wert oder Schlüssel/Wert ausgedrückt. Ein solches Paar enthält einen Feldnamen in Anführungszeichen, gefolgt von einem Doppelpunkt, und einen Wert. Zum Beispiel:

```
"vorName":"Hans"
```

Zur Info: Dieses Beispiel sieht in JavaScript folgendermaßen aus:

```
vorName="Hans"
```

Beachten Sie, dass Eigennamen diakritische Zeichen und Groß-/Kleinschreibung berücksichtigen. Schreiben Sie "VorName" anstelle von "vorname," erhalten Sie ein neues Name/Wert Paar.

JSON Datentypen

JSON unterstützt folgende Wertetypen:

Typ	Beschreibung	Kommentar
String	Jedes Unicode Zeichen mit Ausnahme von " und \. Werte wie Eigennamen, stehen in Anführungszeichen ("), z.B. "Stadt":"Berlin" </td></tr></table>[#/html]	\ wird für Steuerzeichen verwendet: \" = Anführungszeichen \\ = Umgekehrte Schrägstriche \/ = Schrägstrich \b = Rückschritt \f = Seitenvorschub \n = Zeilenvorschub \r = Zeilenschaltung \t = Tabulator \u = vier hexadezimale Stellen
Zahl	Zahl oder Zahl mit Fließkomma	ähnlich zu einer Zahl in C oder Java, mit Ausnahme von oktalen und hexadezimalen Formaten
Objekt	{ }	
Array	[ ]	
Boolean	true oder false	
Null	null	

JSON Objekte

JSON Objekte werden mit Klammern definiert. Sie können eine undefinierte Anzahl von Name/Wert Paaren enthalten, z.B.:

```
{ "vorName":"Hans" , "nachName":"Dampf" }
```

JSON Objekte lassen sich in 4D über Variablen und Felder vom Typ **Objekt** speichern und verwalten (**C_OBJECT**).

JSON Arrays

JSON Arrays werden mit Klammern definiert. Jedes Array kann eine undefinierte Anzahl Objekte enthalten:

```
{    "Angestellte": [        { "vorName":"Hans" , "nachName":"Dampf" },        { "vorName":"Anna" , "nachName":"Schmid" },        { "vorName":"Peter" , "nachName":"Meier" }    ] }
```

JSON Arrays lassen sich in 4D über Variablen und Felder vom Typ **Collection** speichern und verwalten (**C_COLLECTION**).

JSON Zeiger

4D unterstützt JSON Zeiger. Ein JSON Zeiger ist ein String, um im gesamten JSON Dokument auf einen bestimmten Feld- oder Schlüsselwert zuzugreifen. Gemäß Konvention kann eine URI mit einem JSON Zeiger in einer JSON Objekteigenschaft liegen, die den Namen "$ref" haben muss.

```
{    "$ref":<path>#<json_pointer> }
```

JSON Zeiger werden entweder durch Aufrufen des Befehls **JSON Resolve pointers** oder automatisch über **Dynamische Formulare** aufgelöst.

Weitere Informationen dazu finden Sie unter **JSON Resolve pointers**.

Unterstützung der Zeitzone

4D Datumsangaben, die von oder in das JSON Format konvertiert werden, berücksichtigen standardmäßig die Zeitzone des konvertierenden Rechners. Dieses Prinzip entspricht der Standardoperation von Java-Script. Zum Beispiel ergibt Konvertieren des Datums !23/08/2013! in Deutschland (GMT+2) als Ergebnis "2013-08-22T22:00:00Z" und umgekehrt.

Dieses Standardverhalten können Sie mit dem Befehl **SET DATABASE PARAMETER**, Selektor Dates inside objects ändern, z.B. wenn Sie beim Implementieren von Exportoperationen keine Zeitzonen berücksichtigen wollen. Beachten Sie, dass dieser Selektor auch beeinflussen kann, wie Datumsangaben in Objekten gespeichert werden.

Weitere Informationen dazu finden Sie im Abschnitt **Konvertieren von Datum in JavaScript**.

Hinweis: Ab 4D v16 R6 werden auch JSON Datum-Strings im Format "YYYY-MM-DD" unterstützt. Weitere Informationen dazu finden Sie auf der **Seite Kompatibilität**, Option *Verwende Datumstyp statt ISO Datumsformat in Objekten*.

JSON Parse

JSON Parse (jsonString {; Typ}{; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
jsonString	String	➡	Zu analysierender JSON String
Typ	Lange Ganzzahl	➡	Typ, in den die Werte konvertiert werden sollen
*	Operator	➡	Fügt Zeilenposition und Versatz jeder Eigenschaft hinzu, wenn zurückgegebener Wert ein Objekt ist.
Funktionsergebnis	Mixed, Objekt	⮌	Aus JSON String extrahierte Werte

Beschreibung

Die Funktion **JSON Parse** analysiert den Inhalt einer in JSON formatierten Zeichenkette und extrahiert Werte, die Sie in einem 4D Feld oder einer Variablen speichern können. Diese Funktion deserialisiert Daten im JSON-Format; sie führt die entgegengesetzte Aktion der Funktion **JSON Stringify** durch.

In *jsonString* übergeben Sie die in JSON formatierte Zeichenkette, deren Inhalt Sie analysieren wollen. Sie muss korrekt formatiert sein, sonst wird ein Analyse-Fehler erzeugt. **JSON Parse** lässt sich von daher zum Bestätigen von JSON Strings verwenden.

Hinweis: Beim Verwenden von Zeigern müssen Sie zuerst **JSON Stringify** und dann **JSON Parse** aufrufen.

Standardmäßig, d.h. ohne den Parameter *Typ*, versucht 4D, den erhaltenen Wert in den Variablen- oder Feldtypen zu konvertieren, der zum Speichern der Ergebnisse verwendet wird - sofern einer definiert wurde. Andernfalls versucht 4D, den Typ einzuschätzen. Über den Parameter *Typ* können Sie die Interpretation des Typs erzwingen. Übergeben Sie eine der folgenden Konstanten unter dem Thema **Feld und Variablentypen**:

Konstante	Typ	Wert
Is Boolean	Lange Ganzzahl	6
Is collection	Lange Ganzzahl	42
Is date	Lange Ganzzahl	4
Is longint	Lange Ganzzahl	9
Is object	Lange Ganzzahl	38
Is real	Lange Ganzzahl	1
Is text	Lange Ganzzahl	2
Is time	Lange Ganzzahl	11

Hinweise:

- Werte vom Typ Zahl müssen im Bereich ±10.421e±10 liegen
- In Werten vom Typ Text müssen alle Sonderzeichen in Escape-Sequenzen stehen, inkl. Anführungszeichen (siehe Beispiele)
- Standardmäßig (mit der Konstante Is date) geht die Funktion davon aus, dass das 4D Datum eine lokale Zeit enthält und nicht GMT. Sie können diese Einstellung über den Selektor Dates inside objects des Befehls **SET DATABASE PARAMETER** ändern.
- Ab 4D v16 R6 ist die aktuelle Einstellung zum Speichern des Datums vom Typ Datum, **JSON Parse** gibt JSON Strings im Format "YYYY-MM-DD" automatisch als Datumswerte zurück. Weitere Informationen dazu finden Sie auf der **Seite Kompatibilität**, Option "Verwende Datumstyp statt ISO Datumsformat in Objekten".
- Werte vom Typ Zeit lassen sich aus Zahlen in Strings zurückgeben. Standardmäßig wird der analysierte Wert als Anzahl von Sekunden gewertet.

Übergeben Sie den optionalen Parameter * und ist der Parameter *jsonString* ein Objekt, enthält das zurückgegebene Objekt die zusätzliche Eigenschaft *__symbols*, die für jede Eigenschaft und Untereigenschaft des Objekts Pfad, Zeilenposition und Zeilenversatz anzeigt. Diese Information kann beim Debuggen nützlich sein. Die Eigenschaft *__symbols* hat folgende Struktur:

```
__symbols:{//Objektbeschreibung    myAtt.mySubAtt...:{ //Pfad der Eigenschaft    line:10, //Zeilennummer der Eigenschaft    offset:35 //Versatz der Eigenschaft ab Zeilenbeginn    }  }
```

Hinweis: Der Parameter * wird ignoriert, wenn der zurückgegebene Wert nicht Objekt als *Typ* hat.

Beispiel 1

Beispiele für eine einfache Konvertierung:

```
C_REAL($r)
$r:=JSON Parse("42.17")  //$r = 42,17 (Zahl)

C_LONGINT($el)
$el:=JSON Parse("120.13";Is longint)  //$el=120

C_TEXT($t)
$t:=JSON Parse("\"Jahr 42\"";Is text)  // $t="Jahr 42" (Text)

C_OBJECT($o)
$o:=JSON Parse("{\"name\":\"jean\"}")
 // $o = {"name":"jean"} (4D Objekt)
```

```
C_BOOLEAN($b)
$b:=JSON Parse("{\"manager\":true}";ls Boolean)  // $b=true

C_TIME($h)
$h:=JSON Parse("5120";ls time)  //$h=01:25:20
```

Beispiel 2

Beispiel zum Konvertieren eines Datums:

```
$test:=JSON Parse("\"1990-12-25T12:00:00Z\"")
 // $test=1990-12-25T12:00:00Z
C_DATE($date;$date2;$date3)
$date:=JSON Parse("\"2008-01-01T12:00:00Z\"";ls date)
 //$date=01/01/08
$date2:=JSON Parse("\"2017-07-13T23:00:00.000Z\"";ls date)
 //$date2=14/07/17 (Paris Zeitzone)
SET DATABASE PARAMETER(Dates inside objects;String type without time zone)
$date3:=JSON Parse("\"2017-07-13T23:00:00.000Z\"";ls date)
 //$date3=13/07/17
```

Beispiel 3

Ist die aktuelle Datumseinstellung zum Speichern vom Typ Datum, schreiben Sie folgenden Code:

```
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
```

Hinweis: Weitere Informationen dazu finden Sie auf der **Seite Kompatibilität**, Option "Verwende Datumstyp statt ISO Datumsformat in Objekten".

Beispiel 4

Dieses Beispiel zeigt die kombinierte Verwendung der Befehle **JSON Stringify** und **JSON Parse**:

```
C_TEXT($JSONContact)
C_OBJECT($Contact;$Contact2)
$Contact:=New object("name";"Monroe";"firstname";"Alan")

 // JSON Stringify: Ein Objekt in einen JSON String konvertieren
$JSONContact:=JSON Stringify($Contact)

 // JSON Parse: Ein JSON String in ein neues Objekt konvertieren
$Contact2:=JSON Parse($JSONContact)
```

Beispiel 5

Eine 4D Collection aus einem JSON Array erstellen:

```
C_COLLECTION($myCol)
$myCol:=JSON Parse("[\"Monday\",10,\"Tuesday\",11,\"Wednesday\",12,false]")
```

Beispiel 6

Nachfolgenden String durchlaufen und Zeilenposition und Versatz jeder Eigenschaft erhalten:

```
{    "alpha": 4552,    "beta": [        {            "echo": 45,            "delta": "text1"        },        {            "echo": 52,            "golf": "text2"        }    ] }
```

Sie können schreiben:

```
C_OBJECT($obInfo)
$obInfo=JSON Parse("json_string";ls object;*)  //* um im zurückgegebenen Objekt $obInfo
  //die Eigenschaft __symbols zu erhalten
```

Das Objekt *$obInfo* enthält:

{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 (jsonString ; array)

Parameter	Typ		Beschreibung
jsonString	String	➡	Zu analysierender JSON String
array	Array	⬅	Array mit dem Analyseergebnis des JSON String

Beschreibung

Der Befehl **JSON PARSE ARRAY** analysiert den Inhalt einer in JSON formatierten Zeichenkette und setzt die extrahierten Daten in das Array *array*. Dieser Befehl deserialisiert die JSON Daten; er führt die entgegengesetzte Aktion der Funktion **JSON Stringify array** durch.

In *jsonString* übergeben Sie die in JSON formatierte Zeichenkette, deren Inhalt Sie analysieren wollen. Sie muss korrekt formatiert sein, sonst wird ein Analysefehler erzeugt.

In *array* übergeben Sie ein Array vom gewünschten Typ zum Empfangen der Analyseergebnisse.

Hinweis: Ab 4D v16 R4 können Sie **JSON PARSE ARRAY** in der Regel durch einen Aufruf von **JSON Parse** ersetzen, der eine **Collection** zurückgibt. Sie basieren auf JSON Arrays und ermöglichen, Daten unterschiedlicher Typen zu speichern. Das bietet größere Flexibilität als Arrays.

Beispiel

Dieses Beispiel extrahiert die Daten aus Feldern der Datensätze in einer Tabelle und setzt sie dann in Objekt Arrays:

```
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 (Objekt {; Optionen}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Objekt	Objekt	➡	Objekt mit JSON Zeigern zum Auflösen
		⬅	Objekt mit aufgelösten JSON Zeigern (nur wenn Ergebnis ein Objekt ist)
Optionen	Objekt	➡	Optionen für Zeigerauflösung
Funktionsergebnis	Objekt	↩	Objekt mit Ergebnis der Bearbeitung

Beschreibung

Die Funktion **JSON Resolve pointers** löst alle in *Objekt* gefundenen JSON Zeiger auf, optional mit den Einstellungen in *Optionen*.

JSON Zeiger sind besonders nützlich, um:

- einen Teil eines externen JSON Dokuments einzubinden oder einen Teil des JSON Dokuments an anderer Stelle desselben Dokuments wiederzuverwenden
- eine zyklische Struktur in JSON auszudrücken
- ein Vorlageobjekt mit in JSON gespeicherten Standardeigenschaften zu definieren

Im Parameter *Objekt* übergeben Sie ein Objekt mit aufzulösenden JSON Zeigern. Weitere Informationen dazu finden Sie im unteren Abschnitt **JSON Zeiger definieren**.

Hinweis: Nach Ausführen der Funktion wird das ursprüngliche *Objekt* mit den aufgelösten Zeigern aktualisiert (außer, das Ergebnis ist kein Objekt, s.o.). Wollen Sie die Originalversion von *Objekt* beibehalten, müssen Sie zuvor den Befehl **OB Copy** verwenden.

Optional können Sie in *Optionen* ein Objekt mit spezifischen Eigenschaften übergeben, die beim Auflösen der Zeiger verwendet werden sollen. Es gibt folgende Eigenschaften:

Eigenschaft	Wertetyp	Beschreibung
rootFolder	String	Absoluter Pfad (mit standardmäßiger 4D Syntax) zum Ordner, der für Auflösen relativer Zeiger in *Objekt* dient. Standardmäßig ist es der Ordner *Resources* der Anwendung.
merge	Boolean	Objekte und Zeigerobjekte (*true*) zusammenlegen, anstatt diese zu ersetzen (*false*). Standard ist *false*

Nach Ausführen der Funktion gilt folgendes:

- ist das Ergebnis der Zeigerauflösung ein Objekt, wird *Objekt* aktualisiert und enthält das Ergebnisobjekt.
- ist das Ergebnis der Zeigerauflösung ein skalarer Wert (z.B. ein Text, eine Zahl ...), bleibt *Objekt* unverändert und der Ergebniswert wird in der Eigenschaft "value" des Funktionsergebnisses zurückgegeben.

In allen Fällen gibt die Funktion ein Objekt mit folgenden Eigenschaften zurück:

Eigenschaft	Wertetyp	Beschreibung
value	Beliebig	Funktionsergebnis nach Bearbeiten von *Objekt*. Ist das Ergebnis ein Objekt, ist es gleich mit dem Ausgabe *Objekt*.
success	Boolean	Wahr, wenn alle Zeiger erfolgreich aufgelöst wurden
errors	Collection	Fehler-Collection falls zutreffend
errors[].code	Zahl	Fehlercode
errors[].message	String	Fehlermeldung
errors[].pointerURI	String	Zeigerwert
errors[].referredPath	String	Vollständiger Pfad des Dokuments

JSON Zeiger definieren

JSON Zeiger ist ein Standard in Form eines String, um auf einen bestimmten Feld- oder Schlüsselwert im gesamten JSON Dokument zuzugreifen. Weitere Informationen zum Standard finden Sie unter RFC 6901.

Genaugenommen ist ein JSON Zeiger ein String aus mehreren Teilen, getrennt durch '/'. Ein JSON Zeiger liegt gewöhnlich in einer URL zum Dokument, in dem der Zeiger aufgelöst wird. Das Zeichen "#' innerhalb der URI dient zur Kennzeichnung des JSON Zeigers. Gemäß Konvention kann eine URI mit einem JSON Zeiger in einer JSON Objekteigenschaft liegen, die den Namen "$ref" haben muss.

{ "$ref":<path>#<json_pointer> }

Hinweis: 4D unterstützt nicht das Zeichen "-" als Referenz auf nicht-existierende Array-Elemente.

Rekursivität und Pfadauflösung

JSON Zeiger werden rekursiv aufgelöst, d.h. enthält ein aufgelöster Zeiger ebenfalls Zeiger, werden diese wiederum rekursiv aufgelöst, bis alle Zeiger aufgelöst sind. In diesem Kontext können alle JSON Zeiger URIs in Dateipfaden relativ oder absolut sein. Sie müssen das Zeichen '/' als Pfadtrenner nutzen und werden folgendermaßen aufgelöst:

- Ein relativer Pfad darf nicht mit '/' starten. Er wird relativ zum JSON Dokument aufgelöst, in dem der Pfad-String gefunden wurde.
- Ein absoluter Pfad startet mit '/'. Aus Sicherheitsgründen wird nur "/RESOURCES" als absoluter Pfad akzeptiert und bezeichnet den Ordner Ressources der aktuellen Anwendung. Beispiel: "/RESOURCES/templates/myfile.json" zeigt auf die Datei "myfile.json" innerhalb des Ordners Ressources der aktuellen Anwendung.

Hinweise:

- Die Namensauflösung berücksichtigt Groß- und Kleinschreibung
- 4D löst keinen Pfad auf eine Json Datei auf, der über das Netzwerk läuft (beginnt mit "http/https")

Beispiel 1

Dieses grundlegende Beispiel veranschaulicht, wie sich ein JSON Zeiger setzen und in einem Objekt ersetzen lässt:

```
 // Objekt mit Wert erstellen
C_OBJECT($o)
$o:=New object("value";42)

 // das Objekt JSON Zeiger erstellen
C_OBJECT($ref)
$ref:=New object("$ref";"#/value")

 // das Objekt JSON Zeiger als Eigenschaft erstellen
$o.myJSONPointer:=$ref

 // Das Ganze auflösen und prüfen, ob der Zeiger aufgelöst wurde
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
```

Beispiel 2

Im JSON Objekt mit Namen $oMyConfig die Rechnungsadresse auch als Lieferadresse wiederverwenden:

```
{ "lastname": "Doe", "firstname": "John", "billingAddress": { "street": "95 S. Market Street", "city": "San Jose", "state": "California" }, "shippingAddress": { "$ref": "#/billingAddress" } }
```

Nach Ausführen dieses Code:

```
$oResult:=JSON Resolve pointers($oMyConfig)
```

... wird folgendes Objekt zurückgegeben:

```
{ "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" } } }
```

Beispiel 3

Dieses Beispiel zeigt die Auswirkung der Option "merge". Sie wollen die Rechte eines Benutzers anhand einer Standarddatei bearbeiten.

```
{ "rights": { "$ref": "defaultSettings.json#/defaultRights", "delete": true, "id": 456 } }
```

Die Datei *defaultSettings.json* enthält:

```
{ "defaultRights": { "edit": true, "add": false, "delete": false } }
```

Führen Sie folgendes aus:

```
C_OBJECT($options)
$options:=New object("merge";False)  //Inhalt ersetzen
```

```
$oResult:=JSON Resolve pointers($oMyConfig;$options)
```

... ist der Ergebniswert exakt der Inhalt der Datei *defaultSettings.json*:

{ "success": true, "value": { "rights": { "edit": true, "add": false, "delete": false } } }

Führen Sie folgendes aus:

```
C_OBJECT($options)
$options:=New object("merge";True)  //beide Inhalte zusammenlegen
$oResult:=JSON Resolve pointers($oMyConfig;$options)
```

... ist der Ergebniswert eine zusammengeführte Version des Originalobjekts mit der Vorlage:

{ "success": true, "value": { "rights": { "edit": true, "add": false, "delete": true, "id": 456 } } }

JSON Stringify

JSON Stringify (Wert {; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Objekt, Mixed	➡	Daten zum Konvertieren in JSON String
*	Operator	➡	Formatierung aktivieren
Funktionsergebnis	Text	⮌	String mit dem serialisierten JSON Text

Beschreibung

Die Funktion **JSON Stringify** konvertiert den Parameter *Wert* in einen JSON String. Sie serialisiert Daten in JSON; sie führt die entgegengesetzte Aktion der Funktion **JSON Parse** durch.

In *Wert* übergeben Sie Daten zum Serialisieren. Sie lassen sich in skalarer Form, also String, Zahl, Datum oder Zeit, bzw. als 4D Objekt, Objekt Array oder Collection anzeigen.

Hinweis: 4D Datumsangaben werden entweder in das Format "yyyy-mm-dd" oder "YYYY-MM-DDThh:mm:sssZ" konvertiert. Das richtet sich nach der aktuellen Datumseinstellung der Datenbank (siehe auf der **Seite Kompatibilität**, Option "Verwende Datumstyp statt ISO Datumsformat in Objekten").

Bei einem Objekt oder einer Collection sind alle Typen verwendbar (siehe Abschnitt **JSON Datentypen**). Die JSON Formatierung muss folgende Regeln berücksichtigen:

- String Werte stehen in Anführungszeichen. Alle Unicode Zeichen sind verwendbar, bei Sonderzeichen wird ein umgekehrter Schrägstrich \ vorangestellt
- Zahlen: Bereich ±10.421e±10
- Booleans: String "true" oder "false"
- Datum (Format "yyyy-mm-dd" oder "\"YYYY-MM-DDTHH:mm:ssZ"\", gemäß der aktuellen Einstellung der Datenbank; siehe oben)
- Zeit: Typ Zahl (standardmäßig Anzahl von Sekunden)
  Hinweise:
  - Bilder werden in folgenden String konvertiert: "[Objekt Bild]".
  - Zeiger auf Feld, Variable oder Array werden beim Umwandeln in String gewertet

Sie können den optionalen Parameter * übergeben, um im resultierenden String Formatierungszeichen einzuschließen. Diese Option verbessert die Darstellung von JSON Daten (pretty formatting).

Beispiel 1

Skalare Werte konvertieren:

```
$vc:=JSON Stringify("Eureka!")  // "Eureka!"
$vel:=JSON Stringify(120)  // "120"

$vh:=JSON Stringify(?20:00:00?)  // "72000" Sekunden seit Mitternacht
SET DATABASE PARAMETER(Times inside objects;Times in milliseconds)
$vhms:=JSON Stringify(?20:00:00?)  // "72000000" Millisekunden seit Mitternacht

$vd:=JSON Stringify(!28/08/2013!)  // "2013-08-27T22:00:00.000Z" (Paris Zeitzone)
SET DATABASE PARAMETER(Dates inside objects;String type without time zone)
$vdd:=JSON Stringify(!28/08/2013!)  // "2013-08-28T00:00:00.000Z"
```

Beispiel 2

Eine einfache Zeichenkette konvertieren:

```
$s:=JSON Stringify("{\"name\":\"john\"}")
 // $s="{\\"name\\":\\"john\\"}"
$p:=JSON Parse($s)
 // $p={"name":"john"}
```

Beispiel 3

Ein 4D Objekt mit und ohne den Parameter * serialisieren:

```
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}
```

Der Vorteil dieser Formatierung wird deutlich, wenn JSON in einem Web Bereich angezeigt wird:

- Standard Formatierung ohne Option *:

```
{"Name":"Monroe","firstname":"Alan","children":{"firstname":"John","age":12}}
```

- Formatierung mit Option * (pretty formatting):

```
{
        "Name": "Monroe",
        "firstname": "Alan",
        "children": {
                "firstname": "John",
                "age": 12
        }
}
```

Beispiel 4

Beispiel für Zeiger auf eine Variable:

```
C_OBJECT($MyTestVar)
C_TEXT($name ;$jsonstring )
OB SET($MyTestVar;"name";->$name)  // Objekt Definition
 // $MyTestVar= {"name":"->$name"}

$jsonstring :=JSON Stringify($MyTestVar)
 // $jsonstring ="{"name":""}"
 //...

$name:="Smith"
$jsonstring :=JSON Stringify($MyTestVar)
 //$jsonstring = "{"name" : "Smith"}"
```

Beispiel 5

Ein 4D Objekt serialisieren:

```
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"}"
```

Beispiel 6

Serialisierung eines 4D Objekts mit einem Datumswert (Paris Zeitzone). Der Ergebnisstring richtet sich nach der aktuellen Datumseinstellung der Datenbank.

```
C_TEXT($varjsonTextserialized)
C_OBJECT($Contact)
OB SET($Contact;"name";"Smith";"birthday";!22/10/1975!)
$varjsonTextserialized:=JSON Stringify($Contact)
```

- Ist die Option "Verwende Datumstyp statt ISO Datumsformat in Objekten" nicht markiert:

```
"name":"Smith",
"birthday":"1975-10-21T22:00:00.000Z"
```

- Ist die Option "Verwende Datumstyp statt ISO Datumsformat in Objekten" markiert:

```
"name":"Smith",
"birthday":"1975-10-22"
```

Hinweis: Weitere Informationen dazu finden Sie auf der **Seite Kompatibilität**.

Beispiel 7

Eine Collection konvertieren:

```
C_COLLECTION($myCol)
C_TEXT($myTxtCol)
$myCol:=New collection(33;"mike";!28/08/2017!;False)
$myTxtCol:=JSON Stringify($myCol)"
```

- Ist die Option "Verwende Datumstyp statt ISO Datumsformat in Objekten" nicht markiert:

```
$myTxtCol="[33,"mike","2017-08-27T22:00:00.000Z",false]"
```

- Ist die Option "Verwende Datumstyp statt ISO Datumsformat in Objekten" markiert:

```
$myTxtCol="[33,"mike","2017-08-28",false]"
```

Hinweis: Weitere Informationen dazu finden Sie auf der **Seite Kompatibilität**.

JSON Stringify array

JSON Stringify array (Array {; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Array	Array Text, Array Zahl, Array Boolean, Array Zeiger, Array Objekt	➡	Array zum Serialisieren des Inhalts
*	Operator	➡	Formatierung aktivieren
Funktionsergebnis	Text	↩	String mit dem serialisierten JSON Array

Beschreibung

Die Funktion **JSON Stringify array** konvertiert das 4D Array *Array* in ein serialisiertes JSON Array. Diese Funktion führt die entgegengesetzte Aktion des Befehls **JSON PARSE ARRAY** aus.

In *Array* übergeben Sie ein 4D Array mit den Daten zum Serialisieren. Es kann vom Typ Text, Zahl, Boolean, Zeiger oder Objekt sein.

Hinweis: Übergeben Sie eine skalare Variable oder Feld in *Array*, gibt die Funktion einen String mit dem Parameterwert zwischen den Klammern "[ ]" zurück.

Sie können den optionalen Parameter * übergeben, um im resultierenden String Formatierungszeichen einzuschließen. Das verbessert die Darstellung von JSON Daten (pretty formatting) bei der Anzeige in einer Web Seite.

Beispiel 1

Ein Array Text konvertieren:

```
C_TEXT($jsonString)
ARRAY TEXT($ArrayFirstname;2)
$ArrayFirstname{1}:="John"
$ArrayFirstname{2}:="Jim"
$jsonString :=JSON Stringify array($ArrayFirstname)

 // $jsonString = "["John","Jim"]"
```

Beispiel 2

Ein Array Text mit Zahlen konvertieren:

```
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"]"
```

Beispiel 3

Ein Array Objekt konvertieren:

```
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}]"

 // Wollen Sie das Ergebnis als Text anzeigen, übergeben Sie den optionalen Parameter *:
$JsonStringPretty :=JSON Stringify array($myArray;*)
```

```
[
  {
    "name": "John",
    "age": 35
  },
  {
    "name": "Jim",
    "age": 40
  }
]
```

Beispiel 4

Eine 4D Auswahl in ein Array Objekt konvertieren:

```
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)
 // Ohne Kopieren sind die Werte leere Strings
   APPEND TO ARRAY($arraySel;$ref_value)
 // Jedes Element enthält die gewählten Werte, zum Beispiel:
 // $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 (TabelleName ; jsonArray)

Parameter	Typ		Beschreibung
TabelleName	Tabelle	➡	4D Tabelle, in die Elemente kopiert werden
jsonArray	Text	➡	Array mit Objekten in JSON

Beschreibung

Der Befehl **JSON TO SELECTION** kopiert den Inhalt des Array mit JSON Objekten *jsonArray* in eine Datensatzauswahl von *TabelleName*.

Der Parameter *jsonArray* ist ein Text Array mit Objekten, formatiert in JSON und mit einem oder mehreren Elementen. Die Syntax lautet wie folgt:

"[{"attribute1":"value1","attribute2":"value2",...},...,{"attribute1":"valueN","attribute2":"valueN",...}]"

Gibt es für *TabelleName* zum Zeitpunkt des Aufrufs eine Auswahl, werden die Elemente des JSON Array in die Datensätze kopiert, und zwar in der Reihenfolge des Array und der Datensätze. Übersteigt die Anzahl der Elemente im JSON Array die Anzahl der Datensätze in der aktuellen Auswahl, werden neue Datensätze erstellt. Alle Datensätze, egal ob neu oder schon vorhanden, werden automatisch gesichert.

Hinweis: Dieser Befehl unterstützt Felder vom Typ Objekt: JSON Daten werden automatisch konvertiert.

Warnung: Verwenden Sie diesen Befehl mit Bedacht, denn **JSON TO SELECTION** ersetzt alle Informationen in den vorhandenen Datensätzen.

Ist ein Datensatz während der Ausführung des Befehls gesperrt, wird er nicht geändert. Alle gesperrten Datensätze werden in die Systemmenge **LockedSet** gelegt. Nach Ausführen von **JSON TO SELECTION** können Sie testen, ob die Menge **LockedSet** Datensätze enthält, die gesperrt waren.
Weitere Informationen zu dieser Menge finden Sie im Abschnitt **Die Systemmenge LockedSet**.

Beispiel

Mit dem Befehl **JSON TO SELECTION** Datensätze in der Tabelle [Company] hinzufügen:

```
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)
 // Sie erstellen 4 Datensätze in der Tabelle [Company], füllen die ID, die Felder Company Name und City
```

JSON Validate

JSON Validate (vJson ; vSchema) -> Funktionsergebnis

Parameter	Typ		Beschreibung
vJson	Objekt	➡	JSON Objekt zum Bestätigen
vSchema	Objekt	➡	Verwendetes JSON Schema zum Bestätigen von JSON Objekten
Funktionsergebnis	Objekt	↩	Status der Bestätigung und Fehler (falls vorhanden)

Beschreibung

Die Funktion **JSON Validate** prüft die Übereinstimmung des JSON Inhalts *vJson* mit den im JSON Schema *vSchema* definierten Regeln. Bei ungültigem JSON gibt die Funktion eine ausführliche Beschreibung der Fehler zurück.

In *vJson* übergeben Sie ein JSON Objekt mit dem JSON Inhalt, der bestätigt werden soll.

Hinweis: Bestätigen eines JSON String bedeutet die Überprüfung, ob er die in einem JSON Schema definierten Regeln befolgt. Das unterscheidet sich von der Überprüfung, ob das JSON "well-formed" ist. Diese wird mit dem Befehl **JSON Parse** ausgeführt.

In *vSchema* übergeben Sie das JSON Schema, das zur Bestätigung verwendet werden soll. Weitere Informationen zum Erstellen eines JSON Schema erhalten Sie unter der Web Site json-schema.org.

Hinweis: 4D verwendet zum Bestätigen eines JSON Objekts die im Dokument JSON Schema Validation beschriebene Norm (ist noch im Entwurfsstatus und kann noch erweitert werden). 4D aktuelle Implementation basiert auf Version 4 des Entwurfsstatus.

Ist das JSON Schema nicht gültig, gibt 4D ein Objekt **Null** zurück und generiert einen Fehler, der sich über eine Fehlerverwaltungsmethode abfangen lässt.

Die Funktion **JSON Validate** gibt ein Objekt mit dem Status der Bestätigung zurück. Es kann folgende Eigenschaften enthalten:

Name der Eigenschaft	Typ	Beschreibung
success	Boolean	Wahr, wenn *vJson* bestätigt wird, sonst falsch. Bei falsch wird auch die Eigenschaft *errors* zurückgegeben
errors	Objekt collection	Liste der Fehlerobjekte, wenn *vJson* nicht bestätigt wird (siehe unten)

Jedes Fehlerobjekt der Collection *errors* enthält folgende Eigenschaften:

Name der Eigenschaft	Typ	Beschreibung
code	Nummer	Fehlercode
jsonPath	String	JSON Pfad, der sich in *vJson* nicht bestätigen lässt
line	Nummer	Zeilennummer des Fehlers in der JSON Datei. Diese Eigenschaft wird gefüllt, wenn das JSON durch **JSON Parse** mit dem Parameter * analysiert wurde. Andernfalls wird die Eigenschaft weggelassen.
message	String	Fehlermeldung
offset	Nummer	Zeilenversatz des Fehlers in der JSON Datei. Diese Eigenschaft wird gefüllt, wenn das JSON durch **JSON Parse** mit dem Parameter * analysiert wurde. Andernfalls wird die Eigenschaft weggelassen.
schemaPaths	String	JSON Pfad im Schema, der Bestätigungsfehler verursacht

Fehlerverwaltung

Es können folgende Fehlermeldungen erscheinen:

Code	JSON Keyword	Meldung
2	multipleOf	Fehler beim Validieren des 'multipleOf' Schlüssels.
3	maximum	Der angegebene Wert sollte nicht größer sein als im Schema angegeben ("{s1}").
4	exclusiveMaximum	Der angegebene Wert sollte kleiner sein als im Schema angegeben ("{s1}").
5	minimum	Der angegebene Wert sollte nicht kleiner sein als im Schema angegeben ("{s1}").
6	exclusiveMinimum	Der angegebene Wert sollte größer sein als im Schema angegeben ("{s1}").
7	maxLength	Der String ist länger als im Schema angegeben.
8	minLength	Der String ist kürzer als im Schema angegeben.
9	pattern	Der String "{s1}" stimmt nicht mit dem Muster im Schema überein:{s2}.
10	additionalItems	Fehler beim Validieren eines Arrays. JSON enthält mehr Elemente als im Schema angegeben.
11	maxItems	Das Array enthält mehr Elemente als im Schema angegeben.
12	minItems	Das Array enthält weniger Elemente als im Schema angegeben.
13	uniqueItems	Fehler beim Validieren eines Arrays. Elemente sind nicht einmalig. Eine weitere Instanz von "{s1}" befindet sich bereits im Array.
14	maxProperties	Die Anzahl der Eigenschaften ist größer als im Schema angegeben.
15	minProperties	Die Anzahl der Eigenschaften ist kleiner als im Schema angegeben.
16	required	Die erforderliche Eigenschaft "{s1}" fehlt.
17	additionalProperties	Das Schema erlaubt keine zusätzlichen Eigenschaften. Die Eigenschaft(en) {s1} sollte(n) entfernt werden.
18	dependencies	Die Eigenschaft "{s1}" benötigt die Eigenschaft "{s2}".
19	enum	Fehler beim Validieren des 'enum' Schlüssels. "{S1}" passt zu keinem enum-Element im Schema.
20	type	Falscher Typ. Erwarteter Typ ist: {s1}
21	oneOfmany	Fehler beim Validieren des 'oneOf' Schlüssels. Json passt zu mehr als einem Wert.
22	oneOfnone	Fehler beim Validieren des 'oneOf' Schlüssels. Json passt zu keinem Wert.
23	not	Fehler beim Validieren des 'not' Schlüssels. Das Json ist gültig für den Wert von 'not'.
24	format	Der String passt nicht zu ("{s1}")

Beispiel

Ein JSON Objekt mit Schema bestätigen und - falls vorhanden - die Liste der Bestätigungsfehler erhalten. Die Fehlerzeilen und Meldungen in einer Textvariablen speichern:

```
C_OBJECT($oResult)
$oResult:=JSON Validate(JSON Parse(myJson;*);mySchema)
If($oResult.success)  //Bestätigung erfolgreich
   ...
Else  //Bestätigung fehlgeschlagen
   C_LONGINT($vLNbErr)
   C_TEXT($vTerrLine)
   $vLNbErr:=$oResult.errors.length  ///Fehlernummer erhalten
   ALERT(String($vLNbErr)+" Bestätigungsfehler gefunden.")
   For($i;0;$vLNbErr)
      $vTerrLine:=$vTerrLine+$oResult.errors[$i].message+" "+String($oResult.errors[$i].line)+Carriage return
   End for
End if
```

Hinweis: Für dieses Beispiel muss Objektnotation aktiviert sein (siehe **Seite Kompatibilität**).

Selection to JSON

Selection to JSON (Tabellenname {; Feld1... FeldN}{; Feld1... FeldN2 ; ... ; Feld1... FeldNN}{; Vorlage}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Zeiger auf Tabelle
Feld1... FeldN	Feld	➡	Feld(er) deren Inhalt serialisiert werden soll
Vorlage	Objekt	➡	Objekt zum Auswählen von Bezeichnungen und Feldern
Funktionsergebnis	Text	⮌	String mit dem serialisierten JSON Array

Beschreibung

Die Funktion **Selection to JSON** gibt einen String mit einem JSON Array mit sovielen Elementen zurück, wie Datensätze in der aktuellen Auswahl von *Tabellenname* enthalten sind. Jedes Element des Array ist ein JSON Objekt mit den Bezeichnungen und Werten der Felder in der Auswahl.

Übergeben Sie nur den Parameter *Tabellenname*, fügt die Funktion im JSON Array die Werte aller Felder der Tabelle ein, die sich in JSON ausdrücken lassen. Felder vom Typ BLOB und Bild werden ignoriert.

Wollen Sie nicht alle Felder von *Tabellenname* einfügen, können Sie entweder den Parameter *Feld1...FeldN* oder den Parameter *Vorlage* verwenden:

- *Feld1...FeldN*: Hier übergeben Sie ein oder mehrere Felder. Nur die Werte der definierten Felder werden in das JSON Array eingefügt.
- *Vorlage*: Hier übergeben Sie ein 4D Objekt mit einem oder mehreren Name/Wert Paaren. Der Name kann jeder gültige Attributname sein, der Wert enthält einen Zeiger auf ein Feld, das Sie einfügen möchten (siehe Beispiel 3)

Diese Funktion unterstützt Felder vom Typ Objekt: Daten dieser Felder werden automatisch in das JSON Format konvertiert (Werte von Bildattrituten werden als "[object Picture]" Strings konvertiert). So wird die nachfolgende 4D Anweisung interpretiert als "erzeuge JSON von allen Werten in *objectField* in der aktuellen Auswahl der Tabelle":

```
Selection to JSON([aTable];objectField)
```

Hinweis: Nach Aufrufen von **Selection to JSON** bleibt die aktuelle Auswahl gleich, der aktuelle Datensatz wird jedoch nicht länger geladen und kann sich geändert haben (der zuletzt geladene Datensatz der aktuellen Auswahl ist dann der aktuelle Datensatz). Um die Werte der Datenfelder im aktuellen Datensatz zu verwenden, rufen Sie nach **Selection to JSON** den Befehl **LOAD RECORD** in Kombination mit **GOTO SELECTED RECORD** (falls erforderlich) auf.

Beispiel 1

Sie wollen ein JSON String für folgende Auswahl erstellen:

newbase - Mitglieder: 6 von 6

Name :	Vorname :	Adresse :	PLZ :	Stadt :
Maier	Alexander	Ludwigshafen 17	41017	Düsseldorf
Schmidt	Verena	Schloßallee 1	10099	Berlin
Siemens	Fabian	Siemensallee 1	81373	München
Osterman	Heinz	Kaufmannstraße 8	20122	Hamburg
Schneider	Leo	Parzivalplatz 9	70233	Stuttgart
Berger	Walter	Sonnenstraße 100	80333	München

1) Sie wollen die Werte aller Felder der Tabelle [Mitglieder] nutzen:

```
$jsonString :=Selection to JSON([Mitglieder])
 // $jsonString =[{"Name":"Maier","Vorname":"Alexander","Adresse":
 //"Ludwigshafen 1","PLZ":"41017","Stadt":"Düsseldorf"},{"Name":
 //"Schmidt","Vorname":"Verena","Adresse":"Schlossallee 1","PLZ":
 //"10099","Stadt":"Berlin"},{"Name":"Siemens","Vorname"
 //:"Fabian","Adresse":"Siemensallee 1","PLZ":"81373","Stadt":"München"},...]
```

2) Sie wollen die Auswahl verkleinern und nur zwei Felder in den JSON String einfügen. Dafür verwenden Sie die Syntax Feld1...FeldN:

```
QUERY([Mitglieder];[Mitglieder]Name="A@")
$jsonString :=Selection to JSON([Mitglieder];[Mitglieder]Name;[Mitglieder]Stadt)
 // $jsonString = [{"Name":"Schmidt","Stadt":"Berlin"},{"Name":"Siemens","Stadt":"München"}]
```

3) Sie wollen nur ein Feld in den JSON String einfügen und eine andere Bezeichnung verwenden. Sie können die Syntax *template* verwenden:

```
C_OBJECT($template)
OB SET($template;"Name";->[Mitglieder]Name)  // ein einzelnes Feld
ALL RECORDS([Mitglieder])
$jsonString :=Selection to JSON([Mitglieder];$template)
```

```
// $jsonString = [{"Name":"Maier"},{"Name":"Schmidt"},{"Name":"Siemens"},{"Name":"Osterman"},{"Name":"Schneider"},
{"Name":"Berger"}]
```

Beispiel 2

Über die Syntax *template* können Sie Felder von unterschiedlichen Tabellen exportieren:

```
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)  //Eigene Bezeichnung, sonst Konflikt mit dem Feld [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)
```

Kommunikation

- 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 (NumArray ; NameArray)

Parameter	Typ		Beschreibung
NumArray	Array Lange Ganzzahl	⇐	Array mit Portnummer(n)
NameArray	Array String	⇐	Array mit Portname(n)

Beschreibung

Der Befehl **GET SERIAL PORT MAPPING** gibt die beiden Arrays *NumArray* und *NameArray* mit den seriellen Portnummern und Portnamen des aktuellen Rechners zurück.

Dieser Befehl ist hilfreich auf Mac OS X, wo das Betriebssystem bei Verwendung eines seriellen USB Adapters die Portnummer dynamisch zuweist. Sie können jeden erweiterten seriellen Port über seinen Namen (statisch) zuweisen, unabhängig von seiner aktuellen Kennummer.

Hinweis: Dieser Befehl gibt für den Standardport keine signifikanten Werte zurück. Wollen Sie einen Standardport zuweisen, müssen Sie seinen Wert (0 oder 1) direkt über den Befehl **SET CHANNEL** übergeben (frühere Arbeitsweise von 4D).

Beispiel

Diese Projektmethode lässt sich zur Zuweisung desselben seriellen Ports (ohne Protokoll) verwenden, unabhängig von der zugewiesenen Nummer:

```
ARRAY TEXT($arrPortNames;0)
ARRAY LONGINT($arrPortNums;0)
C_LONGINT($vPortNum;$vFinalPortNum))

  `Aktuelle Nummern der seriellen Ports herausfinden
GET SERIAL PORT MAPPING($arrPortNums;$arrPortNames)
$vPortNum:=Find in array($arrPortNames;vPortName)
  `vPortName enthält den Namen des zu verwendenden Ports; es kann von einem Dialogfenster, einem Wert in einem Feld, etc. stammen
If(arrPortNums($vPortNum=0)
   $vFinalPortNum:=0  `Sonderfall auf Mac OS X
Else
   $vFinalPortNum:=arrPortNums{$vPortNum}+100
End if
SET CHANNEL($vFinalPortNum;params)  `params enthält die Kommunikationsparameter
  `Führen Sie die gewünschten Operationen aus.
SET CHANNEL(11)  `Port schließen
```

RECEIVE BUFFER

RECEIVE BUFFER (Puffername)

Parameter	Typ		Beschreibung
Puffername	Textvariable	➡	Variable zum Empfangen von Daten

Beschreibung

Der Befehl **RECEIVE BUFFER** liest die zuvor mit **SET CHANNEL** geöffnete serielle Schnittstelle. Sie hat einen Puffer, der mit Zeichen gefüllt wird, bis ein Befehl den Puffer ausliest. **RECEIVE BUFFER** liest nur die bereits empfangenen Daten aus dem Puffer der seriellen Schnittstelle, setzt sie in die Variable *Puffername* und leert anschließend den seriellen Puffer. Gibt es keine Zeichen im Puffer, enthält *Puffername* nichts.

Unter Windows
Die Größe des seriellen Puffers unter Windows ist auf 10 kb begrenzt, d.h. er kann überfüllt sein. Wenn er voll ist und weiter neue Daten eingehen, ersetzen die neuen Zeichen die ältesten. Diese gehen verloren. Deshalb muss der Puffer schnell ausgelesen werden, wenn weiter neue Zeichen eingehen.

Auf Mac OS
Auf Mac OS X ist die Kapazität theoretisch unbegrenzt (abhängig vom verfügbaren Speicher). Ist er voll und gehen weiter neue Daten ein, ersetzen die neuen Zeichen die ältesten. Diese gehen verloren. Deshalb muss der Puffer schnell ausgelesen werden, wenn weiter neue Zeichen eingehen.

RECEIVE BUFFER nimmt alles, was im Puffer enthalten ist und gibt es sofort zurück. Im Gegensatz dazu wartet **RECEIVE PACKET**, bis ein bestimmtes Zeichen gefunden wird, oder eine vorgegebene Anzahl Zeichen im Puffer enthalten ist.

Der Benutzer kann während der Ausführung von **RECEIVE BUFFER** den Empfang unterbrechen, und zwar unter Windows durch die Tastenkombination **Strg-**, **Alt-** und **Umschalttaste**, auf Macintosh **Wahl-**, **Befehls-** und **Umschalttaste**. Die Unterbrechung generiert den Fehler -9994. Sie können ihn mit einer Methode **ON ERR CALL** abfangen.

Beispiel

Die Projektmethode ***LISTEN TO SERIAL PORT*** erhält über **RECEIVE BUFFER** Text aus der seriellen Schnittstelle und sammelt diesen in einer Interprozessvariablen:

```
  ` LISTEN TO SERIAL PORT
  ` Seriellen Port öffnen
SET CHANNEL(201;Speed 9600+Data bits 8+Stop bits one+Parity none)
<>IP_Listen_Serial_Port:=True
While(<>IP_Listen_Serial_Port)
   RECEIVE BUFFER($vtBuffer)
   If((Length($vtBuffer)+Length(<>vtBuffer))>MAXTEXTLEN)
      <>vtBuffer:=""
   End if
   <>vtBuffer:=<>vtBuffer+$Buffer
End while
```

Ab hier kann jeder andere Prozess die Interprozessvariable ◇*vtBuffer* lesen, um mit den Daten aus der seriellen Schnittstelle zu arbeiten.

Um die Abfrage für die serielle Schnittstelle zu beenden, schreiben Sie:

```
  ` Stoppe listening für seriellen Port
◇IP_Listen_Serial_Port:=False
```

Hinweis: Sie sollten den Zugriff auf die Interprozessvariable ◇*vtBuffer* mit einer lokalen Semaphore schützen, um Konflikte mit anderen Prozessen zu vermeiden. Weitere Informationen dazu finden Sie unter der Funktion **Semaphore**.

RECEIVE PACKET

RECEIVE PACKET ({DokRef ;} EmpfangeVar ; StopChar | NumBytes)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenznummer des Dokuments oder aktueller Kanal (serielle Schnittstelle oder Dokument)
EmpfangeVar	Textvariable, BLOB Variable	⬅	Variable zum Empfangen des Textes
StopChar | NumBytes	String, Lange Ganzzahl	➡	Zeichen, bis zu dem Daten empfangen werden, oder Anzahl zu empfangender Bytes

Beschreibung

Der Befehl **RECEIVE PACKET** liest die Daten von einer seriellen Schnittstelle oder von einem Dokument.

Ist *DokRef* definiert, findet dieser Befehl Daten aus einem Dokument, das mit **Open document**, **Create document** oder **Append document** geöffnet wurde. Ist *DokRef* nicht definiert, findet dieser Befehl Daten aus einer seriellen Schnittstelle oder einem Dokument, das mit **SET CHANNEL** geöffnet wurde.

Die eingelesenen Zeichen werden, unabhängig von der Quelle, in *EmpfangeVar* zurückgegeben. Dies muss eine Variable von Typ Text, String oder BLOB sein. Wurden die Zeichen über den Befehl **SEND PACKET** gesendet, muss der Typ zu dem des gesendeten Pakets passen.

Hinweise:

- Ist das empfangene Paket vom Typ BLOB, berücksichtigt es keine Zeichen, die über **USE CHARACTER SET** angegeben wurden. Das BLOB wird ohne Änderung gesendet.
- Ist das empfangene Paket vom Typ Text, unterstützt **RECEIVE PACKET** BOMs (Byte Order Marks). Ist in diesem Fall der aktuelle Zeichensatz vom Typ Unicode (UTF-8, UTF-16 oder UTF-32), versucht 4D unter den ersten empfangenen Bytes ein BOM zu identifizieren. Wurde es gefunden, wird es aus dem Ergebnis gefiltert und 4D verwendet den hier definierten Zeichensatz statt des aktuellen.

Um eine bestimmte Anzahl Zeichen einzulesen, übergeben Sie diese Anzahl in *NumBytes*. Ist die Variable *EmpfangeVar* vom Typ Text, können Sie in einem Aufruf theoretisch bis zu 2 GB Text einlesen.

Um Zeichen bis zu einem bestimmten String einzulesen (bestehend aus ein oder mehreren Zeichen), übergeben Sie diesen String in *StopChar*. Er wird nicht in *EmpfangeVar* zurückgegeben.

Wird der in *StopChar* angegebene String nicht gefunden, gilt folgendes:

- Beim Lesen eines Dokuments stoppt **RECEIVE PACKET** am Ende des Dokuments.
- Beim Auslesen von Daten aus einer seriellen Schnittstelle wartet **RECEIVE PACKET** unendlich, bis ein Timeout eintritt (siehe **SET TIMEOUT**) oder bis der Benutzer den Empfang abbricht.

Beim Ausführen von **RECEIVE PACKET** kann der Benutzer dem Empfang durch gleichzeitiges Drücken der Strg-, Alt- und Umschalttaste bzw. auf Macintosh der Wahl-, Befehls- und Umschalttaste unterbrechen. Die Unterbrechung generiert den Fehler -9994. Sie können ihn mit einer Fehlerverwaltungsmethode, installiert mit **ON ERR CALL**, abfangen. Eine Unterbrechung des Empfangs ist in der Regel nur bei der Kommunikation über eine serielle Schnittstelle erforderlich.

Beim Auslesen eines Dokuments beginnt das erste **RECEIVE PACKET** am Anfang des Dokuments auszulesen. Alle darauffolgenden Datenpakete beginnen mit dem Zeichen, das auf das zuletzt gelesene Byte folgt.

Hinweis: Sie können diesen Befehl verwenden für ein Dokument, das mit **SET CHANNEL** geöffnet wurde. Für ein Dokument, das mit **Open document**, **Create document** oder **Append document** geöffnet wurde, können Sie mit den Befehlen **Get document position** und **SET DOCUMENT POSITION** die Position festlegen oder ändern, wo das nächste Auslesen (**SEND PACKET**) oder Einlesen (**RECEIVE PACKET**) stattfinden soll.

Versuchen Sie, über das Ende einer Datei hinaus zu lesen, gibt **RECEIVE PACKET** die gelesenen Daten bis zu diesem Punkt zurück und setzt die Systemvariable auf 1. Das nächste **RECEIVE PACKET** gibt eine leere Zeichenkette zurück und setzt die Systemvariable auf 0.

Beispiel 1

Folgendes Beispiel liest 20 Zeichen aus einer seriellen Schnittstelle in die Variable *getTwenty*:

```
RECEIVE PACKET(getTwenty;20)
```

Beispiel 2

Folgendes Beispiel liest Daten aus dem in der Variablen *myDoc* zugewiesenen Dokument in die Variable *vData*. Ausgelesen wird bis zur Zeilenschaltung:

```
RECEIVE PACKET(myDoc;vData;Char(Carriage return))
```

Beispiel 3

Folgendes Beispiel liest Daten aus dem in der Variablen *myDoc* zugewiesenen Dokument in die Variable *vData*. Ausgelesen wird bis zum HTML Tag </TD> (Ende der Tabellenzelle):

```
RECEIVE PACKET(myDoc;vData;"</TD>")
```

Beispiel 4

Folgendes Beispiel liest Daten von einem Dokument in Datenfelder. Die Daten werden in Datenfelder von fester Länge gespeichert. Die darin enthaltene Methode ruft eine Unterroutine auf, die schleifende Leerzeichen entfernt (Leerzeichen am Ende von Zeichenketten):

```
$vhDocRef :=Open document("";"TEXT")  ` Öffne ein TEXT Dokument
If(OK=1)  `Wurde das Dokument geöffnet
   Repeat  ` Durchlaufe bis keine Daten mehr da sind
      RECEIVE PACKET($vhDocRef;$Var1;15)  ` Lies 15 Zeichen
      RECEIVE PACKET($vhDocRef;$Var2;15)  `Führe dasselbe für das 2. Feld aus
      If(($Var1#"")|($Var2#""))  `Ist mindestens eins der Felder nicht leer
         CREATE RECORD([People])  ` Erstelle neuen Datensatz
         [People]First :=Strip($Var1)  ` Sichere den ersten Namen
         [People]Last :=Strip($Var2)  ` Sichere den letzten Namen
         SAVE RECORD([People])  ` Sichere den Datensatz
      End if
   Until(OK=0)
   CLOSE DOCUMENT($vhDocRef)  ` Schließe das Dokument
End if
```

Nachfolgende Methode ***Strip*** entfernt die Leerzeichen am Ende der Daten:

```
For($i;Length($1);1;-1)  ` Durchlaufe vom Ende des Zeichensatzes bis zum Start
   If($1[[$i]]#" ")  ` Ist kein Leerzeichen vorhanden
      $i :=-$i  ` Beende die Schleife
   End if
End for
$0:=Delete string($1;-$i;Length($1))  ` Lösche die Leerzeichen
```

Systemvariablen und Mengen

Nach Aufrufen von **RECEIVE PACKET** hat die Systemvariable OK den Wert 1, wenn das Datenpaket ohne Fehler empfangen wurde. Wurde die Kommunikation unterbrochen oder bei einem Übertragungsfehler hat OK den Wert 0.

RECEIVE RECORD

RECEIVE RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle für den zu empfangenden Datensatz Ohne Angabe Standardtabelle

Beschreibung

Der Befehl **RECEIVE RECORD** empfängt in *Tabellenname* einen Datensatz aus einer seriellen Schnittstelle oder von einem durch den Befehl **SET CHANNEL** geöffneten Dokument. Der Datensatz muss mit **SEND RECORD** gesendet worden sein. Führen Sie **RECEIVE RECORD** aus, wird automatisch eine neuer Datensatz für *Tabellenname* angelegt. Wurde der Datensatz korrekt empfangen, müssen Sie ihn mit dem Befehl **SAVE RECORD** sichern.

Der komplette Datensatz wird empfangen, dazu gehören auch alle im oder mit dem Datensatz gespeicherten Bilder oder BLOBs.

Wichtig: Werden Datensätze mit den Befehlen **SEND RECORD** und **RECEIVE RECORD** gesendet bzw. empfangen, müssen die Struktur der Quell- bzw. der Zieltabelle zueinander kompatibel sein. Ist das nicht der Fall, konvertiert 4D die Werte gemäß den Tabellendefinitionen beim Ausführen von **RECEIVE RECORD**.

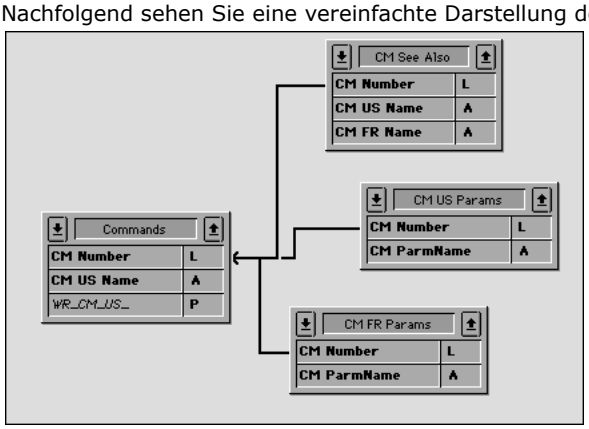
Anmerkungen:

- Empfangen Sie mit diesem Befehl einen Datensatz aus einem Dokument, muss es zuvor mit dem Befehl **SET CHANNEL** geöffnet werden. Sie können **RECEIVE RECORD** nicht für Dokumente verwenden, die mit **Open document**, **Create document** oder **Append document** geöffnet wurden.
- Der Benutzer kann während der Ausführung von **RECEIVE RECORD** den Empfang unterbrechen, und zwar unter Windows durch die Tastenkombination **Strg-**, **Alt-** und **Umschalttaste**, auf Macintosh **Wahl-**, **Befehls-** und **Umschalttaste**. Die Unterbrechung generiert den Fehler -9994. Sie können ihn mit einer Methode **ON ERR CALL** abfangen.

Beispiel

Die Kombination von **SEND VARIABLE**, **SEND RECORD**, **RECEIVE VARIABLE** und **RECEIVE RECORD** ist ideal zum Archivieren von Daten oder für den Austausch von Daten zwischen identischen Datenbanken im Einzelplatzbetrieb an verschiedenen Orten. Sie können Daten zwar auch mit den Befehlen **EXPORT TEXT** und **IMPORT TEXT** austauschen. Sobald sie jedoch Bilder und/oder verknüpfte Tabellen enthalten, sind die Befehle **SEND RECORD** und **RECEIVE RECORD** besser geeignet.

Nehmen wir z.B. eine Dokumentation, die auf 4D und 4D Write basiert. Da mehrere Verfasser an verschiedenen Orten auf der Welt daran arbeiten, benötigen wir einen einfachen Weg für den Datenaustausch zwischen den verschiedenen Datenbanken. Nachfolgend sehen Sie eine vereinfachte Darstellung der Datenbankstruktur:



Die Tabelle *[Commands]* enthält die Beschreibung jedes Befehls oder Abschnitts. Die Tabellen *[CM US Params]* und *[CM FR Params]* enthalten jeweils die Liste der Parameter für jeden Befehl in Englisch und Französisch. Die Tabelle *[CM See Also]* enthält die Befehle, die für jeden Befehl als Referenz aufgeführt werden. Für den Datenaustausch werden die Datensätze aus der Tabelle *[Commands]* und die damit verknüpften Datensätze gesendet. Dazu verwenden wir die Befehle **SEND RECORD** und **RECEIVE RECORD**. Zusätzlich benützen wir **SEND VARIABLE** und **SEND RECORD**, um das Import-/Exportdokument mit Tags zu markieren.

Hier ist die vereinfachte Projektmethode für den Datenexport:

```
  ` Projektmethode CM_EXPORT_SEL
  ` Diese Methode arbeitet mit der aktuellen Auswahl der Tabelle [Commands]

SET CHANNEL(12;"") ` Lässt den Benutzer ein Kanaldokument erstellen und öffnen
If(OK=1)
  ` Markiere Dokument mit Variable, die den Inhalt angibt
  ` Hinweis: Die Prozessvariable BUILD_LANG gibt an, ob US (Englisch) oder FR (Französisch) Daten gesendet werden
   $vsTag:="4DV6COMMAND"+BUILD_LANG
   SEND VARIABLE($vsTag)
  ` Sende Variable, die anzeigt wie viele [Commands] gesendet werden
   $vlNbCmd:=Records in selection([Commands])
   SEND VARIABLE($vlNbCmd)
   FIRST RECORD([Commands])
```

```
    ` Für jeden Befehl
    For($vlCmd;1;$vlNbCmd)
  ` Sende den [Commands] Datensatz
      SEND RECORD([Commands])
  ` Wähle alle verknüpften Datensätze aus
      RELATE MANY([Commands])
      [ ` Sende je nach Sprache eine Variable, welche die Anzahl der folgenden Parameter angibt
      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)
  ` Sende die [CM US Params] Datensätze (falls vorhanden)
      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
  ` Sende eine Variable, die die Anzahl der folgenden Referenzen angibt
      $vlNbSee:=Records in selection([CM See Also])
      SEND VARIABLE($vlNbSee)
  ` Sende die [CM See Also] Datensätze (falls vorhanden)
      For($vlSee;1;$vlNbSee)
        SEND RECORD([CM See Also])
        NEXT RECORD([CM See Also])
      End for
  ` Gehe zum nächsten [Commands] Datensatz und fahre fort mit Export
      NEXT RECORD([Commands])
    End for
    SET CHANNEL(11) `Schließe Dokument
End if
```

Hier ist die vereinfachte Projektmethode für den Datenimport

```
    ` Projektmethode CM_IMPORT_SEL

SET CHANNEL(10;"") ` Lässt den Benutzer ein vorhandenes Dokument öffnen
If(OK=1) ` War ein Dokument offen
    RECEIVE VARIABLE($vsTag) ` Versuche, die erwartete Tag Variable zu erhalten
    If($vsTag="4DV6COMMAND@") ` War es das richtige Tag?
      $CurLang:=Substring($vsTag;Length($vsTag)-1) ` Entnehme Sprache aus dem Tag
      If(($CurLang="US")|($CurLang="FR")) ` War es eine gültige Sprache?
        RECEIVE VARIABLE($vlNbCmd)
          ` Wieviele Befehle gibt es in diesem Dokument?
        If($vlNbCmd>0) ` Ist mindestens einer vorhanden
          For($vlCmd;1;$vlNbCmd) ` Für jeden archivierten [Commands] Datensatz empfange den Datensatz
            RECEIVE RECORD([Commands])
  ` Rufe Unterroutine, die den neuen Datensatz sichert oder seinen Inhalt in bereits vorhandenen Datensatz kopiert
            CM_IMP_CMD($CurLang) ` Empfange Anzahl der Parameter (falls vorhanden)
            RECEIVE VARIABLE($vlNbParm)
            If($vlNbParm>=0)
  ` Rufe Unterroutine, die RECEIVE RECORD aufruft, dann den neuen Datensatz sichert oder in bereits vorhandene Datensätze kopiert
              CM_IMP_PARM($vlNbParm;$CurLang)
            End if
  ` Erhalte Anzahl von “Referenz” (falls vorhanden)
            RECEIVE VARIABLE($vlNbSee)
            If($vlNbSee>0)
  ` Rufe Unterroutine, die RECEIVE RECORD aufruft, dann den neuen Datensatz sichert oder in bereits vorhandene Datensätze kopiert
              CM_IMP_SEEA($vlNbSee;$CurLang)
            End if
          End for
        Else
          ALERT("Anzahl der Befehle in Exportdokument ist ungültig.")
        End if
```

```
        Else
            ALERT("Sprache in Exportdokument ist unbekannt.")
        End if
    Else
        ALERT("Dieses Dokument ist KEIN Exportdokument für Befehle.")
    End if
    SET CHANNEL(11) ` Schließe Dokument
End if
```

Beachten Sie, dass wir beim Datenempfang weder die Systemvariable OK testen, noch versuchen, Fehler abzufangen. Da wir jedoch Variablen im Dokument gespeichert haben, welches das Dokument selbst beschreibt, ist die Fehlerwahrscheinlichkeit sehr gering. Öffnet der Benutzer z.B. ein falsches Dokument, stoppt der erste Test sofort die Operation.

Systemvariablen und Mengen

Die Systemvariable OK hat den Wert 1, wenn der Datensatz empfangen wurde. Wurde die Kommunikation unterbrochen oder trat ein Übertragungsfehler auf, ergibt OK den Wert 0.

RECEIVE VARIABLE

RECEIVE VARIABLE (Variablenname)

Parameter	Typ		Beschreibung
Variablenname	Variable	⇐	Zu empfangende Variable

Beschreibung

Der Befehl **RECEIVE VARIABLE** wartet auf den Empfang der Variablen *Variablenname* von der seriellen Schnittstelle oder von dem durch den Befehl **SET CHANNEL** geöffneten Dokument. **RECEIVE VARIABLE** erhält die Variable im internen 4D-Format. Die Variable muss von einer anderen 4D-Anwendung mit **SEND VARIABLE** oder aus einem Dokument, das von einer 4D Anwendung erzeugt wurde, abgeschickt worden sein.

Existiert die Variable im interpretierten Modus vor Aufrufen von **RECEIVE VARIABLE** noch nicht, wird sie gemäß den empfangenen Daten erstellt, typisiert und zugewiesen. Im kompilierten Modus muss sie vom selben Typ wie die empfangenen Daten sein.

Anmerkungen

1. Senden Sie mit diesem Befehl eine Variable zu einem Dokument, muss es zuvor mit dem Befehl **SET CHANNEL** geöffnet werden. Sie können **RECEIVE VARIABLE** nicht für Dokumente verwenden, die mit **Open document**, **Create document** oder **Append document** geöffnet wurden.
2. Dieser Befehl unterstützt keine Variablen vom Typ Array. Wollen Sie Arrays in bzw. aus einem Dokument oder einer seriellen Schnittstelle ein- bzw. auslesen, verwenden Sie die **BLOB Befehle**.
3. Der Benutzer kann während der Ausführung von **RECEIVE VARIABLE** den Empfang unterbrechen, und zwar unter Windows durch die Tastenkombination **Strg-**, **Alt-** und **Umschalttaste**, auf Macintosh **Wahl-**, **Befehls-** und **Umschalttaste**. Die Unterbrechung generiert den Fehler -9994. Sie können ihn mit einer Methode **ON ERR CALL** abfangen.

Beispiel

Siehe Beispiel zum Befehl **RECEIVE RECORD**.

Systemvariablen und Mengen

Die Systemvariable OK hat den Wert 1, wenn die Variable empfangen wurde. Wurde die Kommunikation unterbrochen oder trat ein Übertragungsfehler auf, ergibt OK den Wert 0.

SEND PACKET

SEND PACKET ({DokRef ;} Datenpaket)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenznummer des Dokuments oder aktueller Kanal (serielle Schnittstelle oder Dokument)
Datenpaket	String, BLOB	➡	Zu sendender String oder BLOB

Beschreibung

Der Befehl **SEND PACKET** sendet ein Datenpaket auf die serielle Schnittstelle oder in das durch den Befehl **SET CHANNEL** geöffnete Dokument. Ist *DokRef* definiert, wird das Datenpaket in das in *DokRef* festgelegte Dokument eingelesen.

Ein *Datenpaket* ist nichts anderes als Daten, normalerweise eine alphanumerische Zeichenkette.
Sie können in *Datenpaket* auch ein BLOB übergeben und so die Einschränkungen zur Codierung von Zeichen, die im Textmodus gesendet werden, umgehen (siehe Beispiel 2).

Hinweis: Übergeben Sie ein BLOB in *Datenpaket*, berücksichtigt der Befehl keinen Zeichensatz, der über den Befehl **USE CHARACTER SET** definiert wurde. Das BLOB wird ohne Änderung gesendet.

Mit diesem Befehl senden Sie Informationen an ein anderes Programm, einen anderen Rechner, einen Drucker mit einer seriellen Schnittstelle, an einen Magnetkartenleser oder an ein anderes Gerät, das eine serielle RS-232-Schnittstelle hat.

Hinweis: Zur Benutzung des parallelen Ports können Sie das Plug-In 4D Pack verwenden. Es enthält hierfür weitere Befehle.

Bevor Sie **SEND PACKET** einsetzen, müssen Sie mit **SET CHANNEL** eine serielle Schnittstelle bzw. ein Dokument öffnen oder ein Dokument mit einem der Dokumentbefehle öffnen.

Beim Einlesen in ein Dokument startet der erste Befehl **SEND PACKET** am Anfang des Dokuments, außer es wurde mit **Append document** geöffnet. Bis zum Schließen des Dokuments wird jedes nachfolgende Datenpaket an das zuvor übertragene angehängt.

Hinweis: Sie können diesen Befehl verwenden für ein Dokument, das mit **SET CHANNEL** geöffnet wurde. Für ein Dokument, das mit **Open document**, **Create document** oder **Append document** geöffnet wurde, können Sie mit den Befehlen **Get document position** und **SET DOCUMENT POSITION** die Position festlegen oder ändern, wo das nächste Auslesen (**SEND PACKET**) oder Einlesen (**RECEIVE PACKET**) stattfinden soll.

Beispiel 1

Folgendes Beispiel liest Daten aus Datenfeldern in ein Dokument ein. Die Felder sind von fester Länge. Ist ein Datenfeld kürzer als die vorgegebene Länge, wird es mit Leerzeichen aufgefüllt. Die Verwendung von Datenfeldern mit fester Länge ist zwar keine effektive Methode zum Speichern von Daten, einige Computersysteme und Anwendungen arbeiten jedoch noch damit:

```
$vhDocRef :=Create document("") ` Erstelle ein Dokument
If(OK=1) ` Wurde das Dokument erstellt?
   For($vlRecord;1;Records in selection([People]))
  ` Durchlaufe einmal für jeden Datensatz
  ` Sende erstes Datenpaket. Erstelle es aus einem Zeichensatz mit 15 Stellen, der das Datenfeld Vorname enthält
      SEND PACKET($vhDocRef;Change string(15*Char(SPACE);[People]First;1))
  ` Sende zweites Datenpaket. Erstelle es aus einem Zeichensatz mit 15 Stellen, der das Datenfeld Nachname enthält
  ` Es könnte zwar im ersten SEND PACKET sein, wird jedoch zur besseren Übersicht getrennt
      SEND PACKET($vhDocRef;Change string(15*Char(SPACE);[People]Last;1))
      NEXT RECORD([People])
   End for
  ` Sende Char(26), das für einige Rechner das Zeichen für Dateiende ist
   SEND PACKET($vhDocRef;Char(SUB ASCII code))
   CLOSE DOCUMENT($vhDocRef) ` Schließe das Dokument
End if
```

Beispiel 2

Dieses Beispiel zeigt das Senden und Empfangen erweiterter Zeichen via BLOB in ein Dokument:

```
C_BLOB($send_blob)
C_BLOB($receive_blob)
TEXT TO BLOB("âzérty";$send_blob;UTF8 text without length)
SET BLOB SIZE($send_blob;16;255)
$send_blob{6}:=0
$send_blob{7}:=1
$send_blob{8}:=2
$send_blob{9}:=3
$send_blob{10}:=0
$vlDocRef:=Create document("blob.test")
If(OK=1)
```

```
    SEND PACKET($vlDocRef;$send_blob)
    CLOSE DOCUMENT($vlDocRef)
End if
$vlDocRef:=Open document(document)
If(OK=1)
    RECEIVE PACKET($vlDocRef;$receive_blob;65536)
    CLOSE DOCUMENT($vlDocRef)
End if
```

SEND RECORD

SEND RECORD {(Tabellenname)}

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, aus der der aktuelle Datensatz gesendet wird Ohne Angabe Standardtabelle

Beschreibung

Der Befehl **SEND RECORD** sendet den aktuellen Datensatz der Tabelle *Tabellenname* zur seriellen Schnittstelle oder in das mit **SET CHANNEL** geöffnete Dokument. Der Datensatz wird in einem spezifischen internen Format gesendet, der sich nur mit **RECEIVE RECORD** lesen lässt. Gibt es keinen aktuellen Datensatz in *Tabellenname*, wird dieser Befehl nicht ausgeführt.

Der komplette Datensatz wird gesendet, dazu gehören auch alle im oder mit dem Datensatz gespeicherten Bilder oder BLOBs.

Wichtig: Werden Datensätze mit den Befehlen **SEND RECORD** und **RECEIVE RECORD** gesendet bzw. empfangen, müssen die Struktur der Quell- bzw. der Zieltabelle zueinander kompatibel sein. Ist das nicht der Fall, konvertiert 4D die Werte gemäß den Definitionen für die Tabelle beim Ausführen von **RECEIVE RECORD**.

Hinweis: Senden Sie mit diesem Befehl einen Datensatz zu einem Dokument, muss es zuvor mit dem Befehl **SET CHANNEL** geöffnet werden. Sie können **SEND RECORD** nicht für Dokumente verwenden, die mit **Open document**, **Create document** oder **Append document** geöffnet wurden.

Hinweis zur Kompatibilität: Ab Version 11 unterstützt dieser Befehl Untertabellen NICHT mehr.

Beispiel

Siehe Beispiel für den Befehl **RECEIVE RECORD**.

SEND VARIABLE

SEND VARIABLE (Variablenname)

Parameter	Typ		Beschreibung
Variablenname	Variable	➡	Zu sendende Variable

Beschreibung

Der Befehl **SEND VARIABLE** sendet die Variable *Variablenname* zur seriellen Schnittstelle oder in das mit **SET CHANNEL** geöffnete Dokument. Die Variable wird in einem spezifischen internen Format gesendet, das nur durch den Befehl **RECEIVE VARIABLE** eingelesen werden kann. **SEND VARIABLE** sendet die komplette Variable inkl. Typ und Wert.

Anmerkung

1. Senden Sie mit diesem Befehl eine Variable zu einem Dokument, muss es zuvor mit dem Befehl **SET CHANNEL** geöffnet werden. Sie können **SEND VARIABLE** nicht für Dokumente verwenden, die mit **Open document**, **Create document** oder **Append document** geöffnet wurden.
2. Dieser Befehl unterstützt keine Variablen vom Typ Array. Wollen Sie Arrays in bzw. aus einem Dokument oder einer seriellen Schnittstelle ein- bzw. auslesen, verwenden Sie die **DISABLE MENU ITEM**.

Beispiel

Siehe Beispiel für den Befehl **RECEIVE RECORD**.

SET CHANNEL

SET CHANNEL (Schnittstelle ; Parameter)

Parameter	Typ		Beschreibung
Schnittstelle	Lange Ganzzahl	➡	Wahl der Schnittstelle
Parameter	Lange Ganzzahl	➡	Parameter der Schnittstelle

SET CHANNEL (Operation ; Dokumentname)

Parameter	Typ		Beschreibung
Operation	Lange Ganzzahl	➡	Operation für das Dokument
Dokumentname	String	➡	Dokumentname

Beschreibung

Der Befehl **SET CHANNEL** öffnet eine serielle Schnittstelle oder ein Dokument. Sie können immer nur eine serielle Schnittstelle oder ein Dokument gleichzeitig öffnen. Um eine geöffnete serielle Schnittstelle zu schließen, übergeben Sie **SET CHANNEL** (11).

Historischer Hinweis: SET CHANNEL war ursprünglich der erste Befehl zum Arbeiten mit seriellen Schnittstellen und Dokumenten auf Festplatten. Seither sind neue Befehle hinzugekommen. Heute verwenden Sie für Dokumente auf der Festplatte vorrangig die Befehle **Open document**, **Create document** und **Append document**. Damit können Sie mit Hilfe von **Create document** oder **RECEIVE PACKET** Zeichen in bzw. aus Dokumenten ein- bzw. auslesen (Diese Befehle arbeiten auch mit **SET CHANNEL**). Für die Befehle **SEND VARIABLE**, **RECEIVE VARIABLE**, **RECEIVE VARIABLE** und **RECEIVE RECORD** können Sie dagegen nur über **SET CHANNEL** auf Dokumente auf der Festplatte zugreifen.

Die Beschreibung von **SET CHANNEL** gliedert sich in zwei Abschnitte:

- Arbeiten mit seriellen Schnittstellen
- Arbeiten mit Dokumenten

Arbeiten mit seriellen Schnittstellen - SET CHANNEL (Schnittstelle;Parameter)

Die erste Syntax des Befehls **SET CHANNEL** öffnet eine serielle Schnittstelle und definiert das Protokoll sowie weitere Informationen der Schnittstelle. Sie können Daten senden mit **SEND PACKET**, **SEND RECORD** oder **SEND VARIABLE**, bzw. Daten empfangen mit **RECEIVE BUFFER**, **RECEIVE PACKET**, **RECEIVE VARIABLE** oder **RECEIVE RECORD**.

Schnittstelle wählt die serielle Schnittstelle und das zu benutzende Protokoll. Sie können bis zu 99 serielle Schnittstellen ansprechen (eine zur gleichen Zeit pro Prozess). Folgende Liste zeigt die möglichen Werte für *Schnittstelle*:

Bereich	Beschreibung
0	COM2 (Windows) ohne Protokoll oder Druckerschnittstelle (Mac)
1	COM1 (Windows) ohne Protokoll oder Modemschnittstelle (Mac)
20	COM2 (Windows) mit Software Protokoll wie XON/XOFF oder Druckerschnittstelle (Mac)
21	COM1 (Windows) mit Software Protokoll wie XON/XOFF oder Modemschnittstelle (Mac)
30	COM2 (Windows) mit Hardware Protokoll wie RTS/CTS oder Druckerschnittstelle (Mac)
31	COM1 (Windows) mit Hardware Protokoll wie RTS/CTS oder Modemschnittstelle (Mac)
101 bis 199	Serielle Kommunikation ohne Protokoll
201 bis 299	Serielle Kommunikation mit Software Protokoll wie XON/XOFF
301 bis 399	Serielle Kommunikation mit Hardware Protokoll wie RTS/CTS

Wichtig: Der in *Schnittstelle* übergebene Wert muss sich auf eine vorhandene COM Schnittstelle beziehen, die Ihr Betriebssystem erkennt. Wollen Sie zum Beispiel die Werte 101, 103 und 125 verwenden, müssen die Schnittstellen COM1, COM3 und COM25 richtig konfiguriert sein.

Anmerkung zu den seriellen Schnittstellen

Die Betriebssysteme erkennen standardmäßig zwei serielle Schnittstellen der Software: unter Windows die Schnittstellen COM1 und COM2, auf Macintosh die Modem- und die Druckerschnittstelle. Sie können natürlich mit Hilfe von zusätzlicher Hardware weitere serielle Schnittstellen hinzufügen. 4D hat ursprünglich nur die o.a. Standard-Schnittstellen unterstützt, nach und nach sind weitere serielle Schnittstellen hinzugekommen. Aus Kompatibilitätsgründen bleiben beide Systeme erhalten.

- Wollen Sie nur eine Standardschnittstelle zuweisen (COM2 /Drucker oder COM1/Modem), können Sie im Parameter den Wert 0, 1, 20, 21, 30 oder 31 übergeben (das entspricht der alten Funktionsweise von 4D) bzw. einen Wert > 100 (siehe oben).
- Wollen Sie erweiterte serielle Schnittstellen zuweisen, müssen Sie in *Schnittstelle* (Zuweisen der n-ten seriellen Schnittstelle) den Wert N+100 übergeben, und diesen noch um 100 oder 200 erhöhen, je nachdem, ob Sie ein Software- oder ein Hardware-Protokoll verwenden.

Beispiel 1

Für COM2 bzw. die Druckerschnittstelle ohne Protokoll verwenden Sie die Syntax:

```
SET CHANNEL(0;param)
```

oder

```
SET CHANNEL(102;param)
```

Beispiel 2

Für COM1 bzw. die Modemschnittstelle mit dem Protokoll XON/XOFF verwenden Sie die Syntax:

```
SET CHANNEL(21;param)
```

oder

```
SET CHANNEL(201;param)
```

Beispiel 3

Für COM25 mit dem Protokoll RTS/CTS verwenden Sie ausschließlich die Syntax:

```
SET CHANNEL(325;param)
```

Mit *Parameter* wählen Sie die Parameter der Kommunikation: Geschwindigkeit, Anzahl der Bits sowie der Stop-Bits und Parität. Sie legen den Wert für *Parameter* durch Hinzufügen der unten aufgeführten Werte fest. Wollen Sie z.B. setzen 1200 Baud, 8 Datenbits, 1 Stop-Bit und keine Parität, fügen Sie hinzu: 94 + 3072 + 16384 + 0 = 19550. In *Parameter* setzen Sie dann 19550 ein.

	Hinzuzufügende Werte für Parameter	Beschreibung
Geschwindigkeit (in Baud)	380	300
	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
Anzahl Datenbits	0	5
	2048	6
	1024	7
	3072	8
Anzahl Stop-Bits	16384	1
	–32768	1.5
	–16384	2
Parität	0	Keine
	4096	Ungerade
	12288	Gerade

Tipp: Die verschiedenen Werte zum Akkumulieren und zur Übergabe in den Parametern *Schnittstelle* und *Parameter* (mit Ausnahme der Werte für COM1...COM99) sind in der Designumgebung im Explorer Fenster als vordefinierte Konstanten unter dem Thema **Kommunikation** verfügbar. Für COM1...COM99 verwenden Sie numerische Entsprechungen.

Arbeiten mit Dokumenten auf der Festplatte - SET CHANNEL(Operation;Dokument)

Die zweite Syntax von **SET CHANNEL** erstellt, öffnet und schließt ein Dokument. Sie können jedoch im Gegensatz zu den Befehlen im Kapitel **Systemdokumente** nur ein Dokument zur gleichen Zeit öffnen. Das Dokument kann aus- oder eingelesen werden.

Mit *Operation* legen Sie fest, was mit dem in *Dokument* festgelegten Dokument ausgeführt wird. Nachfolgende Tabelle zeigt die für *Operation* und *Dokument* zugelassenen Werte und die daraus resultierende Operation.

Operation	Dokument	Ergebnis
10	Alphanumerisch	Öffnet das alphanumerische Dokument. Gibt es das Dokument nicht, wird es erstellt und geöffnet.
10	"" (alphanum. leer)	Zeigt das Dialogfenster Datei öffnen zum Öffnen einer Datei an. Alle Datentypen werden angezeigt.
11	nichts	Schließt eine geöffnete Datei.
12	"" (alphanum. leer)	Zeigt das Dialogfenster Datei sichern zum Erstellen einer neuen Datei an.
13	"" (alphanum. leer)	Zeigt das Dialogfenster Datei öffnen zum Öffnen einer Datei an. Nur Textdateien werden angezeigt.

Wollen Sie z.B. ein Dialogfenster Datei öffnen anzeigen, um eine Textdatei zu öffnen, schreiben Sie folgende Programmierzeile:

```
SET CHANNEL(13;"")
```

Alle hier aufgelisteten Operationen setzen bei Bedarf die Systemvariable *Document*. War die Operation erfolgreich, hat die Systemvariable OK den Wert 1, ansonsten den Wert 0.

Beispiel 4

Siehe Beispiele für die Befehle **RECEIVE BUFFER**, **SET TIMEOUT** und **RECEIVE RECORD**.

SET TIMEOUT

SET TIMEOUT (Sekunden)

Parameter	Typ		Beschreibung
Sekunden	Lange Ganzzahl	➡	Sekunden bis zum Timeout

Beschreibung

Der Befehl **SET TIMEOUT** gibt an, wieviel Zeit ein Befehl der seriellen Schnittstelle für die Ausführung hat. Ein Befehl, der nicht in der in *Sekunden* festgelegten Zeit abgeschlossen ist, wird abgebrochen. Der Fehler -9990 wird generiert, und die Systemvariable OK wird auf 0 gesetzt. Sie können diesen Fehler mit einer Methode **ON ERR CALL** abfangen.

Beachten Sie, dass *Sekunden* die Gesamtausführungszeit für den Befehl angibt, unabhängig von der Anzahl der übertragenden Zeichen. Wollen Sie die vorige Einstellung abbrechen und die Verwaltung der Kommunikation mit der seriellen Schnittstelle stoppen, geben Sie für *Sekunden* den Wert Null ein.

SET TIMEOUT gilt für die Befehle:

- **RECEIVE PACKET**
- **RECEIVE RECORD**
- **RECEIVE VARIABLE**

Beispiel

Folgendes Beispiel setzt die serielle Schnittstelle zum Empfangen von Daten und dann ein Timeout. Die Daten werden mit dem Befehl **RECEIVE PACKET** gelesen. Werden die Daten nicht in der festgelegten Zeit empfangen, erscheint eine Fehlermeldung:

```
SET CHANNEL(MacOS serial port;Speed 9600+Data bits 8+
Stop bits one+Parity none) ` Öffne serielle Schnittstelle
SET TIMEOUT(10) ` Setze Timeout auf 10 Sekunden
ON ERR CALL("CATCH COM ERRORS") ` Lass die Methode ohne Unterbrechung laufen
RECEIVE PACKET(vtBuffer;Char(13)) ` Lese bis Zeilenschaltung gesetzt wird
If(OK=0)
   ALERT("Fehler beim Empfangen von Daten.")
Else
   [People]Name:=vtBuffer ` Sichere empfangene Daten in ein Datenfeld
End if
ON ERR CALL("")
```

USE CHARACTER SET

USE CHARACTER SET (Filtername {; Modus})

Parameter	Typ		Beschreibung
Filtername	String, Operator	➡	Name des zu benutzenden Zeichensatzes oder * für Zurücksetzen auf Standard Zeichensatz
Modus	Lange Ganzzahl	➡	0 = Exportfilter, 1 = Importfilter, ohne Angabe=Exportfilter

Beschreibung

Der Befehl **USE CHARACTER SET** ändert den von 4D verwendeten Zeichensatz beim Übertragen von Daten zwischen der Datenbank und einem Dokument oder einer seriellen Schnittstelle für den aktuellen Prozess. Dazu gehören auch das Importieren bzw. Exportieren von Text-, DIF- und SYLK-Dateien. Ein Zeichensatz funktioniert auch für die Datenübertragung mit den Befehlen **SEND PACKET**, **RECEIVE PACKET** (für Pakete vom TypText) und **RECEIVE BUFFER**. Sie hat keine Auswirkung auf die Datenübertragung mit den Befehlen **SEND RECORD**, **SEND VARIABLE**, **RECEIVE RECORD**, **SEND PACKET**, **RECEIVE PACKET** (für Pakete vom Typ BLOB) und **RECEIVE VARIABLE**.

Der Parameter *Filtername* muss dem „IANA" Namen oder Alias des zu verwendenden Zeichensatzes entsprechen. „iso-8859-1" oder „utf-8" sind z.B. gültige Namen, sowie die Aliasse „latin1" oder „l1". Weitere Informationen dazu finden Sie im Internet unter http://www.iana.org/assignments/character-sets.
Beispiele für IANA Namen finden Sie in der Beschreibung zum Befehl **CONVERT FROM TEXT**.

Ist *Modus* gleich 0, wird der Filter für den Export verwendet. Ist *Modus* gleich 1, wird der Filter für den Import verwendet. Ist der Parameter *Modus* nicht übergeben, wird standardmäßig der Exportfilter verwendet.

Übergeben Sie den Parameter *, wird der standardmäßige Zeichensatz wiederhergestellt. Das ist je nach dem in *Modus* übergebenen Wert entweder der Import- oder der Exportfilter.

In 4D ist der Zeichensatz standardmäßig UTF-8.

Beispiel

Folgendes Beispiel (Unicode Modus) verwendet den Zeichensatz UTF-16, um einen Text zu exportieren. Anschließend wird der Standard Zeichensatz wiederhergestellt:

```
USE CHARACTER SET("UTF-16LE";0) ` Verwende Zeichensatz UTF_16 'LIttle Endian'
EXPORT TEXT([MyTable];"MyText") ` Exportiere Daten durch diesen Filter
USE CHARACTER SET(*;0) ` Stelle wieder den Standard Zeichensatz her
```

Systemvariablen und Mengen

Die Systemvariable OK hat den Wert 1, wenn der Filter erfolgreich geladen wurde, ansonsten den Wert 0.

LDAP

- Einführung in LDP Befehle
- LDAP LOGIN
- LDAP LOGOUT
- LDAP Search
- LDAP SEARCH ALL

Einführung in LDP Befehle

MIt LDAP Befehlen kann sich Ihre 4D Anwendung über ein LDAP Protokoll an ein vorhandenes Verzeichnis des Unternehmens anmelden, wie z.B. Microsoft Active Directory. Sie können auf Daten auf dem Server zugreifen und darin suchen.

Hinweis: LDAP (Lightweight Directory Access Protocol) ist ein offener Standard, um auf verteilte Informationsdienste zuzugreifen oder solche zu unterhalten. Weitere Informationen dazu finden Sie auf der Wikipedia Seite zu LDAP oder auf der Hauptseite OpenLDAP Software.

Mit diesen Befehlen können Sie folgendes durchführen:

- Login und Kennwort der Windows Sitzung (für Microsoft Active Directory) verwenden, um Zugriff auf Ihre 4D Anwendung zu geben, so das der Endanwender sich nur ein einziges Kennwort merken muss.
- Im Unternehmensverzeichnis nach Benutzerinformationen, wie vollständiger Name, E-Mail, Telefonnummer, Gruppenzugehörigkeit, etc. suchen.

In 4D wird eine LDAP-Verbindung über **LDAP LOGIN** geöffnet. Sie wird an den aktuellen 4D Prozess angebunden und muss über **LDAP LOGOUT** geschlossen werden, bzw. wenn der Prozess die Ausführung beendet.

Glossar

Hier ist die Liste der wichtigsten Acronyme, die in der LDAP Umgebung verwendet werden:

Acronym	Definition
LDAP	Lightweight Directory Access Protocol
AD	Active Directory. AD ist eine Datenbank von Microsoft mit Verzeichnisdiensten, LDAP ist eines der Protokolle, um damit zu kommunizieren.
CN	Common Name, zum Beispiel "John Doe"
DN	Distinguished Name, zum Beispiel "cn=John Doe,ou=users,dc=example,dc=com"
SAM-Account-Name	Security Account Manager, logon Name für AD, zum Beispiel "jdoe"
OU	Organizational unit, Gruppen des Server Baums
DC	Domain Components, Root und Hauptzweige des Server Baums
uid	User identifier

LDAP LOGIN

LDAP LOGIN (Url ; Login ; Kennwort {; digest})

Parameter	Typ		Beschreibung
Url	String	➡	URL des LDAP Servers für die Verbindung
Login	String	➡	Login Eingabe
Kennwort	String	➡	Kennwort der Eingabe, defiiert über vorgegebenen DN
digest	Lange Ganzzahl	➡	0 = Sende Kennwort in digest MD5 (Standard) 1 = Sende Kennwort ohne Verschlüsselung

Beschreibung

Der Befehl **LDAP LOGIN** öffnet eine Verbindung zum LDAP Server im Lesemodus, angegeben im Parameter *url* mit den Kennungen *Login* und *Kennwort*. Akzeptiert der Server diese Verbindung, wird sie für alle nachfolgend ausgeführten LDAP Suchen im aktuellen Prozess verwendet, bis der Befehl **LDAP LOGOUT** ausgeführt oder der Prozess geschlossen wird.

Im Parameter *Url* übergeben Sie die komplette Url des LDAP Server für die Verbindung, inkl. Protokoll und Port (standardmäßig 389). Dieser Parameter sollte die Vorgaben von rfc2255 berücksichtigen.
Sie können sichere Verbindungen über TLS über eine URL öffnen, die mit "ldaps" startet und eine spezifische Port Nummer verwendet, z.B. "ldaps://svr.ldap.acme.com:1389". Der LDAP Server muss ein SSL Zertifikat haben (zumindest für Microsoft Active Directory). Eine TLS Verbindung wird dringend empfohlen, wenn das Kennwort in Volltext gesendet wird (siehe unten).

Hinweis: Übergeben Sie in *Url* einen leeren String, versucht der Befehl, sich an den standardmäßig verfügbaren LDAP Server der Domain anzumelden. Dieses Feature dient nur für Testzwecke, aus Performance Gründen sollte es nicht im laufenden Betrieb verwendet werden.

Im Parameter *Login* übergeben Sie das Benutzer Account auf dem LDAP Server, im Parameter *Kennwort* das Benutzerkennwort. Je nach Konfiguration des LDAP Server kann *Login* standardmäßig einer der folgenden Login Strings sein:

- Ein Distinguished Name (DN), zum Beispiel "CN=John Smith,OU=users,DC=example,DC=com"
- Der Benutzername (CN), zum Beispiel "CN=John Smith"
- Eine E-Mail Adresse, zum Beispiel "johnsmith@4d.fr"
- Ein SAM-Account-Name, zum Beispiel "jsmith"

Beachten Sie, dass für *Login* angenommene Werte sich nach dem Übertragungsmodus für Kennwörter richten, definiert im Parameter *digest*. Hier ein Beispiel für die standardmäßige Konfiguration von Microsoft Active Directory:

- Mit dem Übertragungsmodus LDAP password MD5 wird für ein Login allein der SAM-Account-Name akzeptiert.
- Mit dem Übertragungsmodus LDAP password plain text (Klartext) kann der Parameter *Login* entweder DN, CN oder eine E-Mail Adresse sein. Ein SAM-Account-Name wird auch akzeptiert, jedoch nur mit vorangestelltem Domain-Name (zum Beispiel "dc-acme.com/jsmith").

Über den Parameter *digest* lässt sich die Art der Kennwortübertragung über das Netzwerk ändern. Sie können eine der folgenden Konstanten unter dem Thema **LDAP** verwenden:

Konstante	Typ	Wert	Kommentar
LDAP password MD5	Lange Ganzzahl	0	In MD5 verschlüsseltes Kennwort senden
LDAP password plain text	Lange Ganzzahl	1	Kennwort ohne Verschlüsselung senden (TLS Verbindung empfohlen)

Standardmäßig wird *Kennwort* in MD5 übertragen. Bei Bedarf übergeben Sie LDAP password plain text, zum Beispiel, wenn Sie mit dem LDAP Server verschiedene Login Typen verwenden wollen. In einer Produktionsumgebung empfehlen wir, für die *url* eine TLS Verbindung zu verwenden.

Hinweis: Bei Authentifizierung mit einem leeren Kennwort können Sie den anonymen Verbindungsmodus eingeben - sofern er vom LDAP Server zugelassen wird. Hierbei können jedoch Fehler auftreten, wenn Sie versuchen, Operationen auszuführen, die in diesem Modus nicht zugelassen sind.

Bei gültigen Login-Parametern wird im 4D Prozess eine Verbindung zum LDAP Server geöffnet. Sie können dann über LDAP Suchbefehle Information erhalten oder suchen.

Vergessen Sie nicht, den Befehl **LDAP LOGOUT** aufzurufen, wenn die Verbindung zum LDAP Server nicht mehr benötigt wird.

Beispiel 1

In einen LDAP Server einloggen und eine Suche durchführen:

```
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  //Ausloggen nicht vergessen
```

Beispiel 2

Dieses Beispiel versucht, sich an eine Anwendung anzumelden:

```
ON ERR CALL("ErrHdlr")  //Fehler verwalten
errOccured:=False
errMsg:=""
LDAP LOGIN(vUrlLdap;vUserCN;vPwd;LDAP password MD5)
Case of
    :(Not(errOccured))
        ALERT("Anmeldung erfolgreich.")

    :(errOccured)
        ALERT("Fehler in Ihren Parametern")
End case
LDAP LOGOUT
ON ERR CALL("")
```

LDAP LOGOUT

LDAP LOGOUT
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **LDAP LOGOUT** schließt die Verbindung zum LDAP Server im aktuellen Prozess (falls vorhanden). Gibt es keine Verbindung, wird der Fehler 1003 zurückgegeben. Er meldet, dass Sie nicht eingeloggt sind.

LDAP Search

LDAP Search (dnRootEinstieg ; Filter {; Reichweite {; Attribute {; AttributealsArray}}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
dnRootEinstieg	String	➡	Distinguished Name des Root Einstiegs zum Starten der Suche
Filter	String	➡	LDAP Suchfilter
Reichweite	String	➡	Reichweite der Suche: "base" (Standard), "one" oder "sub"
Attribute	Array Text	➡	zu holendes Attribut bzw. Attribute
AttributealsArray	Array Boolean	➡	true = Attribute als Array zurückgeben false = Attribute als einfache Variable zurückgeben
Funktionsergebnis	Objekt	⮌	Schlüssel/Wert Attribute

Beschreibung

Der Befehl **LDAP Search** sucht nach dem ersten Vorkommen, das zum definierten Kriterium im Ziel LDAP Server passt. Dieser Befehl muss innerhalb einer Verbindung zum LDAP Server ausgeführt werden, geöffnet mit **LDAP LOGIN**. Andernfalls wird der Fehler 1003 zurückgegeben.

In *dnRootEinstieg* übergeben Sie den *Distinguished Name* des Root Einstiegs für den LDAP Server; die Suche startet an dieser Stelle.

In *Filter* übergeben Sie den auszuführenden LDAP Suchfilter. Der Filterstring muss die Vorgaben von rfc2225 berücksichtigen. Standardmäßig, d.h. ohne diesen Parameter, wird ein leerer String "" benutzt (kein Filter wird angewendet.).

In *Reichweite* übergeben Sie eine der folgenden Konstanten unter dem Thema **LDAP**:

Konstante	Typ	Wert	Kommentar
LDAP all levels	Zeichenkette	sub	Suche nur in der Root Einstiegsebene, definiert durch *dnRootEinstieg* (Standard, wenn weggelassen)
LDAP root and next	Zeichenkette	one	Suche in der Root Einstiegsebene, definiert durch *dnRootEinstieg* und in den direkt nachfolgenden Einstiegen auf einer Ebene
LDAP root only	Zeichenkette	base	Suche nur in der Root Einstiegsebene, definiert durch *dnRootEinstieg* (Standard, wenn weggelassen)

In *Attribute* übergeben Sie ein Text Array mit der Liste aller LDAP Attribute, die von den passenden Einstiegspunkten zu holen sind. Standardmäßig, d.h. ohne diesen Parameter, werden alle Attribute geholt.

Hinweis: Beachten Sie, dass LDAP Attributnamen zwischen Groß- und Kleinschreibung unterscheiden. Weitere Informationen finden Sie auf der Seite Attributes. Sie zeigt alle verfügbaren Attribute für Microsoft Active Directory.

Standardmäßig gibt der Befehl bei mehreren Ergebnissen Attribute als Collection zurück, bei einem einzelnen Ergebnis als Variable. Mit dem optionalen Parameter *AttributealsArray* können Sie festlegen, ob der Wert bzw. die zurückgegebenen Werte als Collection oder Variable formatiert werden sollen:

- Übergeben Sie **true** in einem Element, wird das entsprechende Element des Parameters *Attribute* in einer Collection zurückgegeben. Wird nur ein Wert gefunden, gibt der Befehl eine Collection mit einem einzelnen Element zurück.
- Übergeben Sie **false** in einem Element, wird das entsprechende Element des Parameters *Attribute* in einer einfachen Variablen zurückgegeben. Werden mehrere Einträge gefunden, gibt der Befehl nur das erste Element zurück.

Beispiel 1

Die Telefonnummer des Benutzers "smith" im Firmenverzeichnis erhalten:

```
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
```

Beispiel 2

Ein Array mit allen Werten für das Attribut "memberOf" erhalten, um die Gruppenzugehörigkeit für einen Benutzer abzufragen:

```
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)
  // in $_arrMemberOf haben wir ein Array mit allen Gruppen
```

LDAP SEARCH ALL

LDAP SEARCH ALL (dnRootEinstieg ; arrErgebnis ; Filter {; Reichweite {; Attribute {; AttributealsArray}}})

Parameter	Typ		Beschreibung
dnRootEinstieg	String	➡	Distinguished Name des Root Einstiegs zum Starten der Suche
arrErgebnis	Array Objekt	⬅	Suchergebnis
Filter	String	➡	LDAP Suchfilter
Reichweite	String	➡	Reichweite der Suche: "base" (Standard), "one" oder "sub"
Attribute	Array Text	➡	Zu holende Attribute
AttributealsArray	Array Boolean	➡	true = Attribute als Array zurückgeben false = Attribute als einfache Variable zurückgeben

Beschreibung

Der Befehl **LDAP SEARCH ALL** sucht nach allen Vorkommen, die zum definierten Kriterium im Ziel LDAP Server passen. Dieser Befehl muss innerhalb einer Verbindung zum LDAP Server ausgeführt werden, geöffnet mit **LDAP LOGIN**, andernfalls wird der Fehler 1003 zurückgegeben.

Beachten Sie, dass LDAP Server in der Regel ein Maximum an Werten vorschreiben, die von einer Suche empfangen werden können. Microsoft Active Directory begrenzt diese z.B. standardmäßig auf 1000 Werte.

In *dnRootEinstieg* übergeben Sie den *Distinguished Name* des Root Einstiegs für den LDAP Server; die Suche startet an dieser Stelle. Sie können eine leeren String "" übergeben, um die Suche nicht zu filtern; mit "*" können Sie nach Unterstrings suchen.

In *tabErgebnis* übergeben Sie ein Objekt Array, das mit allen passenden Werten gefüllt wird; in diesem Array ist jedes Element ein Objekt mit Attribut/Wert Paaren, die pro passendem Wert zurückgegeben werden. Im Parameter *Attribute* können Sie definieren, wie die Attribute zurückgegeben werden.

In *Filter* übergeben Sie den auszuführenden LDAP Suchfilter. Der Filterstring muss die Vorgaben von rfc2225 berücksichtigen.

In *Reichweite* übergeben Sie eine der folgenden Konstanten unter dem Thema "LDAP":

Konstante	Typ	Wert	Kommentar
LDAP all levels	Zeichenkette	sub	Suche nur in der Root Einstiegsebene, definiert durch *dnRootEinstieg* (Standard, wenn weggelassen)
LDAP root and next	Zeichenkette	one	Suche in der Root Einstiegsebene, definiert durch *dnRootEinstieg* und in den direkt nachfolgenden Einstiegen auf einer Ebene
LDAP root only	Zeichenkette	base	Suche nur in der Root Einstiegsebene, definiert durch *dnRootEinstieg* (Standard, wenn weggelassen)

In *Attribute* übergeben Sie ein Text Array mit der Liste aller LDAP Attribute, die von den passenden Einstiegspunkten zu holen sind. Standardmäßig, d.h. ohne diesen Parameter, werden alle Attribute geholt.

Hinweis: Beachten Sie, dass LDAP Attributnamen zwischen Groß- und Kleinschreibung unterscheiden. Weitere Informationen zu LDAP Attributen finden Sie auf der Seite Attributes. Sie zeigt alle verfügbaren Attribute für das Microsoft Active Directory.

Bei mehreren Ergebnissen gibt der Befehl Attribute standardmäßig als Array zurück, bei einem Ergebnis als Variable. Mit dem optionalen Parameter *AttributealsArray* können Sie festlegen, ob der bzw. die zurückgegebenen Werte als Array oder als Variable formatiert werden sollen:

- Übergeben Sie **true** in einem Element, wird das entsprechende Element des Parameters *Attribute* in einem Array zurückgegeben. Wird nur ein Wert gefunden, gibt der Befehl ein Array mit einem einzelnen Element zurück.
- Übergeben Sie **false** in einem Element, wird das entsprechende Element des Parameters *Attribute* in einer einfachen Variable zurückgegeben. Werden mehrere Werte gefunden, gibt der Befehl nur das erste Element zurück.

Beispiel 1

Die Telefonnummer aller Benutzer mit Namen "smith" im Firmenverzeichnis erhalten:

```
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 kann z.B. enthalten
 // $_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"}
 // ...
```

Beispiel 2

Beispiele für die Verwendung des Parameters *attributesAsArray*:

```
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)
 // $_arrMemberOf ist ein Array mit allen Gruppen der Eingabe
```

```
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")
 // $memberOf ist eine Variable mit der ersten Gruppe der Eingabe
```

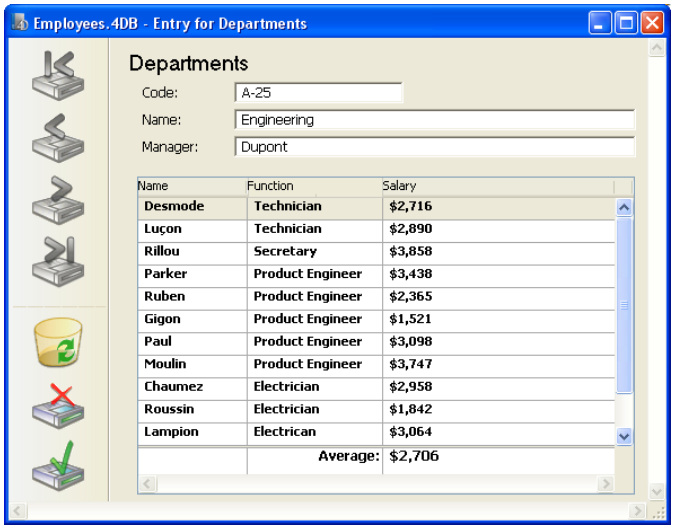
Listbox

- Einführung in Listboxen
- Hierarchische Listboxen verwalten
- Objekt Arrays in Spalten von Listboxen (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

Einführung in Listboxen

Die Befehle und Funktionen dieses Kapitels dienen zum Verwalten von Formularobjekten vom Typ Listbox.

Eine Listbox stellt Daten in Form von Spalten und auswählbaren Zeilen dar und bietet viele Oberflächenfunktionen, wie Werte eingeben, Spalten sortieren, eine Hierarchie anzeigen, wechselnde Farben definieren, etc.

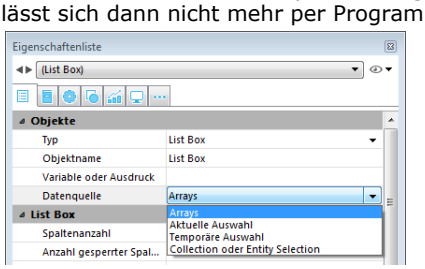


Sie können eine Listbox komplett im 4D Formulareditor einrichten und sie auch über Programmierung verwalten. Weitere Informationen dazu finden Sie im Kapitel **Listboxen** des Handbuchs *4D Designmodus*.

Die Programmierung von Listboxen erfolgt auf dieselbe Art und Weise wie für andere 4D Formularobjekte vom Typ Liste. Hierfür müssen jedoch spezifische Regeln beachtet werden, die nachfolgend erläutert werden.

Datenquellen und Werte verwalten

Eine Listbox kann eine oder mehrere Spalten enthalten, die sich einem 4D Array, einer Datensatzauswahl, einer Collection oder einer Entity-Selection zuordnen lassen. Bei einer Listbox vom Typ Auswahl, Collection, Entity-Selection werden die Spalten Ausdrücken zugewiesen. Beim Typ Auswahl lassen sich Spalten auch direkt Feldern zuweisen.
Es ist nicht möglich, mehrere Arten von Datenquellen (Arrays, Auswahl, Collection oder Entity-Selection) in der gleichen Listbox zu kombinieren. Die Datenquelle wird gesetzt, wenn die Listbox im Formulareditor über die Eigenschaftenliste erstellt wird. Sie lässt sich dann nicht mehr per Programmierung verändern.



Listboxen vom Typ Array

Bei diesem Typ muss jede Spalte einem eindimensionalen 4D Array zugewiesen werden; alle Array-Typen sind möglich, mit Ausnahme von Array mit Zeigern. Das Anzeigeformat für jede Spalte lässt sich im Formulareditor oder über den Befehl **OBJECT SET FORMAT** definieren.

Zum Verwalten der Werte in den Spalten (Eingabe und Anzeige) per Programmierung dienen Befehle für Listboxen, z.B. **LISTBOX INSERT ROWS** oder **LISTBOX DELETE ROWS**, sowie Befehle für Arrays. Sie können z.B. den Inhalt einer Spalte mit folgender Anweisung initialisieren:

```
ARRAY TEXT(ColumnName;size)
```

Sie können auch eine Liste verwenden:

```
LIST TO ARRAY("ListName";ColumnName)
```

Warnung: Enthält die Listbox mehrere Spalten in verschiedenen Größen, erscheint nur die Anzahl der Einträge im kleinsten Array (Spalte). Stellen Sie sicher, dass alle Arrays jeweils dieselbe Anzahl Elemente haben. Auch wenn eine Spalte der Listbox leer ist, zeigt die Listbox nichts an. Das passiert, wenn das Array nicht korrekt deklariert wurde oder über die Programmiersprache in der Größe angepasst wurde.

Listboxen vom Typ Auswahl

Bei diesem Typ lässt sich jede Spalte einem Feld oder Ausdruck zuweisen. Der Inhalt jeder Spalte wird dann gemäß der Datensatzauswahl gewertet, das ist die aktuelle Auswahl einer Tabelle oder eine temporäre Auswahl.

Ist die aktuelle Auswahl die Datenquelle, werden Änderungen in der Datenbank automatisch in der Listbox ausgeführt und umgekehrt. Die aktuelle Auswahl ist also auf beiden Seiten immer gleich. Beachten Sie, dass Sie bei Listboxen vom Typ Auswahl nicht die Befehle **LISTBOX INSERT ROWS** und **LISTBOX DELETE ROWS** verwenden können.
Sie können die Spalte einer Listbox mit einem Ausdruck verbinden, dieser kann auf einem oder mehreren Datenfeldern basieren, z.B. *[Angestellte]Nachname+" "+[Angestellte]VorName*) oder einfach eine Formel sein, z.B. **String**(Milliseconds)). Es kann auch eine Projektmethode, Variable oder ein Array-Element sein.

Über den Befehl **LISTBOX SET TABLE SOURCE** können Sie die der Listbox zugeordnete Tabelle per Programmierung verändern.

Listboxen vom Typ Collection oder Entity-Selection

Bei diesem Typ Listbox muss jeder Spalte ein Ausdruck zugewiesen werden. Der Inhalt jeder Zeile wird dann pro Collection-Element oder pro Entity einer Entity-Selection bewertet.

Jedes Element der Collection oder jede Entity ist als ein Objekt verfügbar, das über den Befehl **This** zugänglich ist.
Der Ausdruck einer Spalte kann eine Projektmethode, eine Variable oder eine Formel sein, die über **This** auf jedes Objekt Entity oder Collection-Element zugreift, z.B. **This.<propertyName>** (oder **This.value** bei einer Collection mit skalaren Werten). Über die Befehle **LISTBOX SET COLUMN FORMULA** und **LISTBOX INSERT COLUMN FORMULA** können Sie Spalten per Programmierung ändern.

Ist die Datenquelle eine Collection, wird jede Änderung in der Collection, z.B. über die verschiedenen Methoden im Kapitel **Collections** in der Listbox und umgekehrt ausgeführt.

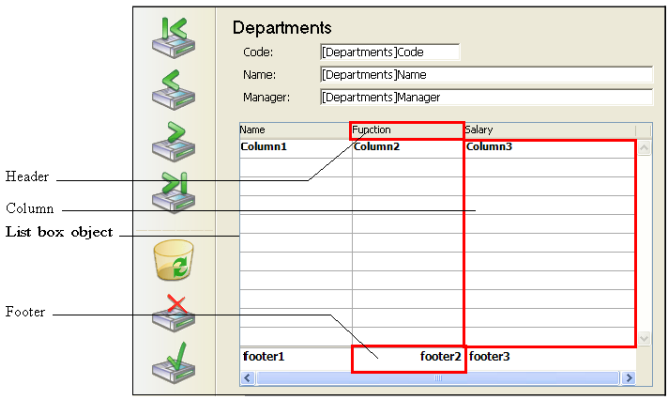
Ist die Datenquelle eine Entity-Selection, werden alle Änderungen in der Listbox automatisch in der Datenbank gesichert. Änderungen in der Datenbank sind in der Listbox nach erneutem Laden der entsprechenden Entities verfügbar.

Objekt, Spalte, Kopfteil und Fußteil

Das Objekt Listbox besteht aus vier voneinander unabhängigen Elementen:

- Das Objekt selbst
- Spalten
- Kopfteile der Spalten. Sie können ein- oder ausgeblendet sein (standardmäßig angezeigt)
- Fußteile der Spalten. Sie können ein- oder ausgeblendet sein (standardmäßig ausgeblendet)

Diese Elemente lassen sich im Formulareditor einzeln auswählen, jedes davon hat einen eigenen Objekt- und Variablennamen und lässt sich separat bearbeiten.



Spalten haben im Formular standardmäßig den Namen *Spalte1, 2 ... X*, *Spaltentitel Header1, 2 ... X*, Fußteile *Footer1, 2 ... X,* sie sind unabhängig vom Objekt Listbox selbst. Beachten Sie, dass derselbe Name für die Objekte und die dazugehörigen Variablen verwendet wird, mit Ausnahme der Fußteile (Variablen für Fußteile sind standardmäßig leer; 4D verwendet dynamische Variablen).

Jede Art Eintrag enthält eigene sowie mit anderen Einträgen geteilte Merkmale. Schriften lassen sich z.B. für die ganze Listbox zuweisen oder für getrennt Spalten und Kopfteil. Eigenschaften für die Eingabe lassen sich hingegen nur für Spalten definieren.

Diese Regeln gelten für die Befehle im Kapitel **Objekte (Formulare)**, die für Listboxen geeignet sind. Je nach Funktionalität gelten sie für die gesamte Listbox, für Spalten, Spaltentitel bzw. Fußteile. Das entsprechende Element übergeben Sie mit seinem Namen oder über die dazuhörige Variable.

Nachfolgende Tabelle zeigt die jeweilige Reichweite der Befehle aus dem Kapitel **Objekte (Formulare)** an, die für Listboxen verwendbar sind:

Befehl Objekteigenschaften	Objekt	Spalten	Kopfteile der Spalten	Fußteile der Spalten
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	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

Hinweis: Bei Listboxen vom Typ Array lassen sich Stil, Farbe für Schrift oder Hintergrund und Sichtbarkeit für jede Zeile getrennt angeben. Dies wird über Arrays verwaltet, die der Listbox in der Eigenschaftenliste zugewiesen werden. Über den Befehl **LISTBOX GET ARRAYS** können Sie die Namen dieser Arrays per Programmierung herausfinden.

Listbox und Programmiersprache

Objektmethoden

Sie können eine Objektmethode einem Objekt bzw. einer Spalte der Listbox hinzufügen. Objektmethoden werden in folgender Reihenfolge aufgerufen:

1. Objektmethode jeder Spalte
2. Objektmethode der Listbox

Die Objektmethode der Spalte erhält Ereignisse, die in deren Kopfteil und Fußteil eintreten

OBJECT SET VISIBLE und Kopf-/Fußteile

Verwenden Sie den Befehl **OBJECT SET VISIBLE** mit einem Spaltentitel, gilt er für alle Titel der Listbox, unabhängig vom Titel, der im Befehl übergeben wurde. So blendet die Anweisung **OBJECT SET VISIBLE(*;"header3";False)** alle Spaltentitel im Objekt Listbox aus, zu denen „header3" gehört und nicht allein diesen Titel.

Um die Sichtbarkeit dieser Objekte über den Befehl **OBJECT SET VISIBLE** zu verwalten, müssen sie in der Listbox auf der Ebene des Formulareditors angezeigt werden, d.h. für das Objekt muss in der Eigenschaftenliste die Option **Kopfteil anzeigen** bzw. **Fußteil anzeigen** markiert sein.

Funktionsweise von OBJECT Get pointer

Die Funktion **OBJECT Get pointer** wird mit der Konstante Object with focus oder Object current verwendet (früher die Funktionen **Focus object** und **Self**). Sie lässt sich in der Objektmethode einer Listbox bzw. einer Spalte der Listbox verwenden. Sie geben einen Zeiger auf die Listbox, die Spalte der Listbox (1) oder die Variable des Kopfteils zurück. Nachfolgende Tabelle zeigt die Funktionsweise im Einzelnen:

Ereignis	Objekt mit Fokus	aktuelles Objekt
On Clicked	Listbox	Spalte
On Double Clicked	Listbox	Spalte
On Before Keystroke	Spalte	Spalte
On After Keystroke	Spalte	Spalte
On After Edit	Spalte	Spalte
On Getting Focus	Spalte oder Listbox (*)	Spalte oder Listbox (*)
On Losing Focus	Spalte oder Listbox (*)	Spalte oder Listbox (*)
On Drop	Listbox Quelle	Listbox (*)
On Drag Over	Listbox Quelle	Listbox (*)
On Begin Drag Over	Listbox	Listbox (*)
On Mouse Enter	Listbox (**)	Listbox (**)
On Mouse Move	Listbox (**)	Listbox (**)
On Mouse Leave	Listbox (**)	Listbox (**)
On Data Change	Spalte	Spalte
On Selection Change	Listbox (**)	Listbox (**)
On Before Data Entry	Spalte	Spalte
On Column Moved	Listbox	Spalte
On Row Moved	Listbox	Listbox
On Column Resize	Listbox	Spalte
On Open Detail	Nil	list box (**)
On Close Detail	Nil	list box (**)
On Header Click	Listbox	Kopfteil
On Footer Click	Listbox	Fußteil
On After Sort	Listbox	Kopfteil

(*) Wird der Fokus innerhalb einer Listbox geändert, wird ein Zeiger auf die Spalte zurückgegeben. Wird der Fokus auf der übergeordneten Formularebene geändert, wird ein Zeiger auf die Listbox zurückgegeben. Bei einer Objektmethode einer Spalte wird ein Zeiger auf die Spalte zurückgegeben.
(**) Wird nicht ausgeführt im Kontext einer Objektmethode einer Spalte.

(1) Wird ein Zeiger auf eine Spalte zurückgegeben, richtet sich das angezeigte Objekt nach der Art der Listbox. Bei einer Listbox vom Typ Array gibt die Funktion **OBJECT Get pointer** einen Zeiger auf die Spalte in der Listbox mit Fokus zurück (z.B. gegen ein Array). Über den Zeiger können Sie die Eintragsnummer des geänderten Array sehen. Nehmen wir an, der Benutzer hat die 5. Zeile in Spalte Sp2 geändert:

```
$Column:=OBJECT Get pointer(Object with focus)
  ` $Column enthält einen Zeiger auf Sp2
$Row:=$Column->  `$Row ist gleich 5
```

Bei einer Listbox vom Typ Auswahl, gibt die Funktion **OBJECT Get pointer** folgendes zurück:

- Für eine Spalte, die einem Datenfeld zugeordnet ist, einen Zeiger auf dieses Datenfeld
- Für eine Spalte, die einer Variablen zugeordnet ist, einen Zeiger auf die Variable
- Für eine Spalte, die einem Ausdruck zugeordnet ist, den Zeiger **Is nil pointer**.

OBJECT SET SCROLL POSITION

Der Befehl **OBJECT SET SCROLL POSITION** lässt sich mit einer Listbox verwenden. Er scrollt in den Zeilen, so dass die erste gewählte oder angegebene Zeile angezeigt wird.

EDIT ITEM

Mit dem Befehl **EDIT ITEM** im Kapitel "Eingabekontrolle" können Sie eine Zelle des Objekts Listbox in den Bearbeitungsmodus setzen.

REDRAW

Wird **REDRAW** (Kapitel **Benutzeroberfläche**) auf eine Listbox vom Typ Auswahl angewandt, löst er die Aktualisierung der in der Listbox angezeigten Daten aus.

Hinweis: Der Befehl **REDRAW** wird mit Listboxen vom Typ Entity-Selection nicht unterstützt.

Displayed line number

Die Funktion **Displayed line number** im Kapitel "Auswahl" funktioniert für Listboxen im Kontext des Formularereignisses *On Display Detail*.

Formularereignisse

Es gibt spezifische Formularereignisse zum Verwalten von Listboxen, insbesondere für Drag-and-Drop und Sortieroperationen. Weitere Informationen dazu finden Sie unter der Funktion **Form event**.

Drag-and-Drop

Drag-and-Drop von Daten in Listboxen wird über die Routinen **Drop position** und **DRAG AND DROP PROPERTIES** verwaltet. Sie wurden speziell an Listboxen angepasst.

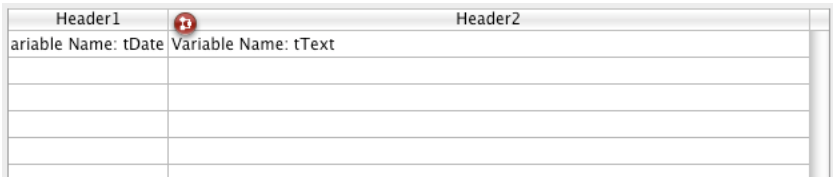
Achtung! Verwechseln Sie nicht Drag-and-Drop mit Bewegen von Zeilen und Spalten. Dies wird von den Befehlen **LISTBOX MOVED ROW NUMBER** und **LISTBOX MOVED COLUMN NUMBER** unterstützt.

Eingabe verwalten

Damit eine Zelle der Listbox eingebbar ist, müssen folgende Bedingungen erfüllt sein:

- Die Zelle der Spalte muss auf **Eingebbar** gesetzt sein (sonst können die Zellen der Spalte nie eingebbar sein).
- Im Ereignis On Before Data Entry gibt $0 nicht -1 zurück.
  Gelangt der Cursor in die Zelle, wird das Ereignis On Before Data Entry in der Methode der Spalte erzeugt. Wird dann im Rahmen dieses Ereignisses $0 auf -1 gesetzt, wird die Zelle als nicht-eingebbar gewertet. Wird das Ereignis nach Drücken der Tasten **Tab** oder **Shift+Tab** erzeugt, geht der Fokus entweder zur nächsten oder zur vorigen Zelle. Ist $0 nicht -1 ($0 ist standardmäßig 0), ist die Zelle eingebbar und wechselt in den Eingabemodus.

Nehmen wir z.B. eine Listbox mit zwei Arrays, eins vom Typ Datum, das andere vom Typ Text. Das Array Datum ist nicht eingebbar, das Array Text ist eingebbar, solange noch kein Datum übertragen wurde.



Hier ist die Methode zur Spalte arrText:

```
Case of
   :(Form event=On Before Data Entry)  // eine Zelle erhält den Fokus
      LISTBOX GET CELL POSITION(*;"lb";$col;$row)
 // Identifikation der Zelle
      If(arrDate{$row}<Current date)  // ist das Datum früher als heute
         $0:=-1  // Zelle ist NICHT eingebbar
      Else
 // sonst ist Zelle eingebbar
      End if
End case
```

Hinweis: Seit 4D v13 wird das Ereignis On Before Data Entry vor On Getting Focus erzeugt.

Um die Datenkonsistenz für Listboxen vom Typ Auswahl zu bewahren, wird ein geänderter Datensatz gesichert, sobald die Zelle bestätigt wird (falls gesetzt, wird der Trigger On saving an existing record aufgerufen), dann wird das Ereignis On Data Change ausgeführt. Der typische Ablauf der Ereignisse, die beim Eingeben oder Ändern von Daten erzeugt werden, ist wie folgt:

Aktion	**Listboxtyp(en)**	**Ablauf der Ereignisse**
Eine Zelle wechselt in den Bearbeitungsmodus	*Alle*	On Before Data Entry
	Alle	On Getting Focus
Ihr Wert wird geändert	*Alle*	On Before Keystroke
	Alle	On After Keystroke
	Alle	On After Edit
Ein Benutzer bestätigt und verlässt die Zelle	*Listboxen vom Typ Auswahl*	***Sichern***
	Listboxen vom Typ Datensatzauswahl	Trigger On saving an existing record trigger (wenn gesetzt)
	Listboxen vom Typ Auswahl	On Data Change(*)
	Listboxen vom Typ Entity-Selection	Entity wird mit Option "automerge" gesichert, optimistisches Sperren (siehe [#title id="9310"/]). Bei erfolgreichem Sichern wird die Entity mit dem zuletzt ausgeführten Update erneuert. Schlägt Sichern fehl, erscheint ein Fehler.
	Alle	On Losing Focus

(*) Bei Listboxen vom Typ Entity-Selection gilt mit dem Ereignis On Data Change folgendes:

- das Objekt **aktueller Eintrag** (siehe **Gruppe Datenquelle**) enthält den Wert *vor* der Änderung.
- das Objekt **This** enthält den geänderten Wert.

Hinweis: Für die Dateneingabe in Listboxen vom Typ Collection/Entity-Selection gibt es eine Einschränkung, wenn der Ausdruck als **null** gewertet wird: der Wert **Null** in der Zelle lässt sich weder bearbeiten noch entfernen.

Sortieren verwalten

Bei Anklicken des Spaltentitels führt die Listbox automatisch eine Standardsortierung der Spalten durch. Das ist die alphanumerische Sortierung der Spaltenwerte in aufsteigender bzw. absteigender Reihenfolge durch aufeinanderfolgende Klicks. Alle Spalten werden immer automatisch aufeinander abgestimmt. Um die Standardsortierung zu unterbinden, deaktivieren Sie für die Listbox die Eigenschaft "Sortierbar".

Der Entwickler kann über den Befehl **LISTBOX SORT COLUMNS** bzw. durch Kombinieren der Formularereignisse On Header Click und On After Sort und 4D Befehlen zur Array-Verwaltung eigene Sortierungen einrichten. Weitere Informationen dazu finden Sie unter der Funktion **Form event**.

Hinweis: Die Spalteneigenschaft „Sortierbar" gilt nur für Standardsortierungen des Benutzers. Der Befehl **LISTBOX SORT COLUMNS** berücksichtigt diese Eigenschaft nicht.

Über den Wert der Variablen für den Spaltentitel können Sie zusätzliche Information verwalten: die aktuelle Sortierung der Spalte im Lesemodus und die Anzeige des Sortierpfeils.

- Hat die Variable den Wert 0 (Null), wird die Spalte nicht sortiert, es erscheint kein Sortierpfeil;

- Hat die Variable den Wert 1, wird die Spalte in aufsteigender Reihenfolge sortiert, der Sortierpfeil erscheint;

- Hat die Variable den Wert 2, wird die Spalte in absteigender Reihenfolge sortiert, der Sortierpfeil erscheint.

  Sie können den Wert der Variablen setzen, damit der Sortierpfeil angezeigt wird, z.B. Header2:=2. In diesem Fall wird die Spaltensortierung selbst nicht geändert; es ist Aufgabe des Entwicklers, das zu verwalten.

Hinweis: Der Befehl **OBJECT SET FORMAT** bietet spezifische Unterstützung für Icons in Kopfteilen von Listboxen. Das ist z.B. hilfreich, wenn Sie einen eigenen Icon zum Sortieren verwenden wollen.

Auswahlen verwalten

Auswahlen werden unterschiedlich verwaltet, je nachdem, ob die Listbox auf einem Array, auf einer Auswahl von Datensätzen oder einer Collection/Entity-Selection basiert.

- **Listbox vom Typ Auswahl**: Auswahlen werden von einer Menge verwaltet, die Sie bei Bedarf verändern können. Sie hat standardmäßig den Namen *$ListboxSetX* (X startet bei 0 und erhöht sich gemäß der Anzahl Listboxen im Formular). Diese Menge wird in den Eigenschaften der Listbox definiert und von 4D automatisch verwaltet: Wählt der Benutzer eine oder mehrere Zeilen in der Listbox, wird die Menge sofort aktualisiert. Sie können auch Befehle aus dem Kapitel **Mengen** verwenden, um die Auswahl der Listbox per Programmierung zu verändern.

- **Listbox vom Typ Collection/Entity-Selection**: Auswahlen werden durch bestimmte Listbox Eigenschaften verwaltet: *Aktueller Eintrag* ist ein Objekt, welches das ausgewählte Element bzw. die Entity empfängt, *Ausgewählte Einträge* ist eine Collection mit ausgewählten Einträgen, *Position aktueller Eintrag* gibt die Position des gewählten Elements bzw. der Entity zurück. Weitere Informationen dazu finden Sie im Abschnitt **Gruppe Datenquelle**.

- **Listbox vom Typ Array**: Mit dem Befehl **LISTBOX SELECT ROW** können Sie per Programmierung eine oder mehrere Zeilen in der Listbox wählen.
  Über die der Listbox zugeordnete Variable können Sie Auswahlen von Objektzeilen erhalten, setzen oder speichern. Diese Variable ist ein Array vom Typ Boolean, die 4D automatisch erstellt und pflegt. Die Größe des Array richtet sich nach der Größe der Listbox: Sie enthält dieselbe Anzahl Elemente wie das kleinste mit Spalten verknüpfte Array.
  Jedes Element des Array gibt *Wahr* zurück, wenn die entsprechende Zeile gewählt ist, sonst *Falsch*. 4D aktualisiert den Inhalt dieses Array gemäß den Benutzeraktionen. Umgekehrt können Sie den Wert von Elementen im Array verändern, um die Auswahl in der Listbox zu ändern.
  Dagegen können Sie in diesem Array weder Zeilen einfügen, überschreiben noch löschen.

Hinweis: Über die Funktion **Count in array** finden Sie die Anzahl der gewählten Zeilen.
Mit der folgenden Methode können Sie die Auswahl der ersten Zeile in der Listbox vom Typ Array invertieren:

```
ARRAY BOOLEAN(tBListBox;10)
\\tBListBox ist der Name der Listbox Variable im Formular
If(tBListBox{1}=True)
   tBListBox{1}:=False
Else
   tBListBox{1}:=True
End if
```

Haben Sie für die Listbox die Option **Auswahlmarkierung ausblenden** markiert, müssen Sie die Auswahlen der Listbox über verfügbare Oberflächenoptionen sichtbar machen. Weitere Informationen dazu finden Sie im Abschnitt **Darstellung von Auswahlen anpassen**.
Weitere Informationen dazu finden Sie im Abschnitt **Hierarchische Listboxen verwalten**.

Listboxen drucken

Ab 4D v12 lassen sich Listboxen auch drucken. Es gibt zwei Druckmodi: Vorschau, um eine Listbox wie ein Formularobjekt zu drucken und Erweitert, um direkt das Drucken des Objekts Listbox innerhalb des Formulars zu steuern. Über eine neue Eigenschaft lässt sich die Druckdarstellung für Listbox Objekte festlegen.

Modus Vorschau

In diesem Modus wird die Listbox über die standardmäßigen Druckbefehle oder den Menübefehl Drucken direkt mit dem Formular gedruckt. Die Listbox wird gedruckt, so wie sie im Formular ist. In diesem Modus lässt sich das Drucken des Objekts nicht präzise steuern, so können Sie bei einer Listbox nicht alle Zeilen drucken, wenn sie mehr enthält als angezeigt werden können.

Erweiterter Modus

Dieser Modus führt das Drucken von Listboxen über den Befehl **Print object** per Programmierung aus (Projektformulare und Tabellenformulare werden unterstützt). Über den Befehl **LISTBOX GET PRINT INFORMATION** lässt sich der Druck des Objekts steuern.
Dieser Modus führt folgendes aus:

- Die Höhe der Listbox wird automatisch verringert, wenn die Anzahl der zu druckenden Zeilen kleiner als die ursprüngliche Höhe des Objekts ist (leere Zeilen werden nicht gedruckt). Die Höhe vergrößert jedoch nicht automatisch gemäß dem Inhalt des Objekts. Die Größe des aktuell gedruckten Objekts lässt sich über den Befehl **LISTBOX GET PRINT INFORMATION** erhalten.

- Das Objekt Listbox wird so wie es ist gedruckt, d.h. unter Berücksichtigung der aktuellen Parameter, wie sichtbare Spaltentitel und Rasterlinien, ausgeblendete und angezeigte Zeilen, etc. Diese Parameter enthalten auch die erste zu druckende Zeile: Rufen Sie den Befehl **OBJECT SET SCROLL POSITION** vor dem Drucken auf, ist die erste gedruckte Zeile in der Listbox die Zeile, welche der Befehl angibt.
- Eine automatische Operation ermöglicht, Listboxen zu drucken, die mehr Zeilen enthalten als angezeigt werden können: durch aufeinanderfolgende Aufrufe der Funktion **Print object** lässt sich jedes Mal ein neuer Satz Zeilen drucken. Über den Befehl **LISTBOX GET PRINT INFORMATION** lässt sich der Status während dem Drucken prüfen.

Schriftstil und -farbe verwalten

Es gibt verschiedene Wege, um Hintergrundfarbe, Schriftfarbe oder -stil für Listboxen zu setzen:

- Eigenschaften des Objekts Listbox
- Eigenschaften der Spalte
- Verwenden von Arrays oder Methoden für die Listbox bzw. pro Spalte
- Textbereich einer Zelle (bei Text mit Mehrfachstil)

Es gibt auch Regeln für Priorität und Vererbung.

Priorität

Wird dieselbe Eigenschaft auf mehr als einer Ebene gesetzt, gilt folgende Priorität:

hohe Priorität	Zelle (bei Text mit Mehrfachstil)
	Spalte Arrays/Methoden
	Listbox Arrays/Methoden
	Spalteneigenschaften
	Listbox Eigenschaften
niedrige Priorität	Meta Info Ausdruck (für Listboxen vom Typ Collection oder Entity-Selection)

Setzen Sie z.B. einen Schriftstil in den Listbox Eigenschaften und einen anderen über ein Array Stil für die Spalte, wird der letzte berücksichtigt.

Wir nehmen eine Listbox, in der die Zeilen in den Farben grau/hellgrau wechseln. Das ist in den Eigenschaften der Listbox definiert. Für diese Listbox wurde auch ein Array Hintergrundfarbe gesetzt, um die Farbe der Zeilen in orange zu wechseln, wenn mindestens ein Wert negativ ist:

```
<>_BgndColors{$i}:=0x00FFD0B0  // orange
<>_BgndColors{$i}:=-255  // Standardwert
```

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

Als nächstes wollen wir die Farbe der Zellen mit negativen Werten in dunkelorange setzen. Dazu setzen Sie ein Array Hintergrundfarbe für jede Spalte, zum Beispiel *<>_BgndColor_1*, *<>_BgndColor_2* und *<>_BgndColor_3*. Die Werte dieser Arrays haben Priorität vor den anderen, die in den Listbox Eigenschaften und im Array Hintergrundfarbe gesetzt wurden:

```
<>_BgndColorsCol_3{2}:=0x00FF8000  // dunkelorange
<>_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

Dasselbe Ergebnis erhalten Sie über die Befehle **LISTBOX SET ROW FONT STYLE** und **LISTBOX SET ROW COLOR**. Diese haben den Vorteil, dass Sie keine Arrays Stil/Farbe für die Spalten definieren müssen, denn die Befehle erstellen diese Arrays dynamisch.

Vererbung

Für jedes Attribut (Stil, Farbe und Hintergrundfarbe) ist mit dem Standardwert eine Vererbung implementiert:

- Für Attribute von Zellen: Attributwerte der Zellen
- Für Attribute von Zeilen: Attributwerte der Spalten
- Für Attribute von Spalten: Attributwerte der Listbox

Soll also ein Objekt den Attributwert einer höheren Ebene erben, können Sie im entsprechenden Befehl oder direkt im Element des Array Stil/Farbe die Konstante lk inherited (Standardwert) übergeben.

Wir nehmen eine Listbox mit einem standardmäßigen Schriftstil mit wechselnden Farben:

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

Führen Sie folgende Änderungen aus:

- Über die Eigenschaft **Zeilen Hintergrundfarbe Array** der Listbox den Hintergrund von Zeile 2 auf rot setzen
- Über die Eigenschaft **Zeilenstil Array** der Listbox den Stil von Zeile 4 in kursiv setzen
- Über die Eigenschaft **Zeilenstil Array** von Spalte 5 zwei Elemente in fett ändern
- Über die Eigenschaft **Zeilen Hintergrundfarbe Array** für Spalte 1 und 2 die beiden Elemente in dunkelblau ändern:

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

Um die ursprüngliche Darstellung der Listbox wiederherzustellen, können Sie folgendes ausführen:

- Die Konstante lk inherited in Element 2 des Array Hintergrundfarbe für Spalte 1 und 2 verwenden: Dann erben sie die rote Hintergrundfarbe der Zeile.
- Die Konstante lk inherited in den Elementen 3 und 4 des Array Stil für Spalte 5 anwenden: Dann erben sie den standardmäßigen Stil, außer für Element 4, es wechselt in kursiv, wie im Array Stil der Listbox angegeben.
- Die Konstante lk inherited in Element 4 des Array Stil für die Listbox anwenden, um den Stil kursiv zu entfernen.
- Die Konstante lk inherited in Element 2 des Array Hintergrundfarbe für die Listbox anwenden, um die ursprünglich wechselnde Farbe der Listbox wiederherzustellen.

Zeilenanzeige verwalten

In einer Listbox vom Typ Array können Sie für jede Zeile die Oberflächeneigenschaften "ausgeblendet", "deaktiviert" und "auswählbar" verwenden.

Diese Einstellung wird über die **Zeilenkontrolle Array** verwaltet, die sie über den Befehl **LISTBOX SET ARRAY** oder die Eigenschaftenliste setzen können:

Objekte	
Typ	List Box
Objektname	List Box
Variablenname	List Box
Datenquelle	Arrays
List Box	
Spaltenanzahl	3
Anzahl gesperrter Spalten	0
Anzahl statische Spalten	0
Zeilenkontrolle-Array	alControlArr
Auswahlmodus	Mehrfach

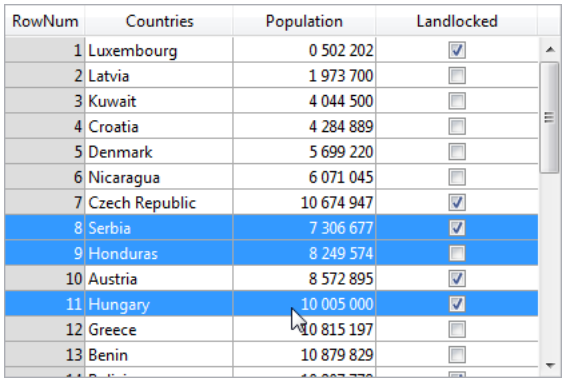
Zeilenkontrolle Array muss vom Typ Lange Ganzzahl sein und die gleiche Anzahl Zeilen haben wie die Listbox. Weitere Informationen dazu finden Sie im Abschnitt **Eigenschaften für Listboxen**.

Jedes Element von **Zeilenkontrolle Array** setzt den Status der Oberfläche für die entsprechende Zeile in der Listbox. Drei Oberflächeneigenschaften sind über Konstanten unter dem Thema "List Box" verfügbar:

Konstante	Typ	Wert	Kommentar
lk row is disabled	Lange Ganzzahl	2	Die entsprechende Zeile ist deaktiviert. Text und Kontrollkästchen sind gedimmt oder in Grauschrift. Eingebbare Textbereiche sind nicht mehr eingebbar. Standardwert: Aktiviert
lk row is hidden	Lange Ganzzahl	1	Die entsprechende Zeile ist ausgeblendet. Das betrifft nur die Darstellung der Listbox. Ausgeblendete Zeilen sind weiterhin in den Arrays vorhanden und lassen sich per Programmierung steuern. Die Befehle der Programmiersprache, insbesondere **LISTBOX Get number of rows** oder **LISTBOX GET CELL POSITION**, ignorieren den Status angezeigt/ausgeblendet von Zeilen. Zum Beispiel gibt in einer Listbox mit 10 Zeilen, von denen 9 ausgeblendet sind, der Befehl **LISTBOX Get number of rows** 10 zurück. Auf Benutzerseite ist nicht ersichtlich, ob es ausgeblendete Zeilen in der Listbox gibt. Nur sichtbare Zeilen lassen sich auswählen, z.B. über den Befehl *All*e auswählen. Standardwert: Sichtbar
lk row is not selectable	Lange Ganzzahl	4	Die entsprechende Zeile ist nicht auswählbar (sie lässt sich nicht markieren). Eingebbare Textbereiche sind nicht länger eingebbar, außer die Option "Einzelklick Editieren" ist aktiviert. Kontrollen wie Optionsfelder und PopUp-Menüs funktionieren jedoch weiterhin. Diese Einstellung wird ignoriert, wenn als Auswahlmodus für Listbox "Nichts" markiert ist. Standard: Auswählbar

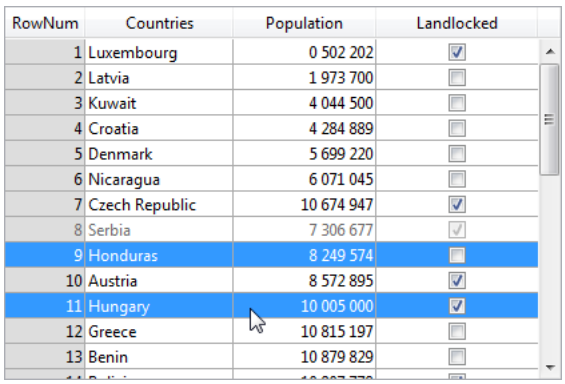
Um den Status für eine Zeile zu ändern, müssen Sie nur die passende Konstante bzw. Konstanten für das entsprechende Array Element setzen. Soll z.B. Zeile 10 nicht auswählbar sein, schreiben Sie folgenden Code:

```
aLControlArr{10}:=lk row is not selectable
```



Sie können auch mehrere Oberflächeneigenschaften gleichzeitig definieren:

```
aLControlArr{8}:=lk row is not selectable+lk row is disabled
```



Beachten Sie, dass Setzen eines Elements die anderen Optionen für dieses Element aufhebt. Zum Beispiel:

```
aLControlArr{6}:=lk row is disabled+lk row is not selectable  //setzt Zeile 6 auf deaktiviert UND nicht auswählbar
aLControlArr{6}:=lk row is disabled  //setzt Zeile 6 auf deaktiviert, aber wieder auswählbar
```

Darstellung von Auswahlen anpassen

Haben Sie für die Listbox die Option **Auswahlmarkierung ausblenden** markiert, müssen Sie die Auswahlen der Listbox über verfügbare Oberflächenoptionen sichtbar machen. Da Auswahlen weiter komplett von 4D verwaltet werden, bedeutet das folgendes:

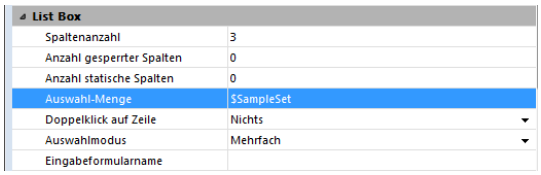
- Für Listboxen vom Typ Array müssen Sie die der Listbox zugeordnete Boolean Array Variable analysieren, um festzustellen, welche Zeilen ausgewählt bzw. nicht ausgewählt sind.
- Für Listboxen vom Typ Auswahl müssen Sie prüfen, ob der aktuelle Datensatz (Zeile) zur Menge gehört, die über die Eigenschaft **Menge markieren** der Listbox definiert ist.

Dann können Sie zur Darstellung der ausgewählten Zeilen spezifische Hintergrundfarben, Schriftfarben bzw. Schriftstile per Programmierung setzen, und zwar über Arrays oder Ausdrücke, je nach Art der angezeigten Listbox (siehe nachfolgende Abschnitte).

Hinweis: Mit der Konstante lk inherited können Sie die aktuelle Darstellung der Listbox gestalten (wie Schriftfarbe, Hintergrundfarbe, Schriftstil, etc.). Weitere Informationen dazu finden Sie im Abschnitt **Vererbung**.

Listboxen vom Typ Auswahl

Um festzulegen, welche Zeilen ausgewählt sind, müssen Sie prüfen, ob sie in der Menge enthalten sind, die über die Eigenschaft **Auswahl-Menge** der Listbox definert ist:



Dann können Sie die Darstellung der ausgewählten Zeilen über folgende Einträge in der Eigenschaftenliste definieren:

- **Ausdruck Hintergrundfarbe** (Gruppe Hintergrund und Rahmen)
- **Ausdruck Schriftfarbe** (Gruppe Text)
- **Ausdruck Stil** (Gruppe Text)

Hinweis: Beachten Sie, dass Ausdrücke automatisch neu bewertet werden, wenn:

- sich die Auswahl in der Listbox verändert
- die Listbox Fokus erhält oder verliert
- das Formularfenster mit der Listbox zum vordersten Fenster wird bzw. nicht mehr vorderstes Fenster ist

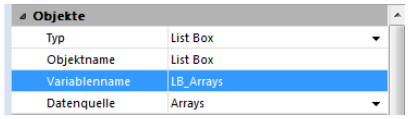
Einschränkung bei hierarchischen Listboxen

Ist die Option **Auswahlmarkierung ausblenden** markiert, lassen sich Umbruchzeilen nicht hervorheben.

Da für Kopfteile auf derselben Ebene keine unterschiedlichen Farben möglich sind, gibt es keine Möglichkeit, eine spezifische Umbruchzeile per Programmierung hervorzuheben.

Listboxen vom Typ Array

Sie müssen die der Listbox zugeordnete Boolean Array Variable analysieren, um festzustellen, ob Zeilen ausgewählt oder nicht ausgewählt sind:



Dann können Sie die Darstellung der ausgewählten Zeilen über ein oder mehrere Arrays definieren, die in der Eigenschaftenliste enthalten sind:

- **Zeilen Hintergrundfarbe Array** (Gruppe Hintergrund und Rahmen)
- **Zeilenschriftfarbe Array** (Gruppe Text)
- **Zeilenstil Array** (Gruppe Text)

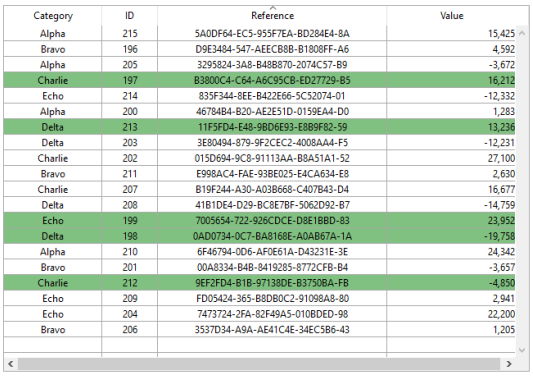
Beachten Sie, dass Listbox Arrays zum Definieren der Darstellung ausgewählter Zeilen während dem Formularereignis On Selection Change neu berechnet werden müssen; diese Arrays können Sie auch über folgende zusätzliche Formularereignisse verändern:

- On Getting Focus (Listboxeigenschaft)
- On Losing Focus (Listboxeigenschaft)
- On Activate (Formulareigenschaft)
- On Deactivate (Formulareigenschaft)

...je nachdem, ob und wie Sie Ändern des Fokus in Auswahlen visuell darstellen wollen.

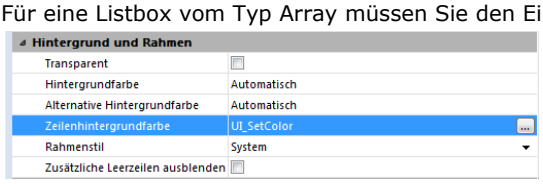
Beispiel

Die vom System vorgegebene Markierung ist ausgeblendet und die Auswahlen in der Listbox sollen mit grüner Hintergrundfarbe erscheinen. Hierzu ein Beispiel:



Category	ID	Reference	Value
Alpha	215	5A0DF64-EC5-955F7EA-BD284E4-8A	15,425
Bravo	196	D9E3484-547-AEECB8B-B1808FF-A6	4,592
Alpha	205	3295824-3A8-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

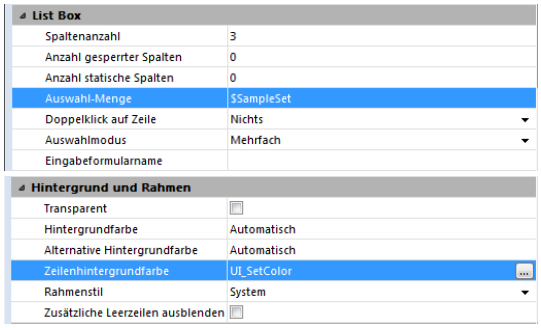
Für eine Listbox vom Typ Array müssen Sie den Eintrag **Zeilen Hintergrundfarbe Array** per Programmierung definieren:



In der Objektmethode der Listbox schreiben Sie:

```
Case of
   :(Form event=On Selection Change)
      $n:=Size of array(LB_Arrays)
      ARRAY LONGINT(_ListboxBackground;$n)  // Hintergrundfarbe für Zeile
      For($i;1;$n)
         If(LB_Arrays{$i}=True)  // ausgewählt
            _ListboxBackground{$i}:=0x0080C080  // grüner Hintergrund
         Else  // nicht ausgewählt
            _ListboxBackground{$i}:=lk inherited
         End if
      End for
End case
```

Für eine Listbox vom Typ Auswahl produzieren Sie denselben Effekt über eine Methode, die den Eintrag **Hintergrundfarbe** für die in **Auswahl-Menge** angegebene Menge aktualisiert. Hierzu ein Beispiel:



In der Methode ***UI_SetColor*** schreiben Sie:

```
If(Is in set("$SampleSet"))
   $color:=0x0080C080  // grüner Hintergrund
Else
   $color:=lk inherited
End if

$0:=$color
```

Ergebnis einer SQL-Anfrage in Listbox anzeigen

Sie können die Ergebnisse einer SQL-Anfrage direkt in ein Array vom Typ Listbox setzen. So können Sie sich rasch die Ergebnisse von SQL Anfragen ansehen. Es sind nur Suchläufe vom Typ **SELECT** möglich (siehe Handbuch *4D SQL Reference*). Dieser Mechanismus lässt sich nicht mit einer externen SQL Datenbank verwenden.

Dabei gilt folgendes:

- Sie erstellen die Listbox zum Empfangen des Suchergebnisses. Die Datenquelle der Listbox muss vom Typ **Array** sein.
- Führen Sie eine SQL-Suche vom Typ **SELECT** aus und weisen Sie das Ergebnis der mit der Listbox verknüpften Variablen zu. Sie können die 4D Tags **Begin SQL**/**End SQL** verwenden.
- Listbox-Spalten lassen sich vom Benutzer sortieren oder ändern.
- Bei erneuter Ausführung einer Anfrage mit **SELECT** werden die Spalten neu gefüllt. Es ist nicht möglich, dieselbe Listbox mit mehreren **SELECT** Anfragen nacheinander zu füllen.
- Es wird empfohlen, der Listbox die gleiche Anzahl Spalten zu geben wie im Ergebnis der SQL Anfrage sind. Gibt es weniger Listbox-Spalten als für die **SELECT** Anfrage erforderlich, werden automatisch Spalten hinzugefügt. Gibt es mehr Spalten als für die **SELECT** Anfrage erforderlich, werden die nicht benötigten Spalten automatisch ausgeblendet.
  Hinweis: Den automatisch hinzugefügten Spalten werden Variablen vom Typ **Dynamische Variablen** zugewiesen, d.h. sie bestehen solange wie das Formular. Eine dynamische Variable wird auch für jeden Kopfteil erstellt. Beim Aufrufen des 4D Befehls **LISTBOX GET ARRAYS** enthält der Parameter *arrColVars* Zeiger auf die dynamischen Arrays und der Parameter *arrHeaderVars* Zeiger auf die dynamischen Kopfteilvariablen. Ist die hinzugefügte Spalte z.B. die fünfte Spalte, lautet ihr Name *sql_column5* und ihr Kopfteil *sql_header5*.
- Im interpretierten Modus lassen sich vorhandene Arrays, welche die Listbox verwendet, automatisch entsprechend der von der SQL-Anfrage gesendeten Daten erneut tippen.

Beispiel:
Wir wollen nach allen Datenfeldern der Tabelle [PEOPLE] suchen und den Inhalt mit der Variablen *vlistbox* in die Listbox setzen. Für die Objektmethode einer Schaltfläche (als eine Möglichkeit) schreiben Sie:

```
Begin SQL
   SELECT * FROM PEOPLE INTO <<vlistbox>>
End SQL
```

Hierarchische Listboxen verwalten

In 4D können Sie hierarchische Listboxen definieren und verwenden. Das ist eine Listbox, in welcher der Inhalt der ersten Spalte in hierarchischer Form erscheint. Diese Art von Darstellung ermöglicht z.B. den Inhalt eines Verzeichnisses anzuzeigen, wie den Finder von Mac OS. Sie ist auch angepasst an die Präsentation von Information, die wiederholte bzw. hierarchisch gegliederte Werte enthält, z.B. Land/Bundesland/Stadt o.ä.

Nur **Listboxen vom Typ Array** können hierarchisch sein.

Hierarchische Listboxen sind eine besondere Darstellungsart von Daten. Sie verändern jedoch nicht die Struktur dieser Daten (die Arrays). Hierarchische Listboxen werden exakt wie normale Listboxen verwaltet. Weitere Informationen dazu finden Sie im Abschnitt **Einführung in Listboxen**.

Um eine hierarchische Listbox zu definieren, gibt es folgende Möglichkeiten:

- Hierarchische Elemente manuell mit der Eigenschaftenliste des Formulareditors konfigurieren. Weitere Informationen dazu finden Sie im Abschnitt **Hierarchische PopUp-Menüs und hierarchische Listen** des Handbuchs *4D Designmodus*.
- MIt den Befehlen **LISTBOX SET HIERARCHY** und **LISTBOX GET HIERARCHY**.

Wird ein Formular mit hierarchischer Listbox zum ersten Mal geöffnet, werden standardmäßig alle Zeilen aufgeklappt.
Ein Umbruch und ein hierarchischer Knotenpunkt werden automatisch in der Listbox hinzugefügt, wenn Werte in den Arrays wiederholt werden. Nehmen wir z.B. eine Listbox mit vier Arrays mit den Angaben Countries, Region, Cities und Population:

Countries	Regions	Cities	Population
Deutschland	Bayern	München	1.326.000
Deutschland	Bayern	Freising	45.000
Deutschland	Bayern	Rosenheim	60.000
Deutschland	Hessen	Frankfurt	675.000
Deutschland	Hessen	Darmstadt	138.000

Wird diese Listbox in hierarchischer Form angezeigt (die drei ersten Arrays sind in der Hierarchie enthalten), erhalten Sie folgendes:

Countries	Population
Deutschland	
Bayern	
München	1.326.000
Freising	45.000
Rosenheim	60.000
Hessen	
Frankfurt	675.000
Darmstadt	138.000
Frankreich	
Normandie	
Caen	220.000
Deauville	4.000

Die Arrays werden vor dem Aufbau der Hierarchie nicht sortiert. Enthält ein Array z.B. die Daten AAABBAACC, lautet die Hierarchie:

> A
> B
> A
> C

Um einen hierarchischen Knotenpunkt auf- oder zuzuklappen, klicken Sie darauf. Klicken Sie mit der Tastenkombination **Alt+Klick** (Windows) oder **Wahltaste+Klick** (Mac OS) auf den Knoten, werden alle Unterelemente auf- oder zugeklappt. Diese Operationen lassen sich auch per Programmierung über die Befehle **LISTBOX EXPAND** und **LISTBOX COLLAPSE** ausführen.

Auswahlen und Positionen verwalten

Eine hierarchische Listbox zeigt je nach Status eine variable Anzahl Zeilen auf dem Bildschirm, je nachdem ob die hierarchischen Knotenpunkte auf- oder zugeklappt sind. Das bedeutet jedoch nicht, dass die Anzahl der Zeilen variiert. Nur die Anzeige verändert sich, nicht die Daten ansich.

Es ist wichtig, dieses Prinzip zu verstehen, weil die programmierte Verwaltung hierarchischer Listboxen immer auf den Daten der Arrays basiert und nicht auf den angezeigten Daten. Speziell die automatisch hinzugefügten Umbruchzeilen werden im Array mit den Anzeigeoptionen nicht berücksichtigt. Weitere Informationen dazu finden Sie im späteren Abschnitt “Umbruchzeilen verwalten”.

Nehmen wir z.B. folgende Arrays:

Deutschland	Bayern	München
Deutschland	Bayern	Freising
Deutschland	Bayern	Rosenheim

Beim Anzeigen in hierarchischer Form erscheint der Eintrag "Freising" nicht in der zweiten, sonderen in der vierten Zeile, weil zwei Umbruchzeilen hinzugefügt werden:

Deutschland
Bayern
München
Freising
Rosenheim

Unabhängig von der Darstellung der Daten in der Listbox (hierarchisch oder nicht), wollen Sie die Zeile mit "Freising" in Fettschrift setzen, müssen Sie die Anweisung Stil{2} = bold schreiben. Denn es wird nur die Position der Zeile in den Arrays berücksichtigt.

Dieses Prinzip gilt für interne Arrays. Sie können damit folgende Elemente verwalten:

- Farben
- Hintergrundfarben
- Stilelemente
- Ausgeblendete Zeilen
- Auswahlen

Um z.B. die Zeile mit Rosenheim auszuwählen, müssen Sie folgendes übergeben:

```
->MyListbox{3}:=True
```

Nicht-hierarchische Darstellung:

Deutschland	Bayern	München
Deutschland	Bayern	Freising
Deutschland	Bayern	Rosenheim

Hierarchische Darstellung:

Deutschland
Bayern
München
Freising
Rosenheim

Hinweis: Sind eine oder mehrere Zeilen ausgeblendet, weil das übergeordnete Element zugeklappt ist, sind sie nicht mehr ausgewählt. Nur sichtbare Zeilen lassen sich auswählen, sei es direkt oder durch Scrollen. Mit anderen Worten, Zeilen können nicht gleichzeitig ausgeblendet und ausgewählt sein.

Analog zu Auswahlen gibt der Befehl **LISTBOX GET CELL POSITION** für eine hierarchische bzw. eine nicht-hierarchische Listbox denselben Wert zurück. Das bedeutet, dass dieser Befehl in beiden nachfolgenden Beispielen dieselbe Position zurückgibt, nämlich: (3;2)

Nicht-hierarchische Darstellung:

Deutschland	Bayern	München	1.326.000
Deutschland	Bayern	Freising	45.000
Deutschland	Bayern	Rosenheim	60.000
Deutschland	Hessen	Frankfurt	675.000

Hierarchische Darstellung:

Deutschland	
Bayern	
München	1.326.000
Freising	45.000
Rosenheim	60.000
Hessen	
Frankfurt	675.000

Umbruchzeilen verwalten

Wählt der Benutzer eine Umbruchzeile, gibt **LISTBOX GET CELL POSITION** das erste Auftreten der Zeile im entsprechenden Array an. Für das folgende Array:

Deutschland	
Bayern	
München	1.326.000
Freising	45.000
Rosenheim	60.000
Hessen	
Frankfurt	675.000

gibt **LISTBOX GET CELL POSITION** die Position (2;4) zurück. Um eine Umbruchzeile per Programmierung auszuwählen, müssen Sie den Befehl **LISTBOX SELECT BREAK** verwenden.

Umbruchzeilen werden in internen Arrays zur Verwaltung der grafischen Darstellung von Listboxen (Stilarten und Farben) nicht berücksichtigt. Sie können die Merkmale für Umbruchzeilen jedoch über die Befehle zur grafischen Verwaltung von Objekten (im Kapitel **Objekte (Formulare)**) verändern. Dazu müssen Sie nur die entsprechenden Befehle für die Arrays ausführen, welche die Hierarchie enthalten.

Nehmen wir z.B. nachfolgende Listbox (die Namen der zugewiesenen Arrays sind in Klammern angegeben):

Nicht-hierarchische Darstellung:

(T1)	(T2)	(T3)	(T4)	(tStyle)	(tColor)
Deutschland	Bayern	München	1.326.000	Normal	0
Deutschland	Bayern	Freising	45.000	Underline	0
Deutschland	Bayern	Rosenheim	60.000	Normal	0xFF0000
Deutschland	Hessen	Frankfurt	675.000	Normal	0
Deutschland	Hessen	Darmstadt	138.000	Normal	0

Hierarchische Darstellung:

Deutschland	
Bayern	
München	1.326.000
Freising	45.000
Rosenheim	60.000
Hessen	
Frankfurt	675.000
Darmstadt	138.000

Im hierarchischen Modus werden Umbruchzeilen nicht von den Arrays für Stiländerungen mit Namen tStyle und tColor berücksichtigt. Um Farbe oder Stil von Umbruchebenen zu ändern, müssen Sie folgende Anweisung ausführen:

```
OBJECT SET RGB COLORS(T1;0x0000FF;0xB0B0B0)
OBJECT SET FONT STYLE(T2;Bold)
```

Hinweis: In diesem Kontext kann nur die Syntax, welche die Array Variable enthält, mit den Befehlen für Objekteigenschaften funktionieren, da die Arrays keine zugewiesenen Objekte haben.

Ergebnis:

Deutschland	
Bayern	
München	1.326.000
Freising	45.000
Rosenheim	60.000
Hessen	
Frankfurt	675.000
Darmstadt	138.000

Ausgeblendete Zeilen

Sind alle Zeilen einer Unterhierarchie ausgeblendet, wird die Umbruchzeile ebenfalls ausgeblendet. Werden in Beispiel Deutschland die Zeilen 1 bis 3 ausgeblendet, erscheint die Umbruchzeile "Bayern" nicht.

Optimiertes Verwalten von Auf- und Zuklappen

Sie können Anzeigen und Arbeiten mit hierarchischen Listboxen mit den beiden Formularereignissen On Expand und On Collapse optimieren.

Eine hierarchische Listbox wird mit dem Inhalt seiner Arrays aufgebaut, es kann also nur angezeigt werden, wenn all diese Arrays in den Speicher geladen sind. Das erschwert den Aufbau umfangreicher hierarchischer Listboxen über Arrays, die aus Daten generiert werden (mit dem Befehl **SELECTION TO ARRAY**), nicht nur in Bezug auf die Anzeigegeschwindigkeit, sondern auch in Bezug auf den verwendeten Speicher.

Diese Einschränkungen können Sie über die Formularereignisse On Expand und On Collapse aufheben. Sie können z.B. nur einen Teil der Hierarchie anzeigen und die Arrays gemäß Benutzeraktionen unmittelbar laden/entladen.

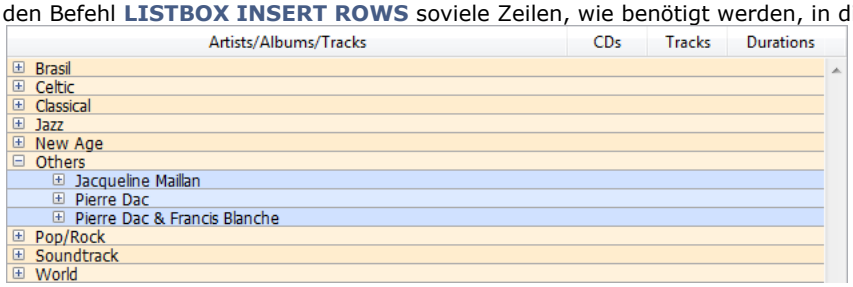
Im Kontext dieser Ereignisse gibt der Befehl **LISTBOX GET CELL POSITION** die Zelle zurück, in die der Benutzer geklickt hat, um eine Zeile auf- oder zuzuklappen.

In diesem Fall müssen Sie Arrays über den Code füllen und leeren. Die Funktionsweise ist wie folgt:

- Wird die Listbox angezeigt, muss nur das erste Array gefüllt werden. Sie müssen jedoch ein zweites Array mit leeren Werten erstellen, damit die Listbox die Icons zum Auf- und Zuklappen anzeigt:

- Klickt ein Benutzer auf das Icon Aufklappen, kann das Ereignis On Expand operieren. Der Befehl **LISTBOX GET CELL POSITION** gibt die entsprechende Zelle zurück und lässt Sie die passende Hierarchie aufbauen: Sie füllen das erste Array mit den wiederholten Werten, das zweite mit den vom Befehl **SELECTION TO ARRAY** gesendeten Werten und fügen über

den Befehl **LISTBOX INSERT ROWS** soviele Zeilen, wie benötigt werden, in die Listbox ein:

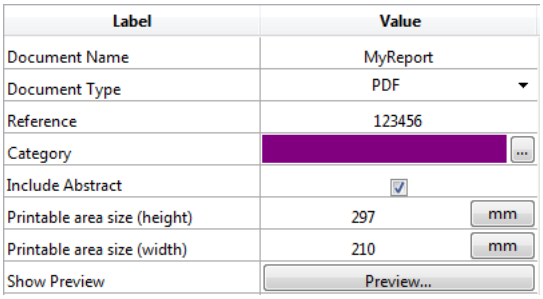


- Klickt ein Benutzer auf das Icon Zuklappen, kann das Ereignis On Collapse operieren. Der Befehl **LISTBOX GET CELL POSITION** gibt die entsprechende Zelle zurück: Sie entfernen über den Befehl **LISTBOX DELETE ROWS** die entsprechenden Zeilen aus der Listbox.

Objekt Arrays in Spalten von Listboxen (4D View Pro)

Ab 4D v15 können Spalten von Listboxen Arrays vom Typ Objekt verwalten. Da diese Arrays unterschiedliche Datentypen erlauben, ermöglicht dieses leistungsstarke Feature, jetzt in den Zeilen einer Spalte verschiedene Eingabetypen zu mischen und auch unterschiedliche Widgets anzuzeigen. Sie können z.B. in der ersten Zeile eine Texteingabe, in der zweiten ein Optionsfeld und in der dritten ein DropDown-Menü setzen. Objekt Arrays ermöglichen auch Zugriff auf neue Widget-Typen, wie Schaltflächen oder Farbpaletten.

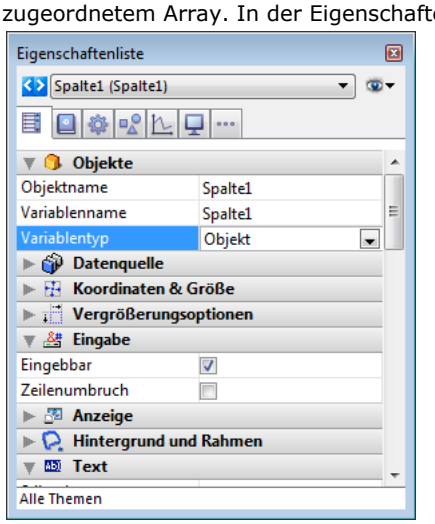
Die folgende Listbox wurde mit einem Objekt Array gestaltet:



Hinweis zur Lizensierung: Der Einsatz von Objekt Arrays in Listboxen ist der erste Schritt für das geplante Tool "4D View Pro", das dann nach und nach das Plug-In 4D View ersetzen wird. Deshalb benötigen Sie zur Nutzung dieser Funktionalität eine gültige 4D View Lizenz. Weitere Informationen dazu finden Sie auf der 4D Web Site.

Objekt Array in Spalte einrichten

Um ein Objekt Array einer Spalte der Listbox zuzuordnen, setzen Sie den Namen des Objekt Array entweder in der Eigenschaftenliste im Feld "Variablenname" oder im Befehl **LISTBOX INSERT COLUMN**, genauso wie für eine Spalte mit zugeordnetem Array. In der Eigenschaftenliste können Sie jetzt für die Spalte als Variablentyp **Objekt** auswählen:



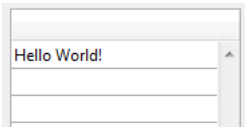
Für Spalten vom Typ Objekt sind Standardeigenschaften zu Koordinaten, Größe und Stil verfügbar. Sie können über die Eigenschaftenliste oder per Programmierung Stil, Schriftfarbe und Ein-/Ausblenden für jede Zeile einer Spalte vom Typ Objekt definieren. Solche Spalten lassen sich auch ausblenden.

Für objektbasierte Spalten der Listbox ist die Gruppe **Datenquelle** nicht verfügbar. Der Inhalt jeder Zelle der Spalte richtet sich nach den Attributen, die dem jeweiligen Element des Objekt Array zugewiesen ist. Jedes Objekt Element kann folgendes definieren:

- Wertetyp (*zwingend*): Text, Farbe, Ereignis...
- Eigentlicher Wert (*optional*): wird für Eingabe/Ausgabe verwendet
- Anzeige des Zelleninhalts (*optional*): Schaltfläche, Liste...
- Zusätzliche Einstellungen (*optional*): richtet sich nach dem Wertetyp

Um diese Eigenschaften zu definieren, müssen Sie im Objekt passende Attribute setzen (Liste siehe unten). Sie können z.B. "Hello World!" in einer Spalte vom Typ Objekt mit folgendem Code schreiben :

```
ARRAY OBJECT(obColumn;0) //Spalten Array
C_OBJECT($ob) //erstes Element
OB SET($ob;"valueType";"text") //definiert den Wertetyp (zwingend)
OB SET($ob;"value";"Hello World!") //definiert den Wert
APPEND TO ARRAY(obColumn;$ob)
```



Hinweis: Für eine Spalte vom Typ Objekt lassen sich keine Anzeigeformate und Eingabefilter setzen. Diese richten sich automatisch nach dem Typ des Werts.

Wertetyp und Anzeige der Daten

Wird eine Spalte der Listbox einem Objekt Array zugewiesen, basiert die Anzeige, Eingabe oder Bearbeitung der Zelle auf dem Attribut "valueType" des Array Elements. Es gibt folgende Werte:

- "text" für einen Wert vom Typ Text
- "real" für eine Zahl inkl. Trennern wie <Abstand>, <.>, oder <,>
- "integer" für einen Wert Ganzzahl
- "boolean" für einen Wert wahr/falsch
- "color" zum Definieren einer Hintergrundfarbe
- "event" zum Anzeigen einer Schaltfläche mit Bezeichnung

4D verwendet Standard Widgets für den Wertetyp, z.B. "text" wird als Widget für Texteingabe angezeigt, "boolean" als Optionsfeld. Über Optionen sind auch Alternativen verfügbar. So lässt sich ein Boolean Wert auch als DropDown-Liste mit den Optionen "ja"/"nein" darstellen. Nachfolgende Tabelle zeigt die verschiedenen Möglichkeiten:

Wertetyp	Standard Widget	Alternative Widget(s)
text	Texteingabe	Dropdown-Menü (Erforderlich-Liste) oder Combobox (Auswahlliste)
real	vorgegebene Eingabe (Zahlen und Trenner)	Dropdown-Menü (Erforderlich-Liste) oder Combobox (Auswahlliste)
integer	vorgegebene Eingabe (nur Zahlen)	Dropdown-Menü (Erforderlich-Liste), Combobox (Auswahlliste) oder Optionsfeld mit drei Stadien
boolean	Optionsfeld	Dropdown-Menü (Erforderlich-Liste)
color	Hintergrundfarbe	Text
event	Schaltfläche mit Bezeichnung	Alle Widgets können eine zusätzliche Schaltfläche zum Wählen der Einheit (**unit toggle button)** oder für weitere Optionen (...) **(ellipsis button)** haben, die der Zelle zugeordnet ist

Sie können die Zellenanzeige und Optionen über spezifische Attribute in jedem Objekt anpassen (siehe unten).

Anzeigeformate und Eingabefilter

Für Spalten vom Typ Objekt können Sie keine Anzeigeformate oder Eingabefilter setzen. Sie werden automatisch gemäß dem Typ des Werts definiert. Es gibt folgende Optionen:

WerteTyp	Standardformat	Eingabekontrolle
text	so wie in Objekt definiert	beliebig (keine Kontrolle)
real	so wie in Objekt definiert (mit Dezimaltrenner des Systems)	"0-9" und "." und "-" "0-9" und "." wenn min>=0
integer	so wie in Objekt definiert	"0-9" und "-" "0-9" wenn min>=0
boolean	Optionsfeld	-
color	-	-
event	-	-

Attribute

Jedes Element des Objekt Array ist ein Objekt mit einem oder mehreren Attributen, die den Inhalt der Zelle und die Anzeige der Daten definieren (siehe Beispiel oben).

Nur das Attribut "valueType" ist zwingend, unterstützt werden "text", "real", "integer", "boolean", "color" und "event". Nachfolgende Tabelle zeigt alle Attribute, die abhängig vom Typ, in Objekt Arrays in Listboxen unterstützt werden (alle anderen Attribute werden ignoriert). Details zu den Anzeigeformaten und Beispiele siehe unten.

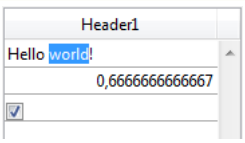
	valueType	text	real	integer	boolean	color	event
Attribut	***Beschreibung***						
value	Zellenwert (Eingabe oder Ausgabe)	x	x	x	x	x	
min	Mindestwert		x	x			
max	Maximumwert		x	x			
behavior	"DreiStadien" Wert			x			
requiredList	Im Objekt definierte Dropdown-Liste	x	x	x			
choiceList	In Objekt definierte Combobox	x	x	x			
requiredListReference	4D Listenref, richtet sich nach "SichernAls" Wert	x	x	x			
requiredListName	4D Listenname, richtet sich nach "SichernAls" Wert	x	x	x			
saveAs	"Referenz" oder "Wert"	x	x	x			
choiceListReference	4D Listenref, Anzeige Combobox	x	x	x			
choiceListName	4D Listenname, Anzeige Combobox	x	x	x			
unitList	Array mit n Elementen	x	x	x			
unitReference	Index des gewählten Elements	x	x	x			
unitsListReference	4D Listenref für Einheiten	x	x	x			
unitsListName	4D Listenname für Einheiten	x	x	x			
alternateButton	Wechsel-Schaltfläche hinzufügen	x	x	x	x	x	

Wert

Zellenwerte werden im Attribut "value" gespeichert. Es wird für Eingabe und Ausgabe verwendet.

Beispiel:

```
ARRAY OBJECT(obColumn;0) //Spalten Array
C_OBJECT($ob1)
$entry:="Hello world!"
OB SET($ob1;"valueType";"text")
OB SET($ob1;"value";$entry)
C_OBJECT($ob2)
OB SET($ob2;"valueType";"integer")
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)
```



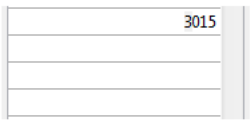
min und max

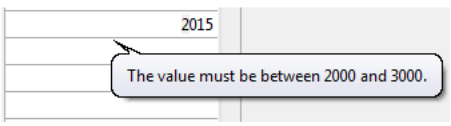
Ist "valueType" vom Typ "real" oder "integer", erlaubt das Objekt auch **min** und **max** Attribute mit passenden Werten (die Werte müssen vom gleichen Typ wie der Wertetyp sein).

Damit können Sie den Bereich von Eingabewerten steuern, wenn die Zelle bestätigt wurde, d.h. wenn sie den Fokus verliert. Ist der Eingabewert kleiner als der Mindestwert **(min)** bzw. größer als der Maximumwert **(max)**, wird er abgewiesen: Der vorige Wert bleibt bestehen und es erscheint eine entsprechende Meldung.

Beispiel:

```
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)
```





Verhalten

Das Attribut"**behavior**" bietet Variationen zur regulären Darstellung von Werten. In 4D v15 ist folgende Variante verfügbar:

Attribut	Verfügbare Werte	WerteTyp(en)	Beschreibung
behavior	threeStates	Ganzzahl	Zeigt einen numerischen Wert als Optionsfeld mit drei Stadien. 2=halb-markiert, 1=markiert, 0=nicht markiert, -1=unsichtbar, -2=nicht markiert ist deaktiviert, -3=markiert ist deaktiviert, -4=halb-markiert ist deaktiviert

Beispiel:

```
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")
```

Auswahlliste und Erforderlich-Liste

Ist im Objekt ein Attribut "choicelist" oder "requiredList" vorhanden, wird die Texteingabe je nach Typ durch eine DropDown-Liste oder eine Combobox ersetzt:

- Beim Attribut "choicelist" wird die Zelle als Combobox angezeigt, d.h. der Benutzer kann einen Wert auswählen oder eingeben.
- Beim Attribut "requiredList" wird die Zelle als DropDown-Liste angezeigt, d.h. der Benutzer kann nur einen der vorgegebenen Werte auswählen.

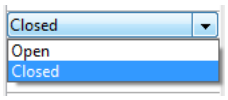
In beiden Fällen wird über das Attribut "value" ein Wert im Widget vorab ausgewählt.

Hinweis: Die Werte des Widget werden über ein Array definiert. Wollen Sie dem Widget eine vorhandene 4D Liste zuweisen, müssen Sie die Attribute "requiredListReference", "requiredListName", "choiceListReference", oder "choiceListName" verwenden.

Beispiele:

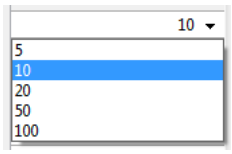
- Eine DropDown-Liste mit den beiden Optionen "Open" oder "Closed" anlegen, wobei "Closed" vorab ausgewählt ist:

```
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)
```



- Jeden Wert vom Typ Ganzzahl zulassen, jedoch eine Combobox mit den am häufigsten verwendeten Werten anzeigen:

```
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($ob;"value";10) //10 als Standardwert
OB SET ARRAY($ob;"choiceList";$ChoiceList)
```



Erforderlich-Liste als Name oder Referenz

Über die Attribute "requiredListName" und "requiredListReference" können Sie in einer Zelle der Listbox eine in 4D definierte Liste anzeigen, die entweder über die Toolbox im Designmodus oder über den Befehl **New list** erstellt wurde. Die Zelle erscheint dann als DropDown-Liste, d.h. der Benutzer kann nur einen der vorgegebenen Werte auswählen.

Je nachdem, wo die Liste angelegt wird, wählen Sie "requiredListName" oder "requiredListReference": Stammt sie von der Toolbox, übergeben Sie einen Namen, wurde Sie per Programmierung erstellt, übergeben Sie eine Referenz. In beiden Fällen lässt sich über das Attribut "value" im Widget ein Wert vorab auswählen.

Hinweis: Wollen Sie die Werte über ein einfaches Array definieren, müssen Sie das Attribut "requiredList" verwenden.

In diesem Fall definiert das Attribut "saveAs", ob der gewählte Eintrag als "value" oder "reference" gespeichert wird.

- Ist "saveAs" = "reference", wird es als Referenz gesichert. "valueType" muss vom Typ Zahl oder Ganzzahl sein.
- Ist "saveAs" = "value", wird der Wert gesichert. "valueType" muss vom gleichen Typ wie die Werte der Liste sein, in der Regel "text" oder "integer", andernfalls versucht 4D, den Wert der Liste in das Objekt "valueType" zu konvertieren (siehe Beispiele unten).

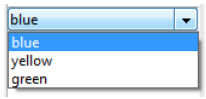
Weitere Informationen dazu finden Sie im Abschnitt **Als Wert oder Referenz sichern** des Handbuchs *Designmodus*.

Hinweis: Enthält die Liste Texteinträge mit Zahlenwerten, muss der Dezimaltrenner ein Punkt (".") sein, unabhängig von den lokalen Einstellungen. Beispiel: "17.6" "1234.456".

Beispiele:

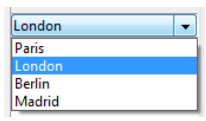
- Eine DropDown Liste mit der Liste "colors" anzeigen, die in der Toolbox definiert wurde (mit den Werten "blue", "yellow" und "green"), als Wert sichern und standardmäßig "blue" anzeigen:


```
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"saveAs";"value")
OB SET($ob;"value";"blue")
OB SET($ob;"requiredListName";"Farben")
```



- Eine DropDown Liste mit einer per Programmierung erstellten Liste anzeigen, als Referenz sichern und standardmäßig London anzeigen:

```
<>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";"Ganzzahl")
OB SET($ob;"saveAs";"reference")
OB SET($ob;"value";2)  //zeigt standardmäßig London an
OB SET($ob;"requiredListReference";<>List)
```



Auswahlliste als Name oder Referenz

Über die Attribute "choiceListName" und "choiceListReference" können Sie in einer Zelle der Listbox eine in 4D definierte Liste anzeigen, die entweder über die Toolbox im Designmodus oder über den Befehl **New list** erstellt wurde. Die Zelle erscheint dann als DropDown-Liste, d.h. der Benutzer kann einen Wert auswählen oder eintippen.

Je nachdem, wo die Liste angelegt wird, wählen Sie "choiceListName" oder "choiceListReference": Stammt sie von der Toolbox, übergeben Sie einen Namen, wurde Sie per Programmierung erstellt, übergeben Sie eine Referenz. In beiden Fällen lässt sich über das Attribut "value" im Widget ein Wert vorab auswählen.

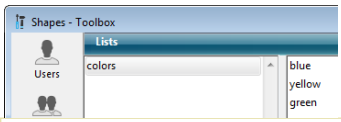
Hinweis: Wollen Sie die Werte über ein einfaches Array definieren, müssen Sie das Attribut "choiceList" verwenden.

In diesem Fall lässt sich das Attribut "saveAs" nicht verwenden. Ausgewählte Einträge werden automatisch als "value" gesichert.

Hinweis: Enthält die Liste Texteinträge mit Zahlenwerten, muss der Dezimaltrenner ein Punkt (".") sein, unabhängig von den lokalen Einstellungen. Beispiel: "17.6" "1234.456".

Beispiel:

Eine Combobox mit der Liste "colors" anzeigen, die in der Toolbox definiert wurde (mit den Werten "blue", "yellow" und "green"), "blue" standardmäßig auswählen:




```
C_OBJECT($ob)
OB SET($ob;"valueType";"text")
OB SET($ob;"value";"blue")
OB SET($ob;"choiceListName";"colors")
```

Listen für Einheiten

Sie können spezifische Attribute für Einheiten verwenden, die Zellenwerten zugeordnet sind, wie z.B. "10 cm", "20 Pixel". Dafür können Sie folgende Attribute verwenden:

- "unitsList": Ein Array mit *n* Elementen zum Definieren der verfügbaren Einheiten, wie "cm", "inches", "km", "miles", etc. Damit definieren Sie Einheiten innerhalb des Objekts.
- "unitsListReference": Referenz auf eine 4D Liste mit den verfügbaren Einheiten. Damit definieren Sie Einheiten über eine 4D Liste, die mit dem Befehl **New list** erstellt wurde.
- "unitsListName": Name einer im Designmodus angelegten 4D Liste mit den verfügbaren Einheiten. Damit definieren Sie Einheiten über eine 4D Liste, die in der Toolbox angelegt wurde.

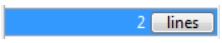
Jede dieser Einheitenliste lässt sich über folgendes Attribut zuordnen:

- "unitsReference": Ein einzelner Wert mit dem Index (von 1 zu *n*) des gewählten Eintrags in der Werteliste "unitsList", "unitsListReference" oder "unitsListName".

Die aktuelle Einheit wird als Schaltfläche angezeigt, die bei jedem Anklicken die Werte von "unitsList", "unitsListReference" oder "unitsListName" durchläuft ("Pixel" -> "Zeilen" -> "cm" -> "Pixel" -> etc.)

Beispiel: Die numerische Eingabe mit den beiden Einheitenvarianten "Zeilen" oder "Pixel" einrichten. Der aktuelle Wert ist "2" + "Zeilen". Wir verwenden Werte, die direkt im Objekt definiert wurden (Attribut "unitsList"):

```
ARRAY TEXT($_units;0)
APPEND TO ARRAY($_units;"Zeilen")
APPEND TO ARRAY($_units;"Pixel")
C_OBJECT($ob)
OB SET($ob;"valueType";"integer")
OB SET($ob;"value";2) // 2 "units"
OB SET($ob;"unitsReference";1) //"Zeilen"
OB SET ARRAY($ob;"unitsList";$_units)
```



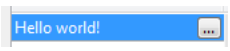
alternateButton

Um in einer Zelle die Schaltfläche [**...**] hinzuzufügen, müssen Sie im Objekt "alternateButton" mit dem Wert **True** übergeben. Die Schaltfläche erscheint automatisch in der Zelle.

Klickt der Benutzer auf diese Schaltfläche, wird das Ereignis **On Alternate Click** generiert, das Sie nach eigenen Wünschen verwalten können. Weitere Informationen dazu finden Sie im Abschnitt "Ereignisverwaltung".

Beispiel:

```
C_OBJECT($ob1)
$entry:="Hello world!"
OB SET($ob;"valueType";"text")
OB SET($ob;"alternateButton";True)
OB SET($ob;"value";$entry)
```



valueType color

Über *valueType* "color" können Sie entweder eine Farbe oder einen Text anzeigen.

- Ist der Wert eine Zahl, wird die Zelle farbig ausgefüllt. Beispiel:

```
C_OBJECT($ob4)
OB SET($ob4;"valueType";"color")
OB SET($ob4;"value";0x00FF0000)
```

  Wird die Zelle angeklickt, öffnet sich die Farbpalette des Systems mit der aktuell gewählten Farbe der Zelle. Der Benutzer kann eine andere Farbe wählen.
- Ist der Wert ein Text, wird der Text angezeigt, z.B. "value";"Automatic".

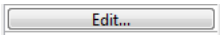
valueType event

valueType "event" zeigt eine einfache Schaltfläche, die bei Anklicken das Ereignis On Clicked generiert. Es lassen sich keine Daten oder Werte übergeben bzw. zurückgeben.

Optional können Sie ein Attribut "label" übergeben.

Beispiel:

```
C_OBJECT($ob)
OB SET($ob;"valueType";"event")
OB SET($ob;"label";"Edit...")
```



Ereignisverwaltung

In Listboxen mit einem Objekt Array lassen sich verschiedene Ereignisse verwalten:

- **On Data Change**: Ein Ereignis On Data Change wird ausgelöst, wenn ein Wert geändert wird in:
  - einem Bereich für Texteingabe
  - einer DropDown Liste
  - einem Bereich Combobox
  - einer Schaltfläche für Einheit (Wechsel von Wert n zu Wert n+1)
  - einem Optionsfeld (Wechsel markiert/nicht markiert)
- **On Clicked**: Klickt der Benutzer auf eine Schaltfläche, die über das Attribut *valueType* "event" installiert wurde, wird ein Ereignis On Clicked generiert. Dieses Ereignis verwaltet der Entwickler.
- **On Alternative Click**: Klickt der Benutzer auf eine Schaltfläche [...] (Attribut "alternateButton"), wird ein Ereignis On Alternative Click generiert. Dieses Ereignis verwaltet der Entwickler.

Hinweis: On Alternative Click ist der neue Name für das Ereignis **On Arrow Click**, das in bisherigen 4D Versionen schon verfügbar war. Es wurde umbenannt, da seine Reichweite in 4D v15 erweitert wurde.

LISTBOX COLLAPSE

LISTBOX COLLAPSE ({* ;} Objekt {; rekursiv {; Selector {; Linie {; Spalte}}}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
rekursiv	Boolean	➡	Wahr = Unterebenen zuklappen, falsch = Unterebenen nicht zuklappen
Selector	Lange Ganzzahl	➡	Teil der Listbox, die zugeklappt werden soll
Linie	Lange Ganzzahl	➡	Nummer der Umbruchzeile bzw. Listboxebene, die zugeklappt werden soll
Spalte	Lange Ganzzahl	➡	Nummer der Umbruchspalte, die zugeklappt werden soll

Beschreibung

Der Befehl **LISTBOX COLLAPSE** klappt die Umbruchzeilen der Listbox zu, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Wurde die Listbox nicht im hierarchischen Modus angelegt, führt der Befehl nichts aus. Weitere Informationen dazu finden Sie im Abschnitt **Hierarchische Listboxen verwalten**.

Mit dem optionalen Parameter *rekursiv* konfigurieren Sie das Zuklappen der hierarchischen Unterebenen der Listbox. Übergeben Sie *Wahr* oder lassen Sie diesen Parameter weg, um alle Ebenen und Unterebenen zuzuklappen. Übergeben Sie *Falsch*, wird nur die erste Ebene zugeklappt.

Mit dem optionalen Parameter *Selector* geben Sie die Reichweite des Befehls an. Sie können eine der folgenden Konstanten unter dem Thema **Listbox** übergeben:

Konstante	Typ	Wert	Kommentar
lk all	Lange Ganzzahl	0	Der Befehl betrifft alle Unterebenen (Standardwert, wird ohne diesen Parameter verwendet)
lk selection	Lange Ganzzahl	1	Befehl gilt für ausgewählte Unterebenen.
lk break row	Lange Ganzzahl	2	Der Befehl gilt für die Unterebene, zu der die Zelle, definiert mit den Parametern *Zeile* und *Spalte*, gehört. Beachten Sie, dass diese Parameter die Zeilen- und Spaltennummern in der Listbox im Standardmodus darstellen und nicht in der hierarchischen Form. In diesem Fall führt der Befehl nichts aus, wenn die Parameter *Zeile* und *Spalte* weggelassen werden.
lk level	Lange Ganzzahl	3	Dieser Befehl betrifft alle Umbruchzeilen für die Spalte *Ebene*. Dieser Parameter bezeichnet eine Spaltennummer in der Listbox im Standardmodus und nicht in seiner hierarchischen Darstellung. In diesem Fall führt der Befehl nichts aus, wenn der Parameter *Ebene* weggelassen wird.

Enthält die Auswahl oder Listbox keine Umbruchzeile oder sind alle Umbruchzeilen bereits zugeklappt, führt der Befehl nichts aus.

Beispiel

Dieses Beispiel klappt die erste Ebene der Umbruchzeilen der Auswahl in der Listbox zu:

```
LISTBOX COLLAPSE(*;"MeineListbox";False;lk selection)
```

LISTBOX DELETE COLUMN

LISTBOX DELETE COLUMN ({* ;} Objekt ; SpaltePos {; Anzahl})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
SpaltePos	Lange Ganzzahl	➡	Zu entfernende Spaltennummer
Anzahl	Lange Ganzzahl	➡	Anzahl der zu entfernenden Spalten

Beschreibung

Der Befehl **LISTBOX DELETE COLUMN** entfernt eine oder mehrere Spalten (ein- und ausgeblendet) in der Listbox, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Übergeben Sie nicht den optionalen Parameter *Anzahl*, entfernt der Befehl nur die in *SpaltePos* angegebene Spalte.

Sonst gibt *Anzahl* die Anzahl der Spalten an, die rechts von der Spalte *SpaltePos* entfernt werden sollen. Diese Spalte wird ebenfalls entfernt.

Ist *SpaltePos* größer als die Gesamtanzahl der Spalten, führt der Befehl nichts aus.

LISTBOX DELETE ROWS

LISTBOX DELETE ROWS ({* ;} Objekt ; ZeilePosition {; AnzZeilen})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
ZeilePosition	Lange Ganzzahl	➡	Position der zu löschenden Zeile
AnzZeilen	Lange Ganzzahl	➡	Anzahl der zu löschenden Zeilen

Beschreibung

Der Befehl **LISTBOX DELETE ROWS** löscht eine oder mehrere Zeilen ab der in *ZeilePosition* angegebenen Zeile (ein- oder ausgeblendet) aus der Listbox, definiert durch die Parameter *Objekt* und *.

Hinweis: Dieser Befehl funktioniert nur mit Listboxen, die auf Arrays basieren. Bei Verwenden von Listboxen, die auf Auswahlen basieren, hat er keine Auswirkung. Die Systemvariable OK gibt 0 (Null) zurück.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable, d.h. Sie übergeben keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Beachten Sie, dass nach dem Ausführen des Befehls in der Listbox keine Elemente mehr ausgewählt sind.

Die Zeile an *ZeilePosition* wird automatisch aus allen Arrays entfernt, welche die Spalten der Listbox verwenden.

Ist der Wert in *ZeilePosition* höher als die Gesamtanzahl der Zeilen in der Listbox, führt der Befehl nichts aus.

Hinweis: Dieser Befehl berücksichtigt nicht den Status ein-/ausgeblendet von Zeilen in der Listbox.

LISTBOX DUPLICATE COLUMN

LISTBOX DUPLICATE COLUMN ({* ;} Objekt ; SpaltePos ; SpalteName ; SpalteVariable ; KopfName ; KopfVariable {; FußName ; FußVariable})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist Objektname (String) Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *) der zu duplizierenden Spalte
SpaltePos	Lange Ganzzahl	➡	Position der neu duplizierten Spalte
SpalteName	String	➡	Name der neuen Spalte
SpalteVariable	Array, Feld, Variable, Nil pointer	➡	Name der Variable Array Spalte oder Feld oder Variable
KopfName	String	➡	Objektname Spaltenkopfteil
KopfVariable	Variable Ganzzahl, Nil pointer	➡	Variable Spaltenkopfteil
FußName	String	➡	Objektname Spaltenfußteil
FußVariable	Variable, Nil pointer	➡	Variable Spaltenfußteil

Beschreibung

Der Befehl **LISTBOX DUPLICATE COLUMN** dupliziert per Programmierung im Formular in der Ausführung die Spalte, definiert durch die Parameter *Objekt* und * (Anwendungsmodus). Das Originalformular, das im Designmodus erstellt wurde, wird nicht verändert.

Hinweis: Diese Funktionalität war bereits im Designmodus von 4D vorhanden, und zwar im Kontextmenü des Formulareditors über den Befehl Spalte duplizieren.

Standardmäßig werden alle Stiloptionen der Ausgangsspalte (Größe, Farbe, Formate, etc.), die über die Eigenschaftenliste oder Befehle zur Objektverwaltung (**OBJECT SET COLOR**, etc.) gesetzt wurden, auch auf die Kopie angewendet. Auch die Objektmethode und die Einstellungen der Formularereignisse werden duplizieren. Die Datenquelle (Listbox vom Typ Array oder Auswahl) sowie Stil und Farb-Arrays werden dagegen nicht dupliziert. Sie müssen selbst dafür sorgen, dass diese nach dem Duplizieren jeder neuen Spalte zugewiesen wird.

Die Parameter *Objekt* und * bestimmen die Spalte zum Duplizieren. Mit dem optionalen Parameter * ist *Objekt* ein Spaltenname (String). Ohne * ist *Objekt* eine Spaltenvariable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Hinweis: Dieser Befehl führt nichts aus, wenn er auf die erste Spalte einer Listbox im hierarchischen Modus angewandt wird.

Die duplizierte Spalte wird genau vor die Spalte gesetzt, die im Parameter *SpaltePosition* angegeben ist. Ist *SpaltePosition* größer als die Gesamtanzahl der Spalten, wird sie nach der letzten Spalte gesetzt.

In den Parametern *SpalteName* und *SpalteVariable* übergeben Sie den Objektnamen und die Variable der neuen duplizierten Spalte.

- Für Listboxen vom Typ Array entspricht der Variablenname dem Namen des Array, dessen Inhalt in der Spalte angezeigt wird. in einem dynamischen Kontext können Sie einen Zeiger Nil (->[]) übergeben (siehe unten).
- Für Listboxen vom Typ Auswahl können Sie im Parameter *SpalteVariable* ein Feld oder eine Variable übergeben. Dann ist der Inhalt der Spalte der Wert dieses Feldes oder der Variable, der für jeden Datensatz in der Auswahl für die Listbox gewertet wird. Dafür muss die Eigenschaft "Datenquelle" der Listbox auf "aktuelle Auswahl" oder "temporäre Auswahl" gesetzt sein.
- Listboxen vom Typ Collection oder Entity-Selection sind mit diesem Befehl nicht kompatibel.

Beachten Sie, dass die Datenquelle der ursprünglichen Spalten nicht dupliziert wird: Sie müssen für die neue duplizierte Spalte ein Quellvariable, Array oder Feld setzen.

In den Parametern *KopfteilName* und *KopfteilVariable* übergeben Sie Objektname und Variable für den Kopfteil der neuen duplizierten Spalte.
In den Parametern *FußteilName* und *FußteilVariable* können Sie Objektname und Variable für den Fußteil der neuen duplizierten Spalte übergeben. Lassen Sie *FußteilVariable* weg, verwendet 4D eine dynamische Variable.

Hinweis: Objektnamen im Formular müssen einmalig sein. Sie müssen sicherstellen, dass die Namen in den Parametern *SpalteName*, *KopfteilName* und *FußteilName* noch nicht verwendet wurden. Andernfalls wird die Spalte nicht dupliziert und ein Fehler wird generiert.

Dieser Befehl muss beim Anzeigen eines Formulars verwendet werden. Er wird normalerweise im Ereignis On Load des Formulars oder in Folge einer Benutzeraktion aufgerufen (Ereignis (On Clicked).

Dynamisches Duplizieren

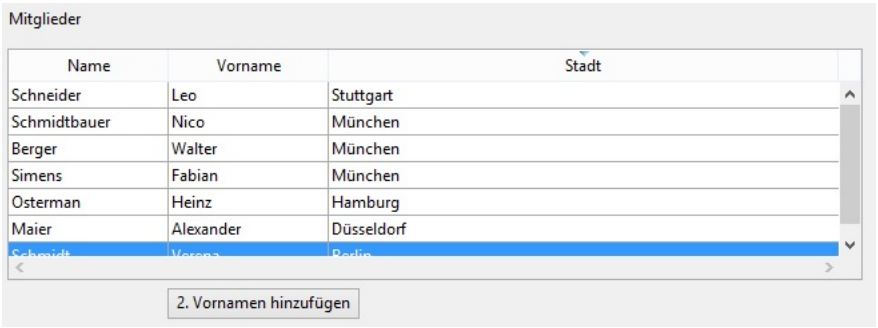
Ab 4D v14 R3 können Sie Spalten von Listboxen dynamisch duplizieren und 4D verwaltet automatisch die Definition der erforderlichen Variablen (Spalten, Fußteil und Kopfteil).

Dazu akzeptiert **LISTBOX DUPLICATE COLUMN** einen Zeiger **Nil (->[])** als Wert für die Parameter *SpalteVariable* (nur für Listboxen vom Typ Array), *KopfVariable* und *FußVariable*. In diesem Fall erstellt 4D beim Ausführen des Befehls die erforderlichen Variablen dynamisch. Weitere Informationen dazu finden Sie im Abschnitt **Dynamische Variablen**.

Beachten Sie, dass Variablen für Kopf- und Fußteil immer mit den spezifischen Typ Lange Ganzzahl bzw. Text erstellt werden. Im Gegensatz dazu lassen sich Variablen für Spalten nicht beim Erstellen typisieren, da die Listbox für diese Variable verschiedene Arraytypen erlaubt (Array Text, Array Ganzzahl, etc.). Es ist wichtig, den Arraytyp vor Verwenden von Befehlen wie **LISTBOX INSERT ROWS** zu setzen, um neue Elemente in das Array einzufügen. Alternativ lässt sich auch der Befehl **APPEND TO ARRAY** verwenden, um gleichzeitig den Arraytyp zu setzen und das Einfügen von Elementen zu starten.

Beispiel 1

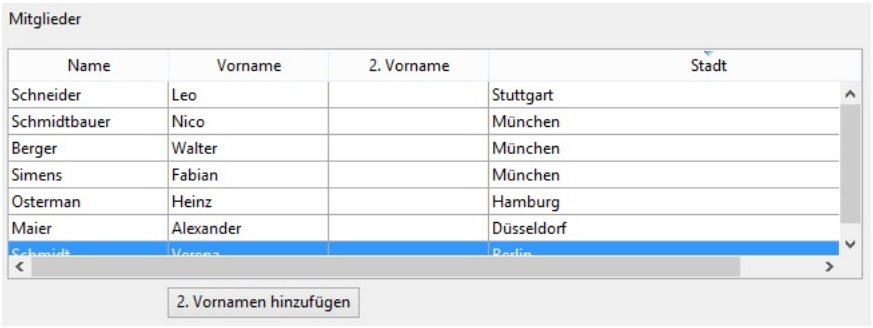
In einer Listbox vom Typ Array die Spalte "Vorname" zur Eingabe des 2. Vornamens duplizieren



Der Code der Schaltfläche lautet:

```
ARRAY TEXT(arrFirstNames2;Records in table([Mitglieder]))
LISTBOX DUPLICATE COLUMN(*;"column2";3;"col2bis";arrFirstNames2;"FirstNameA";vHead2A)
OBJECT SET TITLE(*;"FirstNameA";"2. Vorname")
EDIT ITEM(*;"col2A";0)
```

Klicken Sie auf die Schaltfläche am unteren Rand, erhalten Sie folgende Listbox:



Beispiel 2

Eine Spalte vom Typ Boolean duplizieren und ihren Titel ändern:

```
C_POINTER($ptr)
LISTBOX DUPLICATE COLUMN(*;"boolCol";3;"duplBoolCol";$ptr;"duplBoolHeader";$ptr;"duplBoolFooter";$ptr)
colprt:=OBJECT Get pointer(Object named;"duplBoolCol")
ARRAY BOOLEAN(colprt->;10)
headprt:=OBJECT Get pointer(Object named;"duplBoolHeader")
OBJECT SET TITLE(headprt->;"Neue duplizierte Spalte")
```

LISTBOX EXPAND

LISTBOX EXPAND ({* ;} Objekt {; rekursiv {; Selector {; Linie {; Spalte}}}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
rekursiv	Boolean	➡	Wahr = erweiterte Unterebenen, Falsch = Unterebenen nicht erweitert
Selector	Lange Ganzzahl	➡	Teil der Listbox, die erweitert werden soll.
Linie	Lange Ganzzahl	➡	Nummer der Umbruchzeile
Spalte	Lange Ganzzahl	➡	Nummer der Umbruchspalte

Beschreibung

Der Befehl **LISTBOX EXPAND** erweitert die Umbruchzeilen des Objekts *Listbox*, definiert durch die Parameter Objekt und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String

Ist die Listbox nicht im hierarchischen Modus angelegt, führt der Befehl nichts aus. Weitere Informationen dazu finden Sie im Abschnitt **Hierarchische Listboxen verwalten**.

Mit dem optionalen Parameter *rekursiv* konfigurieren Sie das Erweitern der hierarchischen Unterebenen der Listbox. Übergeben Sie *Wahr* oder lassen Sie diesen Parameter weg, um alle Ebenen und Unterebenen zu erweitern. Übergeben Sie *Falsch*, wird nur die erste Ebene erweitert. Mit dem optionalen Parameter *Selector* geben Sie die Reichweite des Befehls an. Sie können eine der folgenden Konstanten unter dem Thema **Listbox** übergeben:

Konstante	Typ	Wert	Kommentar
lk all	Lange Ganzzahl	0	Der Befehl betrifft alle Unterebenen (Standardwert, wird ohne diesen Parameter verwendet)
lk selection	Lange Ganzzahl	1	Befehl gilt für ausgewählte Unterebenen.
lk break row	Lange Ganzzahl	2	Der Befehl gilt für die Unterebene, zu der die Zelle, definiert mit den Parametern *Zeile* und *Spalte*, gehört. Beachten Sie, dass diese Parameter die Zeilen- und Spaltennummern in der Listbox im Standardmodus darstellen und nicht in der hierarchischen Form. In diesem Fall führt der Befehl nichts aus, wenn die Parameter *Zeile* und *Spalte* weggelassen werden.
lk level	Lange Ganzzahl	3	Dieser Befehl betrifft alle Umbruchzeilen für die Spalte *Ebene*. Dieser Parameter bezeichnet eine Spaltennummer in der Listbox im Standardmodus und nicht in seiner hierarchischen Darstellung. In diesem Fall führt der Befehl nichts aus, wenn der Parameter *Ebene* weggelassen wird.

Dieser Befehl wählt keine Umbruchzeilen aus. Enthält die Auswahl oder Listbox keine Umbruchzeile oder sind alle Umbruchzeilen bereits erweitert, führt der Befehl nichts aus.

Beispiel

Dieses Beispiel zeigt verschiedene Wege für den Befehl. Wir verwenden die folgenden Arrays in einer Listbox:

Deutschland	Bayern	München	1300000
Deutschland	Bayern	Freising	45000
Deutschland	Bayern	Rosenheim	60000
Deutschland	Hessen	Frankfurt	675000
Deutschland	Hessen	Darmstadt	138000
Frankreich	Normandie	Caen	220000
Frankreich	Normandie	Deauville	4000
Frankreich	Bretagne	Brest	120000
Frankreich	Bretagne	Quimper	80000

```
//Erweitere alle Umbruchzeilen und Umbruch-Unterzeilen für die Listbox
LISTBOX EXPAND(*;"MeineListbox")
```

V Deutschland		
V Bayern		
	München	1300000
	Freising	45000
	Rosenheim	60000
V Hessen		
	Frankfurt	675000
	Darmstadt	138000
V Frankreich		
V Normandie		
	Caen	220000
	Deauville	4000
V Bretagne		
	Brest	120000
	Quimper	80000

```
//Erweitere die erste Ebene der Umbruchzeilen für die Auswahl
LISTBOX EXPAND(*;"MeineListbox";False;lk selection)
//wenn die Zeile "Frankreich" ausgewählt wurde:
```


```
//Erweitere die Umbruchzeile Bayern mit Rekursion
LISTBOX EXPAND(*;"MeineListbox";False;lk break row;1;2)
```


```
//Erweitere alle ersten Spalten (Länder) ohne Rekursion
LISTBOX EXPAND(*;"MeineListbox";False;lk level;1)
```

LISTBOX Get array

LISTBOX Get array ({* ;} Objekt ; arrTyp) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
arrTyp	Lange Ganzzahl	➡	Typ des Array
Funktionsergebnis	Zeiger	⬅	Zeiger auf das Array, verbunden mit der Eigenschaft

Beschreibung

Hinweis: Diese Funktion funktioniert nur für Listboxen vom Typ Array.

Die Funktion **LISTBOX Get array** gibt einen Zeiger auf das Array zurück, verbunden mit dem Stil oder der Farbe der Listbox oder Spalte der Listbox, definiert durch die Parameter *Objekt* und *.

Arrays Stil, Farbe, Hintergrundfarbe oder Zeilenkontrolle lassen sich mit Listboxen vom Typ Array oder mit Spalten von Listboxen vom Typ Array (außer für Arrays Zeilenkontrolle) verbinden, entweder über die Eigenschaftenliste im Designmodus oder über den Befehl **LISTBOX SET ARRAY**.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.
In *Objekt* können Sie eine Listbox oder die Spalte einer Listbox angeben.

In *arrTyp* übergeben Sie den Arraytyp für die Eigenschaft, die Sie erhalten wollen. Sie können eine der folgenden Konstanten unter dem Thema **Listbox** verwenden:

Konstante	Typ	Wert	Kommentar
lk background color array	Lange Ganzzahl	1	
lk control array	Lange Ganzzahl	3	
lk font color array	Lange Ganzzahl	0	
lk row height array	Lange Ganzzahl	4	(4D View Pro Lizenz ist erforderlich)
lk style array	Lange Ganzzahl	2	

Die Funktion gibt eine der folgenden Werte zurück:

- **Is nil pointer**, wenn kein Array für die angeforderte Eigenschaft mit der Spalte oder Listbox verbunden ist.
- Ein Zeiger auf das Array der angeforderten Eigenschaft, vom Benutzer definiert.
- Ein Zeiger auf das Array der angeforderten Eigenschaft, dynamisch beim Aufrufen des Befehls **LISTBOX SET ROW COLOR** oder **LISTBOX SET ROW FONT STYLE** definiert.

Beispiel

Typische Anwendungsbeispiele:

```
vPtr:=LISTBOX Get array(*;"MyLB";lk font color array)
 // gibt einen Zeiger auf das Array Schriftfarbe zurück, verbunden mit der Listbox "MeineLB"

vPtr:=LISTBOX Get array(*;"Col4";lk style array)
 // gibt einen Zeiger auf das Array Schriftstil zurück, verbunden mit den Spalten der Listbox "Col4"
```

LISTBOX GET ARRAYS

LISTBOX GET ARRAYS ({* ;} Objekt ; arrSpaNamen ; arrKopfNamen ; arrSpaVars ; arrKopfVars ; arrSpaSichtbar ; arrStile {; arrFußNamen ; arrFußVars})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), Ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
arrSpaNamen	Array String	⬅	Objektnamen der Spalten
arrKopfNamen	Array String	⬅	Objektnamen des Kopfteils
arrSpaVars	Array Zeiger	⬅	Zeiger auf Spaltenvariablen oder Zeiger auf Spaltenfelder oder Nil
arrKopfVars	Array Zeiger	⬅	Zeiger auf Kopfteilvariablen
arrSpaSichtbar	Array Boolean	⬅	Spalte ein- oder ausgeblendet
arrStile	Array Zeiger	⬅	Zeiger auf Arrays oder Variablen für Stil, Farbe bzw. Zeilenkontrolle, oder Nil
arrFußNamen	Array String	⬅	Objektnamen Spaltenfußteile
arrFußVars	Array Zeiger	⬅	Zeiger auf Variablen der Spaltenfußteile

Beschreibung

Der Befehl **LISTBOX GET ARRAYS** gibt einen Satz synchronisierter Arrays zurück mit Information zu jeder Spalte (ein- oder ausgeblendet) in der Listbox, definiert durch die Parameter *Objekt* und *.

Hinweis: Dieser Befehl lässt sich nur verwenden, wenn die Eigenschaft „Datenquelle" der Listbox auf **Aktuelle Auswahl** oder **Temporäre Auswahl** gesetzt ist. Weitere Informationen dazu finden Sie im Abschnitt **Einführung in Listboxen**. Er führt nichts aus, wenn Sie ihn mit einer Listbox vom Typ Array, Collection oder Entity-Selection verwenden.

Mit dem optionalen Parameter* geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable, d.h. Sie übergeben keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Nach Ausführung des Befehls enthält:

- Das Array *arrSpaNamen* die Liste der Objektnamen für jede Spalte in der Listbox.
- Das Array *arrKopfNamen* die Liste der Objektnamen für jeden Spaltentitel in der Listbox.
- Das Array *arrSpaVars* enthält für ein Array vom Typ Listbox Zeiger auf Variablen (Arrays) für jede Spalte in der Listbox. Bei Auswahlen vom Typ Listbox enthält *arrSpaVars*:
  - Für Spalten, die einem Datenfeld zugeordnet sind, einen Zeiger auf das zugewiesene Datenfeld
  - Für Spalten, die einer Variablen zugeordnet sind, einen Zeiger auf eine Variable
  - Für Spalten, die einem Ausdruck zugeordnet sind, den Zeiger **Is nil pointer**

- Das Array *arrKopfVars* die Zeiger auf Variablen (Arrays) für jeden Spaltentitel in der Listbox.
- Das Array *arrSpaSichtbar* einen Boolean Wert für jede Spalte. Bei Wahr erscheint die Spalte in der Listbox, bei Falsch ist sie ausgeblendet.
- Das Array *arrStile* enthält für ein Array vom Typ Listbox vier Zeiger auf vier Arrays, über die Sie jeder Zeile der Listbox einen bestimmten Stil, Farbe für Schrift oder Hintergrund und einen eigenen Status ausgeblendet zuweisen können. Diese Arrays werden der Listbox über die Eigenschaftenliste in der Designumgebung zugewiesen oder über den Befehl **LISTBOX SET ARRAY**. Ist ein Array für die Listbox nicht angegeben, enthält der entsprechende Eintrag in *arrStile* einen Zeiger **Is nil pointer**.
  Der vierte Zeiger entspricht entweder einem Boolean Array (Array ausgeblendete Zeilen) oder einem Array Lange Ganzzahl (Array zum Setzen von ausgeblendeten, deaktivierten und nicht-auswählbaren Zeilen) basierend auf der Variante, die für das Array Zeilenkontrolle verwendet wird (siehe **Eigenschaften für Listboxen**). Ist ein Array für die Listbox nicht angegeben, enthält der entsprechende Eintrag in *arrStile* einen Zeiger **Is nil pointer**.
  Bei einer Listbox vom Typ Auswahl und Collection enthält *arrStile*:
  - Für Konfigurationen über eine Variable einen Zeiger auf eine Variable
  - Für Konfigurationen über einen Ausdruck den Zeiger **Is nil pointer**

LISTBOX Get auto row height

LISTBOX Get auto row height ({* ;} Objekt ; Selector {; Einheit}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Selector	Lange Ganzzahl	➡	Zu erhaltender Wert der Höhe: lk row min height oder lk row max height
Einheit	Lange Ganzzahl	➡	Einheit der Höhe: 0 = Pixel, 1 = Zeilen
Funktionsergebnis	Lange Ganzzahl	⮌	Wert der gewählen Zeilenhöhe

4D View Pro

Für diesen Befehl ist eine 4D View Pro Lizenz erforderlich. Ist sie nicht vorhanden, erscheint beim Ausführen des Formulars ein Fehler in der Listbox. Weitere Informationen dazu finden Sie im Abschnitt **4D View Pro Handbuch**.

Beschreibung

Die Funktion **LISTBOX Get auto row height** gibt den aktuellen Wert für Mindest- oder Maximumhöhe der Zeile zurück für die Listbox, definiert mit den Parametern *Objekt* und *.

Dieser Wert lässt sich entweder in der Eigenschaftenliste (siehe **Automatische Zeilenhöhe**) oder im aktuellen Prozess über die Funktion current **LISTBOX SET AUTO ROW HEIGHT** setzen.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String. Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

Hinweis: Dieser Befehl lässt sich nur für auf Arrays basierende, nicht-hierarchische Listboxen verwenden.

In *Selector* übergeben Sie den gewünschten Typ des Werts. Sie können eine der Konstanten unter dem Thema **Listbox** verwenden:

Konstante	Typ	Wert
lk row max height	Lange Ganzzahl	33
lk row min height	Lange Ganzzahl	32

Standardmäßig gibt die Funktion den Wert in Pixel zurück. Im Parameter *Einheit* können Sie eine der Konstanten unter dem Thema **Listbox** übergeben:

Konstante	Typ	Wert	Kommentar
lk lines	Lange Ganzzahl	1	Höhe ist eine Anzahl Zeilen. 4D berechnet die Zeilenhöhe nach dem Schrifttyp.
lk pixels	Lange Ganzzahl	0	Höhe ist eine Anzahl Pixel (Standard)

Beispiel

Die maximale Anzahl Zeilen für eine Zeile der Listbox erhalten:

```
C_LONGINT(vhMax)
vhMax:=LISTBOX Get auto row height(*;"LB";lk row max height;lk lines)
```

LISTBOX GET CELL COORDINATES

LISTBOX GET CELL COORDINATES ({* ;} Objekt ; Spalte ; Zeile ; links ; oben ; rechts ; unten)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Spalte	Lange Ganzzahl	➡	Spaltennummer
Zeile	Lange Ganzzahl	➡	Zeilennummer
links	Lange Ganzzahl	⬅	Linke Koordinate des Objekts
oben	Lange Ganzzahl	⬅	Obere Koordinate des Objekts
rechts	Lange Ganzzahl	⬅	Rechte Koordinate des Objekts
unten	Lange Ganzzahl	⬅	Untere Koordinate des Objekts

Beschreibung

Der Befehl **LISTBOX GET CELL COORDINATES** gibt die Koordinaten *links*, *oben*, *rechts* und *unten* (in Punkten) in Variablen oder Feldern der Zelle, definiert durch die Parameter *Spalte* und *Zeile*, der Listbox, definiert durch die Parameter *Objekt* und *, zurück..

Mit dem optionalen Parameter * geben Sie an, dass der Parameter ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Zur Wahrung der Konsistenz mit dem Befehl **OBJECT GET COORDINATES** ist der Ausgangspunkt die obere linke Ecke des Formulars, welches die Zelle enthält. Außerdem sind die zurückgegebenen Koordinaten theoretisch, d.h. sie berücksichtigen den Scrollen-Status der Listbox vor dem Zuschneiden. So kann es vorkommen, dass die Zelle nicht oder nur teilweise sichtbar ist, wenn die angegebenen Koordinaten außerhalb der Formulargrenzen oder negativ sind. Um herauszufinden, ob die Zelle bzw. welcher Teil davon sichtbar ist, müssen Sie die zurückgegebenen Koordinaten mit den Koordinaten der Listbox vergleichen. Dabei gelten folgende Regeln:

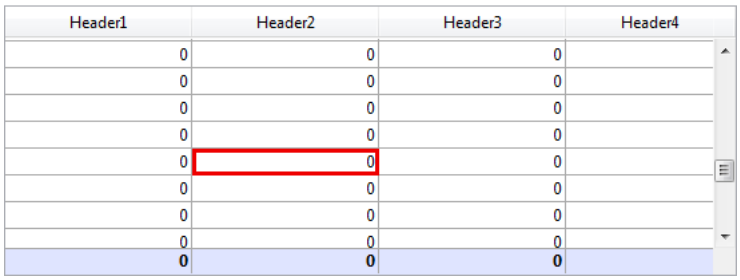
- Alle Zellen werden auf die Koordinaten der zugehörigen Listbox zugeschnitten, d.h. wie sie vom Befehl **OBJECT GET COORDINATES** für die Listbox zurückgegeben werden.
- Die Unterobjekte Kopfteil und Fußteil erscheinen vor dem Inhalt der Spalte: Überschneiden sich die Koordinaten einer Zelle mit den Koordinaten der Zeilen für Kopf- bzw. Fußteil, wird die Zelle an dieser Schnittstelle nicht angezeigt.
- Elemente gesperrter Spalten erscheinen vor den Elementen scrollbarer Spalten: Überschneiden sich die Koordinaten eines Elements in einer scrollbaren Spalte mit den Koordinaten eines Elements in einer gesperrten Spalte, erscheint es nicht an dieser Schnittstelle.

Weitere Informationen dazu finden Sie unter dem Befehl **OBJECT GET COORDINATES**.

Beispiel

Sie wollen in einer Listbox in der ausgewählten Zelle einen roten Rahmen setzen:

```
OBJECT SET VISIBLE(*;"RedRect";False)  //rotes Rechteck setzen
 //das Rechteck ist bereits irgendwo im Formular definiert
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 ({* ;} Objekt {; X ; Y }; Spalte ; Zeile {; SpaltenVar})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern ist Objekt ein Objektname (string), Ohne Stern ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
X	Zahl	➡	Horizontale Koordinate der Maus
Y	Zahl	➡	Vertikale Koordinate der Maus
Spalte	Lange Ganzzahl	⬅	Spaltennummer
Zeile	Lange Ganzzahl	⬅	Zeilennummer
SpaltenVar	Zeiger	⬅	Zeiger auf Spaltenvariable

Beschreibung

Der Befehl **LISTBOX GET CELL POSITION** gibt die Nummern von *Spalte* und *Zeile* zurück, die der Position des letzten Mausklicks, der letzten Auswahl über die Tastatur oder der horizontalen und vertikalen Koordinaten der Maus in der Listbox entsprechen, angegeben durch * und *Objekt*.

Mit dem optionalen Parameter * ist *Objekt* ein Objektname (String), ohne diesen Parameter ist *Objekt* eine Variable.

Sind die Parameter X und Y übergeben, gibt der Befehl die Spalten- und Zeilennummer der Mauskoordinaten zurück, sonst die Spalten- und Zeilennummer eines Klicks oder einer Auswahl. Er gibt auch dann gültige Werte zurück, wenn keine Dateneingabe in die Listbox erlaubt ist.

Hinweise:

- Die in *Zeile* zurückgegebene Nummer berücksichtigt nicht den Status ein-/ausgeblendet der Listbox-Zeilen. Es kann auch der Wert 0 sein, wenn der Klick oder die Y-Koordinate unter der letzten Zeile liegt.
- Wird eine Zelle in einer Proforma Spalte angeklickt oder als X-Koordinate angegeben, enthält der Parameter *Zeile* "N+1". N ist die Anzahl der vorhandenen Spalten bei Klick in eine Zelle oder wenn es keine Spalte an der X-Position gibt. Eine Proforma Spalte lässt sich automatisch hinzufügen, wenn die Option "Spaltenbreite Automatisch" ausgewählt ist; weitere Informationen dazu finden Sie im Abschnitt **Gruppe Vergrößerungsoptionen** des Handbuchs *4D Designmodus*.

Der optionale Parameter *SpaltenVar* gibt einen Zeiger auf die Variable zurück (z.B. Array), die der Spalte zugeordnet ist.

Ohne die Parameter X und Y lässt sich **LISTBOX GET CELL POSITION** nur im Rahmen einer Listbox aufrufen, die eins der folgenden Formularereignisse erzeugt:

- On Clicked und On Double Clicked
- On Before Keystroke und On After Keystroke
- On After Edit
- On Getting Focus und On Losing Focus
- On Data Change
- On Selection Change
- On Before Data Entry

Wird der Befehl in einem anderen Kontext aufgerufen, gibt er in *Spalte* und *Zeile* den Wert 0 zurück.

Er berücksichtigt alle Änderungen an der Auswahl bzw. Abwahl, sei es durch Mausklick, über Tastatur oder über den Befehl **EDIT ITEM**, der das Ereignis On getting Focus erzeugen kann. Wurde die Auswahl über Pfeiltasten der Tastatur verändert, gibt *Spalte* den Wert 0 (Null) zurück. In diesem Fall gibt der Parameter *SpaltenVar* - sofern übergeben - **Is nil pointer** zurück.

LISTBOX Get column formula

LISTBOX Get column formula ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	String	⮌	Der Spalte zugewiesene Formel

Beschreibung

Der Befehl **LISTBOX Get column formula** gibt die der Spalte zugewiesene Formel der Listbox zurück, definiert durch die Parameter *Objekt* und *.

Formeln lassen sich nur verwenden, wenn die Eigenschaft "Datenquelle" der Listbox entweder **aktuelle Auswahl** oder **temporäre Auswahl** bzw. **Collection oder Entity Selection** ist. Ist der Spalte keine Formel zugewiesen, gibt die Funktion einen leeren String zurück.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String. Dieser Parameter muss eine Spalte der Listbox angeben.

LISTBOX Get column width

LISTBOX Get column width ({* ;} Objekt {; minBreite {; maxBreite}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
minBreite	Lange Ganzzahl	⬅	Mindestbreite der Spalte (in Pixel)
maxBreite	Lange Ganzzahl	⬅	Maximale Breite der Spalte (in Pixel)
Funktionsergebnis	Lange Ganzzahl	➲	Spaltenbreite (in Pixel)

Beschreibung

Die Funktion **LISTBOX Get column width** gibt die Breite (in Pixel) der Spalte zurück, definiert durch die Parameter *Objekt* und *.

In *Objekt* können Sie eine Spalte oder einen Spaltentitel der Listbox übergeben.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

LISTBOX Get column width kann die Beschränkungen der Spaltenbreite in den Parametern *minBreite* und *maxBreite* zurückgeben. Diese Beschränkungen lassen sich über den Befehl **LISTBOX SET COLUMN WIDTH** festlegen.
Wurde kein Maximum- bzw. Minimumwert für die Spalte gesetzt, gibt der jeweilige Parameter 0 (Null) zurück.

LISTBOX Get footer calculation

LISTBOX Get footer calculation ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	Lange Ganzzahl	⮌	Art der Berechnung

Beschreibung

Die Funktion **LISTBOX Get footer calculation** gibt die Berechnungsart im Fußteil der Listbox zurück, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Der Parameter *Objekt* kann folgendes festlegen:

- Variable oder Name eines Fußteilbereichs. In diesem Fall gibt der Befehl die Berechnung für diesen Bereich zurück.
- Variable oder Name einer Spalte in der Listbox. In diesem Fall gibt der Befehl die Berechnung für den Fußteil dieser Spalte zurück.

Sie können den zurückgegebenen Wert mit den Konstanten unter dem Thema **Listbox Fußteil Berechnung** vergleichen (siehe Befehl **LISTBOX SET FOOTER CALCULATION**).

LISTBOX Get footers height

LISTBOX Get footers height ({* ;} Objekt {; Einheit}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Einheit	Lange Ganzzahl	➡	Einheit für Wert der Höhe: 0 oder weggelassen = Pixel, 1 = Zeilen
Funktionsergebnis	Lange Ganzzahl	⮌	Zeilenhöhe

Beschreibung

Die Funktion **LISTBOX Get footers height** gibt die Höhe der Fußzeile in der Listbox zurück, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String. Sie können entweder die Listbox oder einen Fußteil in der Listbox angeben.

Standardmäßig, d.h. ohne den Parameter *Einheit* wird die Zeilenhöhe in Pixel gesetzt. Zum Wechseln der Einheit können Sie in *Einheit* eine der folgenden Konstanten unter dem Thema **Listbox** übergeben.

Konstante	Typ	Wert	Kommentar
lk lines	Lange Ganzzahl	1	Höhe ist eine Anzahl Zeilen. 4D berechnet die Zeilenhöhe nach dem Schrifttyp.
lk pixels	Lange Ganzzahl	0	Höhe ist eine Anzahl Pixel (Standard)

Hinweis: Mehr Informationen dazu finden Sie im Abschnitt **Gruppen Kopfteile und Fußteile** des Handbuchs *4D Designmodus*.

LISTBOX GET GRID

LISTBOX GET GRID ({* ;} Objekt ; horizontal ; vertikal)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
horizontal	Boolean	⬅	Wahr = angezeigt, Falsch = ausgeblendet
vertikal	Boolean	⬅	Wahr = angezeigt, Falsch = ausgeblendet

Beschreibung

Der Befehl **LISTBOX GET GRID** zeigt den Status ein-/ausgeblendet der horizontalen bzw. vertikalen Zeilen des Gitters der Listbox, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

In *horizontal* und *vertikal* gibt der Befehl den Wert **Wahr** oder **Falsch** zurück, je nachdem, ob die entsprechenden Zeilen angezeigt (Wahr) oder ausgeblendet (Falsch) sind. Standardmäßig wird das Gitter angezeigt.

LISTBOX GET GRID COLORS

LISTBOX GET GRID COLORS ({* ;} Objekt ; hFarbe ; vFarbe)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
hFarbe	Lange Ganzzahl	⬅	Wert der RGB Farbe für horizontale Zeilen
vFarbe	Lange Ganzzahl	⬅	Wert der RGB Farbe für vertikale Zeilen

Beschreibung

Der Befehl **LISTBOX GET GRID COLORS** setzt die Farbe der horizontalen und vertikalen Zeilen für das Gitter der Listbox, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

In *hFarbe* und *vFarbe* gibt der Befehl die Werte der RGB Farben zurück. Weitere Informationen über RGB Farben finden Sie in der Beschreibung zum Befehl **OBJECT SET RGB COLORS**.

LISTBOX Get headers height

LISTBOX Get headers height ({* ;} Objekt {; Einheit}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Einheit	Lange Ganzzahl	➡	Einheit des Höhenwerts: 0 oder weggelassen = Pixel, 1 = Zeilen
Funktionsergebnis	Lange Ganzzahl	⮌	Zeilenhöhe

Beschreibung

Die Funktion **LISTBOX Get headers height** gibt die Höhe der Kopfzeile in der Listbox an, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String. Sie können entweder die Listbox oder einen Kopfteil in der Listbox angeben.

Standardmäßig, d.h. ohne den Parameter *Einheit* wird diese Höhe in Pixel gesetzt. Zum Wechseln der Einheit können Sie in *Einheit* eine der folgenden Konstanten unter dem Thema **Listbox** übergeben:

Konstante	Typ	Wert	Kommentar
lk lines	Lange Ganzzahl	1	Höhe ist eine Anzahl Zeilen. 4D berechnet die Zeilenhöhe nach dem Schrifttyp.
lk pixels	Lange Ganzzahl	0	Höhe ist eine Anzahl Pixel (Standard)

Hinweis: Weitere Informationen zum Berechnen der Zeilenhöhe finden Sie im Handbuch *4D Designmodus*.

LISTBOX GET HIERARCHY

LISTBOX GET HIERARCHY ({* ;} Objekt ; hierarchisch {; Hierarchie})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
hierarchisch	Boolean	⬅	Wahr = hierarchische Listbox, Falsch = nicht-hierarchische Listbox
Hierarchie	Array Zeiger	⬅	Array mit Zeigern

Beschreibung

Der Befehl **LISTBOX GET HIERARCHY** findet die hierarchischen Eigenschaften der Listbox, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Der Boolean Parameter *hierarchisch* gibt an, ob die Listbox hierarchisch ist oder nicht:

- Gibt er Wahr zurück, erscheint die Listbox im hierarchischen Modus
- Gibt er Falsch zurück, erscheint die Listbox im nicht-hierarchischen Modus (Standard Array Modus)

Ist die Listbox im hierarchischen Modus, gibt der Parameter *hierarchisch* die angeforderte Liste der Arrays der Listbox zum Aufbau der Hierarchie zurück.

Hinweis: Ist die Listbox nicht im hierarchischen Modus, gibt der Befehl im ersten Element des Arrays *Hierarchie* einen Zeiger auf das Array der ersten Spalte der Listbox zurück.

LISTBOX Get locked columns

LISTBOX Get locked columns ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	Lange Ganzzahl	⮌	Anzahl der gesperrten Spalten

Beschreibung

Die Funktion **LISTBOX Get locked columns** gibt die Anzahl der gesperrten Spalten in der Listbox zurück, definiert durch die Parameter *Objek*t und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Spalten lassen sich über die Eigenschaftenliste oder über den Befehl **LISTBOX SET LOCKED COLUMNS** sperren. Weitere Informationen dazu finden Sie im Abschnitt **Gruppe Listbox** des Handbuchs *4D Designmodus*.

Wird eine Spalte per Programmierung in einem Satz gesperrter Spalten eingefügt oder gelöscht, wird diese Änderung in der zurückgegebenen Anzahl Spalten berücksichtigt. Löschen Sie z.B. eine gesperrte Spalte, verringert sich die Anzahl gesperrter Spalten um 1. Fügen Sie eine Spalte per Programmierung in einen gesperrten Bereich ein, wird diese Spalte automatisch gesperrt und die Anzahl gesperrter Spalten erhöht sich um 1.

Die Funktion berücksichtigt dagegen nicht den Status sichtbar/nicht sichtbar der Spalten.

LISTBOX Get number of columns

LISTBOX Get number of columns ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	Lange Ganzzahl	⮌	Anzahl der Spalten

Beschreibung

Die Funktion **LISTBOX Get number of columns** gibt die Gesamtanzahl der Spalten in der Listbox zurück, inkl. ausgeblendeter Spalten, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

LISTBOX Get number of rows

LISTBOX Get number of rows ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	Lange Ganzzahl	⮌	Anzahl der Zeilen

Beschreibung

Die Funktion **LISTBOX Get number of rows** gibt die Anzahl der Zeilen in der Listbox zurück, definiert durch die Parameter *Objekt* und *.

Hinweis: Dieser Befehl berücksichtigt nicht den Status ein-/ausgeblendet von Zeilen in der Listbox. Hat eine Listbox z.B. 10 Zeilen, von denen die ersten 9 ausgeblendet sind, gibt **LISTBOX Get number of rows** 10 zurück.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable, d.h. Sie übergeben keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Hinweis: Haben die Arrays, die den Spalten einer Listbox zugeordnet sind, nicht alle dieselbe Größe, erscheinen in der Listbox nur die Anzahl der Einträge aus dem kleinsten Array. Die Funktion gibt nur diese Einträge zurück.

LISTBOX GET OBJECTS

LISTBOX GET OBJECTS ({* ;} Objekt ; arrObjektNamen)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
arrObjektNamen	Array Text	⬅	Namen der Unterobjekte mit Listbox (Kopfteil, Spalten, Fußteil)

Beschreibung

Der Befehl **LISTBOX GET OBJECTS** gibt ein Array mit den Namen jedes Objekts der Listbox zurück, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

In *arrObjektNamen* übergeben Sie ein Array Text, das der Befehl automatisch füllt. Objektnamen werden in der Reihenfolge der Anzeige zurückgegeben, also wie folgt:

nameCol1
headerNameCol1
footerNameCol1
nameCol2
headerNameCol2
footerNameCol2
...

Das Array gibt die Objektnamen aller Spalten, inkl. Spaltenfußteile zurück, egal ob sie ein- oder ausgeblendet sind.

Dieser Befehl ist hilfreich, wenn ein Formular über die Routinen **FORM LOAD**, **FORM GET OBJECTS** und **OBJECT Get type** analysiert wird. Damit erhalten Sie bei Bedarf die Namen der Unterobjekte der Listbox.

Beispiel

Ein Formular laden und eine Liste mit allen Objekten in der Listbox erhalten.

```
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 ({* ;} Objekt ; Selector ; Info)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Selector	Lange Ganzzahl	➡	Zu erhaltende Information
Info	Lange Ganzzahl	⬅	Aktueller Wert

Beschreibung

Der Befehl **LISTBOX GET PRINT INFORMATION** gibt die aktuelle Information zum Drucken des Objekts Listbox zurück, definiert durch die Parameter *Objekt* und *. Mit diesem Befehl können Sie das Drucken des Inhalts der Listbox kontrollieren.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Dieser Befehl muss im Kontext Drucken einer Listbox über die Funktion **Print object** aufgerufen werden. Außerhalb dieses Kontexts gibt er keine signifikanten Werte zurück.

In *Selector* übergeben Sie einen Wert, der die gesuchte Information angibt, in *Info* eine Variable vom Typ Zahl oder BLOB. In *Selector* können Sie eine der nachfolgenden Konstanten unter dem Thema **Listbox** übergeben:

Konstante	Typ	Wert	Kommentar
lk last printed row number	Lange Ganzzahl	0	Gibt in *Info* die Nummer der zuletzt gedruckten Zeile an. Damit finden Sie die Nummer der nächsten zu druckenden Zeile heraus. Die zurückgegebene Nummer kann größer als die Anzahl der aktuell gedruckten Zeilen sein, wenn die Listbox unsichtbare Zeichen enthält oder wenn der Befehl **OBJECT SET SCROLL POSITION** aufgerufen wurde. Wurden z.B. die Zeilen 1, 18 und 20 gedruckt, gibt *Info* 20 zurück.
lk printed height	Lange Ganzzahl	3	Gibt in *Info* die Höhe in Pixel des aktuell gedruckten Objekts zurück (inkl. Kopfzeilen, Zeilen, etc.). Beachten Sie, dass falls die Anzahl der zu druckenden Zeilen kleiner als die "Kapazität" der Listbox ist, seine Höhe automatisch verringert wird.
lk printed rows	Lange Ganzzahl	1	Gibt in *Info* die Anzahl der aktuell gedruckten Zeilen während dem letzten Aufruf der Funktion **Print object** zurück. Im Falle einer hierarchischen Liste beinhaltet diese Zahl auch alle hinzugefügten Umbruchzeilen. Zum Beispiel ist *Info* 10, bei einer Listbox mit 20 Zeilen, wenn die ungeraden Zeilen ausgeblendet sind.
lk printing is over	Lange Ganzzahl	2	Gibt in *Info* ein Boolean zurück ,der angibt, ob die letzte (sichtbare) Zeile der Listbox gerade gedruckt wurde. Wahr = Zeile wurde gedruckt; Andernfalls falsch.

Weitere Informationen dazu finden Sie im Abschnitt **Listboxen drucken** .

Beispiel 1

Drucken, bis alle Zeilen gedruckt sind:

```
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
```

Beispiel 2

Mindestens 500 Zeilen der Listbox drucken, mit dem Wissen, dass bestimmte Zeilen ausgeblendet sind:

```
$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 ({* ;} Objekt ; Eigenschaft) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Eigenschaft	Lange Ganzzahl	➡	Eigenschaft, dessen Wert Sie erhalten wollen
Funktionsergebnis	Lange Ganzzahl, String	⮌	Aktueller Wert

Beschreibung

Die Funktion **LISTBOX Get property** gibt den Wert des Parameters *Eigenschaft* für die Listbox oder Spalte zurück, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable, d.h. Sie übergeben keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Hinweis: Existiert keine Listbox oder Spalte der Listbox, definiert durch die Parameter *Objekt* und *, gibt die Funktion für numerische Eigenschaften -1 zurück, oder einen leeren String.

In *Eigenschaft* übergeben Sie eine Konstante mit der Eigenschaft, deren Wert Sie erhalten möchten. Sie können eine der vordefinierten Konstanten aus dem Thema **Listbox** verwenden:

Konstante	Typ	Wert	Kommentar
_o_lk display hor scrollbar	Lange Ganzzahl	2	***Konstante ist überholt*** Befehl **OBJECT GET SCROLLBAR** verwenden
_o_lk display ver scrollbar	Lange Ganzzahl	4	***Konstante ist überholt*** Befehl **OBJECT GET SCROLLBAR** verwenden.
_o_lk footer height	Lange Ganzzahl	9	***Konstante ist überholt*** Befehl **LISTBOX Get footers height** verwenden
_o_lk header height	Lange Ganzzahl	1	***Konstante ist überholt*** Befehl **LISTBOX Get headers height** verwenden
_o_lk hor scrollbar position	Lange Ganzzahl	6	***Konstante ist überholt*** Befehl **OBJECT GET SCROLL POSITION** verwenden
_o_lk ver scrollbar position	Lange Ganzzahl	7	***Konstante ist überholt*** Befehl **OBJECT GET SCROLL POSITION** verwenden
lk allow wordwrap	Lange Ganzzahl	14	Eigenschaft **Zeilenumbruch** Gilt für: Spalte* Mögliche Werte: • lk no: (0) • lk yes: (1)
lk auto row height	Lange Ganzzahl	31	Eigenschaft **Automatische Zeilenhöhe** Gilt für: Listbox oder Spalte Mögliche Werte: • lk yes • lk no **Nur 4D View Pro:** Für dieses Feature wird eine 4D View Pro Lizenz benötigt. Weitere Informationen dazu finden Sie im **4D View Pro Handbuch**.
lk background color expression	Zeichenkette	22	Eigenschaft **Hintergrundfarbe Ausdruck** für Listbox vom Typ Auswahl Gilt für: Listbox oder Spalte
lk column max width	Lange Ganzzahl	26	Eigenschaft **Maximale Breite** Gilt für: Spalte*
lk column min width	Lange Ganzzahl	25	Eigenschaft **Minimale Breite** Gilt für: Spalte*
lk column resizable	Lange Ganzzahl	15	Eigenschaft **Vergrößerbar** Gilt für: Spalte * Mögliche Werte: • lk no (0): • lk yes (1):
lk detail form name	Zeichenkette	19	Eigenschaft **Name Detailformular** für Listbox vom Typ Auswahl. Gilt für: Listbox
lk display footer	Lange Ganzzahl	8	Eigenschaft **Fußteil anzeigen** Gilt für: Listbox Mögliche Werte: • lk no (0): Ausgeblendet • lk yes (1): Eingeblendet
lk display header	Lange Ganzzahl	0	Eigenschaft **Kopfteil anzeigen** Gilt für: Listbox Mögliche Werte: • lk no (0): Ausgeblendet • lk yes (1): Eingeblendet
lk display type	Lange Ganzzahl	21	**Eigenschaft Typanzeige** für Spalten der Listbox vom Typ Zahl Gilt für: Spalte* Mögliche Werte: • lk numeric format: (0) Zeigt Werte im Zahlenformat an • lk three states checkbox: (1) Zeigt Werte als Kontrollkästchen mit drei Zuständen an
lk double click on row	Lange Ganzzahl	18	Eigenschaft **Doppelklick auf Zeile** für Listbox vom Typ Auswahl Gilt für: Listbox Mögliche Werte: • lk do nothing (0): Löst keine automatische Aktion aus • lk edit record (1): Zeigt den entsprechenden Datensatz im Lese-/Schreibmodus an • lk display record (2): Zeigt den entsprechenden Datensatz im Nur-Lesen Modus an

Konstante	Typ	Wert	Kommentar
lk extra rows	Lange Ganzzahl	13	Eigenschaft **Zusätzliche Leerzeilen ausblenden** Gilt für für Listbox Mögliche Werte: • lk display: (0) • lk hide: (1)
lk font color expression	Zeichenkette	23	Eigenschaft **Schriftfarbe Ausdruck** für Listbox vom Typ Auswahl Gilt für: Listbox oder Spalte
lk font style expression	Zeichenkette	24	Eigenschaft **Stilausdruck** für Listbox vom Typ Auswahl Gilt für: Listbox oder Spalte
lk hide selection highlight	Lange Ganzzahl	16	Eigenschaft **Markierung Auswahl ausblenden** Gilt für: Listbox Mögliche Werte: • lk no: (0) • lk yes: (1)
lk highlight set	Zeichenkette	27	Eigenschaft **Markierung Menge** für Listbox vom Typ Auswahl Gilt für: Listbox
lk hor scrollbar height	Lange Ganzzahl	3	Höhe in Pixel
lk multi style	Lange Ganzzahl	30	Eigenschaft **Mehrfachstil** Gilt für: Spalte * Mögliche Werte: • lk no (0) • lk yes (1)
lk named selection	Zeichenkette	28	Eigenschaft **temporäre Auswahl** Gilt für: Listbox
lk resizing mode	Lange Ganzzahl	11	Eigenschaft **Spaltenbreite Automatisch** Gilt für: Listbox Mögliche Werte: • lk manual: (0) • lk automatic: (1)
lk row height unit	Lange Ganzzahl	17	Eigenschaft **Einheit für Zeilenhöhe** Gilt für: Listbox Mögliche Werte: • lk lines (1) • lk pixels (0)
lk selection mode	Lange Ganzzahl	10	Eigenschaft **Auswahlmodus** Gilt für: Listbox Mögliche Werte: • lk none (0) • lk single (1) • lk multiple (2)
lk single click edit	Lange Ganzzahl	29	Eigenschaft **Einzelklick editieren** Gilt für: Listbox Mögliche Werte: • lk no (0) • lk yes (1)
lk sortable	Lange Ganzzahl	20	Eigenschaft **Sortierbar** Gilt für: Listbox Mögliche Werte: • lk no: (0) • lk yes: (1)
lk truncate	Lange Ganzzahl	12	Eigenschaft **Abkürzen mit Auslassungspunkten** Gilt für: Listbox oder Spalte Mögliche Werte: • lk without ellipsis: (0) • lk with ellipsis: (1)
lk ver scrollbar width	Lange Ganzzahl	5	Breite in Pixel

* Diese Eigenschaften gelten nur für Spalten der Listbox; übergeben Sie eine Listbox als Parameter mit einer dieser Eigenschaften, gibt **LISTBOX Get property**, je nach übergebener Eigenschaft, -1 oder einen leeren String zurück.

Im allgemeinen gibt **LISTBOX Get property** zum Anzeigen eines ungültigen Ergebnisses bei Eigenschaften mit numerischen Werten -1 zurück, oder einen leeren String; es werden jedoch keine Fehler generiert. Das passiert insbesondere in folgenden

Fällen:

- Wenn Sie eine *Eigenschaft* übergeben, die nicht existiert
- Wenn Sie eine *Eigenschaft* übergeben, die für die angegebene Listbox oder Spalte nicht verfügbar ist, z.B. die Eigenschaft lk font color expression für eine Listbox vom Typ Array
- Wenn Sie eine Spalte als Parameter übergeben mit einer *Eigenschaft*, die für eine Listbox gilt, bzw. eine Listbox mit einer *Eigenschaft*, die für eine Spalte gilt. (siehe * oben)

Außerdem lassen sich zur gleichen Zeit nur Werte für eine Spalte zurückgeben; verwenden Sie z.B. Teile von Spaltennamen mit dem Symbol "@", um mehrere Spalten mit ähnlichen Namen anzugeben, gibt **LISTBOX Get property** den ersten gefundenen Wert an; demzufolge ist der zurückgegebene Wert nicht wirklich signifikant.

Hinweise:

- Die Konstanten lk display footer und lk display header sind hilfreich zum Berechnen der aktuellen Größe eines Listbox-Bereichs in einem Formular.
- Mit den Konstanten _o_lk hor scrollbar position oder _o_lk ver scrollbar position gibt **LISTBOX Get property** die Position des scrollenden Cursors in Bezug auf seine Ausgangsposition an, z.B. die Größe des ausgeblendeten Teils des Fensters in Pixel. Diese Position ist standardmäßig 0 (Null). In Kombination mit Informationen zur Zeilenhöhe können Sie über diesen Wert den in der Listbox angezeigten Inhalt herausfinden. Beachten Sie jedoch, dass diese Konstanten überholt sind und sich besser mit dem Befehl **OBJECT GET SCROLL POSITION** ersetzen lassen.
- Die Anweisung **LISTBOX Get property**(vLB;_o_lk footer height) gibt denselben Wert zurück wie die Funktion **LISTBOX Get footers height**, wenn Fußteile angezeigt werden. Werden dagegen Fußteile nicht angezeigt, gibt **LISTBOX Get property** 0 (Null) zurück, während **LISTBOX Get footers height** weiterhin die Höhe der Fußteile angibt, die in diesem Fall theoretisch ist.

Beispiel 1

Sie führen bei der vorgegebenen Listbox "MyListbox" folgende Anweisung aus:

```
$Value:=LISTBOX Get property(*;"MyListbox";lk selection mode)  // der zurückgegebene Wert gibt den Auswahlmodus an
```

Das zurückgegebene Ergebnis zeigt an, ob sich mehrfache Zeilen auswählen lassen.

Beispiel 2

Sie führen bei der vorgegebenen Listbox "MyListbox" folgende Anweisung aus:

```
$resizable:=LISTBOX Get property(*;"MyListbox";lk column resizable)
```

LISTBOX Get property gibt -1 zurück, da die Eigenschaft lk column resizable für Spalten gilt und eine Listbox als Parameter übergeben wurde.

LISTBOX Get row color

LISTBOX Get row color ({* ;} Objekt ; Zeile {; FarbeTyp}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Zeile	Lange Ganzzahl	➡	Zeilennummer
FarbeTyp	Lange Ganzzahl	➡	Listbox font color (Standard) oder Listbox background color
Funktionsergebnis	Lange Ganzzahl	↩	Color value

Beschreibung

Hinweis: Diese Funktion funktioniert nur für Listboxen vom Typ Array.

Der Befehl **LISTBOX Get row color** gibt die Farbe einer Zeile oder Zelle in der Listbox zurück, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.
In *Objekt* können Sie eine Listbox oder die Spalte einer Listbox angeben:

- Gibt *Objekt* eine Listbox an, gibt die Funktion die Farbe der Zeile zurück
- Gibt *Objekt* die Spalte einer Listbox an, gibt der Befehl die Farbe der Zelle zurück

In *Zeile* übergeben Sie die Nummer der Zeile, deren Farbe Sie erhalten wollen.

Hinweis: Diese Funktion berücksichtigt nicht den Status ein-/ausgeblendet von Zeilen der Listbox.

In *FarbeTyp* können Sie entweder die Konstante lk background color oder lk font color (Thema "**Listbox**") übergeben, um die Hintergrundfarbe oder die Schriftfarbe der Zeile herauszufinden. Lassen Sie diesen Parameter weg, wird die Schriftfarbe zurückgegeben.

Warnung: Eine zugewiesene Farbe wird nicht zwingend in jeder Zelle der Zeile angezeigt (siehe Beispiel). Bei widersprüchlichen Farbwerten in den Eigenschaften für Listboxen und Spalten von Listboxen gibt es eine Prioritätenregelung. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Beispiel

Wir gehen von folgender Listbox aus:

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 enthält 0xFFFF00 (gelb)
 // vColor2 enthält 0x00FF (blau)
 // vColor3 enthält 0x00FF0000 (rot)
```

LISTBOX Get row font style

LISTBOX Get row font style ({* ;} Objekt ; Zeile) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Zeile	Lange Ganzzahl	➡	Zeilennummer
Funktionsergebnis	Lange Ganzzahl	↩	Stilwert

Beschreibung

Hinweis: Diese Funktion funktioniert nur für Listboxen vom Typ Array.

Die Funktion **LISTBOX Get row font style** gibt den Schriftstil einer Zeile oder Zelle in der Listbox zurück, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String. In *Objekt* können Sie eine Listbox oder die Spalte einer Listbox angeben:

- Gibt *Objekt* eine Listbox an, gibt die Funktion den Stil der Zeile zurück
- Gibt *Objekt* die Spalte einer Listbox an, gibt die Funktion den Stil der Zelle zurück.

In *Zeile* übergeben Sie die Nummer der Zeile, deren Stil Sie erhalten wollen.

Hinweis: Diese Funktion berücksichtigt nicht den Status ein-/ausgeblendet von Zeilen der Listbox.

Warnung: Eine zugewiesene Farbe wird nicht zwingend in jeder Zelle der Zeile angezeigt (siehe Beispiel). Bei widersprüchlichen Farbwerten in den Eigenschaften für Listboxen und Spalten von Listboxen gibt es eine Prioritätenregelung. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Beispiel

Wir nehmen folgende Listbox:

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 enthält 1 (Bold)
 // vStyle2 enthält 6 (Italic + Underline)
```

LISTBOX Get row height

LISTBOX Get row height ({* ;} Objekt ; Zeile) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Zeile	Lange Ganzzahl	➡	Zeile der Listbox zum Erhalten der Höhe
Funktionsergebnis	Ganzzahl	↩	Zeilenhöhe

4D View Pro

Für diesen Befehl ist eine 4D View Pro Lizenz erforderlich. Ist sie nicht vorhanden, erscheint beim Ausführen des Formulars ein Fehler in der Listbox. Weitere Informationen dazu finden Sie im Abschnitt **4D View Pro Handbuch**.

Beschreibung

Die Funktion **LISTBOX Get row height** gibt die aktuelle Höhe der angegebenen *Zeile* in der Listbox zurück, definiert durch die Parameter *Objekt* und *. Die Zeilenhöhe lässt sich global über die Eigenschaftenliste bzw. den Befehl **LISTBOX SET ROWS HEIGHT** setzen, oder individuell über **LISTBOX SET ROW HEIGHT**.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String), ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Existiert die angegebene *Zeile* nicht in der Listbox, gibt die Funktion 0 (Null) zurück.

Die Zeilenhöhe wird in der aktuellen Einheit zurückgegeben, die global für Listboxzeilen definiert wurde, entweder in der Eigenschaftenliste oder durch den vorherigen Aufruf von **LISTBOX SET ROWS HEIGHT**.

LISTBOX Get rows height

LISTBOX Get rows height ({* ;} Objekt {; Einheit}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Einheit	Lange Ganzzahl	➡	Einheit für Höhe: 0 oder weggelassen = Pixel, 1 = Zeilen
Funktionsergebnis	Ganzzahl	⬅	Zeilenhöhe

Beschreibung

Die Funktion **LISTBOX Get rows height** gibt die aktuelle Zeilenhöhe (in Pixel oder Zeilen) in der Listbox zurück, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable, d.h. Sie übergeben keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Standardmäßig, d.h. ohne den Parameter *Einheit*, wird die Zeilenhöhe in Pixel angegeben. Um eine andere Einheit zu setzen, können Sie im Parameter *Einheit* eine der folgenden Konstanten unter dem Thema **Listbox** übergeben:

Konstante	Typ	Wert	Kommentar
lk lines	Lange Ganzzahl	1	Höhe ist eine Anzahl Zeilen. 4D berechnet die Zeilenhöhe nach dem Schrifttyp.
lk pixels	Lange Ganzzahl	0	Höhe ist eine Anzahl Pixel (Standard)

Hinweis: Weitere Informationen dazu finden Sie im Abschnitt **Höhe in Pixel oder Zeilen** des Handbuchs *4D Designmodus*.

LISTBOX Get static columns

LISTBOX Get static columns ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	Lange Ganzzahl	⮌	Anzahl der statischen Spalten

Beschreibung

Die Funktion **LISTBOX Get static columns** gibt die Anzahl der statischen Spalten in der Listbox zurück, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Statische Spalten lassen sich über die Eigenschaftenliste oder über den Befehl **LISTBOX SET STATIC COLUMNS** setzen.

Wird eine Spalte per Programmierung innerhalb einem Satz statischer Spalten eingefügt oder gelöscht, wird diese Änderung in der zurückgegebenen Anzahl Spalten berücksichtigt.
Die Funktion berücksichtigt dagegen nicht den Status sichtbar/nicht sichtbar der Spalten.

Hinweis: Statische und gesperrte Spalten sind zwei unabhängige Funktionen. Weitere Informationen dazu finden Sie im Abschnitt **Gesperrte Spalten und statische Spalten** des Handbuchs *4D Designmodus*.

LISTBOX GET TABLE SOURCE

LISTBOX GET TABLE SOURCE ({* ;} Objekt ; TabelleNum {; Auswahlname {; MarkierName}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), Ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
TabelleNum	Lange Ganzzahl	⬅	Tabellennummer der Auswahl
Auswahlname	String	⬅	Name der temporären Auswahl oder "" für die aktuelle Auswahl
MarkierName	String	⬅	Name der Markiermenge

Beschreibung

Der Befehl **LISTBOX GET TABLE SOURCE** findet die aktuelle Quelle der Daten, die in der Listbox angezeigt werden, definiert durch die Parameter *Objekt* und *.

Übergeben Sie den optionalen Parameter *, ist der Parameter *Objekt* ein Objektname (String). Ohne * ist *Objekt* eine Variable. In diesem Fall übergeben Sie keinen String, sondern eine Variablenreferenz. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Der Befehl gibt die Nummer der Haupttabelle zurück, die der Listbox über den Parameter *TabelleNum* zugewiesen ist, sowie den Namen der temporären Auswahl, definiert im optionalen Parameter *Name*.
Sind die Zeilen der Listbox mit der aktuellen Auswahl der Tabelle verknüpft, gibt der Parameter *Name* - sofern er übergeben ist - einen leeren String zurück.
Sind die Zeilen der Listbox mit der aktuellen Auswahl der Tabelle verknüpft, gibt der Parameter *Name* den Namen dieser temporären Auswahl zurück.

Basiert die Listbox auf einem Array, gibt *TabelleNum* -1 und Name - sofern übergeben - einen leeren String zurück.

LISTBOX INSERT COLUMN

LISTBOX INSERT COLUMN ({* ;} Objekt ; SpaltePos ; SpalteName ; SpalteVar ; KopfName ; KopfVar {; FußName ; FußVar})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, Ohne *: Variable
SpaltePos	Lange Ganzzahl	➡	Setzen der einzufügenden Spalte
SpalteName	String	➡	Name des Spaltenobjekts
SpalteVar	Array, Feld, Variable, Nil pointer	➡	Name der Spalte des Array, Feldes oder der Variablen
KopfName	String	➡	Name des Objekts Spaltentitel
KopfVar	Variable Ganzzahl, Nil pointer	➡	Variable des Spaltentitels
FußName	String	➡	Objektname des Spaltenfußteils
FußVar	Variable, Nil pointer	➡	Variable des Spaltenfußteils

Beschreibung

Der Befehl **LISTBOX INSERT COLUMN** fügt eine Spalte in die Listbox ein, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.
Die neue Spalte wird unmittelbar vor der Spalte eingefügt, die im Parameter *SpaltePos* angegeben ist. Ist *SpaltePos* größer als die Gesamtanzahl der Spalten, wird sie nach der letzten Spalte angefügt.

In *SpalteName* und *SpalteVar* übergeben Sie den Namen des Objekts und die Variable der eingefügten Spalte.

- Bei einer Listbox vom Typ Array muss der Variablenname zum Namen des Array passen, dessen Inhalt in der Spalte angezeigt wird. In einem dynamischen Kontext können Sie einen Zeiger **Nil (->[])** übergeben (siehe unten).
- Bei einer Listbox vom Typ Auswahl müssen Sie in *SpalteVar* ein Datenfeld oder eine Variable übergeben. Der Inhalt der Spalte ist dann der Wert des Datenfeldes oder der Variablen, der für jeden Datensatz der der Listbox zugewiesenen Auswahl bewertet wird. Dieser Inhalt ist nur möglich, wenn die Eigenschaft "Datenquelle" der Listbox die aktuelle oder die temporäre Auswahl ist. Weitere Informationen dazu finden Sie im Abschnitt **Einführung in Listboxen**. Sie können Datenfelder oder Variablen vom Typ String, Zahl, Datum, Zeit, Bild und Boolean verwenden.

Bei Listboxen, die auf einer Auswahl von Datensätzen basieren, können Sie über **LISTBOX INSERT COLUMN** einfache Elemente wie Datenfelder oder Variablen einfügen. Zur Verwaltung komplexer Ausdrücke, wie Formeln oder Methoden benötigen Sie den Befehl **LISTBOX INSERT COLUMN FORMULA**.
Es werden auch Listboxen vom Typ Collection oder Entity-Selection unterstützt. Da der Parameter *SpalteName* jedoch keine Ausdrücke akzeptiert, müssen Sie zum Zuweisen der Datenquelle den Befehl **LISTBOX SET COLUMN FORMULA** verwenden. In diesem Fall ist der Befehl **LISTBOX INSERT COLUMN FORMULA** besser geeignet.

Hinweis: Sie können in derselben Listbox nicht Felder bzw. Variablen (Datenquelle Auswahl) und Arrays (Datenquelle Array) miteinander kombinieren.

In *KopfName* und *KopfVar* übergeben Sie den Objektnamen und die Variable für den eingefügten Spaltentitel.

In *FußName* und *FußVar* können Sie den Objektnamen und die Variable des Fußteils der eingefügten Spalte übergeben.

Hinweis: Objektnamen müssen in einem Formular einmalig sein. Stellen Sie sicher, dass die in *SpalteName*, *KopfName* und *FußName* übergebenen Namen noch nicht verwendet wurden. Sonst wird die Spalte nicht erstellt und ein Fehler erscheint.

Dynamisches Einfügen

Ab 4D v14 R3 können Sie mit diesem Befehl Spalten dynamisch in Listboxen einfügen, wenn das Formular ausgeführt wird. 4D verwaltet automatisch die Definition der erforderlichen Variablen (Spalten, Fußteil und Kopfteil).

Dazu akzeptiert **LISTBOX INSERT COLUMN** einen Zeiger **Nil** (**->[]**) als Wert für die Parameter *SpalteVariable* (nur für Listboxen vom Typ Array), *KopfVariable* und *FußVariable*. In diesem Fall erstellt 4D beim Ausführen des Befehls die erforderlichen Variablen dynamisch. Weitere Informationen dazu finden Sie im Abschnitt **Dynamische Variablen**.

Beachten Sie, dass Variablen für Kopf- und Fußteil jeweils mit dem spezifischen Typ Text bzw. Lange Ganzzahl erstellt werden. Im Gegensatz dazu lassen sich Variablen für Spalten nicht beim Erstellen typisieren, da die Listbox für diese Variable verschiedene Arraytypen erlaubt (Array Text, Array Ganzzahl, etc.). Deshalb müssen Sie den Arraytyp manuell setzen (siehe Beispiel 3). Es ist wichtig, den Arraytyp vor Verwenden von Befehlen wie **LISTBOX INSERT ROWS** zu setzen, um neue Elemente in das Array einzufügen. Alternativ lässt sich auch der Befehl **APPEND TO ARRAY** verwenden, um gleichzeitig den Arraytyp zu setzen und das Einfügen von Elementen zu starten.

Beispiel 1

Folgender Code fügt am Ende der Listbox eine Spalte hinzu:

```
C_LONGINT(HeaderVarName;$Last;RecNum)
ALL RECORDS([Table 1])
$RecNum:=Records in table([Table 1])
ARRAY PICTURE(Picture;$RecNum)

$Last:=LISTBOX Get number of columns(*;"ListBox1")+1
LISTBOX INSERT COLUMN(*;"ListBox1";$Last;"ColumnPicture";Picture;
"HeaderPicture";HeaderVarName)
```

Beispiel 2

Folgender Code fügt an die rechte Seite der Listbox eine Spalte hinzu und weist ihr die Werte aus dem Datenfeld [Transport]Fees zu

```
$last:=LISTBOX Get number of columns(*;"ListBox1")+1
LISTBOX INSERT COLUMN(*;"ListBox1";$last;"FieldCol";[Transport]Fees;"HeaderName";HeaderVar)
```

Beispiel 3

Eine Spalte dynamisch in eine Listbox vom Typ Array setzen und ihren Kopfteil definieren:

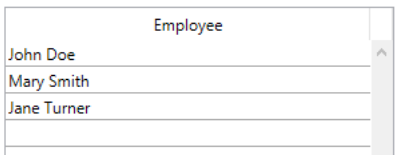
```
C_POINTER($NilPtr)
LISTBOX INSERT COLUMN(*;"MyListBox";1;"MyNewColumn";$NilPtr;"MyNewHeader";$NilPtr)
ColPtr:=OBJECT Get pointer(Object named;"MyNewColumn")
ARRAY TEXT(ColPtr->;10)
 //Kopfteil definieren
headprt:=OBJECT Get pointer(Object named;"MyNewHeader")
OBJECT SET TITLE(headprt->;"Eingefügter Kopfteil")
```

Beispiel 4

In einer Listbox vom Typ Collection eine Spalte hinzufügen:

```
 //Collection erstellen
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"))
```

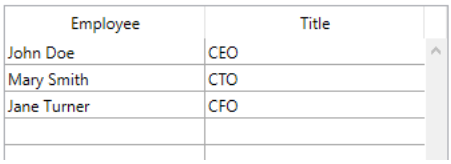
Der Spalteninhalt wird für jedes Element der Collection bewertet und mit dem Ausdruck *This.Employee* der Datenquelle referenziert:



Um eine Spalte mit den Job-Bezeichnungen hinzuzufügen, schreiben Sie:

```
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)
```

Die Spalte wird in der Listbox hinzugefügt:



LISTBOX INSERT COLUMN FORMULA

LISTBOX INSERT COLUMN FORMULA ({* ;} Objekt ; SpaltePosition ; SpalteName ; Formel ; Datentyp ; KopfName ; KopfVar {; FußName ; FußVar})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
SpaltePosition	Lange Ganzzahl	➡	Position der einzufügenden Spalte
SpalteName	String	➡	Name des Objekts Spalte
Formel	String	➡	Der Spalte zugeordnete 4D Formel
Datentyp	Lange Ganzzahl	➡	Typ des Formelergebnisses
KopfName	String	➡	Objektname des Spaltenkopfteils
KopfVar	Variable Ganzzahl, Nil pointer	➡	Variable des Spaltenkopfteils
FußName	String	➡	Objektname des Spaltenfußteils
FußVar	Variable, Nil pointer	➡	Variable des Spaltenfußteils

Beschreibung

Der Befehl **LISTBOX INSERT COLUMN FORMULA** fügt eine Spalte in die Listbox ein, definiert durch die Parameter *Objekt* und *.
Der Befehl arbeitet ähnlich wie der Befehl **LISTBOX INSERT COLUMN**. Der Unterschied ist, dass Sie damit auch eine Formel als Inhalt einer Spalte eingeben können. Eine Formel als Inhalt ist nur möglich, wenn die Eigenschaft „Datenquelle" der Listbox auf **aktuelle Auswahl** oder **temporäre Auswahl** bzw. **Collection oder Entity-Selection** gesetzt ist.
Weitere Informationen dazu finden Sie im Abschnitt **Einführung in Listboxen**.

Mit dem optionalen Parameter * ist der Parameter *Objekt* ein Objektname (String). Ohne * ist *Objekt* eine Variable. In diesem Fall übergeben Sie keinen String, sondern eine Variablenreferenz. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.
Die neue Spalte wird direkt vor der Spalte eingefügt, definiert durch den Parameter *SpaltePosition*. Ist die Nummer von *SpaltePosition* größer als die Gesamtanzahl der Spalten, wird sie an die letzte Spalte angefügt.

In *SpalteName* übergeben Sie den Objektnamen der eingefügten Spalte.

Der Parameter *Formel* kann einen beliebigen gültigen Ausdruck enthalten, z.B.:

- Eine Anweisung
- Eine Formel, die über den Formulareditor erstellt wurde
- Aufruf eines 4D Befehls
- Aufruf einer Projektmethode

Bei Aufruf des Befehls wird die Formel durchlaufen und dann ausgeführt.

Hinweis: Um Formeln unabhängig von der Anwendungssprache zu definieren, verwenden Sie die Funktion **Command name** (wenn Sie 4D Befehle aufrufen).

Mit dem Parameter *Datentyp* können Sie den Typ der Daten bestimmen, die nach Ausführen der Formel zurückgegeben wird. Sie müssen eine der folgenden Konstanten unter dem Thema **Feld und Variablentypen** übergeben:

Konstante	Typ	Wert
Is Boolean	Lange Ganzzahl	6
Is date	Lange Ganzzahl	4
Is picture	Lange Ganzzahl	3
Is real	Lange Ganzzahl	1
Is text	Lange Ganzzahl	2
Is time	Lange Ganzzahl	11

Entspricht das Ergebnis der Formel keinem der erwarteten Datentypen, wird ein Fehler erzeugt.

In den Parametern *KopfName* und *KopfVar* übergeben Sie den Objektnamen und die Variable des eingefügten Kopfteils der Spalte.

In den Parametern *FußName* und *FußVar* können Sie auch den Objektnamen und die Variable für den Fußteil der eingefügten Spalte übergeben. Lassen Sie den Parameter *FußVar* weg, verwendet 4D eine dynamische Variable.

Hinweis: Objektnamen müssen in einem Formular einmalig sein. Sie müssen sicherstellen, dass die in *SpalteName*, *KopfName* und *FußName* übergebenen Namen nicht anderweitig verwendet werden. Ansonsten wird die Spalte nicht erstellt und ein Fehler erzeugt.

Dynamisches Einfügen

Ab 4D v14 R3 können Sie mit diesem Befehl Spalten dynamisch in Listboxen einfügen, wenn das Formular ausgeführt wird. 4D verwaltet automatisch die Definition der erforderlichen Variablen (Spalten, Fußteil und Kopfteil).

Dazu akzeptiert **LISTBOX INSERT COLUMN FORMULA** einen Zeiger **Nil** (**->[]**) als Wert für die Parameter *KopfVariable* und *FußVariable*. In diesem Fall erstellt 4D beim Ausführen des Befehls die erforderlichen Variablen dynamisch. Weitere Informationen dazu finden Sie im Abschnitt **Dynamische Variablen**.
Beachten Sie, dass Variablen für Kopf- und Fußteil immer mit den spezifischen Typ Lange Ganzzahl bzw. Text erstellt werden.

Beispiel 1

Wir wollen rechts vor der Listbox eine neue Spalte hinzufügen mit einer Formel, die das Alter der Angestellten berechnet:

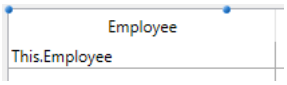
```
vAge:="Current date-[Employees]BirthDate)\365"
$last:=LISTBOX Get number of columns(*;"ListBox1")+1
LISTBOX INSERT COLUMN FORMULA(*;"ListBox1";$last;"ColFormula";vAge;ls real;"Age";HeaderVar)
```

Beispiel 2

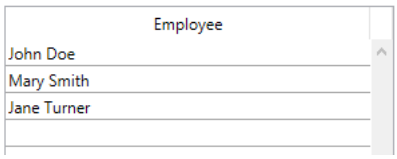
In einer Listbox vom Typ Collection eine Spalte hinzufügen:

```
 //Collection erstellen
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"))
```

Der Spalteninhalt wird für jedes Element der Collection bewertet und mit dem Ausdruck *This.Employee* der Datenquelle referenziert:



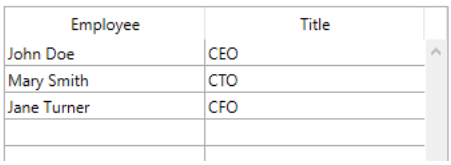
Ausführung:



Um eine Spalte mit den Job-Bezeichnungen hinzuzufügen, schreiben Sie:

```
LISTBOX INSERT COLUMN FORMULA(*;"EmpLB";2;"2nd Column";"This.JobTitle";ls text;"JTHeader";header2)
OBJECT SET TITLE(header2;"Title")
```

Die Spalte wird in der Listbox hinzugefügt:



LISTBOX INSERT ROWS

LISTBOX INSERT ROWS ({* ;} Objekt ; ZeilePosition {; AnzZeilen})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
ZeilePosition	Lange Ganzzahl	➡	Stelle der einzufügenden Zeile
AnzZeilen	Lange Ganzzahl	➡	Anzahl der einzufügenden Zeilen

Beschreibung

Der Befehl **LISTBOX INSERT ROWS** fügt eine oder mehrere Zeilen in der Listbox ein, definiert durch die Parameter *Objekt* und *.

Hinweis: Dieser Befehl funktioniert nur mit Listboxen, die auf Arrays basieren. Bei Verwenden von Listboxen, die auf Auswahlen basieren, hat er keine Auswirkung. Die Systemvariable OK gibt 0 (Null) zurück.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable, d.h. Sie übergeben keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Standardmäßig, d.h. ohne den Parameter *AnzZeilen* wird nur 1 Zeile eingefügt.

Die Zeile wird an der Position eingefügt, definiert durch *ZeilePosition*. An dieser Position wird automatisch eine neue Zeile in allen Arrays hinzugefügt, welche die Spalten der Listbox verwenden, unabhängig vom Typ und der Sichtbarkeit.

Ist der Wert in *ZeilePosition* höher als die Gesamtanzahl der Zeilen in der Listbox, wird die Zeile am Ende jedes Array hinzugefügt. Ist der Wert gleich 0 (Null), wird die Zeile zu Beginn jedes Array hinzugefügt. Ist der Wert negativ, führt der Befehl nichts aus.

LISTBOX MOVE COLUMN

LISTBOX MOVE COLUMN ({* ;} Objekt ; SpaltePosition)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist Objektname (String) Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *) der zu bewegenden Spalte
SpaltePosition	Lange Ganzzahl	➡	Neue Position der bewegten Spalte

Beschreibung

Der Befehl **LISTBOX MOVE COLUMN** verschiebt per Programmierung im Formular in der Ausführung die Spalte, definiert durch die Parameter *Objekt* und * (Anwendungsmodus). Das Originalformular, das im Designmodus erstellt wurde, wird nicht geändert.

Die Parameter *Objekt* und * bestimmen die Spalte zum Verschieben. Mit dem optionalen Parameter * ist *Objekt* ein Spaltenname (String). Ohne * ist *Objekt* eine Spaltenvariable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Die Spalte wird genau vor die Spalte gesetzt, die im Parameter *SpaltePosition* angegeben ist. Ist *SpaltePosition* größer als die Gesamtanzahl der Spalten, wird sie nach der letzten Spalte gesetzt.

Hinweis: Dieser Befehl führt nichts aus, wenn er auf die erste Spalte einer Listbox im hierarchischen Modus angewandt wird.

Der Befehl berücksichtigt die Spalteneigenschaften statisch und gesperrt, d.h. wollen Sie zum Beispiel eine statische Spalte verschieben, führt der Befehl nichts aus.

Diese Funktionalität war bereits in 4D im Anwendungsmodus vorhanden: Der Benutzer kann nicht-statische Spalten mit der Maus bewegen. Dieser Befehl erzeugt, im Gegensatz zum Verschieben von Spalten durch den Benutzer, nicht das Ereignis On Column Moved.

Beispiel

Sie wollen die 2. und 3. Spalte der Listbox tauschen:

```
LISTBOX MOVE COLUMN(*;"Spalte2";3)
```

LISTBOX MOVED COLUMN NUMBER

LISTBOX MOVED COLUMN NUMBER ({* ;} Objekt ; AltePosition ; NeuePosition)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
AltePosition	Lange Ganzzahl	⬅	Vorige Position der bewegten Spalte
NeuePosition	Lange Ganzzahl	⬅	Neue Position der bewegten Spalte

Beschreibung

Der Befehl **LISTBOX MOVED COLUMN NUMBER** gibt in *AltePosition* und *NeuePosition* jeweils eine Nummer zurück, welche die vorige bzw. die neue Position der bewegten Spalte in Listbox angibt, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable, d.h. Sie übergeben keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Dieser Befehl muss zusammen mit dem Formularereignis On column moved verwendet werden. Weitere Informationen dazu finden Sie unter der Funktion **Form event**.

Hinweis: Dieser Befehl berücksichtigt auch ausgeblendete Spalten.

LISTBOX MOVED ROW NUMBER

LISTBOX MOVED ROW NUMBER ({* ;} Objekt ; AltePosition ; NeuePosition)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
AltePosition	Lange Ganzzahl	⬅	Vorige Position der bewegten Zeile
NeuePosition	Lange Ganzzahl	⬅	Neue Position der bewegten Zeile

Beschreibung

Der Befehl **LISTBOX MOVED ROW NUMBER** gibt in *AltePosition* und *NeuePosition* jeweils eine Nummer zurück, welche die vorige bzw. die neue Position der bewegten Zeile in Listbox angibt, definiert durch die Parameter *Objekt* und *.

Hinweis: Sie können nur Zeilen in Listboxen vom Typ Array bewegen.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne * ist *Objekt* eine Variable. In diesem Fall übergeben Sie keinen String, sondern eine Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Dieser Befehl muss zusammen mit dem Formularereignis On row moved verwendet werden (siehe Funktion **Form event**).

Hinweis: Dieser Befehl berücksichtigt nicht den Status ein-/ausgeblendet von Zeilen in der Listbox.

LISTBOX SELECT BREAK

LISTBOX SELECT BREAK ({* ;} Objekt ; Zeile ; Spalte {; Aktion})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (string), ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Zeile	Lange Ganzzahl	➡	Nummer der Umbruchzeile
Spalte	Lange Ganzzahl	➡	Nummer der Umbruchspalte
Aktion	Lange Ganzzahl	➡	Auswahl Aktion

Beschreibung

Der Befehl **LISTBOX SELECT BREAK** ermöglicht in der Listbox, definiert durch die Parameter * und *Objekt*, Umbruchzeilen auszuwählen. Die Listbox muss im hierarchischen Modus sein. Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.
Umbruchzeilen werden zur Darstellung der Hierarchie hinzugefügt, sie entsprechen aber nicht vorhandenen Zeilen im Array. Um eine Umbruchzeile zum Auswählen zu bestimmen, müssen Sie die Zeilen- und Spaltennummer übergeben, die dem ersten Auftreten im entsprechenden Array entspricht. Der Befehl **LISTBOX GET CELL POSITION** gibt diese Werte zurück, wenn der Benutzer eine Umbruchzeile ausgewählt hat. Weitere Informationen dazu finden Sie im Abschnitt **Hierarchische Listboxen verwalten**.

Der Parameter *Aktion* kann die Auswahlaktion so setzen, dass sie ausgeführt wird, wenn in der Listbox bereits eine Auswahl von Umbruchzeilen vorhanden ist. Sie können einen Wert oder eine der nachfolgenden Konstanten unter dem Thema **Listbox** übergeben:

Konstante	Typ	Wert	Kommentar
lk add to selection	Lange Ganzzahl	1	Die ausgewählte Zeile wird der vorhandenen Auswahl hinzugefügt. Gehört die angegebene Zeile bereits zur Auswahl, führt die Konstante nichts aus.
lk remove from selection	Lange Ganzzahl	2	Die ausgewählte Zeile wird aus der vorhandenen Auswahl entfernt. Gehört die angegebene Zeile nicht zur Auswahl, führt die Konstante nichts aus.
lk replace selection	Lange Ganzzahl	0	Die gewählte Zeile wird zur neuen Auswahl und ersetzt die vorhandene Auswahl. Die Konstante hat dieselbe Wirkung wie Anklicken der Zeile durch den Benutzer (Das Ereignis On Clicked wird dagegen nicht generiert). Dies ist die Standardaktion, d.h. wenn der Parameter *Aktion* nicht verwendet wird.

Hinweise: Haben Sie für die Listbox die Option **Auswahlmarkierung ausblenden** markiert:

- Müssen Sie die Auswahlen in der Listbox über verfügbare Oberflächenoptionen sichtbar machen. Weitere Informationen dazu finden Sie im Abschnitt **Darstellung von Auswahlen anpassen**
- Können Sie für hierarchische Listboxen keine Umbruchzeilen hervorheben. Weitere Informationen dazu finden Sie im Abschnitt **Einschränkung bei hierarchischen Listboxen**

Beispiel

Wir verwenden die folgenden Arrays in einer Listbox:

Deutschland	Bayern	München	1300000
Deutschland	Bayern	Freising	45000
Deutschland	Bayern	Rosenheim	60000
Deutschland	Hessen	Frankfurt	675000
Deutschland	Hessen	Darmstadt	138000
Frankreich	Normandie	Caen	220000
Frankreich	Normandie	Deauville	4000
Frankreich	Bretagne	Brest	120000
Frankreich	Bretagne	Quimper	80000

Wir möchten die Umbruchzeile "Hessen" auswählen:

```
$row:=Find in array(T2;"Hessen")
$column:=2
LISTBOX COLLAPSE(*;"MeineListbox")  `alle Ebenen zuklappen
LISTBOX SELECT BREAK(*;"MeineListbox";$row;$column)
```

Hier ist das Ergebnis:

LISTBOX SELECT ROW

LISTBOX SELECT ROW ({* ;} Objekt ; ZeilePosition {; Aktion})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname(string), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
ZeilePosition	Lange Ganzzahl	➡	Nummer der zu wählenden Zeile
Aktion	Lange Ganzzahl	➡	Auswahlaktion

Beschreibung

Der Befehl **LISTBOX SELECT ROW** wählt die Zeile mit der in *PositionZeile* übergebenen Nummer in der Listbox, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable, d.h. Sie übergeben keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Mit dem optionalen Parameter *Aktion* definieren Sie die auszuführende Auswahlaktion, wenn es in der Listbox bereits eine Zeilenauswahl gibt. Sie können eine der vordefinierten Konstanten aus dem Thema **Listbox** übergeben:

Konstante	Typ	Wert	Kommentar
lk add to selection	Lange Ganzzahl	1	Die ausgewählte Zeile wird der vorhandenen Auswahl hinzugefügt. Gehört die angegebene Zeile bereits zur Auswahl, führt die Konstante nichts aus.
lk remove from selection	Lange Ganzzahl	2	Die ausgewählte Zeile wird aus der vorhandenen Auswahl entfernt. Gehört die angegebene Zeile nicht zur Auswahl, führt die Konstante nichts aus.
lk replace selection	Lange Ganzzahl	0	Die gewählte Zeile wird zur neuen Auswahl und ersetzt die vorhandene Auswahl. Die Konstante hat dieselbe Wirkung wie Anklicken der Zeile durch den Benutzer (Das Ereignis On Clicked wird dagegen nicht generiert). Dies ist die Standardaktion, d.h. wenn der Parameter *Aktion* nicht verwendet wird.

Entspricht der Parameter *PositionZeile* nicht exakt einer vorhandenen Zeilennummer, arbeitet der Befehl folgendermaßen:

- Ist *PositionZeile* <0, führt der Befehl nichts aus, unabhängig vom Wert im Parameter *Aktion*.
- Ist *PositionZeile* gleich 0 und enthält der Parameter *Aktion* die Konstante lk replace selection oder wird er nicht verwendet, werden alle Zeilen der Listbox ausgewählt. Enthält der Parameter *Aktion* die Konstante lk remove from selection, werden alle Zeilen der Listbox abgewählt.
- Ist der Wert von *PositionZeile* größer als die Gesamtanzahl der Zeilen in der Listbox, wird das zugewiesene Array vom Typ Boolean automatisch angepasst und dann die Auswahl ausgeführt. Diese Arbeitsweise bedeutet, dass Sie den Befehl LISTBOX SELECT ROW mit der Befehlen zur Standardverwaltung von Arrays verwenden (z.B. **APPEND TO ARRAY**), die nicht automatisch die unmittelbare Anpassung der Listbox auslösen.
  Nach Ausführung der Methode werden die Arrays synchronisiert: Wurde das Ursprungsarray der Listbox tatsächlich angepasst, wird die Aktion Auswahl ausgeführt. Andernfalls kehrt das der Listbox zugewiesene Array vom Typ Boolean zur anfänglichen Größe zurück und der Befehl führt nichts aus.

Hinweise:

- Soll die Listbox automatisch scrollen, um die gewählte Zeile auszuwählen, verwenden Sie den Befehl **OBJECT SET SCROLL POSITION**.
- Soll eine Zeile in den Bearbeitungsmodus wechseln, um Daten eingeben zu können, wählen Sie den Befehl **EDIT ITEM**.
- Gehört die in *PositionZeile* übergebene Nummer zu einer ausgeblendeten Zeile in der Listbox, wird die Zeile ausgewählt, aber nicht angezeigt.
- Haben Sie für die Listbox die Option **Auswahlmarkierung ausblenden** markiert, müssen Sie die Auswahlen der Listbox über verfügbare Oberflächenoptionen sichtbar machen. Weitere Informationen dazu finden Sie im Abschnitt **Darstellung von Auswahlen anpassen**.

LISTBOX SET ARRAY

LISTBOX SET ARRAY ({* ;} Objekt ; arrTyp ; arrPtr)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
arrTyp	Lange Ganzzahl	➡	Typ des Array
arrPtr	Zeiger	➡	Array zum Zuweisen der Eigenschaft

Beschreibung

Hinweis: Dieser Befehl funktioniert nur für Listboxen vom Typ Array.

Der Befehl **LISTBOX SET ARRAY** weist ein Array *arrTyp* der Listbox oder der Spalte der Listbox zu, definiert durch die Parameter *Objekt* und *.

Hinweis: Arrays mit Stilen, Farben, Hintergrundfarben oder Zeilenkontrollen auf Listboxen vom Typ Array lassen sich auch über die Eigenschaftenliste im Designmodus zuweisen.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.
In *Objekt* können Sie eine Listbox oder die Spalte einer Listbox angeben.

In *arrTyp* übergeben Sie den Arraytyp für die Listbox oder Spalte. Sie können eine der Konstanten unter dem Thema **Listbox** verwenden:

Konstante	Typ	Wert	Kommentar
lk background color array	Lange Ganzzahl	1	
lk control array	Lange Ganzzahl	3	
lk font color array	Lange Ganzzahl	0	
lk row height array	Lange Ganzzahl	4	(4D View Pro Lizenz ist erforderlich)
lk style array	Lange Ganzzahl	2	

Im Parameter *arrPtr* übergeben Sie einen Zeiger auf das Array zur Unterstützung des Eigenschaftstyps.

Beispiel 1

Das Array Schriftfarbe der 4. Spalte für die 10. Spalte wiederverwenden:

```
 // Zeiger auf das Array für Spalte 4 suchen
$Pointer:=LISTBOX Get array(*;"Col4";lk font color array)
 // Prüfen, ob vorhanden
If(Not(Nil($Pointer)))
 // auf Spalte 10 übertragen
   LISTBOX SET ARRAY(*;"Col10";lk font color array;$Pointer)
End if
```

Beispiel 2

Ein Array Zeilenhöhen für eine Listbox setzen:

```
LISTBOX SET ARRAY(*;"LB";lk row height array;->RowHeightArray)
```

Hinweis: Für die Eigenschaft **Zeilenhöhe Array** für Listboxen ist eine 4D View Pro Lizenz erforderlich. Weitere Informationen dazu finden Sie im Abschnitt **4D View Pro**.

LISTBOX SET AUTO ROW HEIGHT

LISTBOX SET AUTO ROW HEIGHT ({* ;} Objekt ; Selector ; Wert ; Einheit)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Selector	Lange Ganzzahl	➡	Zu erhaltender Wert der Höhe: lk row min height oder lk row max height
Wert	Lange Ganzzahl	➡	Wert für Mindest- oder Maximumzeilenhöhe
Einheit	Lange Ganzzahl	➡	Einheit der Höhe: 0 = Pixel, 1 = Zeilen

4D View Pro

Für diesen Befehl ist eine 4D View Pro Lizenz erforderlich. Ist sie nicht vorhanden, erscheint beim Ausführen des Formulars ein Fehler in der Listbox. Weitere Informationen dazu finden Sie im Abschnitt **4D View Pro Handbuch**.

Beschreibung

Der Befehl **LISTBOX SET AUTO ROW HEIGHT** ermöglicht, über den Parameter *Wert* die Mindest- oder Maximumzeilenhöhe in der Listbox zu setzen, definiert über die Parameter *Objekt* und *.

Hinweis: Dieser Befehl wird nur berücksichtigt, wenn für die Listbox die Eigenschaft automatische Zeilenhöhe markiert ist. Sie ist nur für auf Arrays basierende nicht-hierarchische Listboxen verfügbar. Weitere Informationen dazu finden Sie im Abschnitt **Automatische Zeilenhöhe**.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String. Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

In *Selector* übergeben Sie den gewünschten Typ des Werts. Sie können eine der Konstanten unter dem Thema **Listbox** verwenden:

Konstante	Typ	Wert
lk row max height	Lange Ganzzahl	33
lk row min height	Lange Ganzzahl	32

In *Wert* übergeben Sie den entsprechenden Wert zur gewählten *Einheit*.

Im Parameter *Einheit* können Sie eine der Konstanten unter dem Thema **Listbox** übergeben:

Konstante	Typ	Wert	Kommentar
lk lines	Lange Ganzzahl	1	Höhe ist eine Anzahl Zeilen. 4D berechnet die Zeilenhöhe nach dem Schrifttyp.
lk pixels	Lange Ganzzahl	0	Höhe ist eine Anzahl Pixel (Standard)

Hinweis: Der Befehl prüft nicht, ob die Werte passend sind. In Echtzeit wird jedoch bei Unstimmigkeiten für beide Werte der Mindestwert angewandt. Ist z.B. der Mindestwert 5 Zeilen und der Maximumwert 3 Zeilen (was inkonsistent ist), wird in diesem Fall als max. Zeilenhöhe für die Listbox 5 Zeilen verwendet.

Beispiel

Die Mindest- und Maximumhöhen einer Listbox über eine automatische Zeilenhöhe setzen:

```
LISTBOX SET AUTO ROW HEIGHT(*;"LB";lk row min height;60;lk pixels)  // 60 Pixel für Mindestwert
LISTBOX SET AUTO ROW HEIGHT(*;"LB";lk row max height;100;lk pixels)  //und 100 Pixel für Maximumwert
```

LISTBOX SET COLUMN FORMULA

LISTBOX SET COLUMN FORMULA ({* ;} Objekt ; Formel ; Datentyp)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable
Formel	String	➡	4D Formel, die den Spalten zugeordnet ist
Datentyp	Lange Ganzzahl	➡	Typ des Formelergebnisses

Beschreibung

Der Befehl **LISTBOX SET COLUMN FORMULA** ändert die der Spalte zugewiesene Formel für die Listbox, definiert durch die Parameter *Objekt* und *. Formeln lassen sich nur verwenden, wenn die Eigenschaft "Datenquelle" der Listbox entweder **Aktuelle Auswahl** oder **temporäre Auswahl** bzw. **Collection oder Entity-Selection** ist. Ist der Spalte keine Formel zugewiesen, gibt die Funktion einen leeren String zurück.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String. Dieser Parameter muss die Spalte einer Listbox angeben.

Der Parameter *Formel* kann ein beliebiger gültiger Ausdruck sein, z.B.:

- eine Anweisung
- eine Formel, die über den Formeleditor erstellt wird
- Aufruf eines 4D Befehls
- Aufruf einer Projektmethode

Bei Aufrufen dieses Befehls wird die Formel analysiert und dann ausgeführt.

Hinweis: Über die Funktion **Command name** können Sie Formeln, die 4D Befehle aufrufen, unabhängig von der Anwendungssprache definieren.

Der Parameter *Datentyp* bestimmt die Art der Daten, die sich durch Ausführen der Formel ergeben. In diesem Parameter übergeben Sie eine der Konstanten unter dem Thema **Feld und Variablentypen**. Passt das Formelergebnis nicht zum erwarteten Datentyp, wird ein Fehler erzeugt.

LISTBOX SET COLUMN WIDTH

LISTBOX SET COLUMN WIDTH ({* ;} Objekt ; Breite {; minBreite {; maxBreite}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Breite	Lange Ganzzahl	➡	Spaltenbreite (in Pixel)
minBreite	Lange Ganzzahl	➡	Mindestbreite der Spalte (in Pixel)
maxBreite	Lange Ganzzahl	➡	Maximale Breite der Spalte (in Pixel)

Beschreibung

Der Befehl **LISTBOX SET COLUMN WIDTH** ändert per Programmierung die Breite einer bzw. aller Spalten in der Listbox, definiert durch die Parameter *Objekt* und *. Das kann die Listbox selbst, eine Spalte oder ein Spaltentitel sein.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable, d.h. Sie übergeben keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Im Parameter *Breite* übergeben Sie die neue Breite (in Pixel) von *Objekt*:

- Definiert *Objekt* die Listbox, werden alle Spalten der Listbox in der Größe angepasst.
- Definiert *Objekt* eine Spalte oder einen Spaltentitel, wird nur das angegebene Element angepasst.

Mit den optionalen Parametern *minBreite* und *maxBreite* können Sie Einschränkungen für das manuelle Anpassen von Spalten setzen. Sie können in *minBreite* und *maxBreite* maximale und minimale Werte in Pixel eintragen. Soll der Benutzer nicht die Möglichkeit haben, die Spaltengröße zu verändern, übergeben Sie in *Breite*, *minBreite* und *maxBreite* denselben Wert.

LISTBOX SET FOOTER CALCULATION

LISTBOX SET FOOTER CALCULATION ({* ;} Objekt ; Berechnung)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Berechnung	Lange Ganzzahl	➡	Berechnung für den Fußteilbereich

Beschreibung

Der Befehl **LISTBOX SET FOOTER CALCULATION** setzt die automatische Berechnung, die dem Fußteil der Listbox, definiert durch die Parameter *Objekt* und *, zugewiesen ist.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Der Parameter *Objekt* kann folgendes festlegen:

- Variable oder Name eines Fußteilbereichs. In diesem Fall gilt der Befehl für diesen Bereich.
- Variable oder Name einer Spalte in der Listbox. In diesem Fall gilt der Befehl für den Fußteil dieser Spalte.
- Variable oder Name einer Listbox. In diesem Fall gilt der Befehl für alle Fußteile der Listbox.

Im Parameter *Berechnung* übergeben Sie eine der folgenden Konstanten unter dem Thema **Listbox Fußteil Berechnung**:

Konstante	Typ	Wert	Kommentar
Listbox footer std deviation	Lange Ganzzahl	7	Für Spalten vom Typ Zahl oder Zeit verwenden (nur für Arrays vom Typ Listboxen). Standardtyp des Berechnungsergebnis: Zahl
lk footer average	Lange Ganzzahl	6	Für Spalten vom Typ Zahl oder Zeit verwenden. Standardtyp des Ergebnisses: Zahl.
lk footer count	Lange Ganzzahl	5	Für Spalten vom Typ Zahl, Text, Datum, Zeit, Boolean oder Bild verwenden. Standardtyp des Berechnungsergebnis: Lange Ganzzahl
lk footer custom	Lange Ganzzahl	1	4D führt keine Berechnung durch. Die Variable im Fußteil muss per Programmierung berechnet werden. Standardtyp des Berechnungsergebnis: Typ der Variable
lk footer max	Lange Ganzzahl	3	Für Spalten vom Typ Zahl, Datum, Zeit oder Boolean verwenden. Standardtyp des Berechnungsergebnis: Spalte Array oder Feldtyp
lk footer min	Lange Ganzzahl	2	Für Spalten vom Typ Zahl, Datum, Zeit oder Boolean verwenden. Standardtyp des Berechnungsergebnis: Spalte Array oder Feldtyp
lk footer sum	Lange Ganzzahl	4	Für Spalten vom Typ Zahl, Zeit oder Boolean verwenden. Standardtyp des Berechnungsergebnis: Spalte Array oder Feldtyp.
lk footer sum squares	Lange Ganzzahl	9	Für Spalten vom Typ Zahl oder Zeit verwenden (nur für Arrays vom Typ Listboxen). Standardtyp des Berechnungsergebnis: Zahl
lk footer variance	Lange Ganzzahl	8	Für Spalten vom Typ Zahl oder Zeit verwenden (nur für Arrays vom Typ Listboxen). Standardtyp des Berechnungsergebnis: Zahl

Beachten Sie, dass vordefinierte Berechnungen alle Werte der Spalte berücksichtigen, d.h. auch Werte von ausgeblendeten Zeilen. Wollen Sie eine Berechnung nur auf sichtbare Zeilen begrenzen, müssen Sie die Konstante lk footer custom verwenden und eine eigene Berechnung durchführen.
Passt der Datentyp einer Spalte oder der Listbox (wenn *Objekt* die Listbox angibt) nicht zur definierten Berechnung, wird der Fußteil nicht geändert und Fehler 18 wird erzeugt. Enthält eine Spalte eine Formel, d.h. die Listbox ist vom Typ Auswahl, wird Fehler 10 erzeugt.

Hinweis: Variablen für Fußteile werden automatisch gemäß der Berechnungsart typisiert, die in der Eigenschaftenliste gesetzt wird (wenn sie nicht per Code typisiert werden). Weitere Informationen dazu finden Sie im Abschnitt **Eigenschaften für Fußteile der Listbox**. Entspricht der Datentyp der Variablen nicht dem vom Befehl **LISTBOX SET FOOTER CALCULATION** erwarteten Ergebnis, wird ein Typisierungsfehler generiert.
Beispiel: Bei einer Spalte, die Datum anzeigt, wird auch die Variable *Fußteil* als Datum typisiert. Wird nun in der Berechnung im Fußteil "Maximum" verwendet und führen Sie die Anweisung **LISTBOX SET FOOTER CALCULATION**(footer;lk footer count) aus, erscheint ein Fehler, da sich der erwartete Ergebnistyp (Lange Ganzzahl) vom aktuellen Datentyp der Variable unterscheidet.

LISTBOX SET FOOTERS HEIGHT

LISTBOX SET FOOTERS HEIGHT ({* ;} Objekt ; Höhe {; Einheit})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Höhe	Lange Ganzzahl	➡	Zeilenhöhe
Einheit	Lange Ganzzahl	➡	Einheit für Wert der Höhe: 0 oder weggelassen = Pixel, 1 = Zeilen

Beschreibung

Der Befehl **LISTBOX SET FOOTERS HEIGHT** ändert per Programmierung die Höhe der Fußzeile in der Listbox, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String. Sie können entweder die Listbox oder nur einen Fußteil der Listbox definieren.

Im Parameter *Höhe* setzen Sie die gewünschte Höhe. Standardmäßig, d.h. ohne den Parameter *Einheit* wird diese Höhe in Pixel gesetzt. Zum Wechseln der Einheit können Sie in *Einheit* eine der folgenden Konstanten unter dem Thema **Listbox** übergeben:

Konstante	Typ	Wert	Kommentar
lk lines	Lange Ganzzahl	1	Höhe ist eine Anzahl Zeilen. 4D berechnet die Zeilenhöhe nach dem Schrifttyp.
lk pixels	Lange Ganzzahl	0	Höhe ist eine Anzahl Pixel (Standard)

Weitere Informationen dazu finden Sie im Abschnitt **Gruppen Kopfteile und Fußteile** des Handbuchs *4D Designmodus*.

LISTBOX SET GRID

LISTBOX SET GRID ({* ;} Objekt ; Horizontal ; Vertikal)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Horizontal	Boolean	➡	Wahr = Zeigen, Falsch = Ausblenden
Vertikal	Boolean	➡	Wahr = Zeigen, Falsch = Ausblenden

Beschreibung

Der Befehl **LISTBOX SET GRID** zeigt in der Listbox, definiert durch die Parameter *Objekt* und *, die horizontalen/vertikalen Gitterlinien an bzw. blendet sie aus.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable, d.h. Sie übergeben keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

In *Horizontal* und *Vertikal* geben Sie Boolean Werte an. Bei Wahr werden die Gitterlinien angezeigt, bei Falsch werden sie ausgeblendet. Das Gitter wird standardmäßig angezeigt.

LISTBOX SET GRID COLOR

LISTBOX SET GRID COLOR ({* ;} Objekt ; Farbe ; Horizontal ; Vertikal)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Farbe	Lange Ganzzahl	➡	Wert der RGB Farbe
Horizontal	Boolean	➡	Farbe für horizontale Gitterlinien
Vertikal	Boolean	➡	Farbe für vertikale Gitterlinien

Beschreibung

Der Befehl **LISTBOX SET GRID COLOR** ändert die Farbe der Gitterlinien in der Listbox, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

In *Farbe* übergeben Sie die RGB Farbe. Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **OBJECT SET RGB COLORS**.

In *Horizontal* und *Vertikal* geben Sie an, für welche Gitterlinien die Farbe gesetzt wird:

- Übergeben Sie Wahr in *Horizontal*, gilt die Farbe für die horizontalen Gitterlinien. Bei Falsch wechselt die Farbe nicht.
- Übergeben Sie Wahr in *Vertikal*, gilt die Farbe für die vertikalen Gitterlinien. Bei Falsch wechselt die Farbe nicht.

LISTBOX SET HEADERS HEIGHT

LISTBOX SET HEADERS HEIGHT ({* ;} Objekt ; Höhe {; Einheit})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Höhe	Lange Ganzzahl	➡	Zeilenhöhe
Einheit	Lange Ganzzahl	➡	Einheit für Wert der Höhe: 0 oder weggelassen = Pixel, 1 = Zeilen

Beschreibung

Der Befehl **LISTBOX SET HEADERS HEIGHT** ändert per Programmierung die Höhe der Kopfzeile in der Listbox, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Sie können entweder die Listbox oder nur einen Kopfteil der Listbox definieren.

Im Parameter *Höhe* setzen Sie die gewünschte Höhe. Standardmäßig, d.h. ohne den Parameter *Einheit* wird diese Höhe in Pixel gesetzt. Zum Wechseln der Einheit können Sie in *Einheit* eine der folgenden Konstanten unter dem Thema **Listbox** übergeben:

Konstante	Typ	Wert	Kommentar
lk lines	Lange Ganzzahl	1	Höhe ist eine Anzahl Zeilen. 4D berechnet die Zeilenhöhe nach dem Schrifttyp.
lk pixels	Lange Ganzzahl	0	Höhe ist eine Anzahl Pixel (Standard)

Kopfteile müssen die vom Betriebssystem gesetzte Mindesthöhe berücksichtigen. Das sind 24 Pixel unter Windows, 17 Pixel auf Mac OS. Übergeben Sie im Parameter *Höhe* einen kleineren Wert, wird die jeweilige Mindesthöhe angewandt.

Weitere Informationen dazu finden Sie im Abschnitt **Gruppen Kopfteile und Fußteile** des Handbuchs *4D Designmodus*.

LISTBOX SET HIERARCHY

LISTBOX SET HIERARCHY ({* ;} Objekt ; hierarchisch {; Hierarchie})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
hierarchisch	Boolean	➡	Wahr = hierarchische Listbox, Falsch = nicht-hierarchische Listbox
Hierarchie	Array Zeiger	➡	Array mit Zeigern

Beschreibung

Der Befehl **LISTBOX SET HIERARCHY** setzt die Listbox, definiert durch die Parameter *Objekt* und *, im hierarchischen bzw. nicht-hierarchischen Modus.

Hinweis: Dieser Befehl funktioniert nur mit Listboxen, die auf Arrays basieren. Bei Listboxen, die auf Auswahlen basiert, führt er nichts aus.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Über den Boolean Parameter *hierarchisch* definieren Sie den Modus der Listbox:

- Übergeben Sie *Wahr*, erscheint die Listbox im hierarchischen Modus
- Übergeben Sie *Falsch*, erscheint die Listbox im nicht-hierarchischen Modus (Standard Array Modus)

Übergeben Sie eine Listbox im hierarchischen Modus, sind automatisch bestimmte Eigenschaften eingeschränkt. Weitere Informationen dazu finden Sie im Abschnitt **Hierarchische Listboxen verwalten**.

Mit dem Parameter *Hierarchie* definieren Sie die Arrays der Listbox zum Aufbau der Hierarchie. Lassen Sie diesen Parameter weg, passiert folgendes:

- Ist die Listbox bereits im hierarchischen Modus, führt der Befehl nichts aus.
- Ist die Listbox im nicht-hierarchischen Modus und wurde nie als hierarchisch deklariert, wird standardmäßig das erste Array für die Hierarchie verwendet.
- Ist die Listbox im nicht-hierarchischen Modus, wurde aber zuvor als hierarchisch deklariert, wird die letzte Hierarchie wiederhergestellt.

Beispiel

Definition der Arrays arrLand, arrBundesland und arrStadt als Hierarchie einer Listbox:

```
ARRAY POINTER($ArrHierarch;3)
$ArrHierarch{1}:=->arrLand  `Erste Umbruchebene
$ArrHierarch{2}:=->arrBundesland  `Zweite Umbruchebene
$ArrHierarch{3}:=->arrStadt  `Dritte Umbruchebene
LISTBOX SET HIERARCHY(*;"Meinelistbox";True;$ArrHierarch)
```

LISTBOX SET LOCKED COLUMNS

LISTBOX SET LOCKED COLUMNS ({* ;} Objekt ; AnzSpalten)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable
AnzSpalten	Lange Ganzzahl	➡	Anzahl der zu sperrenden Spalten

Beschreibung

Der Befehl **LISTBOX SET LOCKED COLUMNS** sperrt die *AnzSpalten* (beginnend mit der ersten Spalte links) in der Listbox, definiert durch die Parameter *Objekt* und *.

Gesperrte Spalten erscheinen im linken Teil der Listbox und scrollen nicht mit den restlichen Spalten der Listbox. Weitere Informationen dazu finden Sie im Abschnitt **Gesperrte Spalten und statische Spalten** des Handbuchs *4D Designmodus*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

In *AnzSpalten* können Sie einen beliebigen Wert zwischen 1 und der Gesamtzahl der Spalten minus 1 in der Listbox übergeben. Setzen Sie für eine Listbox mit x Spalten in *AnzSpalten* den Wert x-1, wird er automatisch auf den Wert x-1 reduziert.

Um die Spaltensperre aufzuheben, übergeben Sie in *AnzSpalten* 0 oder einen negativen Wert.

LISTBOX SET PROPERTY

LISTBOX SET PROPERTY ({* ;} Objekt ; Eigenschaft ; Wert)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Eigenschaft	Lange Ganzzahl	➡	Eigenschaft der Listbox oder Spalte
Wert	Lange Ganzzahl, String	➡	Wert der Eigenschaft

Beschreibung

Der Befehl **LISTBOX SET PROPERTY** setzt den *Wert* für die *Eigenschaft* der Listbox oder Spalte der Listbox, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Hinweis: Existiert keine Listbox oder Spalte der Listbox, definiert durch die Parameter *Objekt* und *, führt der Befehl nichts aus. Beachten Sie, dass kein Fehler ausgelöst wird.

Im Parameter *Eigenschaft* setzen Sie die Eigenschaft mit einer der folgenden Konstanten unter dem Thema **Listbox**:

Konstante	Typ	Wert	Kommentar
lk allow wordwrap	Lange Ganzzahl	14	Eigenschaft **Zeilenumbruch** Gilt für: Spalte* Mögliche Werte: • lk no: (0) • lk yes: (1)
lk auto row height	Lange Ganzzahl	31	Eigenschaft **Automatische Zeilenhöhe** Gilt für: Listbox oder Spalte Mögliche Werte: • lk yes • lk no
lk background color expression	Zeichenkette	22	**Nur 4D View Pro:** Für dieses Feature wird eine 4D View Pro Lizenz benötigt. Weitere Informationen dazu finden Sie im **4D View Pro Handbuch**. Eigenschaft **Hintergrundfarbe Ausdruck** für Listbox vom Typ Auswahl Gilt für: Listbox oder Spalte
lk column max width	Lange Ganzzahl	26	Eigenschaft **Maximale Breite** Gilt für: Spalte*
lk column min width	Lange Ganzzahl	25	Eigenschaft **Minimale Breite** Gilt für: Spalte*
lk column resizable	Lange Ganzzahl	15	Eigenschaft **Vergrößerbar** Gilt für: Spalte * Mögliche Werte: • lk no (0): • lk yes (1):
lk detail form name	Zeichenkette	19	Eigenschaft **Name Detailformular** für Listbox vom Typ Auswahl. Gilt für: Listbox
lk display type	Lange Ganzzahl	21	**Eigenschaft Typanzeige** für Spalten der Listbox vom Typ Zahl Gilt für: Spalte* Mögliche Werte: • lk numeric format: (0) Zeigt Werte im Zahlenformat an • lk three states checkbox: (1) Zeigt Werte als Kontrollkästchen mit drei Zuständen an
lk double click on row	Lange Ganzzahl	18	Eigenschaft **Doppelklick auf Zeile** für Listbox vom Typ Auswahl Gilt für: Listbox Mögliche Werte: • lk do nothing (0): Löst keine automatische Aktion aus • lk edit record (1): Zeigt den entsprechenden Datensatz im Lese-/Schreibmodus an • lk display record (2): Zeigt den entsprechenden Datensatz im Nur-Lesen Modus an
lk extra rows	Lange Ganzzahl	13	Eigenschaft **Zusätzliche Leerzeilen ausblenden** Gilt für für Listbox Mögliche Werte: • lk display: (0) • lk hide: (1)
lk font color expression	Zeichenkette	23	Eigenschaft **Schriftfarbe Ausdruck** für Listbox vom Typ Auswahl Gilt für: Listbox oder Spalte
lk font style expression	Zeichenkette	24	Eigenschaft **Stilausdruck** für Listbox vom Typ Auswahl Gilt für: Listbox oder Spalte
lk hide selection highlight	Lange Ganzzahl	16	Eigenschaft **Markierung Auswahl ausblenden** Gilt für: Listbox Mögliche Werte: • lk no: (0) • lk yes*:* (1)
lk highlight set	Zeichenkette	27	Eigenschaft **Markierung Menge** für Listbox vom Typ Auswahl Gilt für: Listbox
lk multi style	Lange Ganzzahl	30	Eigenschaft **Mehrfachstil** Gilt für: Spalte * Mögliche Werte: • lk no (0) • lk yes (1)
lk named selection	Zeichenkette	28	Eigenschaft **temporäre Auswahl** Gilt für: Listbox

Konstante	Typ	Wert	Kommentar
lk resizing mode	Lange Ganzzahl	11	Eigenschaft **Spaltenbreite Automatisch** Gilt für: Listbox Mögliche Werte: • lk manual: (0) • lk automatic: (1)
lk row height unit	Lange Ganzzahl	17	Eigenschaft **Einheit für Zeilenhöhe** Gilt für: Listbox Mögliche Werte: • lk lines (1) • lk pixels (0)
lk selection mode	Lange Ganzzahl	10	Eigenschaft **Auswahlmodus** Gilt für: Listbox Mögliche Werte: • lk none (0) • lk single (1) • lk multiple (2)
lk single click edit	Lange Ganzzahl	29	Eigenschaft **Einzelklick editieren** Gilt für: Listbox Mögliche Werte: • lk no (0) • lk yes (1)
lk sortable	Lange Ganzzahl	20	Eigenschaft **Sortierbar** Gilt für: Listbox Mögliche Werte: • lk no: (0) • lk yes: (1)
lk truncate	Lange Ganzzahl	12	Eigenschaft **Abkürzen mit Auslassungspunkten** Gilt für: Listbox oder Spalte Mögliche Werte: • lk without ellipsis: (0) • lk with ellipsis: (1)

* Diese Eigenschaft gilt für eine Spalte der Listbox; übergeben Sie hier eine Listbox als Parameter, setzt der Befehl **LISTBOX SET PROPERTY** diese *Eigenschaft* für alle Spalten der Listbox.

Hinweis: Übergeben Sie eine *Eigenschaft*, die nicht existiert bzw. für die angegebene Listbox oder Spalte nicht verfügbar ist, z.B. die Konstante lk font style expression für eine Listbox vom Typ Array, führt der Befehl nichts aus und es wird kein Fehler ausgelöst.

Im Parameter *Wert* übergeben Sie den gewünschten Wert für *Eigenschaft*.

Beispiel 1

Alle Spalten der Listbox "MyListbox" in der Größe anpassbar machen:

```
LISTBOX SET PROPERTY(*;"MyListbox";lk column resizable;lk yes)  //Alle Spalten von "MyListbox" werden auf Größe anpassbar gesetzt
```

Beispiel 2

Eine maximale Breite für die Spalte mit Namen "ProductNumber" setzen:

```
LISTBOX SET PROPERTY(*;"ProductNumber";lk column max width;200)  //Diese Spalte erhält die maximale Breite 200
```

LISTBOX SET ROW COLOR

LISTBOX SET ROW COLOR ({* ;} Objekt ; Zeile ; Farbe {; FarbeTyp})

Parameter	Typ		Beschreibung
*	Operator	⇒	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	⇒	Objektname (mit *) oder Variable (ohne *)
Zeile	Lange Ganzzahl	⇒	Zeilennummer
Farbe	Lange Ganzzahl	⇒	Wert für RGB Farbe
FarbeTyp	Lange Ganzzahl	⇒	Schriftfarbe(Standard) oder Hintergrundfarbe in der Listbox

Beschreibung

Hinweis: Dieser Befehl funktioniert nur für Listboxen vom Typ Array.

Der Befehl **LISTBOX SET ROW COLOR** setzt die Farbe für eine Zeile oder Zelle in der Listbox vom Typ Array, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

In *Objekt* können Sie eine Listbox oder die Spalte einer Listbox angeben:

- Gibt *Objekt* eine Listbox an, gilt der Befehl für die Zeile
- Gibt *Objekt* die Spalte einer Listbox an, gilt der Befehl für die Zelle am Schnittpunkt von Spalte und Zeile

In *Zeile* übergeben Sie die Nummer der Zeile, für welche die neue Farbe angewandt werden soll.

Hinweis: Dieser Befehl berücksichtigt nicht den Status ein-/ausgeblendet von Zeilen der Listbox.

In *Farbe* übergeben Sie einen RGB Farbwert. Weitere Informationen dazu finden Sie unter dem Befehl **OBJECT SET RGB COLORS**. Soll die Zeile die Farbe aus der nächsthöheren Ebene erben, können Sie in *Farbe* die Konstante lk inherited übergeben.

In *FarbeTyp* übergeben Sie die Konstanten lk background color oder lk font color, um anzugeben, ob die Farbe für den Hintergrund oder die Schrift in der Zeile gelten soll. Lassen Sie diesen Parameter weg, wird die Farbe auf die Schrift angewendet.

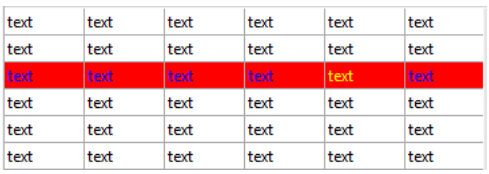
Dieser Befehl verändert die Werte im Array mit Farben, die bereits für die Spalte oder Listbox definiert wurden. Wurden diese Arrays noch nicht definiert, erstellt der Befehl dynamisch Arrays, auf die Sie über die neue Funktion **LISTBOX Get array** zugreifen können.

Bei Farbwerten, die Konflikte mit anderen Eigenschaften der Listbox hervorrufen, wie allgemeine Eigenschaften, Arrays mit farbigen Spalten, etc., gibt es bestimmte Prioritäten. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Beispiel

Wir wollen in einer Listbox vom Typ Array Farben für eine Zeile, sowie eine Zelle in dieser Zeile setzen:

```
// Schriftfarbe für Zelle definieren (gelb)
LISTBOX SET ROW COLOR(*;"Col5";3;0x00FFFF00)

// Hintergrund und Schriftfarbe für Zeile 3 definieren
// roter Hintergrund, blaue Schrift
LISTBOX SET ROW COLOR(*;"ListBox";3;0x00FF0000;lk background color)
LISTBOX SET ROW COLOR(*;"List Box";3;0x000000FF)
```



LISTBOX SET ROW FONT STYLE

LISTBOX SET ROW FONT STYLE ({* ;} Objekt ; Zeile ; Stil)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Zeile	Lange Ganzzahl	➡	Zeilennummer
Stil	Lange Ganzzahl	➡	Schriftstil

Beschreibung

Hinweis: Dieser Befehl funktioniert nur für Listboxen vom Typ Array

Der Befehl **LISTBOX SET ROW FONT STYLE** setzt einen Schriftstil für eine Zeile oder Zelle in der Listbox vom Typ Array, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

In *Objekt* können Sie eine Listbox oder die Spalte einer Listbox angeben:

- Gibt *Objekt* eine Listbox an, gilt der Befehl für die Zeile
- Gibt *Objekt* die Spalte einer Listbox an, gilt der Befehl für die Zelle am Schnittpunkt von Spalte und Zeile

In *Zeile* übergeben Sie die Nummer der Zeile, für welche der neue Stil angewandt werden soll.

Hinweis: Dieser Befehl berücksichtigt nicht den Status ein-/ausgeblendet von Zeilen der Listbox.

In *Stil* übergeben Sie einen Stilwert. Dafür verwenden Sie eine Konstante bzw. miteinander kombinierte Konstanten unter dem Thema **Schriftstile**:

Konstante	Typ	Wert
Bold	Lange Ganzzahl	1
Italic	Lange Ganzzahl	2
Plain	Lange Ganzzahl	0
Underline	Lange Ganzzahl	4

Wurde der Listbox oder Spalte ein Array Schriftstile zugewiesen, wird nur das zur Zeile passende Element geändert. Mit anderen Worten, dieser Befehl führt dasselbe aus wie ein Element im Array Schriftstile ändern. Ist der Listbox oder Spalte kein Array Schriftstile zugewiesen, wird es beim Aufrufen dieses Befehls dynamisch angelegt. Sie können mit der Funktion **LISTBOX Get array** auf dieses Array zugreifen.

Bei Farbwerten, die Konflikte mit anderen Eigenschaften der Listbox hervorrufen, wie allgemeine Eigenschaften, Arrays mit farbigen Spalten, etc., gibt es bestimmte Prioritäten. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Hinweis: Da Stil Arrays für Spalten Vorrang vor Stil Arrays für Listboxen haben, wird dieser Befehl für eine Listbox nur ausgeführt, wenn den Spalten kein Array zugewiesen wurde.

Beispiel

Wir haben eine Listbox vom Typ Array mit folgenden Merkmalen:

- Der Listbox ist ein Array Schriftstil zugewiesen (*ArrGlobalStyle*)
- Der Spalte 5 ist ein Array Schriftstil zugewiesen (*ArrCol5Style*)
- Die anderen Spalten haben keine Arrays Schriftstil

```
LISTBOX SET ROW FONT STYLE(*;"Col5";3;Bold)
 // entspricht 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)
 // entspricht 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

Nach der zweiten Anweisung ändern sich alle Zellen der dritten Zeile in Kursiv und Unterstrichen, mit Ausnahme der 5. Spalte. Sie bleibt in Fettschrift, da das Array Schriftstil für die Spalte Vorrang hat vor dem Array für Listbox.

LISTBOX SET ROW HEIGHT

LISTBOX SET ROW HEIGHT ({* ;} Objekt ; Zeile ; Höhe)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Zeile	Lange Ganzzahl	➡	Zeile der Listbox zum Setzen der Höhe
Höhe	Lange Ganzzahl	➡	Höhe der Listboxzeile

4D View Pro

Für diesen Befehl ist eine 4D View Pro Lizenz erforderlich. Ist sie nicht vorhanden, erscheint beim Ausführen des Formulars ein Fehler in der Listbox. Weitere Informationen dazu finden Sie im Abschnitt **4D View Pro Handbuch**.

Beschreibung

Der Befehl **LISTBOX SET ROW HEIGHT** ändert die Höhe der angegebenen *Zeile* in der Listbox, definiert durch die Parameter *Objekt* und *.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Existiert die angegebene *Zeile* nicht in der Listbox, führt der Befehl nichts aus.

Die in *Höhe* verwendete Einheit richtet sich nach der global für Listboxzeilen festgelegten Einheit, entweder in der Eigenschaftenliste oder durch einen vorangegangenen Aufruf von **LISTBOX SET ROWS HEIGHT**.

Der Befehl **LISTBOX SET ROW HEIGHT** ändert das in der Eigenschaftenliste angegebene Array für Zeilenhöhe - sofern vorhanden (Weitere Informationen dazu finden Sie im Abschnitt **Eigenschaft Zeilenhöhe Array** des Handbuchs *4D Designmodus)*. Andernfalls erstellt er dynamisch ein Array für Zeilenhöhe. Dieser Befehl zum Setzen individueller Zeilenhöhen führt zum gleichen Ergebnis wie Zuweisen eines Array für Zeilenhöhe über die Eigenschaftenliste; jedoch ist das Füllen eines Array für Zeilenhöhe mit Werten viel schneller, als diesen Befehl in einer Schleife aufzurufen, um die Zeilenhöhe eine nach der anderen für die Listbox zu setzen.

Wichtiger Hinweis: Wird der globale Befehl **LISTBOX SET ROWS HEIGHT** nachfolgend mit einer anderen Einheit als zuvor definiert aufgerufen, ersetzt und reinitialisiert der von diesem Befehl gesetzte Standardwert die Zeilenhöhen, die über **LISTBOX SET ROW HEIGHT** gesetzt wurden (siehe Beispiel 2).

Beispiel 1

Sie wollen die Höhe einzelner Zeilen in dieser Listbox ändern:

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

Führen Sie diesen Code aus:

```
//aktuelle Einheit ist Pixel
LISTBOX SET ROW HEIGHT(*;"listboxname";3;40) //Kuwait
LISTBOX SET ROW HEIGHT(*;"listboxname";7;14) //Serbia
```

... erhalten Sie folgendes Ergebnis:

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

Beispiel 2

Sie haben eine standardmäßige Zeilenhöhe gesetzt und dann über den Befehl **LISTBOX SET ROW HEIGHT** für Zeilenhöhe einzelne abweichende Werte:

```
LISTBOX SET ROWS HEIGHT(*;"listboxname";25;lk pixels)  // globale Höhe gesetzt in Pixel

LISTBOX SET ROW HEIGHT(*;"listboxname";1;30)  // Zeile 1: 30 Pixel
LISTBOX SET ROW HEIGHT(*;"listboxname";5;40)  // Zeile 5: 40 Pixel
LISTBOX SET ROW HEIGHT(*;"listboxname";11;50)  // Zeile 11: 50 Pixel
```

Wird später folgender Code ausgeführt …

```
LISTBOX SET ROWS HEIGHT(*;"listboxname";18;lk pixels)
```

…wird die globale Zeilenhöhe auf 18 Pixel gesetzt; da jedoch die Einheit gleichgeblieben ist, behalten die Zeilen 1, 5 und 11 ihre eigenen Werte für die Höhe, nämlich 30, 40 und 50 Pixel, wie oben durch den Befehl **LISTBOX SET ROW HEIGHT** definiert.

Wird dagegen später folgender Code ausgeführt …

```
LISTBOX SET ROWS HEIGHT(*;"listboxname";2;lk lines)
```

…werden die Zeilen 1, 5 und 11 zurückgesetzt auf die globale Zeilenhöhe von **LISTBOX SET ROWS HEIGHT** (hier 2 Zeilen), da die Einheit von Pixel in Zeilen gewechselt hat. Da es hier keine automatische Konvertierung gibt, wird beim Ändern der Einheit die Zeilenhöhe immer auf den neuen Standardwert reinitialisiert.

LISTBOX SET ROWS HEIGHT

LISTBOX SET ROWS HEIGHT ({* ;} Objekt ; Höhe {; Einheit})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String) Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Höhe	Lange Ganzzahl	➡	Zeilenhöhe (in Pixel)
Einheit	Lange Ganzzahl	➡	Einheit für Wert der Höhe: 0 oder weggelassen = Pixel, 1 = Zeilen

Beschreibung

Der Befehl **LISTBOX SET ROWS HEIGHT** verändert per Programmierung die Zeilenhöhe in der Listbox, definiert durch die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie keinen String, sondern die Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Standardmäßig, d.h. ohne den Parameter *Einheit*, wird die Höhe in Pixel ausgedrückt. Um die Einheit zu verändern, können Sie in *Einheit* eine der folgenden Konstanten unter dem Thema **Listbox** übergeben:

Konstante	Typ	Wert	Kommentar
lk lines	Lange Ganzzahl	1	Höhe ist eine Anzahl Zeilen. 4D berechnet die Zeilenhöhe nach dem Schrifttyp.
lk pixels	Lange Ganzzahl	0	Höhe ist eine Anzahl Pixel (Standard)

Hinweis: Weitere Informationen dazu finden Sie im Abschnitt **Höhe in Pixel oder Zeilen** des Handbuchs *4D Designmodus*.

LISTBOX SET STATIC COLUMNS

LISTBOX SET STATIC COLUMNS ({* ;} Objekt ; AnzSpalten)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
AnzSpalten	Lange Ganzzahl	➡	Anzahl Spalten, die statisch werden sollen

Beschreibung

Der Befehl **LISTBOX SET STATIC COLUMNS** setzt die Spalten in *AnzSpalten* auf statisch (beginnend mit der ersten Spalte links) in der Listbox, definiert durch die Parameter *Objekt* und *.

Statische Spalten lassen sich innerhalb der Listbox nicht bewegen.

Hinweis: Statische und gesperrte Spalten sind zwei unabhängige Funktionen. Weitere Informationen dazu finden Sie im Abschnitt **Gesperrte Spalten und statische Spalten** des Handbuchs *4D Designmodus*.

LISTBOX SET TABLE SOURCE

LISTBOX SET TABLE SOURCE ({* ;} Objekt ; TabelleNum | Name {; MarkierName})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), Ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
TabelleNum | Name	Lange Ganzzahl, String	➡	Nummer der Tabelle, deren aktuelle oder temporäre Auswahl verwendet werden soll.
MarkierName	String	➡	Name der Markiermenge

Beschreibung

Der Befehl **LISTBOX SET TABLE SOURCE** verändert die Quelle der Daten, die in der Listbox, definiert durch die Parameter * und *Objekt*, angezeigt werden.

Hinweis: Dieser Befehl lässt sich nur verwenden, wenn die Eigenschaft „Datenquelle" der Listbox auf **Aktuelle Auswahl** oder **Temporäre Auswahl** gesetzt ist. Weitere Informationen dazu finden Sie im Abschnitt **Einführung in Listboxen**. Er führt nichts aus, wenn Sie ihn mit einer Listbox vom Typ Array, Collection oder Entity-Selection verwenden.

Übergeben Sie den optionalen Parameter *, ist der Parameter *Objekt* ein Objektname (String). Ohne * ist *Objekt* eine Variable. In diesem Fall übergeben Sie keinen String, sondern eine Variablenreferenz. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

Übergeben Sie im Parameter *TabelleNum* eine Tabellennummer, wird die Listbox mit den Daten der Datensätze in der aktuellen Auswahl der Tabelle gefüllt.
Übergeben Sie im Parameter *Name* eine temporäre Auswahl, wird die Listbox mit den Daten der Datensätze aus der temporären Auswahl gefüllt.

Der optionale Parameter *MarkierName* weist der Listbox eine Markiermenge zu. Sie verwaltet das Markieren von Datensätzen durch den Benutzer in der Listbox.

Enthält die Listbox bereits Spalten, wird ihr Inhalt nach Ausführen des Befehls aktualisiert.

Hinweis: Dieser Befehl läuft zur Optimierung asynchron ab, d.h. die Quelle der Listbox wird erst nach vollständiger Ausführung der Methode geändert, in welcher der Befehl aufgerufen wird.

LISTBOX SORT COLUMNS

LISTBOX SORT COLUMNS ({* ;} Objekt ; SpaltenNr ; Richtung {; SpaltenNr2 ; Richtung2 ; ... ; SpaltenNrN ; RichtungN})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist Objektname (String), Ohne *: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
SpaltenNr	Lange Ganzzahl	➡	Nummer(n) der zu sortierenden Spalte(n)
Richtung	Operator	➡	">" Aufsteigend sortieren "<" Absteigend sortieren

Beschreibung

Der Befehl **LISTBOX SORT COLUMNS** sortiert die Zeilen der Listbox, definiert durch die Parameter *Objekt* und *, nach einem oder mehreren Spaltenwerten.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname ist. Sonst ist es eine Variable, d.h. Sie übergeben keinen String, sondern eine Referenz auf eine Variable. Weitere Informationen zu Objektnamen finden Sie im Abschnitt **Objekteigenschaften**.

In *SpaltenNr* übergeben Sie die Variable, d.h. die Nummer der zu sortierenden Spalte. Sie können jede Art von Array-Daten verwenden bis auf Bilder und Zeiger.

In *Richtung* übergeben Sie das Symbol > oder < für die Sortierrichtung. Enthält *Richtung* das Symbol "größer als" (>), wird in aufsteigender Richtung sortiert. Enthält *Richtung* das Symbol "kleiner als" (<), wird in absteigender Reihenfolge sortiert.

Sie können auch mehrstufig sortieren: Dazu übergeben Sie soviele Paare *SpaltenNr*; *Richtung* wie erforderlich. Die Ebene wird durch die Position des Parameters im Aufruf definiert.

Gemäß dem Prinzip für Listbox Operationen werden die Spalten synchronisiert, d.h. das Sortieren einer Spalte wirkt sich automatisch auf alle anderen Spalten des Objekts aus.

Mathematische Funktionen

- Abs
- Arctan
- Cos
- Dec
- Euro converter
- Exp
- Int
- Log
- Mod
- Random
- Round
- SET REAL COMPARISON LEVEL
- Sin
- Square root
- Tan
- Trunc

Abs

Abs (Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	→	Umzuwandelnder Wert
Funktionsergebnis	Zahl	↩	Absoluter Wert von Wert

Beschreibung

Die Funktion **Abs** gibt den absoluten Wert von *Wert* zurück. Mit anderen Worten, ist der Wert in *Wert* negativ, gibt **Abs** den positiven Wert von *Wert* zurück, ist *Wert* positiv, wird der unveränderte Wert zurückgegeben.

Beispiel

Folgendes Beispiel gibt den absoluten Wert von -10,3 zurück, also 10,3:

```
vlVector:=Abs(-10,3)
```

Arctan

Arctan (Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	⇒	Wert, für den der Winkel im Bogenmaß errechnet werden soll
Funktionsergebnis	Zahl	⇐	Winkel, angegeben im Bogenmaß

Beschreibung

Die Funktion **Arctan** gibt den Arcustangens (Winkel, angegeben im Bogenmaß) des übergebenen Winkels zurück.

Hinweis: 4D bietet die vordefinierten Konstanten Pi, Degree und Radian. Pi gibt den Pi-Wert (3,14159...) zurück, Degree den Wert von 1 Grad angegeben im Bogenmaß (0,01745...) und Radian den Wert von 1 Radiant angegeben in Grad (57,29577...).

Beispiel

Folgendes Beispiel zeigt den Wert von Pi:

```
ALERT("Pi ist gleich: "+String(Arctan(1)*4))
```

Cos

Cos (Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	➡	Winkel, angegeben im Bogenmaß
Funktionsergebnis	Zahl	⮌	Cosinus des Werts

Beschreibung

Die Funktion **Cos** gibt den Cosinus des im Bogenmaß angegebenen Winkels zurück.

Hinweis: 4D bietet die vordefinierten Konstanten Pi, Degree und Radian. Pi gibt den Pi-Wert (3,14159...) zurück, Degree den Wert von 1 Grad angegeben im Bogenmaß (0,01745...) und Radian den Wert von 1 Radiant angegeben in Grad (57,29577...).

Dec

Dec (Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	⇒	Umzuwandelnder Wert
Funktionsergebnis	Zahl	⮌	Dezimalstellen von Wert

Beschreibung

Die Funktion **Dec** gibt die Nachkommastellen des Wertes *Wert* zurück. Das Ergebnis ist immer positiv oder null.

Beispiel

Folgendes Beispiel nimmt den als Zahl ausgedrückten Geldwert und teilt in auf in Euro und Cent. Ist *vrAmount* z.B. 7,31, ergibt *vlEuro* 7 und *vlCent* 31:

```
vlEuro:=Int(vrAmount) ` Erhält Euro
vlCent:=Dec(vrAmount)*100 ` Erhält Stellen nach dem Komma
```

Euro converter

Euro converter (Wert ; VonWährung ; InWährung) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	➡	Umzurechnender Wert
VonWährung	String	➡	Code der Währung, in welcher der Wert angezeigt wird
InWährung	String	➡	Code der Währung, in welche der Wert umzurechnen ist
Funktionsergebnis	Zahl	⬅	Umgerechneter Wert

Beschreibung

Die Funktion **Euro converter** rechnet jeden Wert in einer für den Euro relevanten Währung in Euro um und umgekehrt.
Es gibt folgende Möglichkeiten:

- Eine nationale Währung in Euro umrechnen,
- Euro in eine nationale Währung umrechnen,
- Eine nationale Währung in eine andere nationale Währung umrechnen. In diesem Fall wird erst nach den festgelegten Kursen in Euro umgerechnet. Wollen Sie beispielsweise belgische Franc in Deutsche Mark umrechnen, führt 4D folgende Berechnung durch: Belgische Franc -> Euro -> Deutsche Mark.

Im ersten Parameter übergeben Sie den umzurechnenden Wert.
Der zweite Parameter gibt den Code der Währung an, in welcher der Wert angezeigt wird.
Der dritte Parameter gibt den Code der Währung an, in welche der Wert umgerechnet werden soll.
4D bietet zur Definition eines Währungscodes unter dem Thema **Euro Währungen** folgende vordefinierten Konstanten an:

Konstante	Typ	Wert
Austrian Schilling	Zeichenkette	ATS
Belgian Franc	Zeichenkette	BEF
Deutsche Mark	Zeichenkette	DEM
Euro	Zeichenkette	EUR
Finnish Markka	Zeichenkette	FIM
French Franc	Zeichenkette	FRF
Greek Drachma	Zeichenkette	GRD
Irish Pound	Zeichenkette	IEP
Italian Lira	Zeichenkette	ITL
Luxembourg Franc	Zeichenkette	LUF
Netherlands Guilder	Zeichenkette	NLG
Portuguese Escudo	Zeichenkette	PTE
Spanish Peseta	Zeichenkette	ESP

4D rundet das Ergebnis der Umrechnung bei Bedarf automatisch auf zwei Stellen nach dem Komma — davon ausgenommen ist die Umwandlung in italienische Lira, belgische und luxemburgische Franc, sowie spanische Pesetas. Hier behält 4D keine Stellen nach dem Komma, da das Ergebnis eine Ganzzahl ist.
Die Umrechungskurse für Euro und die Währungen der 11 beteiligten Länder sind folgendermaßen festgelegt:

Währung	Wert für 1 Euro
Austrian Schilling	13,7603
Belgian Franc	40,3399
Deutsche Mark	1,95583
Finnish Markka	5,94573
French Franc	6,55957
Greek drachma	340,750
Irish Pound	0,787564
Italian Lira	1936,27
Luxembourg Franc	40,3399
Netherlands Guilder	2,20371
Portuguese Escudo	200,482
Spanish Peseta	166,386

Beispiel

Mit diesem Befehl können Sie z.B. folgende Umrechnungen ausführen:

```
$value:=10000  `Wert in französischen Franc
  `Konvertiere den Wert in Euro
$InEuros:=Euro converter($value;French Franc;Euro)
  `Konvertiere den Wert in italienische Lire
$InLires:=Euro converter($value;French Franc;Italian Lira)
```

Exp

Exp (Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	⇒	Positive Zahl außer Null
Funktionsergebnis	Zahl	⇐	Natürliche Exponentialfunktion des Werts

Beschreibung

Exp gibt den Exponent von *Wert*, d.h. den Wert eZahl zurück. **Exp** ist die Umkehrung der Funktion **Log**.
Hinweis: 4D bietet die vordefinierte Konstante e number (2.71828...).

Beispiel

Folgendes Beispiel weist *vrE* den Exponent von 1 zu (der Logarithmus von *vrE* ist 1):

```
vrE:=Exp(1) ` vrE erhält 2.17828...
```

Int

Int (Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	→	Umzuwandelnder Wert
Funktionsergebnis	Zahl	↩	Ganzzahl von Wert

Beschreibung

Die Funktion **Int** gibt den auf die niedrigere Ganzzahl gerundeten Wert von *Wert* zurück.

Beispiel

Folgendes Beispiel zeigt, wie **Int** für positive und negative Zahlen funktioniert. Beachten Sie, dass der Dezimalanteil der Zahl entfernt wird:

```
vlResult:=Int(123,4)  ` vlResult erhält 123
vlResult:=Int(-123,4)  ` vlResult erhält –124
```

Log

Log (Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	➡	Positive Zahl ungleich Null
Funktionsergebnis	Zahl	⮌	Logarithmus der Zahl

Beschreibung

Die Funktion **Log** gibt den natürlichen Logarithmus von *Wert* zurück.
Log ist die Umkehrung der Funktion **Exp**.

Hinweis: 4D liefert die vordefinierte Konstante e number (2,71828...).

Beispiel

Folgender Code ergibt 1:

```
ALERT(String(Log(Exp(1))))
```

Mod

Mod (Wert ; Divisor) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Lange Ganzzahl	⇒	Zu dividierende Zahl
Divisor	Lange Ganzzahl	⇒	Divisor
Funktionsergebnis	Zahl	⇐	Gibt den Restwert zurück

Beschreibung

Die Funktion **Mod** gibt als Ergebnis den ganzzahligen Rest der Division aus *Wert* durch *Divisor* zurück.
Statt dieser Funktion können Sie auch den Operator % verwenden. Er liefert jedoch nur für Ganzzahlen und Lange Ganzzahlen gültige Ergebnisse. Für Werte vom Typ Zahl müssen Sie die Funktion **Mod** einsetzen. Weitere Informationen dazu finden Sie im Abschnitt **Numerische Operatoren**.

Hinweise:

- **Mod** lässt Werte vom Typ Ganzzahl, Lange Ganzzahl und Zahl zu. Werte vom Typ Zahl werden jedoch zuerst gerundet.
- Vorsicht ist geraten, wenn Sie **Mod** mit sehr hohen Zahlen verwenden, d.h. größer als 2^31. In diesem Fall kann die Ausführung die Grenzen der Berechnungskapazität von Standardprozessoren erreichen.

Beispiel

Folgendes Beispiel weist der Variablen je nach Argument einen anderen Wert zu:

```
vlResult:=Mod(3;2)  ` vlResult erhält 1
vlResult:=Mod(4;2)  ` vlResult erhält 0
vlResult:=Mod(3,5;2)  ` vlResult erhält 0
```

Random

Random -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Zufallszahl

Beschreibung

Die Funktion **Random** gibt eine Ganzzahl zwischen 0 und 32767 (einschließlich) zurück.
Der Bereich für die Zufallszahl wird mit folgender Formel definiert:

```
(Random%(vEnd-vStart+1))+vStart
```

Dabei ist *vStart* die erste Zahl, *vEnd* die letzte Zahl des Bereichs.

Beispiel

Folgendes Beispiel weist der Variablen *vlResult* einen Wert zwischen 10 und 30 zu:

```
vlResult:=(Random%21)+10
```

Round

Round (Wert ; Dezimalstellen) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	→	zu rundender Wert
Dezimalstellen	Lange Ganzzahl	→	Zu rundende Dezimalstellen
Funktionsergebnis	Zahl	↩	Wert gerundet auf die in Dezimalstellen angegebene Anzahl

Beschreibung

Die Funktion **Round** gibt den gerundeten Wert von *Wert* bis auf die in *Dezimalstellen* angegebene Anzahl zurück.

- Ist *Dezimalstellen* positiv, werden die Stellen nach dem Komma gerundet.
- Ist *Dezimalstellen* negativ, wird die ganze Zahl gerundet.

Werte zwischen 5 und 9 werden aufgerundet, Werte zwischen 0 und 4 werden abgerundet.

Beispiel

Folgendes Beispiel weist der Variablen je nach Argument einen anderen Wert zu:

```
vlResult:=Round(16,857;2) ` vlResult erhält 16,86
vlResult:=Round(32345,67;-3) ` vlResult erhält 32000
vlResult:=Round(29,8725;3) ` vlResult erhält 29,873
vlResult:=Round(-1,5;0) ` vlResult erhält –2
```

SET REAL COMPARISON LEVEL

SET REAL COMPARISON LEVEL (Epsilon)

Parameter	Typ		Beschreibung
Epsilon	Zahl	⇒	Epsilon Wert, um Zahlen miteinander auf Gleichheit zu prüfen

Beschreibung

Der Befehl **SET REAL COMPARISON LEVEL** setzt den Epsilon Wert von 4D, um Werte und Ausdrücke vom Typ Zahl miteinander auf Gleichheit zu prüfen.

Ein Rechner kann nur annähernde Werte berechnen; das sollte beim Vergleichen von Werten berücksichtigt werden. Beim Vergleichen von Werten vom Typ Zahl kann 4D testen, ob der Unterschied zwischen zwei Werten einen bestimmten Wert nicht überschreitet. Dieser Wert wird Epsilon genannt. Er funktioniert folgendermaßen:

Wir gehen von den beiden Werten *a* und *b* vom Typ Zahl aus. Ist *Abs(a-b)* größer als Epsilon, sind die Werte ungleich; Ist *Abs(a-b)* kleiner als Epsilon, sind die Werte gleich.

4D setzt Epsilon standardmäßig auf 10 hoch minus 6 (10^-6). Beispiele:

- *0.00001=0.00002* gibt Falsch zurück, da die Differenz *0.00001* größer ist als *10^-6*.
- *0.000001=0.000002* gibt Wahr zurück, da die Differenz *0.000001* nicht größer ist als *10^-6*.
- *0.000001=0.000003* gibt Falsch zurück, da die Differenz *0.000002* größer ist als *10^-6.*

Mit **SET REAL COMPARISON LEVEL** können Sie den Epsilon-Wert je nach Bedarf erhöhen oder verringern.

Sie benötigen diesen Befehl natürlich nur, wenn Sie nicht mit dem Standardwert von Epsilon arbeiten.

WICHTIG: Der Epsilon-Wert gilt nur für Zahlenvergleiche im Bezug auf Gleichheit. Er hat keine Auswirkung auf andere Vergleiche bzw. die Anzeige von Werten vom Typ Zahl.

Hinweis: SET REAL COMPARISON LEVEL hat keine Auswirkung auf Such- und Sortierläufe, die mit Feldern vom Typ Zahl durchgeführt werden. Er gilt nur für die 4D Programmiersprache.

Sin

Sin (Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	➡	Winkel, angegeben im Bogenmaß
Funktionsergebnis	Zahl	⮌	Sinus des Werts

Beschreibung

Die Funktion **Sin** gibt den Sinus des im Bogenmaß angegebenen Winkels zurück.

Hinweis: 4D bietet die vordefinierten Konstanten Pi, Degree und Radian. Pi gibt den Pi-Wert (3,14159...) zurück, Degree den Wert von 1 Grad angegeben im Bogenmaß (0,01745...) und Radian den Wert von 1 Radiant angegeben in Grad (57,29577...).

Square root

Square root (Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	➡	Wert, dessen Quadratwurzel berechnet wird
Funktionsergebnis	Zahl	➲	Quadratwurzel des Wertes

Beschreibung

Die Funktion **Square root** gibt die Quadratwurzel von *Wert* zurück.

Beispiel 1

Die Zeile:

```
$vrSquareRootOfTwo :=Square root(2)
```

weist der Variablen *$vrSquareRootOfTwo* den Wert *1,414213562373* zu.

Beispiel 2

Folgende Methode gibt die Hypotenuse des rechtwinkligen Dreiecks zurück, dessen beide Schenkel als Parameter übergeben wurden:

```
  ` Hypotenuse method
  ` Hypotenuse ( real ; real ) -> real
  ` Hypotenuse ( legA ; legB ) -> Hypotenuse
C_REAL($0;$1;$2)
$0:=Square root(($1^2)+($2^2))
```

Hypotenuse (4;3) gibt 5 zurück.

Tan

Tan (Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	⇒	Tangens zum Winkel, angegeben im Bogenmaß
Funktionsergebnis	Zahl	⇐	Tangens des Werts

Beschreibung

Die Funktion **Tan** gibt die Tangente des im Bogenmaß angegebenen Winkels zurück.

Hinweis: 4D bietet die vordefinierten Konstanten Pi, Degree und Radian. Pi gibt den Pi-Wert (3,14159...) zurück, Degree den Wert von 1 Grad angegeben im Bogenmaß (0,01745...) und Radian den Wert von 1 Radiant angegeben in Grad (57,29577...).

Trunc

Trunc (Wert ; Dezimalstellen) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Wert	Zahl	➡	Abzuschneidender Wert
Dezimalstellen	Lange Ganzzahl	➡	Anzahl der Dezimalstellen
Funktionsergebnis	Zahl	↩	Wert abgeschnitten auf die in Dezimalstellen angegebene Anzahl

Beschreibung

Die Funktion **Trunc** gibt den "abgeschnittenen" Wert von *Wert* bis auf die im Parameter *Dezimalstellen* angegebene Stelle genau zurück. Beim Abschneiden wird immer auf die niedrigere Zahl gerundet.

- Ist *Dezimalstellen* positiv, werden die Stellen nach dem Komma abgeschnitten.
- Ist *Dezimalstellen* negativ, wird die Funktion auf die ganze Zahl angewandt.

Beispiel

Folgendes Beispiel weist der Variablen je nach Argument einen anderen Wert zu:

```
vlResult:=Trunc(216,897;1) ` vlResult erhält 216,8
vlResult:=Trunc(216,897;-1) ` vlResult erhält 210
vlResult:=Trunc(-216,897;1) ` vlResult erhält –216,9
vlResult:=Trunc(-216,897;-1) ` vlResult erhält –220
```

Mehrfachstil Text

Das Kapitel "**Mehrfachstil Text**" enthält die 4D Befehle zum Einsetzen und Verwalten von Textbereichen mit Mehrfachstil, auch bekannt als *Rich Text Bereiche*.

Sie erhalten einen Rich Text Bereich, wenn Sie in der Eigenschaftenliste die Option Mehrfachstil aktivieren (siehe Abschnitt **Text mit Stil (Rich Text)** im Handbuch *4D Designmodus*). Über die Funktion **OBJECT Is styled text** im Kapitel "**Objekte (Formulare)**" können Sie prüfen, ob ein Textbereich im Modus Mehrfachstil ist oder nicht.

4D Write Pro Areas

Viele Befehle aus dem Kapitel "**Mehrfachstil Text**" unterstützen 4D Write Pro Areas. Weitere Informationen dazu finden Sie im Abschnitt **Befehle aus dem Kapitel Mehrfachstil Text verwenden** des *4D Write Pro Handbuchs*.

- Programmierhinweise
- Unterstützte Tags
- 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

Programmierhinweise

Befehle zum Verwalten von Textobjekten

Befehle, die zum Verwalten von Textobjekten per Programmierung dienen, berücksichtigen keine im Text integrierten Stil Tags. Sie verfahren mit angezeigtem Text wie in bisherigen Versionen von 4D. Das gilt für folgende Befehle:

- Kapitel **Benutzeroberfläche**
- 4D Befehl **HIGHLIGHT TEXT**
- 4D Befehl **GET HIGHLIGHT**

Beachten Sie, dass Sie bei Verwendung dieser Befehle mit Befehlen zum Verwalten von Zeichenketten die Formatierungszeichen mit denen der 4D Funktion **ST Get plain text** filtern müssen:

```
HIGHLIGHT TEXT([Products]Notes;1;Length(ST Get plain text([Products]Notes))+1)
```

- Kapitel **Objekte (Formulare)**
  Die Befehle zum Verändern des Stils von Objekten (z.B. 4D Befehl **OBJECT SET FONT**) gelten für das gesamte Objekt und nicht nur für die Auswahl. Beachten Sie, dass bei der Ausführung des Befehls bei einem Objekt, das keinen Fokus hat, die Änderung gleichzeitig auf das Objekt, d.h. den Textbereich, und seine zugeordnete Variable angewandt wird. Hat das Objekt den Fokus, wird die Änderung für das Objekt, jedoch nicht für die zugeordnete Variable ausgeführt. Die Änderung wird nur auf die Variable angewandt, wenn das Objekt den Fokus verliert. Berücksichtigen Sie dieses Prinzip beim Programmieren von Textbereichen.

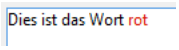
Ist die Option "Mit Standard Stil Tags speichern" für das Objekt markiert, wird beim Verwenden dieser Befehle eine Änderung der Tags mit jedem Objekt gesichert.

Generische Befehle in Bereichen mit Mehrfachstil

Ab 4D v14 gibt es neue Interaktionen zwischen generischen Befehlen wie **OBJECT SET RGB COLORS** oder **OBJECT SET FONT STYLE** und Textbereichen mit Mehrfachstil.

In bisherigen 4D Versionen hat das Ausführen solcher Befehle den Inhalt eigener Stilelemente im Bereich verändert. Jetzt werden nur standardmäßige Eigenschaften und über standardmäßige Tags gesicherte Eigenschaften durch diese Befehle beeinflusst. Eigene Stilelemente bleiben unverändert erhalten.

Nehmen wir z.B. einen Bereich mit Mehrfachstil, in dem Standard Tags gesichert wurden:



Die Textformatierung dafür lautet:

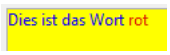
```
<span style="text-align:left;font-family:'Segoe UI';font-size:9pt;color:#009900">Dies ist das Wort <span
style="color:#D81E05">rot</span></span>
```

Führen Sie folgenden Code aus:

```
OBJECT SET COLOR(*;"myArea";-(Blue+(256*Yellow)))
```

bleibt in 4D v14 die rote Farbe erhalten:

4D v14




```
<span style="text-align:left;font-family:'Segoe
UI';font-size:9pt;color:#0000FF">Dies ist das
Wort <span style="color:#D81E05">rot</span>
</span>
```

bisherige Versionen


```
<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">Dies ist das Wort rot</span>
</span>
```

Das gilt für folgende generische Befehle:

OBJECT SET RGB COLORS
OBJECT SET COLOR
OBJECT SET FONT
OBJECT SET FONT STYLE
OBJECT SET FONT SIZE

Für Bereiche mit Mehrfachstil sollten generische Befehle nur zum Setzen von Standard Stilarten verwendet werden. Zum Verwalten von Stilarten während der Ausführung der Datenbank empfehlen wir, die o.a. Befehle zu verwenden (siehe **Mehrfachstil Text**).

Funktion Get edited text

Beim Verwenden mit einem "Rich Text" Bereich gibt die Funktion **Get edited text** aus dem Kapitel **Formularereignisse** den Text des aktuellen Bereichs mit allen darin enthaltenen Stil Tags zurück.

Um bearbeiteten reinen Text, d.h. Text ohne Tags wiederzufinden, müssen Sie die 4D Funktion **ST Get plain text** verwenden:

```
ST Get plain text(Get edited text)
```

Such- und Sortierbefehle

Such- und Sortierläufe in Objekten mit Mehrfachstil berücksichtigen alle im Objekt gesicherten Stil Tags. Wurde der Stil innerhalb eines Wortes verändert, ist die Suche nach diesem Wort nicht erfolgreich.
Zum Ausführen gültiger Such- und Sortierläufe müssen Sie die Funktion **ST Get plain text** verwenden. Zum Beispiel:

```
QUERY BY FORMULA([MyTable];ST Get plain text([MyTable]MyFieldStyle)="very well")
```

Automatische Vereinheitlichung von Zeilenenden

Damit bei Texten in Datenbanken größere Kompatibilität mit anderen Plattformen gegeben ist, vereinheitlicht 4D ab Version 14 automatisch Zeilenenden mit dem Zeichen '\r'. Das gilt für Formularobjekte (Variablen oder Felder) in Plain Text oder Text mit Mehrfachstil. Zeilenenden, die nicht nativ sind bzw. eine Mischung aus mehreren Zeichen (z.B. '\r\n') werden als ein einzelnes '\r'. gewertet.

Beachten Sie, dass in Übereinstimmung mit XML Standards (Mehrfachstil Textformat) Befehle für Texte mit Mehrfachstil ebenfalls Zeilenenden für Textvariablen, die keinen Objekten zugeordnet sind, vereinheitlichen. Das ist die Funktionsweise wie in früheren 4D Versionen.

Das vereinfacht die plattformübergreifende Verwendung von Befehlen für Text mit Mehrfachstil oder Befehle wie **HIGHLIGHT TEXT**. Das müssen Sie jedoch in Ihren Abläufen berücksichtigen, wenn Sie mit Texten aus heterogenen Quellen arbeiten.

Unterstützte Tags

Dynamische Tags

In 4D können Sie folgende Tags in formatierten Textbereichen (Mehrfachstil) verwenden:

4D Expression

Fügt einen 4D Ausdruck in den Text ein, wie Ausdruck, Methode, Feld, Variable, Befehl, etc. Der Ausdruck wird tokenized und bewertet:

- Beim Einfügen des Ausdrucks
- Beim Laden des Objekts
- Beim Aufrufen der Standardaktion *computeExpressions* über ein Objekt der Oberfläche oder über den Befehl **INVOKE ACTION**
- Beim Ausführen des Befehls **ST COMPUTE EXPRESSIONS**
- Beim Ausführen des Befehls **ST FREEZE EXPRESSIONS**, wenn der zweite Parameter * übergeben wird

Im Tag wird nicht der bewertete Wert des Ausdrucks gesichert, sondern nur seine Referenz.

Hinweis: Um einen 4D Befehl unabhängig von der Version der 4D Programmiersprache einzufügen, verwenden Sie einen Ausdruck vom Typ '<*command_name*>:C<*command_number*>'. Beispiel: Um den Befehl **Current time** einzufügen, schreiben Sie '**Current time:C178**'. Weitere Informationen dazu finden Sie im Abschnitt **Tokens in Formeln verwenden**.

URL

Visible label

fügt eine URL in den Text ein. Beispiel:

```
<span><a href="http://www.4d.com/">4D Web Site</a></span>
```

Benutzerlink

Click here

"Benutzerlinks" sehen genauso aus wie URLs, öffnen jedoch beim Anklicken nicht automatisch die Quelle. Sie können als Referenz einen beliebigen String übergeben. Der Entwickler kann selbst eigene Aktionen programmieren, die beim Anklicken passieren.
Sie können also Links erstellen, die keine URLs sind, sondern Referenzen auf Dateien, 4D Methoden, o.ä. die Sie bei Anklicken öffnen oder ausführen können. Der Befehl **ST Get content type** findet heraus, ob ein Benutzerlink angeklickt wurde.

Benutzerlinks werden mit dem Befehl **ST SET TEXT** definiert. Zum Beispiel:

```
ST SET TEXT(txtVar;"This is a user link: <span style=\"-d4-ref-user:'UserLink'\">User Label</span>";$start;$end)
```

Eigene Tags

Sie können in Plain Text beliebige Tags einfügen, z.B. <img src="http://doc.4d.com/pictures/ja.png">. Dies wird im Code des Plain Textes gespeichert, aber weder interpretiert noch angezeigt. Das ist z.B. hilfreich bei E-Mails im HTML Format oder zum Einfügen von Bildern.

Stil Tags

4D unterstützt in Rich Text Bereichen verschiedene Attribute für <SPAN> Tags. Sie können diese Tags zum Verwalten eigener Stilelemente verwenden. 4D unterstützt nur die nachfolgend aufgelisteten Tags für Stilvariationen.

Schriftname

<SPAN STYLE="font-family: DESDEMONA"> ... </SPAN>

Schriftgröße

<SPAN STYLE="font-size: 20pt"> ... </SPAN>

Schriftstil

- **Fett**
  <SPAN STYLE="font-weight: bold"> ... </SPAN>
- **Kursiv oder Normal**
  <SPAN STYLE="font-style: italic"> ... </SPAN>
  <SPAN STYLE="font-style: normal"> ... </SPAN>
- **Unterstrichen**
  <SPAN STYLE="text-decoration: underline"> ... </SPAN>
- **Durchgestrichen**
  <SPAN STYLE="text-decoration:line-through">...</SPAN>
  Hinweis: Auf macOS wird dieser Stil nicht unterstützt, das Tag lässt sich aber weiter per Programmierung verwalten.

Schriftfarbe

<SPAN STYLE="color:green"> ... </SPAN>
oder
<SPAN STYLE="color:#006CCC">...</SPAN>

Hintergrundfarben (nur Windows)

<SPAN STYLE="background-color:green"> ... </SPAN>
oder
<SPAN STYLE="background-color:#006CCC">...</SPAN>

Hinweis: Auf Mac OS wird dieses Attribut ignoriert. Es wird beim Ändern des Objekts entfernt.

Farbwerte

Für Attribute zu Schrift- und Hintergrundfarben kann der Farbwert entweder hexadezimal Code für eine RGB Farbe oder der Name einer der 16 HTML Farben sein, der für standardmäßige CSS über W3C definiert wird:

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 ({* ;} Objekt {; StartAusw {; EndeAusw}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
StartAusw	Lange Ganzzahl	➡	Start der Auswahl
EndeAusw	Lange Ganzzahl	➡	Ende der Auswahl

Beschreibung

Der Befehl **ST COMPUTE EXPRESSIONS** aktualisiert die dynamischen 4D Ausdrücke im Text mit Mehrfachstil oder in 4D Write Pro Feld oder Variable ein, definiert im Parameter *Objekt*.

Weitere Informationen zu 4D Ausdrücken in formatierten Textbereichen oder 4D Write Pro Bereichen finden Sie unter dem Befehl **ST INSERT EXPRESSION**.

Der Befehl bewertet das Ergebnis von Ausdrücken in *Objekt* gemäß dem aktuellen Kontext neu und zeigt den resultierenden Wert an. Gibt es z.B. einen Ausdruck mit der Uhrzeit, wird der Wert bei jedem Aufruf von **ST COMPUTE EXPRESSIONS** geändert. Ausdrücke werden auch in folgenden Situationen neu berechnet:

- Beim Einfügen
- Beim Laden
- Beim "Einfrieren" mit dem Befehl **ST FREEZE EXPRESSIONS**, wenn der zweite Parameter * übergeben ist.

ST COMPUTE EXPRESSIONS ändert nicht formatierten Text, d.h. mit *span* Tags, sondern nur Plain Text in *Objekt*. Im formatierten Text werden nicht die berechneten Werte, sondern nur ihre Referenzen gespeichert.
Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String.

Objekt muss nicht zwingend den Fokus haben; bei einem Textbereich mit Mehrfachstil muss es jedoch in einem Formular liegen, sonst hat der Befehl **ST COMPUTE EXPRESSIONS** keine Auswirkung.

Hinweis: Gibt *Objekt* ein 4D Write Pro Dokument an, wird es vom Befehl bewertet, selbst wenn es nicht in einem Formularobjekt geöffnet wird. Das ist notwendig, da **ST COMPUTE EXPRESSIONS** das Dokument durch Konvertieren von Ausdrücken oder Bildausdrücken nach der Bewertung (falls zutreffend) verändern kann, um die Ausdrücke korrekt zu verwalten (siehe **Bildausdrücke einfügen**).

Die optionalen Parameter *StartAusw* und *EndeAusw* definieren eine Textauswahl in *Objekt*. Die Werte *StartAusw* und *EndeAusw* geben eine Auswahl im Plain Text, ohne evtl. vorhandene Stil Tags oder Referenzen zu berücksichtigen. Beachten Sie, dass eine Referenz einem einzelnen Zeichen entspricht.

- Mit *StartAusw* und *EndeAusw* aktualisiert **ST COMPUTE EXPRESSIONS** nur die Ausdrücke innerhalb dieser Auswahl.
- Übergeben Sie nur *StartAusw* oder ist der Wert von *EndeAusw* größer als die Gesamtanzahl der Zeichen in *Objekt*, werden alle Ausdrücke zwischen *StartAusw* und dem Textende berechnet.
- Lassen Sie *StartAusw* und *EndeAusw* weg, werden alle Ausdrücke innerhalb der Benutzerauswahl von *Objekt* berechnet.

4D bietet vordefinierte Konstanten, so dass Sie die Auswahlgrenzen in den Parametern *StartAusw* und *EndeAusw* automatisch setzen können.
Diese Konstanten finden Sie unter dem Thema **Mehrfachstil Text**:

Konstante	Typ	Wert	Kommentar
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt

(*) Um diese Konstante zu nutzen, müssen Sie in *Objekt* einen Objektnamen verwenden. Übergeben Sie eine Referenz auf ein Feld oder eine Variable, wird der Befehl auf den gesamten Text des Objekts angewendet.

Hinweis: Ist *StartAusw* größer als *EndeAusw*, führt der Befehl nichts aus und die Variable OK wird auf 0 gesetzt.

Beispiel

Die Referenzen innerhalb einer Textauswahl aktualisieren:

```
ST COMPUTE EXPRESSIONS(*;"myText";ST Start highlight;ST End highlight)
```

ST FREEZE EXPRESSIONS

ST FREEZE EXPRESSIONS ({* ;} Objekt {; StartAusw {; EndeAusw}}{; *})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
StartAusw	Lange Ganzzahl	➡	Start der Auswahl
EndeAusw	Lange Ganzzahl	➡	Ende der Auswahl
*	Operator	➡	Mit *: Ausdrücke vor dem Einfrieren aktualisieren

Beschreibung

Der Befehl **ST FREEZE EXPRESSIONS** friert den Inhalt der Ausdrücke im Text mit Mehrfachstil bzw. in 4D Write Pro Feld oder Variable ein, definiert im Parameter *Objekt*. Diese Aktion konvertiert dynamische Ausdrücke in statischen Text oder Bild (nur 4D Write Pro Bereiche) und entfernt zugeordnete Referenzen aus *Objekt*.

Weitere Informationen zu 4D Ausdrücken in Text mit Mehrfachstil oder in 4D Write Pro Bereichen finden Sie unter dem Befehl **ST INSERT EXPRESSION**.

Der Befehl **ST FREEZE EXPRESSIONS** speichert die berechneten Werte eines Ausdrucks zu einer bestimmten Zeit. Diese Operation ist insbesondere vor dem Verwenden von *Objekt* außerhalb des Bereichs (Export, Speichern in einer Datei auf der Festplatte, Drucken, etc.) notwendig, da nur die Referenz des Ausdrucks im Bereich selbst beibehalten wird.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Die optionalen Parameter *StartAusw* und *EndeAusw* definieren eine Textauswahl in *Objekt*. Die Werte *StartAusw* und *EndeAusw* geben eine Auswahl im Plain Text, ohne evtl. vorhandene Stil Tags oder Referenzen zu berücksichtigen.

- Übergeben Sie StartAusw und EndeAusw, friert **ST FREEZE EXPRESSIONS** nur die Ausdrücke innerhalb dieser Auswahl ein.
- Übergeben Sie nur *StartAusw* oder ist der Wert von *EndeAusw* größer als die Gesamtanzahl der Zeichen in *Objekt*, werden alle Ausdrücke zwischen *StartAusw* und dem Textende eingefroren
- Lassen Sie *StartAusw* und *EndeAusw* weg, werden alle Ausdrücke innerhalb der Benutzerauswahl von *Objekt* eingefroren

4D bietet vordefinierte Konstanten, so dass Sie die Auswahlgrenzen in den Parametern *StartAusw* und *EndeAusw* automatisch setzen können. Diese Konstanten finden Sie unter dem Thema "**Mehrfachstil Text**":

Konstante	Typ	Wert	Kommentar
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt

(*) Um diese Konstante zu nutzen, müssen Sie in Objekt einen Objektnamen verwenden. Übergeben Sie eine Referenz auf ein Feld oder eine Variable, wird der Befehl auf den gesamten Text des Objekts angewendet.

Hinweis: Ist *StartAusw* größer als *EndeAusw* (außer *EndeAusw* ist 0), führt der Befehl nichts aus und die Variable OK wird auf 0 gesetzt.

Standardmäßig werden Ausdrücke vor dem Einfrieren nicht neu berechnet. Übergeben Sie den zweiten Parameter *, wenn der Ausdruck erst neu berechnet und dann eingefroren werden soll.

Beispiel

Die aktuelle Zeit am Textanfang einfügen und dann vor Sichern des Datensatzes einfrieren:

```
//Die Uhrzeit am Textanfang einfügen
ST INSERT EXPRESSION(*;StyledText_t;"Current time";1)
//Dann Ausdruck einfrieren
ST FREEZE EXPRESSIONS(*;"StyledText_t";1)
```

ST GET ATTRIBUTES

ST GET ATTRIBUTES ({* ;} Objekt ; StartAusw ; EndeAusw ; attrName ; attrWert {; attrName2 ; attrWert2 ; ... ; attrNameN ; attrWertN})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Feld oder Variable
StartAusw	Lange Ganzzahl	➡	Start der Textauswahl
EndeAusw	Lange Ganzzahl	➡	Ende der Textauswahl
attrName	Lange Ganzzahl	➡	Zu erhaltendes Attribut
attrWert	Variable	⬅	Aktueller Wert des Attributs

Beschreibung

Der Befehl **ST GET ATTRIBUTES** erhält den aktuellen Wert eines Stilattributs in einer Textauswahl des Formularobjekts, definiert durch *Objekt*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Hat das Objekt während der Ausführung den Fokus, gibt die Funktion Information über das Objekt in Bearbeitung zurück; hat das Objekt keinen Fokus, gibt die Funktion Information über die Datenquelle (Feld oder Variable) des Objekts zurück.

Ohne * geben Sie an, dass der Parameter *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstatt eines String. Die Funktion gibt während der Ausführung Information über dieses Feld oder diese Variable zurück.

Mit den Parametern *StartAusw* und *EndeAusw* können Sie die Textauswahl von *Objekt* festlegen, dessen Stilattribut gelesen werden soll. In *StartAusw* übergeben Sie die Position des ersten Zeichens, in *EndeAusw* die Position des letzten Zeichens + 1 (das letzte übergebene Zeichen ist in der Auswahl nicht enthalten). Sie können in *EndeAusw* 0 übergeben, um automatisch das letzte Zeichen des Textes anzugeben (Übergeben Sie 1 in *StartAusw* , um das erste Zeichen im Text anzugeben.) Sind die Werte *StartAusw* und *EndeAusw* gleich oder ist *StartAusw* größer als *EndeAusw* (außer der Wert von *EndeAusw* ist 0, siehe oben), wird ein Fehler zurückgegeben.
Die Werte *StartAusw* und *EndeAusw* berücksichtigen nicht bereits vorhandene Stil Tags im Bereich. Sie werden vielmehr auf Basis von Rohtext bewertet, d.h. Text, aus dem Stil Tag herausgefiltert wurden.

4D bietet vordefinierte Konstanten, so dass Sie die Auswahlgrenzen in den Parametern *StartAusw* und *EndeAusw* automatisch setzen können. Diese Konstanten finden Sie unter dem Thema **Mehrfachstil Text**:

Konstante	Typ	Wert	Kommentar
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt

(*) Für diese Konstante müssen Sie in *Objekt* einen Objektnamen übergeben. Übergeben Sie eine Referenz auf ein Feld oder eine Variable, wird die Funktion auf den gesamten Text des Objekts angewandt.
Im Parameter *attrName* übergeben Sie den Namen des zu lesenden Attributs. Im Parameter *attrWert* übergeben Sie eine Variable, die den aktuellen Wert des Attributs wiedergibt.
Sie können beliebig viele Paare Attribut/Wert übergeben. Im Parameter *attrName* müssen Sie eine vordefinierte Konstante unter dem Thema **Mehrfachstil Textattribute** verwenden.

Konstante	Typ	Wert	Kommentar
Attribute background color	Lange Ganzzahl	8	(nur Windows) Hexadezimale Werte oder HTML Farbnamen
Attribute bold style	Lange Ganzzahl	1	*attrWert*=0: Attribut fett aus Auswahl entfernen *attrWert*=1: Attribut fett auf Auswahl anwenden
Attribute font name	Lange Ganzzahl	5	*attrWert*=Schriftfamilienname (String)
Attribute italic style	Lange Ganzzahl	2	*attrWert*=0: Attribut kursiv aus Auswahl entfernen *attrWert*=1: Attribut kursiv auf Auswahl anwenden
Attribute strikethrough style	Lange Ganzzahl	3	*attrWert*=0: Attribut durchgestrichen aus Auswahl entfernen *attrWert*=1: Attribut durchgestrichen auf Auswahl anwenden
Attribute text color	Lange Ganzzahl	7	Hexadezimale Werte oder Konstanten mit hexadezimalen Werten
Attribute text size	Lange Ganzzahl	6	*attrWert*=Anzahl Punkte (Zahl)
Attribute underline style	Lange Ganzzahl	4	*attrWert*=0: Attribut unterstrichen aus Auswahl entfernen *attrWert*=1: Attribut unterstrichen auf Auswahl anwenden

Ist der Wert von *attrName* für die gesamte Auswahl gleich, wird er in *attrWert* zurückgegeben. Ist er unterschiedlich oder enthält *Objekt* keine SPAN Tags, werden folgende Werte zurückgegeben:

attrName	attrWert, wenn Attribut in Auswahl unterschiedlich oder ohne SPAN Tags
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

Beispiel

Wir nehmen das Feld [Table_1]StyledText aus einem Formular. Das Objekt hat die Eigenschaft **Mehrfachstil** und den Namen "StyledText_t". Sie wollen den markierten Text und den Status von Attribute bold style erhalten. Es gibt zwei unterschiedliche Vorgehensweisen, je nachdem, ob Sie den Objektnamen oder die Referenz auf das Feld verwenden.

- Den Objektnamen verwenden:

```
$text:=ST Get text(*;"StyledText_t";ST Start highlight;ST End highlight)
ST GET ATTRIBUTES(*;"StyledText_t";ST Start highlight;ST End highlight;Attribute bold style;$bold)
```

- Den Feldnamen verwenden:

```
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)
```

Systemvariablen und Mengen

Nach Ausführen dieses Befehls wird die Variable OK auf 1 gesetzt, wenn kein Fehler aufgetreten ist; andernfalls wird sie auf 0 gesetzt. Das ist insbesondere der Fall, wenn Stil Tags nicht korrekt gewertet werden (inkorrekte oder fehlende Tags).

Bei einem Fehler wird die Variable nicht geändert. Tritt ein Fehler in einer Variablen auf, während der Text gewertet wird, wandelt 4D den Text in Plain Text um; als Ergebnis werden die Zeichen <, > und & in HTML Einheiten umgewandelt.

ST Get content type

ST Get content type ({* ;} Objekt {; StartAusw {; EndeAusw {; StartBlock {; EndeBlock}}}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
StartAusw	Lange Ganzzahl	➡	Start der Auswahl
EndeAusw	Lange Ganzzahl	➡	Ende der Auswahl
StartBlock	Lange Ganzzahl	⬅	Startpunkt des ersten Auswahltyps
EndeBlock	Lange Ganzzahl	⬅	Endpunkt des ersten Auswahltyps
Funktionsergebnis	Lange Ganzzahl	↩	Typ des Inhalts

Beschreibung

Die Funktion **ST Get content type**gibt den Typ des Inhalts vom Feld zurück, definiert im Parameter *Objekt* und vom Typ formatierter Text oder Variable.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Hat das Objekt den Fokus, gibt die Funktion beim Ausführen die Information zum Objekt in Bearbeitung zurück. Hat das Objekt keinen Fokus, gibt die Funktion die Information der Datenquelle des Objekts zurück (Variable oder Feld).

Die optionalen Parameter *StartAusw* und *EndeAusw* definieren eine Textauswahl in *Objekt*. Die Werte *StartAusw* und *EndeAusw* geben eine Auswahl im Plain Text, ohne evtl. vorhandene Stil Tags zu berücksichtigen.

- Übergeben Sie *StartAusw* und *EndeAusw*, bewertet **ST Get content type** den Inhalt innerhalb dieser Auswahl
- Übergeben Sie nur *StartAusw* oder ist der Wert von *EndeAusw* größer als die Gesamtanzahl der Zeichen in *Objekt*, wird der Inhalt zwischen *StartAusw* und dem Textende bewertet
- Lassen Sie *StartAusw* und *EndeAusw* weg, wird der Inhalt in der aktuellen Textauswahl bewertet.

4D bietet vordefinierte Konstanten, so dass Sie die Auswahlgrenzen in den Parametern *StartAusw* und *EndeAusw* automatisch setzen können. Diese Konstanten finden Sie unter dem Thema **Mehrfachstil Text**:

Konstante	Typ	Wert	Kommentar
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt

(*) Um diese Konstante zu nutzen, müssen Sie in Objekt einen Objektnamen verwenden. Übergeben Sie eine Referenz auf ein Feld oder eine Variable, wird der Befehl auf den gesamten Text des Objekts angewendet.

Hinweis: Ist *StartAusw* größer als *EndeAusw* (außer *EndeAusw* ist 0), führt der Befehl nichts aus und die Variable OK wird auf 0 gesetzt.

Die optionalen Parameter *StartBlock* und *EndeBlock* finden die Position des ersten und des letzten Zeichens des ersten homogenen Blocks wieder, definiert in *Objekt* oder in der Auswahl von *Objekt*. Enthält die Auswahl z.B. einen Ausdruck und dann Plain Text, geben *StartBlock* und *EndeBlock* die Grenzen des Ausdruck zurück. Sie können eine Schleife setzen, um alle Blöcke der Auswahl zu bearbeiten.

Dieser Befehl gibt einen Wert für den Typ des identifizierten Inhalts zurück. Sie können diesen Wert mit einer der folgenden Konstanten unter dem Thema **Mehrfachstil Text** vergleichen:

Konstante	Typ	Wert	Kommentar
ST Expression type	Lange Ganzzahl	2	Auswahl enthält nur eine Referenz für Ausdruck
ST Mixed type	Lange Ganzzahl	3	Auswahl enthält mindestens zwei unterschiedliche Typen im Inhalt
ST Picture type	Lange Ganzzahl	6	Auswahl enthält nur ein Bild (nur 4D Write Pro Areas)
ST Plain type	Lange Ganzzahl	0	Auswahl enthält Text und keine Referenzen
ST Unknown tag type	Lange Ganzzahl	4	Auswahl enthält nur einen unbekannten Tag Typ
ST URL type	Lange Ganzzahl	1	Auswahl enthält nur eine Referenz für URL
ST User type	Lange Ganzzahl	5	Auswahl enthält nur eine eigene Referenz

Beispiel

Befehle des Kontextmenüs gemäß dem im Bereich gewählten Inhaltstyp anzeigen.

```
Case of
   :(Form event=On Clicked)
 //die Auswahl finden
      GET HIGHLIGHT(*;"myText";startSel;endSel)
      If(Contextual click & (Macintosh control down=False)) //ruft das Kontextmenü auf
         If(startSel=endSel) // kein Inhalt ausgewählt
 //nur bestimmte Befehle aktivieren
            DISABLE MENU ITEM(<>menu_STYLEDTEXT;2)
            DISABLE MENU ITEM(<>menu_STYLEDTEXT;4)
            ENABLE MENU ITEM(<>menu_STYLEDTEXT;6)
            ...
```

```
        Else  // den Inhaltstyp erhalten
            CT_Texttype:=ST Get content type(*;"myText";startSel;endSel)
            Case of  // verschiedene Typen bearbeiten
                :(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 ({* ;} Objekt {; StartAusw {; EndeAusw}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
StartAusw	Lange Ganzzahl	➡	Start der Auswahl
EndeAusw	Lange Ganzzahl	➡	Ende der Auswahl
Funktionsergebnis	Text	⬅	Bezeichnung des Ausdrucks

Beschreibung

Die Funktion **ST Get expression** gibt den ersten Ausdruck in der aktuellen Auswahl im Feld zurück, definiert im Parameter *Objekt* und vom Typ formatierter Text oder Variable.

Die Funktion gibt die Bezeichnung des Ausdrucks zurück, der im Objekt eingefügt wurde (zum Beispiel "mymethod" oder "[table1]field1"). Es wird nicht der aktuelle Wert des Ausdrucks zurückgegeben.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Hat das Objekt den Fokus, gibt die Funktion beim Ausführen die Information zum Objekt in Bearbeitung zurück. Hat das Objekt keinen Fokus, gibt die Funktion die Information der Datenquelle des Objekts zurück (Variable oder Feld). Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt). Die Funktion gibt dann beim Ausführen die Information der Variablen oder des Feldes zurück.

Die optionalen Parameter *StartAusw* und *EndeAusw* definieren eine Textauswahl in *Objekt*. Die Werte *StartAusw* und *EndeAusw* geben eine Auswahl im Plain Text, ohne evtl. vorhandene Stil Tags zu berücksichtigen.

- Übergeben Sie *StartAusw* und *EndeAusw*, sucht **ST Get expression** den Ausdruck innerhalb dieser Auswahl
- Übergeben Sie nur *StartAusw* oder ist der Wert von *EndeAusw* größer als die Gesamtanzahl der Zeichen in *Objekt*, sucht die Funktion den Ausdruck zwischen *StartAusw* und dem Textende
- Lassen Sie *StartAusw* und *EndeAusw* weg, sucht die Funktion den Ausdruck in der aktuellen Textauswahl

4D bietet vordefinierte Konstanten, so dass Sie die Auswahlgrenzen in den Parametern *StartAusw* und *EndeAusw* automatisch setzen können. Diese Konstanten finden Sie unter dem Thema "**Mehrfachstil Text**":

Konstante	Typ	Wert	Kommentar
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt

(*) Um diese Konstante zu nutzen, müssen Sie in *Objekt* einen Objektnamen verwenden. Übergeben Sie eine Referenz auf ein Feld oder eine Variable, wird der Befehl auf den gesamten Text des Objekts angewendet.

Hinweis: Ist *StartAusw* größer als *EndeAusw* (außer *EndeAusw* ist 0), führt der Befehl nichts aus nicht und die Variable OK wird auf 0 gesetzt.

Wird in der Auswahl kein Ausdruck gefunden, gibt die Funktion einen leeren String zurück.

Beispiel 1

Bei einem Doppelklick Ereignis prüfen Sie, ob es einen Ausdruck gibt. Wenn ja, zeigen Sie, wo seine Werte gefunden wurden, so dass der Benutzer den eingefügten Ausdruck ändern kann:

```
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
```

Beispiel 2

Eine 4D Methode ausführen, wenn ein Benutzerlink angeklickt wurde:

```
Case of
    :(Form event=On Clicked)
 //Die Auswahl finden
       HIGHLIGHT TEXT(*;"myText";startSel;endSel)
       If(startSel#endSel)  //Es gibt ausgewählten Inhalt
 //Die Art des Inhalts erhalten
          $CT_type:=ST Get content type(*;"myText";startSel;endSel)
          If($CT_type=ST User type)  //Dies ist ein Benutzerlink
             MyMethod  //Eine 4D Methode ausführen
          End if
       End if
End case
```

ST GET OPTIONS

ST GET OPTIONS ({* ;} Objekt ; Option ; Wert {; Option2 ; Wert2 ; ... ; OptionN ; WertN})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable)
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Option	Lange Ganzzahl	➡	Zu erhaltende Option
Wert	Lange Ganzzahl	⬅	Aktueller Wert der Option

Beschreibung

Der Befehl **ST GET OPTIONS** erhält den aktuellen Wert einer oder mehrerer Optionen im Feld, definiert im Parameter *Objekt* und vom Typ formatierter Text oder Variable.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Hat das Objekt den Fokus, gibt die Funktion beim Ausführen die Information zum Objekt in Bearbeitung zurück. Hat das Objekt keinen Fokus, gibt die Funktion die Information der Datenquelle des Objekts zurück (Variable oder Feld).
Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt). Die Funktion gibt dann beim Ausführen die Information der Variablen oder des Feldes zurück.

Im Parameter *Option* übergeben Sie den Code der zu erhaltenden Option. Der Befehl gibt den aktuellen Wert dieser Option in *Wert* zurück. Für beide Parameter können Sie eine der folgenden Konstanten unter dem Thema **Mehrfachstil Text**:

Konstante	Typ	Wert	Kommentar
ST Expressions display mode	Lange Ganzzahl	1	Der Parameter *Wert* kann ST Values oder ST References enthalten
ST References	Lange Ganzzahl	1	Zeigt Quelle der Strings für Ausdrücke
ST Values	Lange Ganzzahl	0	Zeigt berechnete Werte von Ausdrücken

ST Get plain text

ST Get plain text ({* ;} Objekt {; RefModus}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
RefModus	Lange Ganzzahl	➡	Modus für die Anzeige der im Text gefundenen Referenzen
Funktionsergebnis	Text	⮌	Text ohne Tags

Beschreibung

Die Funktion **ST Get plain text** entfernt alle Stil Tags aus der Textvariablen oder dem Feld, definiert in den Parametern * und *Objekt* und gibt den Plain Text zurück.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Hat das Objekt während der Ausführung den Fokus, gibt die Funktion Information über das Objekt in Bearbeitung zurück; hat das Objekt keinen Fokus, gibt die Funktion Information über die Datenquelle (Feld oder Variable) des Objekts zurück.

Ohne * geben Sie an, dass der Parameter *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstatt eines String. Die Funktion gibt während der Ausführung Information über dieses Feld oder diese Variable zurück.

Der optionale Parameter *RefModus* gibt an, wie in *Objekt* gefundene Referenzen zurückgegeben werden sollen. In *RefModus* übergeben Sie eine der folgenden Konstanten unter dem Thema **Mehrfachstil Text** (Sie können eine einzelne Konstante oder mehrere kombiniert übergeben):

Konstante	Typ	Wert	Kommentar
ST 4D Expressions as sources	Lange Ganzzahl	2	Der ursprüngliche String der Referenzen auf 4D Ausdrücke wird zurückgegeben
ST 4D Expressions as values	Lange Ganzzahl	1	Referenzen auf 4D Ausdrücke werden in interpretierter Form zurückgegeben (standardmäßige Funktionsweise in Formularen)
ST References as spaces	Lange Ganzzahl	0	Jede Referenz wird in Form geschützter Leerzeichen zurückgegeben (Standardoperation, wird von den anderen Befehlen verwendet)
ST Tags as plain text	Lange Ganzzahl	64	Die Bezeichnung des Tag wird in Plain Text zurückgegeben. Beispiel: Für das Tag '<img src="test.jpg" alt="picture">my picture</img>', ist Plain Text "my picture" (standardmäßige Funktionsweise in Formularen)
ST Tags as XML code	Lange Ganzzahl	128	Der XML Code des Tag wird in Plain Text zurückgegeben. Beispiel: Für das Tag '<img src="test.jpg" alt="picture">my picture</img>', ist der Plain Text '<img src="test.jpg" alt="picture">my picture</img>
ST Text displayed with 4D Expression sources	Lange Ganzzahl	86	Gibt den Text so zurück, wie er im Formular angezeigt wird, mit den ursprünglichen Strings der 4D Ausdrücke. Entspricht einer vordefinierten Kombination der Konstanten 2+4+16+64
ST Text displayed with 4D Expression values	Lange Ganzzahl	85	Gibt den Text so zurück, wie er im Formular angezeigt wird, mit interpretierten 4D Ausdrücken. Entspricht einer vordefinierten Kombination der Konstanten 1+4+16+64
ST URL as labels	Lange Ganzzahl	4	Die sichtbare Bezeichnung von URLs wird zurückgegeben, z.B. "Besuchen Sie unsere Web Site" (standardmäßige Funktionsweise in Formularen)
ST URL as links	Lange Ganzzahl	8	Der Link wird zurückgegeben, z.B. "http://www.4d.com"
ST User links as labels	Lange Ganzzahl	16	Die sichtbare Bezeichnung des Benutzer-Links wird zurückgegeben (standardmäßige Funktionsweise in Formularen)
ST User links as links	Lange Ganzzahl	32	Der Inhalt des Benutzer-Links wird zurückgegeben

Hinweise:

- Da Plain Text gleich bleibt, unabhängig von den Werten im Parameter *RefModus*, ist dieser optionale Parameter nur für Text mit Referenzen sinnvoll.
- Bei einem 4D Write Pro Dokument mit Tabellen gilt der Inhalt jeder Zelle als individueller Absatz und wird als Text getrennt durch Tabs zurückgegeben. Zeilen werden durch Zeilenschaltung getrennt.

Beispiel 1

Sie suchen unter den Werten eines Textfeldes mit Mehrfachstil nach dem Text "sehr schön". Der Wert wurde in folgender Form gespeichert: "Das Wetter ist sehr schön **heute**".

```
QUERY BY FORMULA([Comments];ST Get plain text([Comments]Weather)="@sehr schön@")
```

Hinweis: In diesem Kontext gibt nachfolgende Anweisung nicht das gewünschte Ergebnis zurück, weil der Text mit Stil Tags gesichert wird:

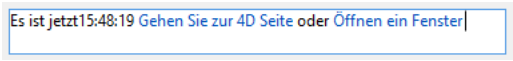
```
QUERY([Comments];[Comments]Weather="@sehr schön@")
```

Beispiel 2

Nehmen wir folgenden Text im Bereich mit Mehrfachstil mit Namen "meineZone"

```
<span>Es ist jetzt <span style="-d4-ref:'Current time:C178'"> </span> <a href="http://www.4d.com">Gehen Sie zur 4D Seite</a> oder
<span style="-d4-ref-user:'openW'">Öffnen ein Fenster</span></span>
```

Dieser Text erscheint wie folgt:



Führen Sie dazu folgenden Code aus:

```
$txt :=ST Get plain text(*;"meineZone";ST References as spaces)
 // $txt = "Es ist jetzt      oder " (Abstände)
$txt :=ST Get plain text(*;"meineZone";ST 4D Expressions as values)
 // $txt = "Es ist jetzt 15:48:19   oder   "
$txt :=ST Get plain text(*;"meineZone";ST 4D Expressions as sources)
 // $txt = "Es ist jetzt Current time   oder   "
$txt :=ST Get plain text(*;"meineZone";ST URL as links)
 //$txt = "Es ist jetzt    http://www.4d.com or   "
$txt :=ST Get plain text(*;"meineZone";ST Text displayed with 4D Expression values)
 //$txt = "Es ist jetzt 15:48:19 Gehen Sie zur 4D Seite oder Öffnen ein Fenster"
$txt :=ST Get plain text(*;"meineZone";ST Text displayed with 4D Expression sources)
 //$txt = "Es ist jetzt Current time Gehen Sie zur 4D Seite oder Öffnen ein Fenster"
$txt :=ST Get plain text(*;"meineZone";ST User links as labels)
 //$txt = "Es ist jetzt     oder Öffnen ein Fenster"
$txt :=ST Get plain text(*;"meineZone";ST User links as links)
 //$txt = "Es ist jetzt     oder openW"
```

Systemvariablen und Mengen

Nach Ausführen dieses Befehls wird die Variable OK auf 1 gesetzt, wenn kein Fehler aufgetreten ist; andernfalls wird sie auf 0 gesetzt. Das ist insbesondere der Fall, wenn Stil Tags nicht korrekt gewertet werden (inkorrekte oder fehlende Tags).

Bei einem Fehler wird die Variable nicht geändert. Tritt ein Fehler in einer Variablen auf, während der Text gewertet wird, wandelt 4D den Text in Plain Text um; als Ergebnis werden die Zeichen <, > und & in HTML Einheiten umgewandelt.

ST Get text

ST Get text ({* ;} Objekt {; StartAusw {; EndeAusw}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Textfeld oder Variable
StartAusw	Lange Ganzzahl	➡	Start der Auswahl
EndeAusw	Lange Ganzzahl	➡	Ende der Auswahl
Funktionsergebnis	Text	⮌	Text mit Stil Tags

Beschreibung

Die Funktion **ST Get text** gibt den gefundenen formatierten Text im Textfeld oder der Variablen zurück, definiert im Parameter *Objekt*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Hat das Objekt während der Ausführung den Fokus, gibt die Funktion Information über das Objekt in Bearbeitung zurück; hat das Objekt keinen Fokus, gibt die Funktion Information über die Datenquelle (Feld oder Variable) des Objekts zurück.

Ohne * geben Sie an, dass der Parameter *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstatt eines String. Die Funktion gibt während der Ausführung Information über dieses Feld oder diese Variable zurück.

Die Funktion gibt den Text mit den enthaltenen Stil Tags zurück. So können Sie z.B. Text mitsamt den Stilelementen kopieren und einsetzen.

Über die optionalen Parameter *StartAusw* und *EndeAusw* definieren Sie eine Auswahl des Textes in *Objekt*. Die Werte in *StartAusw* und *EndeAusw* geben eine Auswahl von Plain Text, ohne im Text gefundene Stil Tags zu berücksichtigen.

- Lassen Sie *StartAusw* und *EndeAusw* weg, gibt **ST Get text** den gesamten Text von *Objekt* zurück
- Übergeben Sie *StartAusw* und *EndeAusw*, gibt **ST Get text** den Volltext dazwischen zurück

Sind die Werte von *StartAusw* und *EndeAusw* gleich oder ist *StartAusw* größer als *EndeAusw*, wird ein Fehler zurückgegeben.

4D bietet vordefinierte Konstanten, so dass Sie die Auswahlgrenzen in den Parametern *StartAusw* und *EndeAusw* automatisch setzen können. Diese Konstanten finden Sie unter dem Thema **Mehrfachstil Text**:

Konstante	Typ	Wert	Kommentar
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt

(*) Für diese Konstante müssen Sie in *Objekt* einen Objektnamen übergeben. Übergeben Sie eine Referenz auf ein Feld oder eine Variable, wird die Funktion auf den gesamten Text des Objekts angewandt.

Systemvariablen und Mengen

Nach Ausführen dieses Befehls wird die Variable OK auf 1 gesetzt, wenn kein Fehler aufgetreten ist; andernfalls wird sie auf 0 gesetzt. Das ist insbesondere der Fall, wenn Stil Tags nicht korrekt gewertet werden (inkorrekte oder fehlende Tags).

Bei einem Fehler wird die Variable nicht geändert. Tritt ein Fehler in einer Variablen auf, während der Text gewertet wird, wandelt 4D den Text in Plain Text um; als Ergebnis werden die Zeichen <, > und & in HTML Einheiten umgewandelt.

ST GET URL

ST GET URL ({* ;} Objekt ; urlText ; urlAdresse {; StartAusw {; EndeAusw}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
urlText	Text	⬅	Sichtbarer Text der URL
urlAdresse	Text	⬅	URL Adresse
StartAusw	Lange Ganzzahl	➡	Start der Auswahl
EndeAusw	Lange Ganzzahl	➡	Ende der Auswahl

Beschreibung

Die Funktion **ST GET URL** gibt die Textbezeichnung und Adresse der ersten URL im Feld zurück, definiert im Parameter *Objekt* und vom Typ formatierter Text oder Variable.

Die Parameter *urlText* und *urlAdresse* geben die Textbezeichnung und Adresse zurück. Enthält die Auswahl keine URL, werden leere Strings zurückgegeben.

Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt). Die Funktion gibt dann beim Ausführen die Information der Variablen oder des Feldes zurück.

Die optionalen Parameter StartAusw und EndeAusw definieren eine Textauswahl in *Objekt*. Die Werte *StartAusw* und *EndeAusw* geben eine Auswahl im Plain Text, ohne evtl. vorhandene Stil Tags zu berücksichtigen

- Übergeben Sie *StartAusw* und *EndeAusw*, sucht **ST GET URL** n dieser Auswahl nach der URL
- Übergeben Sie nur *StartAusw* oder ist der Wert von *EndeAusw* größer als die Gesamtanzahl der Zeichen in Objekt, sucht der Befehl zwischen *StartAusw* und dem Textende nach der URL
- Lassen Sie *StartAusw* und *EndeAusw* weg, sucht der Befehl in der aktuellen Textauswahl nach der URL

4D bietet vordefinierte Konstanten, so dass Sie die Auswahlgrenzen in den Parametern *StartAusw* und *EndeAusw* automatisch setzen können. Diese Konstanten finden Sie unter dem Thema **Mehrfachstil Text**:

Konstante	Typ	Wert	Kommentar
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt

(*) Um diese Konstante zu nutzen, müssen Sie in *Objekt* einen Objektnamen verwenden. Übergeben Sie eine Referenz auf ein Feld oder eine Variable, wird der Befehl auf den gesamten Text des Objekts angewendet

Hinweis: Ist *StartAusw* größer als *EndeAusw* (außer *EndeAusw* ist 0), führt der Befehl nichts aus und die Variable OK wird auf 0 gesetzt.

Beispiel

Bei einem Doppelklick Ereignis prüfen Sie, ob es tatsächlich eine URL gibt. Wenn ja, zeigen Sie einen Dialog mit den Werten an, so dass der Benutzer diese ändern kann:

```
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 ({* ;} Objekt ; Ausdruck {; StartAusw {; EndeAusw}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Objekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Ausdruck	Text	➡	Ausdruck und (optional) Format zum Einfügen
StartAusw	Lange Ganzzahl	➡	Start der Auswahl
EndeAusw	Lange Ganzzahl	➡	Ende der Auswahl

Beschreibung

Der Befehl **ST INSERT EXPRESSION** fügt eine Referenz von *Ausdruck* in das Feld ein, definiert im Parameter *Objekt* und vom Typ formatierter Text oder Variable.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Im Parameter Ausdruck übergeben Sie den 4D Ausdruck, den Sie in *Objek*t bewerten wollen. Ein gültiger 4D Ausdruck ist ein String, der einen Wert zurückgibt. Ausdruck kann ein Feld, eine Variable, ein 4D Befehl, eine Anweisung, die einen Wert zurückgibt, eine Projektmethode, o.ä. sein.

Der Ausdruck muss in Anführungszeichen ("") stehen.

Hinweis: Das Einfügen von Bildausdrücken (z.B. Variablen vom Typ Bild) wird in 4D Write Pro Bereichen unterstützt (siehe **Bildausdrücke einfügen**), aber nicht in Textbereichen mit Mehrfachstil.

Gibt *Ausdruck* einen Wert mit Zeilenschaltung und Tabulatoren zurück, formatiert 4D den Text gemäß dem Objekt, das den Ausdruck hostet; Zeilenschaltungen werden als Zeilenvorschub interpretiert.

Sie können den Inhalt von *Ausdruck* auch formatieren. Dann muss er folgende Form haben:

```
"String(value;format)"
```

... wobei *Wert* den Ausdruck selbst enthält und Format das anzuwendende Format:

- Für Zahlen: jedes Anzeigeformat für Zahlen (vorhanden oder nicht), zum Beispiel "###,##".
- Für Datum: Einen Wert für ein vorhandenes Datumsformat, z.B. 1 für die Konstante System date short
- Für Zeit: Einen Wert für ein vorhandenes Zeitformat, z.B. 9 für die Konstante System time short

Zum Beispiel:

```
"String([Table_1]Field_1;1)"  // 1 ist der Wert der Konstante System date short
```

Der Ausdruck mit Werten zeigt standardmäßig Textbereich mit Mehrfachstil an. Sie können die Anzeige mit Referenzen erzwingen, wenn Sie stattdessen den Befehl **ST SET OPTIONS** verwenden.

Die optionalen Parameter *StartAusw* und *EndeAusw* definieren eine Textauswahl in *Objekt*. Die Werte *StartAusw* und *EndeAusw* geben eine Auswahl im Plain Text, ohne evtl. vorhandene Stil Tags zu berücksichtigen.

- Übergeben Sie nur *StartAusw*, wird das Ergebnis von *Ausdruck* an der angegebenen Position eingefügt.
- Lassen Sie *StartAusw* und *EndeAusw* weg, wird das Ergebnis von *Ausdruck* an der Cursor-Position eingefügt.
- Übergeben Sie *StartAusw* und *EndeAusw*, ersetzt **ST INSERT EXPRESSION** den Text innerhalb dieser Auswahl mit dem Ergebnis von *Ausdruck*. Ist der Wert von *EndeAusw* größer als die Gesamtanzahl der Zeichen im Objekt, werden alle Zeichen zwischen *StartAusw* und dem Textende durch das Ergebnis von *Ausdruck* ersetzt.

4D bietet vordefinierte Konstanten, so dass Sie die Auswahlgrenzen in den Parametern *StartAusw* und *EndeAusw* automatisch setzen können. Diese Konstanten finden Sie unter dem Thema **Mehrfachstil Text**:

Konstante	Typ	Wert	Kommentar
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt

(*) Um diese Konstante zu nutzen, müssen Sie in *Objekt* einen Objektnamen verwenden. Übergeben Sie eine Referenz auf ein Feld oder eine Variable, wird der Befehl auf den gesamten Text des Objekts angewendet.

Hinweis: Ist *StartAusw* größer als *EndeAusw* (außer *EndeAusw* ist 0), führt der Befehl nichts aus und die Variable OK wird auf 0 gesetzt.

Beispiel

Den ausgewählten Text mit dem Ergebnis einer Projektmethode ersetzen:

```
ST INSERT EXPRESSION(*;"myText";"MyMethod";ST Start highlight;ST End highlight)
```

ST INSERT URL

ST INSERT URL ({* ;} Objekt ; urlText ; urlAdresse {; StartAusw {; EndeAusw}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
urlText	Text	➡	Sichtbarer Text der URL
urlAdresse	Text	➡	URL Adresse
StartAusw	Lange Ganzzahl	➡	Start der Auswahl
EndeAusw	Lange Ganzzahl	➡	Ende der Auswahl

Beschreibung

Der Befehl **ST INSERT URL** fügt einen URL Link in das Feld ein, definiert im Parameter *Objekt* und vom Typ formatierter Text oder Variable.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Im Parameter *urlText* übergeben Sie den sichtbaren Text der URL, wie sie in *Objekt* erscheinen soll. Das können z.B. Textbezeichnungen sein, wie "4D Web Site" oder "Für weitere Informationen klicken Sie auf diesen Link". Sie können auch die Adresse selbst verwenden, z.B. "http://www.4d.com".

Im Parameter *urlAdresse* übergeben Sie die komplette Adresse zur Seite, die der Browser aufrufen soll, z.B."http://www.4D.com".

Die optionalen Parameter *StartAusw* und *EndeAusw* definieren eine Textauswahl in *Objekt*. Die Werte *StartAusw* und *EndeAusw* geben eine Auswahl im Plain Text, ohne evtl. vorhandene Stil Tags zu berücksichtigen.

- Übergeben Sie nur *StartAusw*, wird *urlText* an der angegebenen Position eingefügt.
- Lassen Sie *StartAusw* und *EndeAusw* weg, wird *urlText* an der Cursorposition eingefügt.
- Übergeben Sie *StartAusw* und *EndeAusw*, ersetzt **ST INSERT URL** den Text innerhalb dieser Auswahl mit *urlText*. Ist der Wert von *EndeAusw* größer als die Gesamtanzahl der Zeichen im Objekt, werden alle Zeichen zwischen *StartAusw* und dem Textende mit *urlText* ersetzt.

4D bietet vordefinierte Konstanten, so dass Sie die Auswahlgrenzen in den Parametern *StartAusw* und *EndeAusw* automatisch setzen können. Diese Konstanten finden Sie unter dem Thema **Mehrfachstil Text**:

Konstante	Typ	Wert	Kommentar
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt

(*) Um diese Konstante zu nutzen, müssen Sie in *Objekt* einen Objektnamen verwenden. Übergeben Sie eine Referenz auf ein Feld oder eine Variable, wird der Befehl auf den gesamten Text des Objekts angewendet.

Hinweis: Ist *StartAusw* größer als *EndeAusw* (außer *EndeAusw* ist 0), führt der Befehl nichts aus und die Variable OK wird auf 0 gesetzt.

Der eingefügte Link ist dann aktiv: Über **Strg+click** (Windows) oder **Befehl+Klick** (OS X) auf die Bezeichnung öffnet der Standardbrowser die Seite zur Adresse, die im Parameter *urlAdresse* angegeben ist.

Beispiel

Einen Link zur 4D Web Site einfügen, um den im Objekt ausgewählten Text zu ersetzen:

```
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 ({* ;} Objekt ; StartAusw ; EndeAusw ; attrName ; attrWert {; attrName2 ; attrWert2 ; ... ; attrNameN ; attrWertN})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern ist Objekt ein Objektname (String), Ohne Stern ist Objekt eine Variable oder ein Feld
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable bzw. Feld (ohne *)
StartAusw	Lange Ganzzahl	➡	Start der neuen Textauswahl
EndeAusw	Lange Ganzzahl	➡	Ende der neuen Textauswahl
attrName	String	➡	Zu setzendes Attribut
attrWert	String, Lange Ganzzahl	➡	Neuer Wert des Attributs

Beschreibung

Der Befehl **ST SET ATTRIBUTES** verändert eine oder mehrere Stilattribute in den Formularobjekten, definiert durch *Objekt*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Hat das Objekt während der Ausführung den Fokus, gilt der Befehl nur für das Objekt in Bearbeitung und nicht seine Datenquelle (Feld oder Variable). Die Änderungen werden nur auf die Quelle (sowie alle anderen Objekte, die dieselbe Datenquelle nutzen) übertragen, wenn das Objekt in Bearbeitung entweder durch Verlieren des Fokus oder mit der **Eingabetaste** bestätigt wird. Hat das Objekt keinen Fokus, wird der Befehl direkt auf die Datenquelle und die Änderungen werden sofort an alle anderen Objekte mit derselben Datenquelle übertragen.

Ohne * geben Sie an, dass der Parameter *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstatt eines String. Der Befehl wird direkt auf das Feld oder die Variable angewandt. Änderungen werden sofort auf alle Objekte übertragen, die diese Datenquelle nutzen, inkl. das Objekt mit Fokus.

Hinweis: Stilattribute können Sie nur mit Feldern vom Typ Text verwenden. Da alphanumerische Felder eine vordefinierte Länge haben, würde das Hinzufügen von Stil Tags zu Datenverlust führen.

Die Definition eines Attributs erfolgt durch Einfügen oder Ändern der HTML Stil Tags im Text. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*. Beachten Sie, dass der Befehl **ST SET ATTRIBUTES** in allen Fällen Stil Tags einfügt, selbst wenn *Objekt* Textobjekte ohne die Eigenschaft Mehrfachstil angibt.

Mit den Parametern *StartAusw* und *EndeAusw* können Sie die Textauswahl festlegen, für welche die Stiländerungen in *Objekt* gelten sollen. In *StartAusw* übergeben Sie die Position des ersten zu ändernden Zeichens, in *EndeAusw* die Position des letzten zu ändernden Zeichens + 1 (das letzte übergebene Zeichen ist in der Auswahl nicht enthalten). Sie können in *EndeAusw* 0 übergeben, um automatisch das letzte Zeichen des Textes anzugeben (Übergeben Sie 1 in *StartAusw*, um das erste Zeichen im Text anzugeben.)

Ist der Wert von *EndeAusw* größer als die Anzahl Zeichen in *Objekt*, werden alle Zeichen zwischen *StartAusw* und dem Textende geändert. Ist *StartAusw* *größer als* *EndeAusw* (außer der Wert von *EndeAusw* ist 0, siehe oben), führt der Befehl nichts aus und die Variable OK wird auf 0 (Null) gesetzt.

Die Werte *StartAusw* und *EndeAusw* berücksichtigen nicht bereits vorhandene Stil Tags im Bereich. Sie werden vielmehr auf Basis von Rohtext bewertet, d.h. Text, aus dem Stil Tag herausgefiltert wurden.

4D bietet vordefinierte Konstanten, so dass Sie die Auswahlgrenzen in den Parametern *StartAusw* und *EndeAusw* automatisch setzen können. Diese Konstanten finden Sie unter dem Thema "**Mehrfachstil Text**":

Konstante	Typ	Wert	Kommentar
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt

(*) Für diese Konstante müssen Sie in *Objekt* einen Objektnamen übergeben. Übergeben Sie eine Referenz auf ein Feld oder eine Variable, wird die Funktion auf den gesamten Text des Objekts angewandt.

In *attrName* und *attrWert* übergeben Sie jeweils Name und Wert des zu ändernden Attributs. Sie können beliebig viele Paare Attribut/Wert übergeben. Für den Parameter *attrName* übergeben Sie vordefinierte Konstanten unter dem Thema **Mehrfachstil Textattribute**. Der in *attrWert* übergebene Parameter richtet sich nach dem Parameter *attrName*:

Konstante	Typ	Wert	Kommentar
Attribute background color	Lange Ganzzahl	8	(nur Windows) Hexadezimale Werte oder HTML Farbnamen
Attribute bold style	Lange Ganzzahl	1	*attrWert*=0: Attribut fett aus Auswahl entfernen *attrWert*=1: Attribut fett auf Auswahl anwenden
Attribute font name	Lange Ganzzahl	5	*attrWert*=Schriftfamilienname (String)
Attribute italic style	Lange Ganzzahl	2	*attrWert*=0: Attribut kursiv aus Auswahl entfernen *attrWert*=1: Attribut kursiv auf Auswahl anwenden
Attribute strikethrough style	Lange Ganzzahl	3	*attrWert*=0: Attribut durchgestrichen aus Auswahl entfernen *attrWert*=1: Attribut durchgestrichen auf Auswahl anwenden
Attribute text color	Lange Ganzzahl	7	Hexadezimale Werte oder Konstanten mit hexadezimalen Werten
Attribute text size	Lange Ganzzahl	6	*attrWert*=Anzahl Punkte (Zahl)
Attribute underline style	Lange Ganzzahl	4	*attrWert*=0: Attribut unterstrichen aus Auswahl entfernen *attrWert*=1: Attribut unterstrichen auf Auswahl anwenden

Farben
Übergeben Sie im Parameter *attrName* Attribute text color oder Attribute background color, müssen Sie in *attrWert* einen String übergeben, der entweder einen HTML Farbnamen oder einen hexadezimalen Farbwert enthält:

HTML Farbname	Hexadezimaler Wert
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

Beispiel

In diesem Beispiel ändern wir Größe und Farbe des Textes, sowie die Attribute fett und unterstrichen der Zeichen 2 bis 4 des Feldes:

```
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")
```

Systemvariablen und Mengen

Nach Ausführen dieses Befehls wird die Variable OK auf 1 gesetzt, wenn kein Fehler aufgetreten ist; andernfalls wird sie auf 0 gesetzt. Das ist insbesondere der Fall, wenn Stil Tags nicht korrekt gewertet werden (inkorrekte oder fehlende Tags).

Bei einem Fehler wird die Variable nicht geändert. Tritt ein Fehler in einer Variablen auf, während der Text gewertet wird, wandelt 4D den Text in Plain Text um; als Ergebnis werden die Zeichen <, > und & in HTML Einheiten umgewandelt.

ST SET OPTIONS

ST SET OPTIONS ({* ;} Objekt ; Option ; Wert {; Option2 ; Wert2 ; ... ; OptionN ; WertN})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Option	Lange Ganzzahl	➡	Zu setzende Option
Wert	Lange Ganzzahl	➡	Neuer Wert der Option

Beschreibung

Der Befehl **ST SET OPTIONS** ändert eine oder mehrere Optionen für das Feld, definiert im Parameter *Objekt* und vom Typ formatierter Text oder Variable.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Übergeben Sie den Code der zu ändernden Option in *Option* und ihren neuen Wert in *Wert*.

Der Parameter *Option* unterstützt folgende Konstante unter dem Thema **Mehrfachstil Text**:

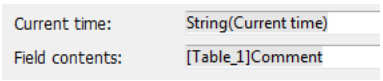
Konstante	Typ	Wert	Kommentar
ST Expressions display mode	Lange Ganzzahl	1	Der Parameter *Wert* kann ST Values oder ST References enthalten

Im Parameter *Wert* können Sie folgende Konstanten übergeben:

Konstante	Typ	Wert	Kommentar
ST References	Lange Ganzzahl	1	Zeigt Quelle der Strings für Ausdrücke
ST Values	Lange Ganzzahl	0	Zeigt berechnete Werte von Ausdrücken

Werte anzeigen:

Ausdrücke anzeigen:



Beispiel

Mit folgendem Code können Sie den Anzeigemodus des Bereichs wechseln:

```
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 ({* ;} Objekt ; NeuerText {; StartAusw {; EndeAusw}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
NeuerText	Text	➡	einzufügender Text
StartAusw	Lange Ganzzahl	➡	Beginn der Auswahl
EndeAusw	Lange Ganzzahl	➡	Ende der Auswahl

Beschreibung

Der Befehl **ST SET PLAIN TEXT** fügt den Text, der im Parameter *NeuerText* übergeben wurde, in das Textfeld oder die Variable mit Stil ein, definiert im Parameter *Objekt* . Dieser Befehl gilt nur für den reinen Text (Plain Text) des Parameters *Objekt*, ohne Ändern der darin enthaltenen Stil Tags.

Im Gegensatz zum Befehl **ST SET TEXT** fügt **ST SET PLAIN TEXT** nur Plain Text ein. *NeuerText* darf keine Stil Tags enthalten. Enthält er die Zeichen <, > oder &, werden sie als Standardzeichen gewertet und in HTML Einheiten konvertiert:

- '&' wird konvertiert in &
- '<' wird konvertiert in <
- '>' wird konvertiert in >

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Hat das Objekt während der Ausführung den Fokus, gilt der Befehl nur für das Objekt in Bearbeitung und nicht seine Datenquelle (Feld oder Variable). Die Änderungen werden nur auf die Quelle (sowie alle anderen Objekte, die dieselbe Datenquelle nutzen) übertragen, wenn das Objekt in Bearbeitung entweder durch Verlieren des Fokus oder mit der **Eingabetaste** bestätigt wird. Hat das Objekt keinen Fokus, wird der Befehl direkt auf die Datenquelle und die Änderungen werden sofort an alle anderen Objekte mit derselben Datenquelle übertragen.

Ohne * geben Sie an, dass der Parameter *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstatt eines String. Der Befehl wird direkt auf das Feld oder die Variable angewandt. Änderungen werden sofort auf alle Objekte übertragen, die diese Datenquelle nutzen, inkl. das Objekt mit Fokus.

In *NeuerText* übergeben Sie den einzufügenden Volltext.

Über die optionalen Parameter *StartAusw* und *EndeAusw* können Sie in *Objekt* eine Textauswahl angeben. Die Werte *StartAusw* und *EndeAusw* geben eine Auswahl im Plain Text, ohne im Text gefundene Stil Tags zu berücksichtigen. Die Aktion des Befehls variiert je nach den optionalen Parametern *StartAusw* und *EndeAusw*:

- Lassen Sie *StartAusw* und *EndeAusw* weg, ersetzt **ST SET PLAIN TEXT** den gesamten Text von *Objekt* durch *NeuerText*,
- Übergeben Sie nur *StartAusw* oder sind die Werte von *StartAusw* und *EndeAusw* gleich, fügt **ST SET PLAIN TEXT** den Text *in NeuerText* in *Objekt* ein, beginnend bei *StartAusw*,
- Übergeben Sie *StartAusw* und *EndeAusw*, ersetzt **ST SET PLAIN TEXT** den Plain Text, der durch diese Elemente begrenzt wird, durch den Text in *NeuerText*,
- Sie können in *EndeAusw* 0 übergeben, um automatisch das letzte Zeichen des Textes anzugeben. (Übergeben Sie 1 in *StartAusw*, um das erste Zeichen im Text anzugeben.)

4D bietet vordefinierte Konstanten, so dass Sie die Auswahlgrenzen in den Parametern *StartAusw* und *EndeAusw* automatisch setzen können. Diese Konstanten finden Sie unter dem Thema **Mehrfachstil Text**:

Konstante	Typ	Wert	Kommentar
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt

(*) Um diese Konstante zu nutzen, müssen Sie in *Objekt* einen Objektnamen verwenden. Übergeben Sie eine Referenz auf ein Feld oder eine Variable, wird der Befehl auf den gesamten Text des Objekts angewendet.

Der Stil des ersten ersetzten Zeichens wird für den gesamten Text in *NeuerText* verwendet.

Ist *StartAusw* größer *als EndeAusw* (außer der Wert von *EndeAusw* ist 0, siehe oben), wird der Text nicht geändert und die Systemvariable OK wird auf 0 gesetzt.

Beispiel

Nehmen wir z.B. folgende Rich Text (multi-style) Variable:

Please remember that the **X company has made a commitment** to you.

Sie wollen Firmennamen einfügen, die in einem Textfeld gespeichert sind. Diese Namen können z.B. das Zeichen "&" enthalten. In diesem Fall müssen Sie den Befehl **ST SET PLAIN TEXT** verwenden:

```
ST SET PLAIN TEXT(myStyledTex;[Company]Name;33;34)
```

Hier ist das Ergebnis:

Please remember that the **Smith & Jones company has made a commitment** to you.

Hier ist der Plain Text in der Variablen:

```
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.
```

Sie können sehen, dass der eingefügte Text innerhalb zusätzlicher Stil Tags steht. Diese Tags entsprechen dem Stil der Zeichen, bevor sie eingefügt wurden. Auf diese Weise können Sie die korrekte Anzeige von Feldern mit Rich Text in allen Fällen sicherstellen.

Hinweis: Würden Sie hier den Befehl **ST SET TEXT** verwenden, würde 4D nichts einfügen, da das nicht-codierte Zeichen "&" die Interpretation von Stil Tags in der Variablen verhindert. Weitere Informationen dazu finden in der Beschreibung zu diesem Befehl.

Systemvariablen und Mengen

Nach Ausführen dieses Befehls wird die Variable OK auf 1 gesetzt, wenn kein Fehler aufgetreten ist; andernfalls wird sie auf 0 gesetzt. Das ist insbesondere der Fall, wenn Stil Tags nicht korrekt gewertet werden (inkorrekte oder fehlende Tags).

Bei einem Fehler wird die Variable nicht geändert. Tritt ein Fehler in einer Variablen auf, während der Text gewertet wird, wandelt 4D den Text in Plain Text um; als Ergebnis werden die Zeichen <, > und & in HTML Einheiten umgewandelt.

ST SET TEXT

ST SET TEXT ({* ;} Objekt ; NeuerText {; StartAusw {; EndeAusw}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Textfeld oder Variable
NeuerText	Text	➡	Einzufügender Text
StartAusw	Lange Ganzzahl	➡	Start der Auswahl
EndeAusw	Lange Ganzzahl	➡	Ende der Auswahl

Beschreibung

Der Befehl **ST SET TEXT** fügt den Text im Parameter *NeuerText* in das formatierte Textfeld oder die Variable ein, die im Parameter *Objekt* übergeben ist. Dieser Befehl wird nur auf Plain Text in *Objekt* angewandt, ohne darin enthaltene Stil Tags zu verändern. Damit können Sie per Programmierung auf dem Bildschirm angezeigten formatierten Text ändern.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Hat das Objekt während der Ausführung den Fokus, gilt der Befehl nur für das Objekt in Bearbeitung und nicht seine Datenquelle (Feld oder Variable). Die Änderungen werden nur auf die Quelle (sowie alle anderen Objekte, die dieselbe Datenquelle nutzen) übertragen, wenn das Objekt in Bearbeitung entweder durch Verlieren des Fokus oder mit der **Eingabetaste** bestätigt wird. Hat das Objekt keinen Fokus, wird der Befehl direkt auf die Datenquelle und die Änderungen werden sofort an alle anderen Objekte mit derselben Datenquelle übertragen.

Ohne * geben Sie an, dass der Parameter *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstatt eines String. Der Befehl wird direkt auf das Feld oder die Variable angewandt. Änderungen werden sofort auf alle Objekte übertragen, die diese Datenquelle nutzen, inkl. das Objekt mit Fokus.

In *NeuerText* übergeben Sie den einzufügenden Text. Der Befehl **ST SET TEXT** dient zum Arbeiten mit Rich Text (multistyle) mit Tags vom Typ . In allen anderen Fällen, insbesondere beim Arbeiten mit Plain Text, der die Zeichen <, > oder & enthält, müssen Sie den Befehl **ST SET PLAIN TEXT** verwenden. Übergeben Sie Plain Text mit den Zeichen <, > oder & im Befehl **ST SET TEXT**, führt er nichts aus. Das muss so sein, denn wenn Sie einen String wie "a<b" direkt in Rich Text einfügen, würde er die interne Analyse der Tags beeinträchtigen. In diesem Fall müssen "<" Zeichen zuvor als "<" codiert werden. Das ist mit dem Befehl **ST SET PLAIN TEXT** möglich (siehe auch dazugehöriges Beispiel).

Über die optionalen Parameter *StartAusw* und *EndeAusw* können Sie in *Objekt* eine Textauswahl angeben. Die Werte *StartAusw* und *EndeAusw* geben eine Auswahl im Plain Text, ohne im Text gefundene Stil Tags zu berücksichtigen. Die Aktion des Befehls variiert je nach den optionalen Parametern *StartAusw* und *EndeAusw*:

- Lassen Sie *StartAusw* und *EndeAusw* weg, ersetzt **ST SET TEXT** den gesamten Text von *Objekt* durch *NeuerText*,
- Übergeben Sie nur *StartAusw* oder sind die Werte von *StartAusw* und *EndeAusw* gleich, fügt **ST SET TEXT** den Text *NeuerText* in *Objekt* ein, beginnend bei *StartAusw*,
- Übergeben Sie *StartAusw* und *EndeAusw*, ersetzt **ST SET TEXT** den Plain Text, der durch diese Elemente begrenzt wird, durch den Text in *NeuerText*.
- Sie können in *EndeAusw* 0 übergeben, um automatisch das letzte Zeichen des Textes anzugeben. (Übergeben Sie 1 in *StartAusw*, um das erste Zeichen im Text anzugeben.)

4D bietet vordefinierte Konstanten, so dass Sie die Auswahlgrenzen in den Parametern *StartAusw* und *EndeAusw* automatisch setzen können. Diese Konstanten finden Sie unter dem Thema **Mehrfachstil Text**:

Konstante	Typ	Wert	Kommentar
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt

(*) Beachten Sie, dass diese Konstanten nur funktionieren, wenn Sie in *Objekt* einen Objektnamen übergeben. Übergeben Sie eine Referenz auf ein Feld oder eine Variable, wird die Funktion auf den gesamten Text im Objekt angewandt.

Hinweis: Ist *StartAusw* größer als *EndeAusw* (außer wenn der Wert von *EndeAusw* 0 ist, siehe oben), wird der Text nicht geändert und die Systemvariable OK wird auf 0 gesetzt.

Systemvariablen und Mengen

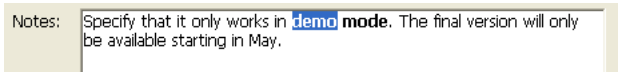
Nach Ausführen dieses Befehls wird die Variable OK auf 1 gesetzt, wenn kein Fehler aufgetreten ist; andernfalls wird sie auf 0 gesetzt. Das ist insbesondere der Fall, wenn Stil Tags nicht korrekt gewertet werden (inkorrekte oder fehlende Tags).

Bei einem Fehler wird die Variable nicht geändert. Tritt ein Fehler in einer Variablen auf, während der Text gewertet wird, wandelt 4D den Text in Plain Text um; als Ergebnis werden die Zeichen <, > und & in HTML Einheiten umgewandelt.

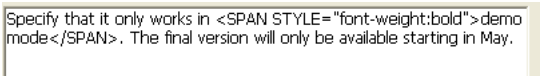
Beispiel 1

Sie wollen den vom Benutzer ausgewählten Text mit Formatierung durch den Inhalt einer Variablen ersetzen.

Hier ist der ausgewählte Text:



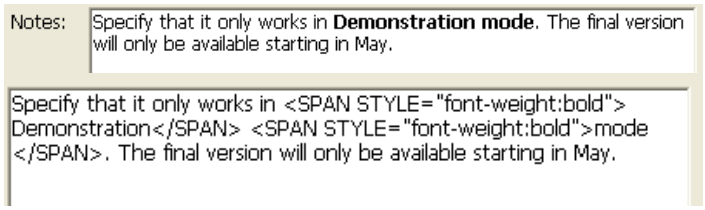
Der nachfolgende Inhalt wird im Feld gespeichert:



Nach Ausführen des Code:

```
vtempo:="Demonstration"
GET HIGHLIGHT([Products]Notes;vStart;vEnd)
ST SET TEXT([Products]Notes;vtemp;vStart;vEnd)
```

sehen Feld und Feldinhalt wie folgt aus:



Beispiel 2

Siehe Beispiel zum Befehl **ST SET PLAIN TEXT**.

Meldungen

- ALERT
- CONFIRM
- DISPLAY NOTIFICATION
- GOTO XY
- MESSAGE
- MESSAGES OFF
- MESSAGES ON
- Request

ALERT

ALERT (Meldung {; OKTitel})

Parameter	Typ		Beschreibung
Meldung	String	➡	Text der Warnung
OKTitel	String	➡	Titel der Schaltfläche OK

Beschreibung

Der Befehl **ALERT** zeigt ein Dialogfenster an, das einen Icon, eine Meldung und eine Schaltfläche OK enthält.. Mit **ALERT** können Sie dem Anwender z.B. mitteilen, dass ein Fehler aufgetreten ist oder eine Operation nicht korrekt abgeschlossen werden konnte.

Im Parameter *Meldung* setzen Sie die anzuzeigende Meldung. Sie kann bis zu 255 Zeichen enthalten. Ist sie länger als dieser Bereich, wird sie abgeschnitten.

Die Schaltfläche OK hat standardmäßig den Titel **OK**. Mit dem optionalen Parameter *OKTitel* können Sie einen eigenen Titel festlegen. Die Schaltfläche wird je nach Textlänge nach links erweitert.

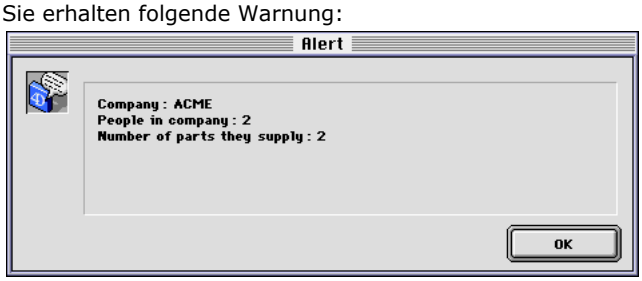
Hinweis: Rufen Sie **ALERT** nicht von Formular- oder Objektmethoden aus auf, die die Ereignisse On Activate oder On Deactivate verwalten; denn das verursacht eine Endlosschleife.

Beispiel 1

Dieses Beispiel zeigt eine Meldung mit Informationen zur Firma. Die Funktion Char(13) erzeugt einen Zeilenumbruch:

```
ALERT("Company: "+[Companies]Name+Char(13)+"People in company: "+String(Records in selection([People]))+Char(13)+"Number of parts they supply:"+String(Records in selection([Parts])))
```

Sie erhalten folgende Warnung:

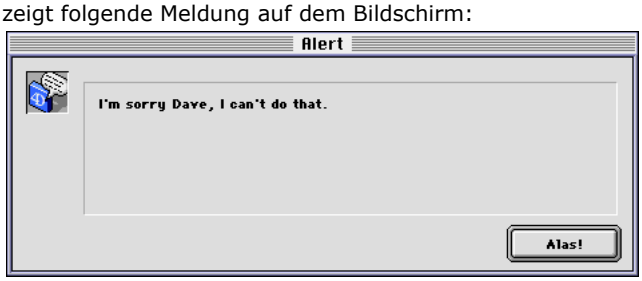


Beispiel 2

Die Zeile:

```
ALERT("I'm sorry Dave, I can't do that.";"Alas!")
```

zeigt folgende Meldung auf dem Bildschirm:

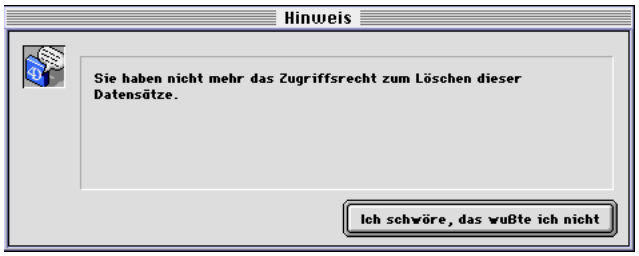


Beispiel 3

Die Zeile:

```
ALERT("Sie haben nicht mehr das Zugriffsrecht zum Löschen dieser Datensätze.";"Ich schwöre, das wußte ich nicht")
```

zeigt folgende Meldung auf dem Bildschirm:


Hinweis
Sie haben nicht mehr das Zugriffsrecht zum Löschen dieser Datensätze.
Ich schwöre, das wußte ich nicht

CONFIRM

CONFIRM (Meldung {; OKTitel {; AbbrechenTitel}})

Parameter	Typ		Beschreibung
Meldung	String	➡	Zu bestätigende Meldung
OKTitel	String	➡	Titel der Schaltfläche OK
AbbrechenTitel	String	➡	Titel der Schaltfläche Abbrechen

Beschreibung

Der Befehl **CONFIRM** zeigt einen Dialog mit dem Text *Meldung* und den beiden Schaltflächen **OK** und **Abbrechen**. Klicken Sie auf die Schaltfläche **OK** oder drücken Sie die Zeilenschaltung bzw. Eingabetaste, nimmt die Systemvariable OK den Wert 1 an. Klicken Sie auf die Schaltfläche **Abbrechen**, nimmt die Systemvariable OK den Wert 0 an.
Benutzen Sie **CONFIRM** zum Bestätigen einer Anweisung.

Im Parameter *Meldung* setzen Sie die anzuzeigende Meldung. Sie kann bis zu 255 Zeichen enthalten. Ist sie länger als dieser Bereich, wird sie abgeschnitten.

Die Schaltfläche **OK** hat standardmäßig den Titel "OK", die Schaltfläche **Abbrechen** den Titel "Abbrechen". Mit den optionalen Parametern *OKTitel* und *AbbrechenTitel* können Sie einen eigenen Titel festlegen. Die Schaltfläche wird bei Bedarf nach links erweitert.

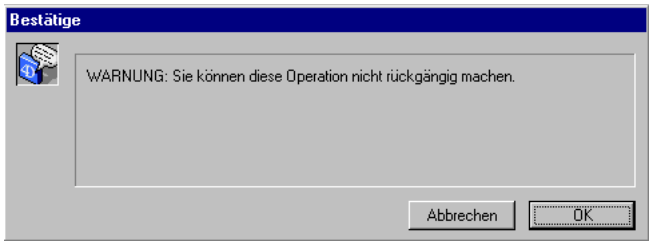
Tipp: Rufen Sie **CONFIRM** nicht in Formular- oder Objektmethoden auf, die die Ereignisse On Activate oder On Deactivate verwalten; denn das verursacht eine Endlosschleife.

Beispiel 1

Die Zeile:

```
CONFIRM("WARNUNG: Sie können diese Operation nicht rückgängig machen.")
If(OK=1)
   ALL RECORDS([Old Stuff])
   DELETE SELECTION([Old Stuff])
Else
   ALERT("Operation abgebrochen.")
End if
```

zeigt folgende Meldung auf dem Bildschirm (unter Windows):

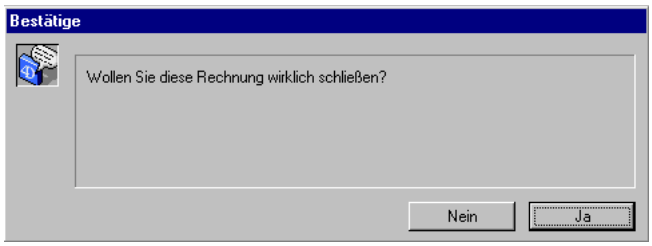


Beispiel 2

Die Zeile:

```
CONFIRM("Wollen Sie diese Rechnung wirklich schließen?";"Ja";"Nein")
```

zeigt folgende Meldung auf dem Bildschirm (unter Windows):

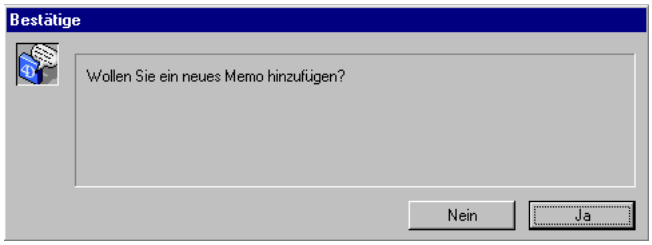


Beispiel 3

Sie entwickeln eine internationale 4D Anwendung. Sie enthält eine Projektmethode, die den korrekt lokalisierten Text der englischen Version wiedergibt. Außerdem haben Sie die gängigsten Meldungen in einem Array mit Namen *<>asLocalizedUIMessages* gespeichert. So kann die Zeile:

```
CONFIRM(INTL Text("Wollen Sie ein neues Memo hinzufügen?");<>asLocalizedUIMessages{kLoc_YES};<>asLocalizedUIMessages{kLoc_NO})
```

die Meldung in Deutsch auf dem Bildschirm anzeigen (unter Windows):



Beispiel 4

Die Zeile:

```
CONFIRM("WARNUNG: Wenn Sie fortfahren, werden einige Datensätze "+"unwiderrufbar verändert."+Char(13)+"Was wollen Sie tun?";"NICHT fortfahren";"Fortfahren")
```

zeigt folgende Meldung auf dem Bildschirm (auf Macintosh):

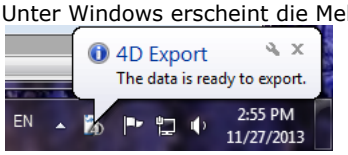
DISPLAY NOTIFICATION

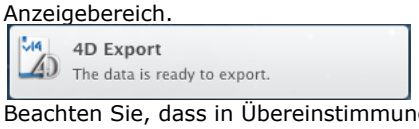
DISPLAY NOTIFICATION (Titel ; Text {; Dauer})

Parameter	Typ		Beschreibung
Titel	Alpha	➡	Titel der Nachricht
Text	Alpha	➡	Text der Nachricht
Dauer	Lange Ganzzahl	➡	Dauer der Anzeige in Sekunden

Beschreibung

Der Befehl **DISPLAY NOTIFICATION** zeigt eine Nachricht im Anzeigebereich der Task-Leiste:
Diese Art Nachricht verwendet in der Regel das Betriebssystem oder ein Programm, um den Benutzer über ein externes Ereignis zu informieren, wie Abmeldung vom Netzwerk, Verfügbarkeit eines Upgrades, o.ä..

- Unter Windows erscheint die Meldung im Anzeigebereich der Task-Leiste:


- Auf OS X (ab Version 10.8) erscheint die Meldung in der oberen rechten Ecke des Bildschirm in einem kleinen Anzeigebereich.


Beachten Sie, dass in Übereinstimmung mit Apple Spezifikationen eine Nachricht nur angezeigt wird, wenn die Anwendung nicht im Vordergrund ist. Sie erscheint dagegen weiterhin in der Liste der "Mitteilungszentrale".

In *Titel* und *Text* übergeben Sie Titel und Text der anzuzeigenden Nachricht. Sie können bis zu 255 Zeichen eingeben. Im obigen Beispiel lautet der Titel “4D Export”.
Unter Windows bleibt der Anzeigebereich erhalten, solange auf dem Rechner keine Aktivität stattfindet oder bis der Benutzer auf das Schließkästchen klickt. Der optionale Parameter *Dauer* schließt den Bereich automatisch nach Ablauf der definierten Zeitspanne. Beachten Sie, dass sich die Anzeige der Meldung nach der Systemkonfiguration richtet.

Beispiel

```
DISPLAY NOTIFICATION("4D Export";"The data is ready to export.")
```

GOTO XY

GOTO XY (PositionX ; PositionY)

Parameter	Typ		Beschreibung
PositionX	Lange Ganzzahl	➡	x (horizontal) Cursorposition
PositionY	Lange Ganzzahl	➡	y (vertikal) Cursorposition

Beschreibung

Der Befehl **GOTO XY** wird zusammen mit dem Befehl **MESSAGE** verwendet, wenn Sie Meldungen in einem Fenster anzeigen, das mit **Open window** geöffnet wurde.

GOTO XY setzt den Zeichen-Cursor (ein unsichtbarer Cursor), um die Position der nächsten Meldung im Fenster anzugeben.

Die obere linke Ecke ist Position 0,0. Der Cursor wird automatisch auf 0,0 gesetzt, wenn ein Fenster geöffnet und der Befehl **ERASE WINDOW** ausgeführt wird.

Tipp: Für die Befehle **GOTO XY** und **MESSAGE** erhalten Sie eine optimale Anzeige, wenn Sie eine Schrift mit fester Breite verwenden, wie Courier New unter Windows und Monaco auf Macintosh. Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **MESSAGE**.

Beispiel 1

Siehe Beispiel zum Befehl **MESSAGE**.

Beispiel 2

Siehe Beispiel zur Funktion **Milliseconds**.

Beispiel 3

Folgendes Beispiel:

```
Open window(50;50;300;300;5;"Dies ist nur ein Test")
For($vlRow;0;9)
   GOTO XY($vlRow;0)
   MESSAGE(String($vlRow))
End for
For($vlLine;0;9)
   GOTO XY(0;$vlLine)
   MESSAGE(String($vlLine))
End for
$vhStartTime:=Current time
Repeat
Until((Current time-$vhStartTime)>†00:00:30†)
```

zeigt folgendes Fenster 30 Sekunden lang an (auf Macintosh):

MESSAGE

MESSAGE (Meldung)

Parameter	Typ		Beschreibung
Meldung	String	➡	Zu zeigende Meldung

Beschreibung

Der Befehl **MESSAGE** zeigt *Meldung* auf dem Bildschirm in einem eigenen Fenster an. Die Meldung ist temporär und verschwindet, sobald ein Formular aufgerufen oder eine Methode mit einem Formularaufruf ausgeführt wird. Führt **MESSAGE** eine andere Meldung aus, wird die alte entfernt.

Wird ein Fenster mit der Funktion **Open window** geöffnet, erscheinen alle später über **MESSAGE** aufgerufenen Meldungen in diesem Fenster. Das Fenster verhält sich wie ein Terminal:

- Eine nachfolgende Meldung in diesem Fenster hebt die vorige Meldung nicht auf. Die Meldungen erscheinen nacheinander.
- Ist eine Meldung breiter als das Fenster, bricht 4D den Text automatisch um.
- Hat eine Meldung mehr Zeilen als das Fenster zulässt, scrollt 4D automatisch im Meldungsfenster.
- Fügen Sie Zeilenumbrüche – Char(13) – in Ihre Meldungen ein, um den Textumbruch zu steuern.
- Rufen Sie den Befehl **GOTO XY** auf, um Text an einer bestimmten Stelle im Fenster anzuzeigen.
- Rufen Sie den Befehl **ERASE WINDOW** auf, um den Inhalt des Fensters zu entfernen.
- Das Fenster ist nur ein Ausgabefenster, d.h. bei Überlappung durch andere Fenster wird es nicht neu gezeichnet.
- Über die Seite "Oberfläche" in den Datenbankeigenschaften können Sie Schrift und Größe der angezeigten Zeichen verändern.

Hinweis: MESSAGE ist kompatibel mit dem Befehl **Open form window**. Beachten Sie jedoch, dass in diesem Kontext der zweite Parameter von * von **Open form window**, der Größe und Position des Fensters sichert, **nicht** unterstützt wird.

Beispiel 1

Folgendes Beispiel bearbeitet eine Datensatzauswahl und informiert den Benutzer über **MESSAGE** über das Fortschreiten der Operation:

```
For($vlRecord;1;Records in selection([anyTable]))
   MESSAGE("Datensatz wird bearbeitet Nr."+String($vlRecord))
   ` Tu etwas mit dem Datensatz
   NEXT RECORD([anyTable])
End for
```

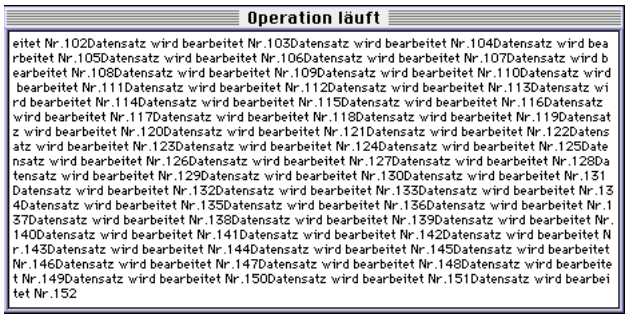
Folgendes Fenster erscheint und verschwindet bei jedem Aufruf von **MESSAGE**:

Beispiel 2

Um dieses "blinkende" Fenster zu vermeiden, können Sie die Meldungen mit der Funktion **Open window** in einem Fenster anzeigen:

```
Open window(50;50;500;250;5;"Operation läuft")
For($vlRecord;1;Records in selection([anyTable]))
   MESSAGE("Datensatz wird bearbeitet Nr."+String($vlRecord))
   ` Tu etwas mit dem Datensatz
   NEXT RECORD([anyTable])
End for
CLOSE WINDOW
```

Sie erhalten folgendes Ergebnis:



Beispiel 3

Fügen Sie die Zeilenschaltung ein:

```
Open window(50;50;500;250;5;"Operation läuft")
For($vlRecord;1;Records in selection([anyTable]))
   MESSAGE("Datensatz wird bearbeitet Nr."+String($vlRecord)+Char(13))
   ` Tu etwas mit dem Datensatz
   NEXT RECORD([anyTable])
End for
CLOSE WINDOW
```

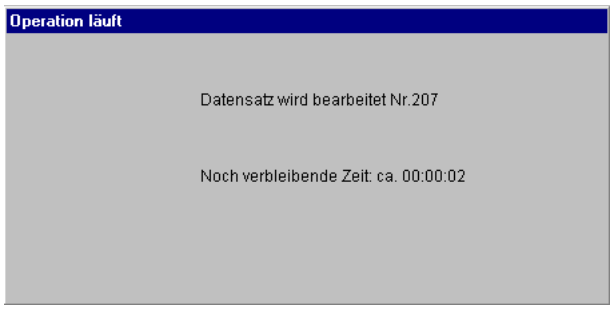
erhalten Sie ein besseres Ergebnis:

Beispiel 4

Mit **GOTO XY** und weiteren Programmierzeilen:

```
Open window(50;50;500;250;5;"Operation läuft")
$vlNbRecords:=Records in selection([anyTable])
$vhStartTime:=Current time
For($vlRecord;1;$vlNbRecords)
   GOTO XY(5;2)
   MESSAGE("Datensatz wird bearbeitet Nr."+String($vlRecord)+Char(13))
   ` Tu etwas mit dem Datensatz
   NEXT RECORD([anyTable])
   GOTO XY(5;5)
   $vlRemaining:=(($vlNbRecords/$vlRecord)-1)*(Current time-$vhStartTime)
   MESSAGE("Noch verbleibende Zeit: ca. "+Time string($vlRemaining))
End for
CLOSE WINDOW
```

erhalten Sie folgendes Ergebnis (unter Windows):



MESSAGES OFF

MESSAGES OFF
Dieser Befehl benötigt keine Parameter

Beschreibung

Die Befehle **MESSAGES OFF** und **MESSAGES ON** schalten die Ablaufanzeige aus bzw. ein, die 4D bei länger dauernden Operationen anzeigt. Die Ablaufanzeige wird standardmäßig angezeigt.

Die Ablaufanzeige erscheint bei folgenden Operationen:

Formel anwenden	Schnellbericht	Sortieren
Daten exportieren	Daten importieren	Diagramm
Suchen	Nach Formel suchen	Sucheditor

und bei folgenden Befehlen:

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

Hinweis für 4D Server: Ab 4D Server v14 R3 erscheinen die Fenster mit Ablaufmeldungen nicht mehr auf dem Server Rechner, da diese Operationen automatisch auf der **4D Server Verwaltungsfenster** des Verwaltungsfensters angezeigt werden. Wenn Sie Ablaufmeldungen anzeigen wollen, müssen Sie explizit den Befehl **MESSAGES ON** auf dem Server.

Beispiel

Folgendes Beispiel blendet die Ablaufanzeige für die Dauer des Sortierlaufs aus:

```
MESSAGES OFF
ORDER BY([Addresses];[Addresses]ZIP;>;[Addresses]Name2;>)
MESSAGES ON
```

MESSAGES ON

MESSAGES ON
Dieser Befehl benötigt keine Parameter

Beschreibung

Siehe Beschreibung zum Befehl **MESSAGES OFF**.

Request

Request (Meldung {; Standardantwort {; OKTitel {; AbbrechenTitel}}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Meldung	String	➡	Meldung zum Anzeigen im Dialogfenster der Anfrage
Standardantwort	String	➡	Standarddaten für eingebbaren Textbereich
OKTitel	String	➡	Titel für Schaltfläche OK
AbbrechenTitel	String	➡	Titel für Schaltfläche Abbrechen
Funktionsergebnis	String	↩	Vom Benutzer eingegebener Wert

Beschreibung

Die Funktion **Request** zeigt einen Anfragedialog mit einer Meldung, einem Eingabebereich für Text und den beiden Schaltflächen **OK** und **Abbrechen**.

Im Parameter *Meldung* übergeben Sie die anzuzeigende Meldung. Passt sie nicht in den Anzeigebereich (in der Regel ca. 50 Zeichen je nach Betriebssystem und verwendeter Schrift), wird sie abgeschnitten.
Die Schaltfläche OK hat standardmäßig den Titel "OK", die Schaltfläche **Abbrechen** den Titel "Abbrechen". Mit den optionalen Parametern *OKTitel* und *AbbrechenTitel* können Sie einen eigenen Titel festlegen. Die Schaltfläche wird bei Bedarf nach links erweitert.

Die Schaltfläche **OK** ist die Standardschaltfläche. Klicken Sie auf die Schaltfläche **OK** oder drücken die **Eingabetaste**, um den Dialog zu bestätigen, wird die Systemvariable OK auf 1 gesetzt. Klicken Sie auf die Schaltfläche **Abbrechen**, um den Dialog zu annullieren, wird die Systemvariable OK auf 0 gesetzt

Der Benutzer kann Text in den Eingabebereich des Dialogs eingeben oder mit dem optionalen Parameter *Standardantwort* einen Standardtext festlegen. Klickt er auf die Schaltfläche **OK**, gibt **Request** den Text zurück. Klickt er auf **Abbrechen**, gibt **Request** einen leeren String ("") zurück. Soll die Antwort ein Zahlen- oder Datumswert sein, benutzen Sie zum Umformen die Funktionen **Num** oder **Date**.

Tipp: Rufen Sie **Request** nicht in Formular- oder Objektmethoden auf, die die Formularereignisse On Activate oder On Deactivate verwalten; denn das verursacht eine Endlosschleife.

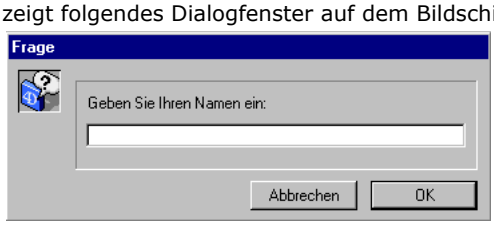
Tipp: Benötigen Sie vom Benutzer mehrere Teilinformationen, legen Sie ein Formular an, das Sie mit dem Befehl **DIALOG** aufrufen. Mit der Funktion **Request** bräuchten Sie hierfür mehrere Dialogfenster.

Beispiel 1

Die Zeile:

```
$vsPrompt :=Request("Geben Sie Ihren Namen ein:")
```

zeigt folgendes Dialogfenster auf dem Bildschirm:

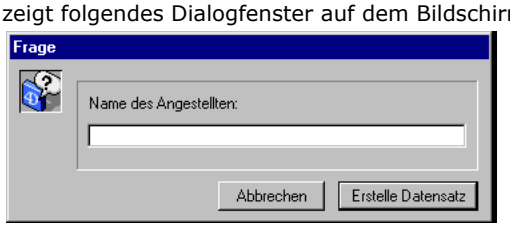


Beispiel 2

Die Zeile:

```
vsPrompt:=Request("Name des Angestellten:";"";"Erstelle Datensatz";"Abbrechen")
If(OK=1)
   ADD RECORD([Employees])
   ` Hinweis: vsPrompt wird dann in der Formularmethode während dem Ereignis On Load in das Datenfeld [Employees]Nachname kopiert
End if
```

zeigt folgendes Dialogfenster auf dem Bildschirm:

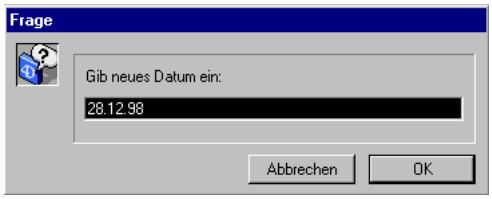


Beispiel 3

Die Zeile:

```
$vdPrompt:=Date(Request("Gib neues Datum ein:";String(Current date)))
```

zeigt folgendes Dialogfenster auf dem Bildschirm:



Mengen

- Einführung in Mengen
- 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

Einführung in Mengen

Mengen sind ein leistungsstarkes und effizientes Werkzeug zum Bearbeiten von Datensatzauswahlen. Sie können Mengen erstellen, speichern, laden und löschen, sowie der aktuellen Auswahl zuordnen. Mit 4D können Sie außerdem folgende Standardmengen bilden:

- Schnittmenge
- Vereinigungsmenge
- Differenzmenge

Mengen und die aktuelle Auswahl

Eine Menge ist die kompakte Darstellung einer aktuellen Auswahl. Menge und Auswahlen haben gewisse Ähnlichkeiten. Eine Menge ist eine boolesche Tabelle, die ausgewählte Datensätze durch 1 bzw. nicht ausgewählte durch 0 anzeigt.
Benutzen Sie Mengen hauptsächlich dann, wenn Sie:

- Im gleichen Prozess mit mehreren Auswahlen arbeiten.
- Die Auswahl kurzfristig sichern und dann wiederherstellen möchten.
- Auf eine vom Benutzer angeklickte Teilauswahl zugreifen möchten (Menge *UserSet*).
- Aus zwei Auswahlen die Vereinigungs-, Schnitt- oder Differenzmenge bilden möchten.

Sie können sich die aktuelle Auswahl als eine Liste oder Tabelle, die auf jeden Datensatz der Auswahl zeigt, vorstellen. In dieser Liste sind nur die Datensätze der aktuellen Auswahl aufgeführt. Die aktuelle Auswahl enthält nicht die Datensätze selbst, sondern nur die Zeiger auf ihre Adressen. Jeder Zeiger auf einen Datensatz belegt im Speicher 32 Bits oder 4 Bytes. Arbeiten Sie in einer Tabelle, arbeiten Sie immer mit einer aktuellen Auswahl. Sortieren Sie die aktuelle Auswahl, sortieren Sie in Wirklichkeit nur die Zeiger. Es gibt in jedem Prozess nur eine aktuelle Auswahl pro Tabelle.

Eine Menge ist die Beschreibung einer Auswahl. Jeder Datensatz wird mit einem Bit (1/8 eines Byte) dargestellt: Dieses Bit enthält die Ziffer 1 für die ausgewählten und die Ziffer 0 für die nicht ausgewählten Datensätze. Das Arbeiten mit Mengen ist daher sehr schnell. 4D muss nur wenige Informationen verarbeiten.

Die Größe einer Menge in Bits entspricht immer der Anzahl der Datensätze der Tabelle. Erzeugen Sie beispielsweise eine Menge aus einer Tabelle mit 10 000 Datensätzen, umfasst die Menge 10 000 Bits bzw. 1 250 Bytes oder 1,2 KB im RAM.

Für eine Tabelle können Sie beliebig viele Mengen erzeugen.
Mengen lassen sich unabhängig von der Datenbank auf der Festplatte speichern. Wollen Sie einen Datensatz ändern, der zu einer Menge gehört, müssen Sie die Menge zuerst zur aktuellen Auswahl machen. Hier können Sie dann einen oder mehrere Datensätze ändern.

Eine Menge ist nie sortiert, da sie nur die Information enthält, ob der Datensatz enthalten oder nicht enthalten ist. Im Gegensatz dazu ist eine temporäre Auswahl sortiert, sie benötigt jedoch in den meisten Fällen mehr Speicher. Weitere Informationen dazu finden Sie im Abschnitt **Einführung in temporäre Auswahl**.

Eine Menge merkt sich, welcher Datensatz beim Erstellen der Menge der aktuelle Datensatz war. Folgende Tabelle zeigt die Unterschiede zwischen der aktuellen Auswahl und der Menge:

Vergleichskriterium	Temporäre Auswahl	Menge
Anzahl pro Tabelle und Prozess	1	Beliebig
Sortieren	Ja	Nein
Sichern auf Festplatte	Nein	Ja
RAM pro Datensatz	(4 Bytes) pro ausgewähltem Datensatz	1 Bit (1/8 Byte) pro vorhandenem Datensatz
Kombinierbar	Nein	Ja
Rechenoperationen	Nein	Ja
Enthält aktuellen Datensatz	Ja	Ja, beim Erzeugen der Menge

Eine Menge bezieht sich immer auf die Tabelle, für die sie erzeugt wurde. Bei Operationen mit mehreren Mengen, wie Schnitt- oder Vereinigungsmengen, müssen alle Mengen der gleichen Tabelle angehören.

Mengen sind von den Datensätzen abhängig. Nach einer Änderung in der aktuellen Auswahl kann die Menge veraltet sein, vor allem beim Anlegen bzw. Löschen von Datensätzen.
Nehmen Sie beispielsweise eine Auswahl mit allen Einwohnern von Köln. Sie legen davon eine Menge an. Zieht nun ein Einwohner weg, ist die Menge nicht mehr gültig. Sie repräsentiert nicht mehr die Einwohner von Köln. Eine Menge beschreibt nur die aktuelle Auswahl zu einem bestimmten Zeitpunkt. Wird der Inhalt der Datensätze verändert oder werden Datensätze in der aktuellen Auswahl gelöscht bzw. hinzugefügt, stellt die Menge nicht mehr die aktuelle Auswahl dar.

Typen von Mengen

Es gibt folgende Arten:

- **Prozessmenge**: Sie gilt nur für den Prozess, in dem sie erzeugt wurde. Sie benötigt keine spezielle Vorsilbe im Namen und wird gelöscht, sobald die Prozessmethode endet. *LockedSet* ist eine Prozessmenge. Prozessmengen werden gelöscht, sobald die Prozessmethode endet.
- **Interprozessmenge**: Sie gilt für alle Prozesse der Arbeitsstation. Der Interprozessmenge werden die Zeichen "kleiner als, größer als" (<>) vorangestellt. Diese Syntax gilt sowohl für Windows als auch für Macintosh. Auf Macintosh können Sie auch das Zeichen ◊ benutzen. Sie erhalten es mit der Kombination Wahl- + Umschalttaste + Buchstabe v).
  Eine Interprozessmenge ist für alle Prozesse der Datenbank sichtbar.

Im Client/Server-Betrieb ist eine Interprozessmenge für Prozesse des Rechners sichtbar, wo sie erstellt wurde (Client oder Server).
Ihr Name muss in der Datenbank einmalig sein.

- **Lokale Mengen/Client Mengen**: Lokale bzw. Client Mengen dienen zur Verwendung im Client/Server-Modus. Dem Namen einer lokalen bzw. Client Menge wird immer das Dollarzeichen (*$*) vorangestellt -- mit Ausnahme der Systemmenge *UserSet*. Sie werden im Gegensatz zu anderen Mengen auf dem Client-Rechner gespeichert.

Hinweise:

- Mengennamen können max. 255 Zeichen lang sein (ohne die Symbole <> und $).
- Weitere Informationen zur Verwendung im Client/Server Betrieb finden Sie im Handbuch *4D Server* im Abschnitt **4D Server, Mengen und temporäre Auswahlen**.

Sichtbarkeit von Mengen

Nachfolgende Tabelle zeigt, wie temporäre Auswahlen sichtbar sind, je nachdem, wo sie erstellt wurden:

Sichtbarkeit temporärer Auswahlen und Mengen					
	Client Prozess	Andere Client Prozesse	Andere Clients	Server Prozess	Andere Server Prozesse
Erstellung in einem Client Prozess					
$test	x				
test	x			x (Trigger)	
<>test	x	x			
Erstellung in einem Server Prozess					
$test				x	
test				x	
<>test				x	x

Menge und Transaktion

Eine Menge lässt sich innerhalb einer Transaktion erzeugen. Sie können eine Menge mit Datensätzen erstellen, die innerhalb einer Transaktion erzeugt wurden oder eine Menge mit Datensätzen, die außerhalb einer Transaktion erstellt oder geändert wurden. Endet die Transaktion, sollten Sie die während der Transaktion erstellte Menge löschen, da sie keine zuverlässige Darstellung der Datensätze ist, insbesondere wenn die Transaktion abgebrochen wurde.

Beispiel

Folgendes Beispiel löscht doppelte Datensätze aus der Tabelle [People]. Eine Schleife *For...End for* durchläuft alle Datensätze und vergleicht den aktuellen Datensatz mit dem vorigen. Sind Name, Straße und Postleitzahl gleich, wird der Datensatz in die Menge gelegt. Am Ende der Schleife wird diese Menge die aktuelle Auswahl, die alte aktuelle Auswahl wird gelöscht:

```
CREATE EMPTY SET([People];"Duplikate")
  ` Erstelle leere Menge für doppelte Datensätze
ALL RECORDS([People])
  ` Wähle alle Datensätze aus
  ` Sortiere Datensätze nach PLZ, Straße, Name etc.
  ` so daß die doppelten aufeinander folgen
ORDER BY([People];[People]PLZ;>;[People]Address;>;[People]Name;>)
  ` Initialisiere Variablen mit den Feldern des vorigen Datensatzes
$Name:=[People]Name
$Address:=[People]Address
$PLZ:=[People]PLZ
  ` Gehe zum zweiten Datensatz, um ihn mit dem ersten zu vergleichen
NEXT RECORD([People])
For($i;2;Records in table([People]))
  ` Durchlaufe Datensätze beginnend mit 2
  ` Sind Name, Straße und PLZ gleich mit dem vorigen Datensatz,
  ` ist es ein doppelter Datensatz.
   If(([People]Name=$Name) & ([People]Address=$Address) & ([People]PLZ=$PLZ))
  ` Füge aktuellen Datensatz (den doppelten) hinzu
      ADD TO SET([People];"Duplikate")
   Else
  ` Sichere Name, Straße und PLZ dieses Datensatzes zum Vergleichen mit dem nächsten Datensatz
      $Name:=[People]Name
      $Address:=[People]Address
      $ZIP:=[People]PLZ
   End if
  ` Gehe zum nächsten Datensatz
   NEXT RECORD([People])
End for
  ` Verwende gefundene doppelte Datensätze
USE SET("Duplikate")
  ` Lösche doppelte Datensätze
DELETE SELECTION([People])
```

```
` Entferne die Menge aus dem Speicher
CLEAR SET("Duplikate")
```

Alternativ dazu können Sie am Ende der Methode die Datensätze auch zuerst auf dem Bildschirm anzeigen oder ausdrucken, um so einen genaueren Vergleich auszuführen.

Die Systemmenge UserSet

4D unterhält die Menge *UserSet*, die automatisch die aktuellste Datensatzauswahl speichert, die der Benutzer auf dem Bildschirm markiert hat. So können Sie mit **MODIFY SELECTION** oder **DISPLAY SELECTION** eine Gruppe Datensätze anzeigen, den Benutzer auffordern, davon welche auszuwählen und das Ergebnis dieser manuellen Auswahl in einer Auswahl oder eine temporären Menge zurückzugeben.

4D Server: Die Systemmenge *UserSet* ist lokal, auch wenn ihr Name nicht mit dem Dollarzeichen ($) beginnt. Stellen Sie deshalb beim Aufrufen von **INTERSECTION**, **UNION** und **DIFFERENCE** sicher, dass *UserSet* nur mit lokalen Mengen genutzt wird. Weitere Informationen dazu finden Sie in der Beschreibung dieser Befehle und im Handbuch *4D Server* im Abschnitt **4D Server, Mengen und temporäre Auswahlen**.

Es gibt nur eine Menge *UserSet* pro Prozess. Eine Tabelle hat keine eigene Menge *UserSet*. *UserSet* wird sozusagen Eigentum einer Tabelle, wenn für die Tabelle eine Datensatzauswahl angezeigt wird.

4D verwaltet die Menge *UserSet* für Listenformulare, die im Designmodus oder über die Befehle **MODIFY SELECTION** bzw. **DISPLAY SELECTION** angezeigt werden. Diese Funktionsweise ist jedoch nicht für Unterformulare aktiv.

Folgende Methode zeigt Datensätze an, lässt den Benutzer einige auswählen und zeigt dann die ausgewählten Datensätze in *UserSet* an:

```
` Zeige alle Datensätze und lass den Benutzer eine beliebige Anzahl auswählen.
` Zeige diese Auswahl in UserSet, um die aktuelle Auswahl zu ändern.
OUTPUT FORM([People];"Display") ` Setze Ausgabeformular
ALL RECORDS([People]) ` Wähle alle Personen
ALERT("Wähle die gewünschten Personen mit der Kombination Ctrl- bzw. Befehlstaste und Klick.")
DISPLAY SELECTION([People]) ` Zeige die Personen
USE SET("UserSet") ` Verwende die ausgewählten Personen
ALERT("Sie haben folgende Personen gewählt.")
DISPLAY SELECTION([People]) ` Zeige die ausgewählten Personen
```

Die Systemmenge LockedSet

Die Befehle **APPLY TO SELECTION**, **DELETE SELECTION**, **ARRAY TO SELECTION** und **JSON TO SELECTION** erzeugen in der Multiprozess-Umgebung automatisch die Menge *LockedSet*. Auch Suchbefehle erstellen eine solche Menge, wenn sie im Kontext "Suchen und Sperren" gesperrte Datensätze finden (siehe Befehl **SET QUERY AND LOCK**).
LockedSet enthält die Datensätze, die beim Ausführen des Befehls gesperrt waren.

ADD TO SET

ADD TO SET ({Tabellenname ;} Mengenname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle, zu der die Menge gehört oder Haupttabelle ohne Angabe
Mengenname	String	⇒	Name der Menge

Beschreibung

Der Befehl **ADD TO SET** fügt den aktuellen Datensatz der Tabelle *Tabellenname* zur Menge *Mengenname* hinzu. Die Menge muss bereits vorhanden sein, ansonsten erhalten Sie eine Fehlermeldung. Gibt es keinen aktuellen Datensatz für *Tabellenname*, wird der Befehl nicht ausgeführt.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich **ADD TO SET** auf die Haupttabelle.

CLEAR SET

CLEAR SET (Mengenname)

Parameter	Typ		Beschreibung
Mengenname	String	➡	Name der zu löschenden Menge

Beschreibung

Der Befehl **CLEAR SET** löscht die Menge *Mengenname* aus dem Arbeitsspeicher. Der Befehl löscht nicht die Datensätze der Menge, sondern nur die Menge selbst. Mit dem Befehl **SAVE SET** können Sie eine Menge vor dem Löschen sichern. Da Mengen Speicherplatz belegen, sollten Sie die nicht mehr benötigten Mengen immer gleich löschen.

Beispiel

Siehe Beispiel für **USE SET**.

COPY SET

COPY SET (Quellmenge ; Zielmenge)

Parameter	Typ		Beschreibung
Quellmenge	String	➡	Name der Quellmenge
Zielmenge	String	➡	Name der Zielmenge

Beschreibung

Der Befehl **COPY SET** kopiert den Inhalt von *Quellmenge* in *Zielmenge*.

Beide Mengen können Prozess-, Interprozess- oder lokale Mengen sein. Sie müssen nicht vom selben Typ sein, solange beide auf dem Rechner sichtbar sind. Weitere Informationen dazu finden Sie im Abschnitt **Sichtbarkeit von Mengen**.

Beispiel 1

Folgendes Beispiel operiert im Client/Server-Betrieb. Die lokale Menge "*$SetA"* wird vom Client-Rechner in die Prozessmenge *"SetB"* auf dem Server-Rechner kopiert:

```
COPY SET("$SetA";"SetB")
```

Beispiel 2

Folgendes Beispiel operiert im Client/Server-Betrieb. Die Prozessmenge *"SetA"* wird vom Server-Rechner in die lokale Prozessmenge *"$SetB"* auf dem Client-Rechner kopiert:

```
COPY SET("SetA";"$SetB")
```

CREATE EMPTY SET

CREATE EMPTY SET ({Tabellenname ;} Mengenname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle der zu erzeugenden Menge oder Haupttabelle ohne Angabe
Mengenname	String	⇒	Name der Menge

Beschreibung

Der Befehl **CREATE EMPTY SET** legt eine leere Menge *Mengenname* für *Tabellenname* für den laufenden Prozess an. Werden der Menge *Mengenname* die <>-Zeichen vorangestellt, gilt diese für alle Prozesse der Arbeitsstation.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich **CREATE EMPTY SET** auf die Haupttabelle. Anschließend können Sie mit dem Befehl **ADD TO SET** Datensätze zu dieser Menge hinzufügen. Gibt es bereits eine Menge mit demselben Namen, wird sie überschrieben.

Hinweis: CREATE EMPTY SET müssen Sie nicht vor **CREATE SET** aufrufen.

Beispiel

Siehe Beispiel über doppelten Datensatz im Abschnitt **Einführung in Mengen**.

CREATE SET

CREATE SET ({Tabellenname ;} Mengenname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	⇒	Tabelle der zu erzeugenden Menge, ohne Angabe Haupttabelle
Mengenname	String	⇒	Name der Menge

Beschreibung

Der Befehl **CREATE SET** erstellt die Menge *Mengenname* mit den Datensätzen der aktuellen Auswahl von *Tabellenname* für den laufenden Prozess. Werden *Mengenname* die < > -Zeichen vorangestellt, gilt diese für alle Prozesse der Arbeitsstation.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich **CREATE SET** auf die aktuelle Auswahl der Haupttabelle.

Die erzeugte Menge enthält den aktuellen Datensatz von *Tabellenname*. Besteht *Mengenname* bereits, löscht **CREATE SET** diese Menge und ersetzt sie durch die neue.

Mit **USE SET** stellen Sie die alte Auswahl und den alten aktuellen Datensatz wieder her. Es gibt keine Sortierreihenfolge. *Mengenname* verwendet die Standardreihenfolge.
Die Menge wird im Arbeitsspeicher erstellt und belegt für jeden vorhandenen Datensatz ein Bit.

Beispiel

Folgendes Beispiel erstellt nach der Suche eine Menge, die dann auf der Festplatte gespeichert wird:

```
QUERY([People]) ` Lass Benutzer eine Suche ausführen
CREATE SET([People];"SearchSet") ` Erstelle neue Menge
SAVE SET("SearchSet";"MySearch") ` Sichere Menge auf der Festplatte
```

CREATE SET FROM ARRAY

CREATE SET FROM ARRAY (Tabellenname ; DatensArray {; Mengenname})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle der Menge
DatensArray	Lange Ganzzahl, Array Boolean	➡	Array der Datensatznummern, oder Array von Booleans (True = Datensatz ist in Menge, False = Datensatz ist nicht in Menge)
Mengenname	String	➡	Name der zu erstellenden Menge, ohne Angabe verwendet Befehl UserSet

Beschreibung

Der Befehl **CREATE SET FROM ARRAY** erstellt *Mengenname* aus:

- Einem Array mit den absoluten Datensatznummern *DatensArray* aus *Tabellenname*,
- Oder einem Array Boolean *DatensArray*. In diesem Fall geben die Werte des Array an, ob die Datensätze in der Tabelle zu *MengenName* gehören (*True*) oder nicht (*False*).

Verwenden Sie diesen Befehl und übergeben in *DatensArray* ein Array Lange Ganzzahl, entsprechen alle Nummern im Array der Liste der Datensatznummern in *Mengenname*. Ist eine Nummer ungültig (zum Beispiel, wenn ein Datensatz nicht erstellt wurde), wird der Fehler -10503 erzeugt.
Verwenden Sie diesen Befehl und übergeben ein Array Boolean in *DatensArray*, gibt das N-te Element des Array an, ob der Datensatz mit Nummer N in *Mengenname* enthalten ist (*True*) oder nicht (*False*). Im Normalfall muss die Anzahl der Elemente im Array mit der Anzahl der Datensätze in der Tabelle übereinstimmen. Ist das Array kleiner als die Anzahl der Datensätze, sind in der Menge nur die Datensätze enthalten, die vom Array definiert wurden.

Hinweis: Mit dem Array Boolean verwendet **CREATE SET FROM ARRAY** Elemente mit den Nummern 0 bis N-1.

Übergeben Sie den Parameter *MengenName* nicht bzw. einen leeren String, verwendet der Befehl die Systemmenge *UserSet*.

Fehlerverwaltung

Ist eine Datensatznummer in einem Array Lange Ganzzahl ungültig, d.h. der Datensatz wird nicht angelegt, wird der Fehler -10503 generiert.

DIFFERENCE

DIFFERENCE (Menge1 ; Menge2 ; Ergebnis)

Parameter	Typ		Beschreibung
Menge1	String	➡	1. Menge
Menge2	String	➡	2. Menge
Ergebnis	String	➡	Differenzmenge

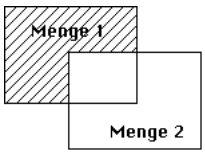
Beschreibung

Der Befehl **DIFFERENCE** ergibt die Differenz zwischen *Menge1* und *Menge2* und setzt das Ergebnis in *Ergebnis*. Beachten Sie, dass *Menge1* und *Menge2* in **DIFFERENCE** nicht vertauscht werden können, wie bei den Befehlen **UNION** oder **INTERSECTION**. Die Differenz zwischen *Menge1* und *Menge2* ist nicht gleich der Differenz zwischen *Menge2* und *Menge1*.

Folgende Tabelle zeigt die mit dem Befehl **DIFFERENCE** möglichen Ergebnisse:

Menge1	Menge2	Ergebnis
Ja	Nein	Ja
Ja	Ja	Nein
Nein	Ja	Nein
Nein	Nein	Nein

Die Differenzmenge ist in der folgenden Grafik schattiert dargestellt:



DIFFERENCE erstellt den Parameter *Ergebnis*. Dieser ersetzt alle vorhandenen Mengen mit demselben Namen, inkl. *Menge1* und *Menge2*. Beide Mengen müssen derselben Tabelle angehören. *Ergebnis* gehört dann ebenfalls zu dieser Tabelle.

4D Server: Im Client/Server-Betrieb sind Mengen sichtbar, abhängig vom Typ (Interprozess- und Prozess und lokal) und vom Erstellungsort (Server oder Client). **DIFFERENCE** wird nur ausgeführt, wenn alle drei Mengen auf demselben Rechner sichtbar sind. Weitere Informationen dazu finden Sie im Handbuch *4D Server* im Abschnitt **4D Server, Mengen und temporäre Auswahlen**.

Beispiel

Dieses Beispiel schließt die Datensätze aus, die ein Benutzer in einer angezeigten Auswahl auswählt. Die Datensätze erscheinen auf dem Bildschirm mit folgender Zeile:

```
DISPLAY SELECTION([Customers])  `Zeige Kunden in einer Liste an
```

Am Ende der Liste ist eine Schaltfläche mit Objektmethode. Diese Methode schließt die Datensätze vom Benutzer ausgewählten Datensätze aus (die Menge "UserSet") und zeigt die reduzierte Auswahl:

```
CREATE SET([Customers];"$Current")  `Menge der aktuellen Auswahl erstellen
DIFFERENCE("$Current";"UserSet";"$Current")  `Ausgewählte Datensätze ausschließen
USE SET("$Current")  `Neue Menge nehmen
CLEAR SET("$Current")  `Menge entfernen
```

INTERSECTION

INTERSECTION (Menge1 ; Menge2 ; Ergebnis)

Parameter	Typ		Beschreibung
Menge1	String	➡	1. Menge
Menge2	String	➡	2. Menge
Ergebnis	String	➡	Schnittmenge

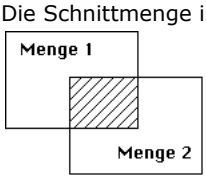
Beschreibung

Der Befehl **INTERSECTION** ergibt die Schnittmenge aus *Menge1* und *Menge2* und speichert die Ergebnismenge in *Ergebnis*. Der Parameter *Ergebnis* enthält nur die Datensätze, die in *Menge1* und *Menge2* enthalten sind.

Folgende Tabelle zeigt die mit dem Befehl **INTERSECTION** möglichen Ergebnisse:

Menge1	Menge2	Ergebnis
Ja	Nein	Nein
Ja	Ja	Ja
Nein	Ja	Nein
Nein	Nein	Nein

Die Schnittmenge ist in der folgenden Grafik schraffiert dargestellt:



INTERSECTION erstellt den Parameter *Ergebnis*. Dieser ersetzt alle vorhandenen Mengen mit demselben Namen, inkl. *Menge1* und *Menge2*. Beide Mengen müssen derselben Tabelle angehören. *Ergebnis* gehört dann ebenfalls zu dieser Tabelle. Wird derselbe aktuelle Datensatz in beiden Mengen *Menge1* und *Menge2* gesetzt, bleibt er in *Ergebnis* gespeichert. Andernfalls hat *Ergebnis* keinen aktuellen Datensatz.

4D Server: Im Client/Server-Betrieb sind Mengen sichtbar, abhängig vom Typ (Interprozess, Prozess und lokal) und vom Erstellungsort (Server oder Client). **INTERSECTION** wird nur ausgeführt, wenn alle Mengen auf demselben Rechner sichtbar sind. Weitere Informationen dazu finden Sie im Handbuch *4D Server* im Abschnitt **4D Server, Mengen und temporäre Auswahlen**.

Beispiel

Folgendes Beispiel findet die Kunden, die die beiden Vertriebsmitarbeiterinnen Susi und Andrea gemeinsam betreuen. Jede Vertriebsmitarbeiterin hat eine Menge mit ihren Kunden, dargestellt in den Mengen "Susi" und "Andrea". Die gemeinsamen Kunden erscheinen in der Menge "Beide":

```
INTERSECTION("Susi";"Andrea";"Beide") ` Legen Kunden aus beiden Mengen in Beide
USE SET("Beide") ` Benutze diese Menge
CLEAR SET("Beide") ` Lösche diese Menge, aber sichere die anderen
DISPLAY SELECTION([Customers]) ` Zeige Kunden, die von beiden betreut werden
```

Is in set

Is in set (Mengenname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Mengenname	String	⇒	Name der Menge
Funktionsergebnis	Boolean	⇐	Aktueller Datensatz der Tabellenmenge ist in der Menge (True) oder Aktueller Datensatz ist nicht in der Menge (False)

Beschreibung

Die Funktion **Is in set** prüft, ob in *Mengenname* ein aktueller Datensatz für die Tabelle enthalten ist. Sie ergibt TRUE, wenn der aktuelle Datensatz in der Menge *Mengenname* enthalten ist, sonst FALSE.
Der Tabellenname muss nicht angegeben werden, er wird durch *Mengenname* eindeutig bestimmt.

Beispiel

Folgendes Beispiel ist eine Objektmethode für eine Schaltfläche. Sie prüft, ob der aktuell angezeigte Datensatz in der Menge "Beste" enthalten ist:

```
If(Is in set("Beste"))  ` Prüfe, ob guter Kunde
    ALERT("Gehört zu Ihren besten Kunden.")
Else
    ALERT("Gehört nicht zu Ihren besten Kunden.")
End if
```

LOAD SET

LOAD SET ({Tabellenname ;} Mengenname ; Dokumentname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, zu der die Menge gehört ohne Angabe Haupttabelle
Mengenname	String	➡	Name der erzeugten Menge
Dokumentname	String	➡	Name des Dokumentes auf der Festplatte

Beschreibung

Der Befehl **LOAD SET** lädt die mit dem Befehl **SAVE SET** gespeicherte Menge *Mengenname* wieder in den Arbeitsspeicher. Die Menge **SAVE SET** für die Tabelle *Tabellenname* wird aus dem Dokument *Dokumentname* gelesen. *Dokumentname* muss nicht denselben Namen wie die Menge haben. Ist *Dokumentname* ein leerer Text, wird das Standardfenster zum Sichern der Dokumente geöffnet, so dass der Benutzer den Dokumentnamen eingeben kann. Ist die Menge bereits vorhanden, wird sie überschrieben.

Warnung: Beachten Sie, dass eine Menge die Datensatzauswahl zum Zeitpunkt der Erstellung darstellt. Sobald sich einer dieser Datensätze ändert, ist die Menge nicht mehr aktuell. Speichern Sie deshalb nur dann Mengen auf der Festplatte, wenn sich die darin enthaltenen Datensätze nicht so häufig ändern. Eine Menge wird ungültig, wenn Sie einen Datensatz der Menge ändern bzw. löschen oder ein Kriterium für die Menge ändern.

Beispiel

Folgendes Beispiel lädt mit **LOAD SET** die Menge der Zweigstellen von der Firma Acme in München:

```
LOAD SET([Companies];"MUC Acme";"MUC AcmeSt")
  ` Lade die Menge in Arbeitsspeicher
USE SET("MUC Acme")  ` Ändert die aktuelle Auswahl um in MUC Acme
CLEAR SET("MUC Acme")  ` Lösche die Menge aus dem Speicher
```

Systemvariablen und Mengen

Klickt der Benutzer im Standardfenster zum Sichern der Dokumente auf die Schaltfläche **Abbrechen** oder tritt beim Laden ein Fehler auf, nimmt die Systemvariable OK den Wert Null (0) an, sonst den Wert 1.

Records in set

Records in set (Mengenname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Mengenname	String	➡	Name der Menge
Funktionsergebnis	Lange Ganzzahl	⮌	Anzahl der zu prüfenden Datensätze

Beschreibung

Die Funktion **Records in set** gibt die Anzahl der in *Mengenname* enthaltenen Datensätze zurück. Gibt es *Mengenname* nicht oder sind in *Mengenname* keine Datensätze enthalten, gibt die Funktion den Wert Null (0) zurück.

Beispiel

Folgendes Beispiel zeigt in einer Meldung, wieviel Prozent der Kunden zu den Besten gehören:

```
  ` Berechne zuerst den Prozentsatz
$Percent :=(Records in set("Beste")/Records in table([Customers]))*100
  ` Zeige eine Meldung mit dem Prozentsatz
ALERT(String($Percent;"##0%")+" Ihrer Kunden gehören zu den Besten.")
```

REMOVE FROM SET

REMOVE FROM SET ({Tabellenname ;} Mengenname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, zu der die Menge gehört ohne Angabe Haupttabelle
Mengenname	String	➡	Name der Menge

Beschreibung

Der Befehl **REMOVE FROM SET** entfernt den aktuellen Datensatz von *Tabellenname* aus *Mengenname*. Die Menge muss bereits vorhanden sein, ansonsten erhalten Sie eine Fehlermeldung. Gibt es keinen aktuellen Datensatz für *Tabellenname*, wird der Befehl nicht ausgeführt.

SAVE SET

SAVE SET (Mengenname ; Dokumentname)

Parameter	Typ		Beschreibung
Mengenname	String	➡	Name der Menge
Dokumentname	String	➡	Name des Dokuments auf der Festplatte

Beschreibung

SAVE SET speichert die Menge *Mengenname* auf der Festplatte unter dem Namen *Dokumentname*. *Dokumentname* muss nicht denselben Namen wie die Menge haben. Ist *Dokumentname* ein leerer Text, wird das Standardfenster zum Sichern der Dokumente geöffnet, so dass der Benutzer den Dokumentnamen eingeben kann. Mit dem Befehl **LOAD SET** können Sie eine gespeicherte Menge laden.

Durch Überprüfen der Systemvariablen OK können Sie feststellen, ob die Menge gesichert wurde. OK nimmt dann den Wert 1 an, sonst den Wert 0. Die Systemvariable *Document* enthält den Namen des Dokumentes.

SAVE SET wird oft verwendet, um die Ergebnisse einer zeitaufwendigen Suche auf der Festplatte zu sichern.

Warnung: Beachten Sie, dass eine Menge die Datensatzauswahl zum Zeitpunkt der Erstellung darstellt. Sobald sich einer dieser Datensätze ändert, ist die Menge nicht mehr aktuell. Speichern Sie deshalb nur dann Mengen auf der Festplatte, wenn sich die darin enthaltenen Datensätze nicht so häufig ändern. Eine Menge wird ungültig, wenn Sie einen Datensatz der Menge ändern bzw. löschen oder ein Kriterium für die Menge ändern.

Beispiel

Folgendes Beispiel zeigt das Standardfenster zum Sichern der Dokumente, in das der Benutzer den Dokumentnamen eingeben kann:

```
SAVE SET("EineMenge";"")
```

Systemvariablen und Mengen

Klickt der Benutzer im Standardfenster zum Sichern der Dokumente auf die Schaltfläche **Abbrechen** oder tritt beim Laden ein Fehler auf, nimmt die Systemvariable OK den Wert Null (0) an, sonst den Wert 1.

UNION

UNION (Menge1 ; Menge2 ; Ergebnis)

Parameter	Typ		Beschreibung
Menge1	String	➡	1. Menge
Menge2	String	➡	2. Menge
Ergebnis	String	➡	Vereinigungsmenge

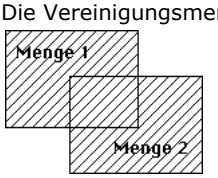
Beschreibung

UNION verknüpft *Menge1* mit *Menge2* und speichert die Gesamtmenge in *Ergebnis*. Die Menge *Ergebnis* enthält dann alle Datensätze aus *Menge1* und *Menge2*.

Folgende Tabelle zeigt die mit dem Befehl **UNION** möglichen Ergebnisse:

Menge1	Menge2	Ergebnis
Ja	Nein	Ja
Ja	Ja	Ja
Nein	Ja	Ja
Nein	Nein	Nein

Die Vereinigungsmenge ist in der folgenden Grafik schraffiert dargestellt:



UNION erstellt den Parameter *Ergebnis*. Dieser ersetzt alle vorhandenen Mengen mit demselben Namen, inkl. *Menge1* und *Menge2*. Beide Mengen müssen derselben Tabelle angehören. *Ergebnis* gehört dann ebenfalls zu dieser Tabelle.

4D Server: Im Client/Server-Betrieb sind Mengen sichtbar, abhängig vom Typ (Interprozess- und Prozess und lokal) und vom Erstellungsort (Server oder Client). **UNION** wird nur ausgeführt, wenn alle Mengen auf demselben Rechner sichtbar sind. Weitere Informationen dazu finden Sie im 4D Server Handbuch im Abschnitt **4D Server, Mengen und temporäre Auswahlen**.

Beispiel

Folgendes Beispiel fügt einer Menge mit den besten Kunden Datensätze hinzu. Die Datensätze erscheinen als Liste auf dem Bildschirm. Dann wird die Menge mit den besten Kunden von der Festplatte geladen und die Menge "UserSet", d.h. alle vom Benutzer ausgewählten Datensätze, hinzugefügt. Die neue Menge wird auf der Festplatte gespeichert:

```
ALL RECORDS([Customers])  ` Wähle alle Kunden
DISPLAY SELECTION([Customers])  ` Zeige alle Kunden als Liste an
LOAD SET("$Best";"$SaveBest")  ` Lade die Menge der besten Kunden
UNION("$Best";"UserSet";"$Best")  ` Füge alle ausgewählten der Menge hinzu
SAVE SET("$Best";"$SaveBest")  ` Sichere die neue Menge
```

USE SET

USE SET (Mengenname)

Parameter	Typ		Beschreibung
Mengenname	String	➡	Name der Menge

Beschreibung

Der Befehl **USE SET** ersetzt die aktuelle Auswahl der zur Menge gehörenden Tabelle mit der Auswahl *Mengenname*. Der aktuelle Datensatz von *Mengenname* wird zum aktuellen Datensatz der Tabelle. Löschen Sie diesen Datensatz vor Ausführen von **USE SET**, wählt 4D den ersten Datensatz der Menge zum aktuellen Datensatz. Auch beim Erstellen einer Menge, die nicht die Position des aktuellen Datensatzes enthält, wählt **USE SET** den ersten Datensatz zum aktuellen Datensatz.
Die Mengenbefehle **UNION**, **INTERSECTION**, **DIFFERENCE** und **ADD TO SET** verwerfen den alten aktuellen Datensatz.

Warnung: Beachten Sie, dass eine Menge die Datensatzauswahl zum Zeitpunkt der Erstellung darstellt. Sobald sich einer dieser Datensätze ändert, ist die Menge nicht mehr aktuell. Speichern Sie deshalb nur dann Mengen auf der Festplatte, wenn sich die darin enthaltenen Datensätze nicht ändern. Eine Menge wird ungültig, wenn Sie einen Datensatz der Menge ändern bzw. löschen oder ein Kriterium für die Menge ändern.

Beispiel

Folgendes Beispiel lädt mit **LOAD SET** die Menge der Zweigstellen von der Firma Acme in München. **USE SET** macht dann die geladene Menge zur aktuellen Auswahl:

```
LOAD SET([Companies];"MUC Acme";"MUCAcmeSt") ` Lade die Menge in den Arbeitsspeicher
USE SET("MUC Acme") ` Ändere die aktuelle Auswahl auf MUC Acme
CLEAR SET("MUC Acme") ` Lösche die Menge aus dem Speicher
```

Menüs

- Menüs verwalten
- 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

Menüs verwalten

Terminologie
Die Begriffe **Menübefehl** und **Menüzeile** sind gleichbedeutend. Sie werden bei der Beschreibung der Menübefehle abwechselnd verwendet.

MenüRef und Menü Nummern

Die Programmiersprache von 4D bietet zwei Arten zum Verwalten von Menüs und Menüleisten: über Referenzen oder Nummern.

- Die Verwaltung über Referenzen (*MenüRef*) ist die neue Art der Menüverwaltung, die mit 4D Version 11 eingeführt wird. Dieser Modus ermöglicht den Zugriff auf fortschrittliche Funktionen. Sie können vollkommen dynamische Oberflächen erstellen, also Menüs "on the fly", die nicht zuvor im Menüeditor definiert werden müssen. Sie können mehrstufige hierarchische Untermenüs verwalten.
- Die Verwaltung über Nummern basiert auf Menüs, die im Menüeditor des Designmodus erstellt werden. Jede Menüleiste und jedes Menü erhält eine feste Nummer, die sich nach der jeweiligen Position im Editor richtet. Über diese Nummer spezifizieren die Befehle der 4D Programmiersprache die Menüleiste oder das Menü, der Einsatzbereich ist beschränkt auf die aktuelle Menüleiste. Dieses Verhalten entspricht den früheren 4D Versionen und hat bestimmte Regeln (siehe unten). Die Verwaltung über Nummern ist weiter möglich, kann aber die in Version 11 neu eingeführten Funktionalitäten nicht nutzen, insbesondere die dynamische Verwaltung von Menüs oder hierarchische Untermenüs. Sie können nicht über Nummern darauf zugreifen.

Beide Arten der Menüverwaltung sind zueinander kompatibel und gleichzeitig verwendbar. Die meisten Befehle dieses Kapitels akzeptieren Menünummern und Referenzen ohne Unterschied. Wir empfehlen jedoch die Verwaltung von Menüs über Referenz, da sie mehr Möglichkeiten bietet. Wurde Ihre Menüoberfläche teilweise oder vollständig über den Menüeditor definiert, lassen sich die Menüs über die Routinen **Get menu bar reference** und **GET MENU ITEMS** über Referenzen verwalten.

Menüs über Referenz verwalten

Bei Verwalten durch Referenzen über *MenüRef* gibt es per se keine Unterscheidung zwischen Menü und Menüleiste. Beide Fälle sind Listen mit Bezeichnungen für Einträge, die sich lediglich in ihrer Verwendung unterscheiden. Bei einer Menüleiste entspricht jeder Eintrag einem Menü, das selbst aus Bezeichnungen für Einträge besteht. Dieses Prinzip verbirgt sich auch hinter hierarchischen Menüs: Jede Bezeichnung für einen Eintrag kann selbst ein Menü sein, usw.

Wird ein Menü über Referenz verwaltet, wird jede während der Sitzung ausgeführte Änderung sofort auf alle anderen Instanzen dieses Menüs innerhalb aller Prozesse der Datenbank übertragen.

MenüRef

Ein per Programmierung erstelltes Menü empfängt, wie hierarchische Listen, eine Referenznummer, auf die Sie während der Sitzung Bezug nehmen können. Diese Referenz, genannt *MenüRef*, ist eine 16-stellige alphanumerische Nummer. Alle Befehle dieses Kapitels akzeptieren entweder diese Referenz oder eine Menünummer, um ein Menü oder eine Menüleiste anzugeben. Menüreferenzen erhalten Sie über die Routinen **Create menu**, **Get menu bar reference** oder **GET MENU ITEMS**.

Menüs über Nummer verwalten

Menüleisten

Menüleisten werden im Menüeditor des Designmodus definiert. Bei Verwaltung über Nummern wird jede Menüleiste über eine Nummer und einen Namen identifiziert. Die erste Menüleiste legt 4D automatisch an, sie hat die Nummer 1 und lautet standardmäßig Menüleiste # 1. Sie können die Menüleiste im Menüeditor umbenennen. Ihr Name muss einmalig sein und kann bis zu 31 Zeichen lang sein.

Menüleiste #1 ist auch die standardmäßige Menüleiste.Wollen Sie Ihre Anwendung nicht mit Menüleiste 1, sondern mit einer anderen Menüleiste starten, definieren Sie in der **Datenbankmethode On Startup** über den Befehl **SET MENU BAR** die gewünschte Menüleiste. Sie können den Inhalt der Menüleiste nicht per Programmierung ändern, jedoch die darin enthaltenen Menüs. Die Reichweite der Programmiersprachebefehle für statische Menüs ist die aktuelle Menüleiste. Bei jedem Aufruf von **SET MENU BAR** (ohne den Parameter *) kehren alle Menüs und Menübefehle zum ursprünglichen Status zurück, wie er im Menüeditor definiert wurde.

Jede Menüleiste hat standardmäßig drei Menüs, und zwar die Menüs **Datei/Ablage**, **Bearbeiten** und **Modus**.

- Das Menü **Datei/Ablage** enthält nur eine Menüzeile: **Beenden**. Dieser ist die Standardaktion Beenden zugewiesen. Sie zeigt die Meldung "Sind Sie sicher?" an und beendet die 4D Anwendung bei Bestätigen dieses Fensters. Andernfalls wird die Aktion abgebrochen.
  Sie können das Menü umbenennen und weitere Menübefehle hinzufügen. Dabei sollte **Beenden** immer die letzte Zeile im Menü sein.

  Hinweis: Auf Mac OS X wird der Menübefehl mit der zugewiesenen Aktion **Beenden** automatisch in das Menü der Anwendung gelegt, wenn die Datenbank auf diesem System ausgeführt wird.

  Sie können das Menü umbenennen, Menübefehle hinzufügen oder es unverändert lassen. Der Menübefehl Beenden sollte aber immer der letzte Eintrag im Menü **Datei/Ablage** sein.

- Das Menü **Bearbeiten** enthält standardmäßig die Zeilen zum Bearbeiten. Jeder Zeile ist eine Standardaktion zugewiesen, z.B. Ausschneiden, Kopieren. Sie können weitere Menübefehle hinzufügen oder die Aktionen über eigene Methoden verwalten.

- Das Menü **Modus** enthält den Menübefehl Zurück zur Designumgebung. Damit können Sie von der Anwendungsumgebung zur Designumgebung zurückkehren (sofern verfügbar).

4D verwaltet standardmäßig die Systemmenüs **Hilfe** und **Anwendung** (Mac OS X). Die Menüzeile **Über 4D...** können Sie mit dem Befehl **SET ABOUT** an Ihre Anwendung anpassen. Die anderen Menüzeilen lassen sich nicht verändern.

Warnung: Menüleisten arbeiten auf Interprozessebene, d.h. jede Änderung in einer Menüleiste im Designmodus spiegelt sich in allen Prozessen wieder, die diese Menüleiste verwenden.

Nummern für Menüs und Menübefehle

Wie die Menüleisten werden auch die Menüs von links nach rechts durchnummeriert. Daher hat das Menü **Datei/Ablage** die Nummer 1, das folgende die Nummer 2 usw. Davon ausgenommen sind das Anwendungsmenü auf Mac OS und das Menü Hilfe auf beiden Plattformen. Beachten Sie, dass die Funktion **Count menus** diese Menüs nicht berücksichtigt. Enthält Ihre Menüleiste z.B. die Menüs Datei, Bearbeiten, Kunde, Rechnung und Hilfe, gibt **Count menus** 4 zurück (die von 4D verwalteten Systemmenüs werden nicht mitgezählt).
Diese Nummerierung ist z.B. für die Funktion **Menu selected** von Bedeutung. Wird ein Menü mit einem Formular zugeordnet, ist die Nummerierung anders. Das erste angehängte Menü beginnt mit der Nummer 2049, die anderen Menüs werden um 1 hochgezählt. Um auf ein angehängtes Menü zu verweisen, fügen Sie 2048 zur normalen Menünummer hinzu.

Die Menüzeilen werden ebenfalls von oben nach unten durchnummeriert, inkl. der dazwischenliegenden Trenner. Das oberste Menü beginnt mit 1.

Verbundene Menüleisten

Sie können einem Formular eine Menüleiste zuweisen. Rufen Sie dazu auf der Seite Allgemein die Formulareigenschaften auf. Wir nennen solche Menüleisten im folgenden "Formularmenüleiste". Erscheint das Formular als Ausgabeformular im Anwendungsmodus, werden die Menüs der Formularmenüleiste bei Aufruf des entsprechenden Formulars an die aktuelle Menüleiste angefügt.

Wird ein Formular mit einer eigenen Menüleiste angezeigt, sind die Befehle der aktuellen Menüleiste standardmäßig deaktiviert, d.h. sie können nicht gewählt werden. Dieses Verhalten können Sie ändern, wenn Sie in den Formulareigenschaften die Option "Aktive Menüleiste" markieren. In diesem Fall bleiben auch die Befehle der aktuellen Menüleiste aktiv.
In beiden Fällen bewirkt die Auswahl eines Menübefehls, dass das Ereignis On Menu Selected an die Formularmethode gesendet wird; Sie können dann mit der Funktion **Menu selected** das gewählte Menü testen.

Angefügte Menüs

Sie können Menüs an Menüleisten anfügen. Ändern Sie ein angefügtes Menü über einen Befehl dieses Kapitels, wirkt sich das auf alle anderen Instanzen dieses Menüs aus. Weitere Informationen finden Sie im Abschnitt **Menü an Menüleiste anfügen** des Handbuchs *4D Designmodus*.

Den Menüs zugewiesene Standardaktionen und Methoden

Jeder Menüzeile können Sie eine Projektmethode oder eine Standardaktion zuweisen. Die Projektmethode wird ausgeführt, sobald die Menüzeile ausgewählt wurde. Weisen Sie einer Menüzeile keine Projektmethode oder Standardaktion zu und wählt der Anwender diese Zeile aus, verlässt 4D die Anwendungsumgebung und wechselt zur Designumgebung. Ist nur die Anwendungsumgebung verfügbar oder hat der Benutzer keinen Zugriff auf die Designumgebung, wird das Programm beendet.
Über Standardaktionen können Sie Operationen ausführen, die zum System gehören (Kopieren, Beenden, etc.) oder zur 4D Datenbank (Datensatz hinzufügen, Alle auswählen, etc.)
Sie können einem Menübefehl auch eine Standardaktion und eine Projektmethode zuweisen. Dann wird die Standardaktion nie ausgeführt; 4D benutzt die Aktion jedoch, um den Menübefehl je nach Kontext zu aktivieren/deaktivieren und um je nach Plattform eine spezifische Operation zuzuweisen. Im Anwendungsmenü unter Mac OS wird z.B. die Aktion Voreinstellungen übergeben. Ist ein Menübefehl deaktiviert, kann die zugeordnete Projektmethode nicht ausgeführt werden.

Menüzeile = -1

Um die Verwaltung von Menüs zu vereinfachen, bietet 4D eine Abkürzung, um den zuletzt im Menü hinzugefügten Eintrag anzugeben: Dafür müssen Sie lediglich im Parameter *MenüZeile* den Wert -1 übergeben. Diese Vorgehensweise können Sie in allen Befehlen dieses Kapitels anwenden, die mit Menüeinträgen arbeiten.

APPEND MENU ITEM

APPEND MENU ITEM (Menü ; ZeileText {; Untermenü {; Prozess {; *}}})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl	➡	Menünummer oder Menüreferenz
ZeileText	Text	➡	Text für neuen Menüeintrag
Untermenü	MenüRef	➡	Referenz des der Zeile zugeordneten Untermenüs
Prozess	Lange Ganzzahl	➡	Referenznummer des Prozesses
*	Operator	➡	Mit *: Metazeichen als Standardzeichen werten

Beschreibung

Der Befehl **APPEND MENU ITEM** hängt neue Menüzeilen an das Menü an mit der Nummer oder Referenz, übergeben in *Menü*.

Prozess ist optional. Geben Sie den Parameter nicht an, gilt **APPEND MENU ITEM** für die Menüleiste des aktuellen Prozesses. Sonst gilt der Befehl für die Menüleiste für den Prozess mit der Referenznummer, übergeben in *Prozess*.

Hinweis: Übergeben Sie in *Menü* den Parameter *MenüRef*, hat der Parameter *Prozess* keine Verwendung und wird ignoriert.

Übergeben Sie nicht den Parameter *, ermöglicht **APPEND MENU ITEM**, eine oder mehrere Menüzeilen in einem Aufruf anzuhängen.

Sie können anzuhängende Einträge mit dem Parameter *ZeileText* wie folgt anhängen:

- Trennen Sie die Zeilen mit einem Strichpunkt (*;*) voneinander. Zum Beispiel *"Zeilentext1;Zeilentext2;Zeilentext3".*
- Deaktivieren Sie eine Zeile durch eine geöffnete Klammer (*(*) im Zeilentext
- Für eine Trennungslinie geben Sie "-" oder *"(-"* ein
- Den Schriftstil für eine Zeile definieren Sie mit dem kleiner als Zeichen (<) und folgenden Buchstaben:
  - <B Fett
  - <I Kursiv
  - <U Unterstrichen

- Eine Markierung für eine Zeile fügen Sie mit dem Ausrufezeichen (!) und dem gewünschten Zeichen hinzu. Das Zeichen wird nur auf Macintosh berücksichtigt, unter Windows erscheint immer die Standardmarkierung.
- Ein Icon fügen Sie mit dem Circumflex Zeichen (^) und dem Zeichen mit dem Code + 208 hinzu. Das ist die Ressourcen ID für das Icon auf Macintosh.
- Ein Tastaturkürzel fügen Sie mit dem Schrägstrich (/) und dem gewünschten Kürzel hinzu.
- (**Ab 4D v16 R3**) Ist dem Eintrag eine Standardaktion zugewiesen, übergeben Sie in *ZeileText* die Konstante ak standard action title, um automatisch die lokalisierte Aktion und optional weitere Angaben zu verwenden, z.B. "Erneut <vorige Aktion>".

Hinweis: Verwenden Sie überschaubare Menüs. Bei mehr als 50 Menüzeilen sollten Sie anstatt eines Menüs eine Listbox in einem Formular einsetzen.

Übergeben Sie den Parameter *, werden Sonderzeichen im Text, wie z.B. "(; !...", für Einträge als standardmäßige Zeichen und nicht als Metazeichen behandelt, d.h. Sie können für Einträge Bezeichnungen wie "**Kopieren (spezial)...**" oder "**Suchen/Ersetzen...**" vergeben. Ist der Parameter * übergeben, können Sie in einem Aufruf nicht mehrere Einträge erstellen, da das Zeichen ";" als standardmäßiges Zeichen gewertet wird.

Hinweis: Die Routinen **GET MENU ITEMS** und **Get menu item** geben je nach Erstellung Metazeichen oder keine Metazeichen zurück: Wurden sie mit der Option * angelegt, werden die Metazeichen als Standardzeichen zurückgegeben.

Der optionale Parameter *UnterMenü* kann ein Menü als hinzugefügten Eintrag angeben und so ein hierarchisches Untermenü anhängen. Sie müssen eine Menüreferenz vom Typ String übergeben, die ein erstelltes Menü angibt (z.B. über die Funktion **Create menu**). Fügt der Befehl mehr als einen Menüeintrag hinzu, wird das Untermenü dem ersten Eintrag zugeordnet.

Wichtig: Den neuen Menüzeilen sind keine Methoden oder Aktionen zugeordnet. Diese müssen Sie den Menüzeilen über die Befehl **SET MENU ITEM PROPERTY** oder **SET MENU ITEM METHOD** zuordnen. Sie können die Zeilen auch über eine Formularmethode mit der Funktion **Menu selected** verwalten.

Beispiel

Dieses Beispiel hängt die Namen der verfügbaren Schriften im Schriftenmenü an. Hier ist es das sechste Menü der aktuellen Menüleiste:

```
` In der Datenbankmethode On Startup
` wird die Schriftenliste geladen und der Text für Menüzeilen aufgebaut
FONT LIST(◇asAvailableFont)
◇atFontMenuItems:=""
For($vlFont;1;Size of array(◇asAvailableFont))
   ◇atFontMenuItems:=◇atFontMenuItems+";"+◇asAvailableFont{$vlFont}
End for
```

Sie können nun in jeder Formular- oder Projektmethode schreiben:

```
APPEND MENU ITEM(6;◇atFontMenuItems)
```

Count menu items

Count menu items (Menü {; Prozess}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüreferenz
Prozess	Lange Ganzzahl	➡	Referenznummer des Prozesses
Funktionsergebnis	Lange Ganzzahl	⮌	Anzahl der Menübefehle im Menü

Beschreibung

Die Funktion **Count menu items** gibt die Anzahl der Menübefehle für das Menü zurück mit der in *Menü* übergebenen Nummer oder Referenz.

Prozess ist optional. Geben Sie den Parameter nicht an, gilt **Count menu items** für die Menüleiste des aktuellen Prozesses. Sonst gilt **Count menu items** für die Menüleiste des Prozesses mit der in *Prozess* übergebenen Referenznummer

Hinweis: Übergeben Sie in *Menü* einen Parameter *MenüRef*, hat der Parameter *Prozess* keine Verwendung und wird ignoriert.

Count menus

Count menus {(Prozess)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Prozess	Lange Ganzzahl	⇒	Nummer des Prozesses
Funktionsergebnis	Lange Ganzzahl	⇐	Anzahl der Menüs in der aktuellen Menüleiste.

Beschreibung

Die Funktion **Count menus** gibt die Anzahl der Menüs in der Menüleiste zurück.
Prozess ist optional. Geben Sie den Parameter nicht an, gilt **Count menus** für die Menüleiste des aktuellen Prozesses.

Create menu

Create menu {(Menü)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Menü	MenüRef, Lange Ganzzahl, String	➡	Menüreferenz oder Nummer oder Name des Menüs
Funktionsergebnis	MenüRef	⮌	Menüreferenz

Beschreibung

Die Funktion **Create menu** erstellt ein neues Menü im Speicher, das dann nur im Speicher existiert und nicht dem Menüeditor im Designmodus hinzugefügt wird. Alle während der Sitzung ausgeführten Änderungen in diesem Menü werden sofort auf alle anderen Instanzen dieses Menüs innerhalb aller Prozesse der Datenbank übertragen.

Die Funktion gibt die Referenznummer vom Typ *MenüRef* für das neue Menü zurück.

- Übergeben Sie nicht den optionalen Parameter *Menü*, wird ein leeres Menü erstellt. Sie müssen es dann mit den Befehlen **RELEASE MENU**, **SET MENU ITEM** einrichten und verwalten.
- Übergeben Sie den optionalen Parameter *Menü*, ist das erstellte Menü die exakte Kopie des Quellmenüs, das durch diesen Parameter bestimmt wird. Alle Eigenschaften des Quellmenüs, inkl. zugewiesener Untermenüs, werden auf das neue Menü angewandt. Beachten Sie, dass eine neue Referenz *MenüRef* für das Quellmenü und für alle vorhandenen Untermenüs, die ihm zugewiesen sind, erstellt wird.

Im Parameter *Menü* können Sie entweder eine gültige Menüreferenz, oder Nummer bzw. Name einer Menüleiste übergeben, die im Designmodus definiert wurde. Dann wird das neue Menü aus den Menüs und Untermenüs der Quellmenüleiste zusammengestellt.

Hinweis: Übergeben Sie in *Menü* einen ungültigen Wert, wird ein leeres Menü angelegt.

Ein mit dieser Funktion erstelltes Menü lässt sich über den Befehl **SET MENU BAR** als Menüleiste verwenden.

Benötigen Sie das mit **Create menu** erstellte Menü nicht mehr, denken Sie daran, den Befehl **RELEASE MENU** aufzurufen, um den verwendeten Speicher wieder frei zu machen.

Beispiel

Siehe Beispiel unter dem Befehl **SET MENU BAR**.

DELETE MENU ITEM

DELETE MENU ITEM (Menü ; MenüZeile {; Prozessnr})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüreferenz
MenüZeile	Lange Ganzzahl	➡	Nummer der Menüzeile oder -1 für zuletzt hinzugefügte Menüzeile
Prozessnr	Lange Ganzzahl	➡	Referenznummer des Prozesses

Beschreibung

Der Befehl **DELETE MENU ITEM** löscht die in *MenüZeile* übergebene Menüzeile aus der in *Menü* übergebenen Menünummer oder Menüreferenz. Sie können in *MenüZeile* -1 übergeben, um die zuletzt im Menü hinzugefügte Zeile zu spezifizieren.

Ist die Menüzeile, definiert durch *Menü* und *MenüZeile* selbst ein Menü, das über Referenz verwaltet und z.B. mit der Funktion **Create menu** erstellt wurde, löscht **DELETE MENU ITEM** im Parameter *Menü* nur die Instanz von *MenüZeile*. Das Untermenü, auf das *MenüZeile* Bezug nimmt, ist im Speicher weiterhin vorhanden. Um ein über Referenz verwaltetes Menü definitiv zu löschen, müssen Sie den Befehl **RELEASE MENU** verwenden. Der Befehl arbeitet auch mit Menüleisten, die über die Funktion **Create menu** erstellt und über **SET MENU BAR** installiert wurden.

Prozessnr ist optional. Geben Sie den Parameter nicht an, gilt **DELETE MENU ITEM** für die Menüleiste des aktuellen Prozesses. Sonst gilt der Befehl für den Prozess mit der in *Prozessnr* übergebenen Referenznummer.

Hinweis: Löschen Sie zur Wahrung der Benutzeroberfläche Menüs ohne Menüzeilen.

Hinweis: Übergeben Sie in *Menü* den Parametertyp *MenüRef*, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert.

DISABLE MENU ITEM

DISABLE MENU ITEM (Menü ; MenüZeile {; Prozessnr})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüeintrag
MenüZeile	Lange Ganzzahl	➡	Nummer der Zeile oder -1 für zuletzt hinzugefügte Menüzeile
Prozessnr	Lange Ganzzahl	➡	Referenznummer des Prozesses

Beschreibung

Der Befehl **DISABLE MENU ITEM** deaktiviert die in *MenüZeile* übergebene Menüzeile aus der in *Menü* übergebenen Menünummer oder Menüreferenz. Sie können in *MenüZeile* -1 übergeben, um die zuletzt im Menü hinzugefügte Zeile zu spezifizieren.

Prozessnr ist optional. Geben Sie den Parameter nicht an, gilt **DISABLE MENU ITEM** für die Menüleiste des aktuellen Prozesses. Sonst gilt der Befehl für den Prozess mit der in *Prozessnr* übergebenen Referenznummer.

Gibt der Parameter *MenüZeile* ein hierarchisches Untermenü an, werden alle Zeilen dieses Menüs und dazugehörige Untermenüs deaktiviert. Der Befehl arbeitet auch mit Menüleisten, die über die Funktion **Create menu** erstellt und über **SET MENU BAR** installiert wurden.

Hinweis: Übergeben Sie in *Menü* den Parametertyp *MenüRef*, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert.

Tipp: Um alle Zeilen eines Menüs auf einmal zu aktivieren/deaktivieren, übergeben Sie in *MenüZeile* den Wert 0 (Null).

Dynamic pop up menu

Dynamic pop up menu (Menü {; Standard {; xKoord ; yKoord}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Menü	MenüRef	➡	Menüreferenz
Standard	String	➡	Parameter des standardmäßig ausgewählten Eintrags
xKoord	Lange Ganzzahl	➡	X Koordinate der oberen linken Ecke
yKoord	Lange Ganzzahl	➡	Y Koordinate der oberen linken Ecke
Funktionsergebnis	String	⬅	Parameter des gewählten Menüeintrags

Beschreibung

Die Funktion **Dynamic pop up menu** zeigt ein hierarchisches PopUp-Menü an der aktuellen Mausposition oder an der Position, die durch die optionalen Parameter *xKoord* und *yKoord* festgelegt wird.
Sie müssen das hierarchische Menü zuvor mit der Funktion **Create menu** erstellen. Die von **Create menu** zurückgegebene Referenz wird dann im Parameter *Menü* übergeben.

Dieser Befehl wird, in Übereinstimmung mit den geltenden Regeln für die Oberfläche, in der Regel durch einen rechten Mausklick oder längeres Drücken der Maustaste (z.B. Kontextmenü) aufgerufen.

Hinweis: Mit der Funktion **Pop up menu** (Kapitel "Benutzeroberfläche") können Sie PopUp-Menüs erstellen, die auf Text basieren.

Der optionale Parameter *Standard* ermöglicht, einen Eintrag im PopUp-Menü als Standardauswahl festzulegen, immer wenn das Menü erscheint. In diesem Parameter übergeben Sie einen eigenen Text für den Menüeintrag. Diesen müssen Sie zuvor über den Befehl **SET MENU ITEM PARAMETER** definiert haben.
Übergeben Sie diesen Parameter nicht, wird standardmäßig der erste Eintrag im Menü ausgewählt.

Über die optionalen Parameter *xKoord* und *yKoord* können Sie die Position des anzuzeigenden PopUp-Menüs angeben. Hier übergeben Sie jeweils die waagrechten und senkrechten Koordinaten der oberen linken Ecke des Menüs. Die Koordinaten müssen im lokalen Koordinatensystem des aktuellen Formulars in Pixel angegeben werden. Beide Parameter müssen zusammen übergeben werden; ist nur einer übergeben, wird er ignoriert.

Wollen Sie ein PopUp-Menü anzeigen, das einer 3D Schaltfläche zugewiesen ist, übergeben Sie diese Parameter nicht. In diesem Fall berechnet 4D automatisch die Position des Menüs in Bezug auf die Schaltfläche gemäß den Oberflächen-Standards der aktuellen Plattform.

Wurde ein Menüeintrag ausgewählt, gibt der Befehl den eigenen zugewiesenen Text zurück, und zwar so, wie er mit dem Befehl **SET MENU ITEM PARAMETER** definiert wurde. Sonst gibt die Funktion einen leeren String zurück.

Ab 4D v16 R3: Ist einem Menüeintrag eine Standardaktion zugewiesen, berücksichtigt die Funktion **Dynamic pop up menu** das auf verschiedenen Ebenen:

- Ist eine Standardaktion im Kontext des PopUp-Menüs nicht aktiviert, sie kann z.B. nicht ausgelöst werden, wird der Eintrag automatisch ausgeblendet. Über die Funktion **Get action info** erfahren Sie, ob eine Aktion aktiviert ist.
- Einträge mit einer zuwiesenen Umschaltaktion sind je nach Auswahl automatisch markiert, nicht markiert oder "gemischt".
- Wurde der Aktionstitel für den Eintrag über die Konstante [#cst id="3300366"/] gesetzt, erscheint im Menü der lokalisierte Titel.
- Ist ein Eintrag ausgewählt, wird die zugewiesene Standardaktion ausgelöst. Die Ausführung ist asynchron.

Beispiel

Dieser Code erstellt ein hierarchisches dynamisches PopUp-Menü mit Standardaktionen:

```
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;"-")
 //Untermenü Text für Bearbeiten
$refMenuEdit:=Create menu
APPEND MENU ITEM($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 (Menü ; MenüZeile {; Prozessnr})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüreferenz
MenüZeile	Lange Ganzzahl	➡	Nummer der Zeile oder -1 für zuletzt hinzugefügte Menüzeile
Prozessnr	Lange Ganzzahl	➡	Referenznummer des Prozesses

Beschreibung

Der Befehl **ENABLE MENU ITEM** aktiviert die in *MenüZeile* übergebene Menüzeile aus der in *Menü* übergebenen Menünummer oder Menüreferenz. Sie können in *MenüZeile* -1 übergeben, um die zuletzt im Menü hinzugefügte Zeile zu spezifizieren.

Gibt der Parameter *MenüZeile* ein hierarchisches Untermenü an, werden alle Zeilen dieses Menüs und dazugehörige Untermenüs aktiviert. Der Befehl arbeitet auch mit Menüleisten, die über die Funktion **Create menu** erstellt und über **SET MENU BAR** installiert wurden.

Prozessnr ist optional. Geben Sie den Parameter nicht an, gilt **ENABLE MENU ITEM** für die Menüleiste des aktuellen Prozesses. Sonst gilt der Befehl für den Prozess mit der in *Prozessnr* übergebenen Referenznummer.

Hinweis: Übergeben Sie in *Menü* den Parametertyp *MenüRef*, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert.

Tipp: Um alle Einträge eines Menüs auf einmal zu aktivieren/deaktivieren, übergeben Sie in *MenüZeile* den Wert 0 (Null).

Get menu bar reference

Get menu bar reference {(Prozessnr)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Prozessnr	Lange Ganzzahl	➡	Referenznummer des Prozesses
Funktionsergebnis	MenüRef	⮌	Referenznummer der Menüleiste

Beschreibung

Die Funktion **Get menu bar reference** gibt die Referenznummer der aktuellen Menüleiste oder der Menüleiste eines bestimmtes Prozesses zurück.

Wurde die Menüleiste mit der Funktion **Create menu** erstellt, entspricht die Nummer der Referenznummer des erstellten Menüs. Andernfalls gibt der Befehl eine spezifische interne Kennummer(*) zurück. Diese Nummer kann in jedem Fall für alle anderen Befehle dieses Kapitels als Referenz auf die Menüleiste dienen.

(*) In 64-bit Versionen von 4D ist diese spezifische Nummer temporär und wird ungültig, sobald mit **SET MENU BAR** eine andere Menüleiste aufgerufen wird. Wollen Sie die Referenz auf ein Menü im Menüeditor beibehalten, müssen Sie diese mit **Create menu** in den Speicher kopieren. Beispiel:

```
$vEditorRef:=Get menu bar reference(Frontmost process)  //Menü aus dem Menüleisteneditor
$vMenuRef:=Create menu($vEditorRef)  //das Menü in den Speicher kopieren
SET MENU BAR(2)  //eine andere Menüleiste installieren
... // Code ausführen
SET MENU BAR($vMenuRef)  //zurück zur ersten Menüleiste
```

Der Parameter *Prozessnr* bezeichnet den Prozess, in welchem Sie die Referenznummer der aktuellen Menüleiste erhalten wollen. Lassen Sie diesen Parameter weg, gibt die Funktion die Kennummer der Menüleiste des aktuellen Prozesses zurück.

Beispiel

Siehe Beispiel zum Befehl **GET MENU ITEMS**.

Get menu item

Get menu item (Menü ; MenüZeile {; Prozessnr}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüreferenz
MenüZeile	Lange Ganzzahl	➡	Nummer der Zeile oder -1 für zuletzt hinzugefügte Menüzeile
Prozessnr	Lange Ganzzahl	➡	Referenznummer des Prozesses
Funktionsergebnis	String	↩	Text der Menüzeile

Beschreibung

Die Funktion **Get menu item** gibt den Text von *MenüZeile* in *Menü* zurück. Sie können in *MenüZeile* -1 übergeben, um die zuletzt im Menü hinzugefügte Zeile zu definieren.

Prozessnr ist optional. Geben Sie den Parameter nicht an, gilt **Get menu item** für die Menüleiste des aktuellen Prozesses. Sonst gilt die Funktion für den Prozess mit der in *Prozessnr* übergebenen Referenznummer.

Hinweis: Übergeben Sie in *Menü* den Parametertyp *MenüRef*, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert.

GET MENU ITEM ICON

GET MENU ITEM ICON (Menü ; MenüEintrag ; IconRef {; Prozessnr})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menüreferenz oder Menünummer
MenüEintrag	Lange Ganzzahl	➡	Nummer des Menüeintrags oder -1 für den zuletzt hinzugefügten Menüeintrag
IconRef	Textvariable, Variable Lange Ganzzahl	⬅	Name oder Nummer des Bildes, das dem Menüeintrag zugeordnet ist.
Prozessnr	Lange Ganzzahl	➡	Prozessnummer

Beschreibung

Der Befehl **GET MENU ITEM ICON** gibt in der Variablen *IconRef* die Referenz jedes Icons zurück, der dem Menüeintrag zugewiesen ist, definiert durch die Parameter *Menü* und *MenüEintrag*. Diese Referenz ist Name oder Nummer des Bildes. Sie können in Menüeintrag -1 übergeben, um den zuletzt im Menü hinzugefügten Eintrag zu spezifizieren.

In *Menü* übergeben Sie eine Menüreferenz (*MenüRef*) oder eine Menünummer.
Übergeben Sie eine Menüreferenz, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert, falls er übergeben ist. Übergeben Sie eine Menünummer, berücksichtigt der Befehl das entsprechende Menü in der Hauptmenüleiste des aktuellen Prozesses. Wollen Sie einen anderen Prozess festlegen, übergeben Sie dessen Nummer im optionalen Parameter *Prozessnr*.

Wurde das Icon über ein Bild der Bildbibliothek definiert, gibt der Befehl je nach dem in *IconRef* übergebenem Variablentyp entweder Name oder Nummer des Bildes zurück. Wurde das Icon über ein Bild im Ordner **Resources** der Datenbank definiert, gibt der Befehl in *IconRef* den Pfadnamen des Bildes zurück.
Geben Sie in *IconRef* keinen bestimmten Typ an, wird standardmäßig der Bildname zurückgegeben, d.h. der Typ Text.

Ist dem Menüeintrag kein Icon zugewiesen, gibt der Befehl einen leeren Wert zurück.

Get menu item key

Get menu item key (Menü ; MenüZeile {; Prozessnr}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüreferenz
MenüZeile	Lange Ganzzahl	➡	Nummer der Zeile oder -1 für zuletzt hinzugefügte Menüzeile
Prozessnr	Lange Ganzzahl	➡	Referenznummer des Prozesses
Funktionsergebnis	Lange Ganzzahl	↩	Code für Standard Tastenkürzel der Zeile

Beschreibung

Die Funktion **Get menu item key** gibt den Code für das Tastaturkürzel (**ctrl-Taste** unter Windows, **Befehlstaste** auf Macintosh) der in *MenüZeile* übergebenen Menüzeile aus der in *Menü* übergebenen Menünummer oder Menüreferenz zurück. Sie können in *MenüZeile* -1 übergeben, um die zuletzt im Menü hinzugefügte Zeile zu spezifizieren.

Prozessnr ist optional. Geben Sie den Parameter nicht an, gilt **Get menu item key** für die Menüleiste des aktuellen Prozesses. Sonst gilt der Befehl für den Prozess mit der in *Prozessnr* übergebenen Referenznummer.

Hinweis: Übergeben Sie in *Menü* den Parametertyp *MenüRef*, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert.

Hat die Menüzeile kein zugewiesenes Tastenkürzel oder gibt der Parameter *MenüZeile* ein hierarchisches Untermenü an, gibt **Get menu item key** 0 (Null) zurück.

Beispiel

Um das einer Menüzeile zugeordnete Tastenkürzel zu erhalten, ist es hilfreich, nachfolgenden Programmcode zu integrieren:

```
If(Get menu item key(mymenu;1)#0)
   $modifiers:=Get menu item modifiers(mymenu;1)
   Case of
      :($modifiers=Option key mask)
         ...
      :($modifiers=Shift key mask)
         ...
      :($modifiers=Option key mask+Shift key mask)
         ...
   End case
End if
```

Get menu item mark

Get menu item mark (Menü ; MenüZeile {; Prozessnr}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüreferenz
MenüZeile	Lange Ganzzahl	➡	Nummer der Zeile oder -1 für zuletzt hinzugefügte Menüzeile
Prozessnr	Lange Ganzzahl	➡	Referenznummer des Prozesses
Funktionsergebnis	String	↩	Aktuelle Markierung der Menüzeile

Beschreibung

Die Funktion **Get menu item mark** gibt die Markierung der in *MenüZeile* übergebenen Menüzeile aus der in *Menü* übergebenen Menünummer oder Menüreferenz zurück. Sie können in *MenüZeile* -1 übergeben, um die zuletzt im Menü hinzugefügte Zeile zu spezifizieren.

Prozessnr ist optional. Geben Sie den Parameter nicht an, gilt **Get menu item mark** für die Menüleiste des aktuellen Prozesses. Sonst gilt der Befehl für den Prozess mit der in *Prozessnr* übergebenen Referenznummer.

Hinweis: Übergeben Sie in *Menü* den Parametertyp *MenüRef*, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert.

Hat die Menüzeile keine Markierung oder gibt der Parameter *MenüZeile* ein hierarchisches Untermenü an, gibt **Get menu item mark** einen leeren String zurück.

Hinweis: Erläuterungen zu Markierungen auf Macintosh und Windows finden Sie in der Beschreibung zum Befehl **SET MENU ITEM MARK**.

Beispiel

Folgendes Beispiel wechselt die Markierung einer Menüzeile:

```
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 (Menü ; MenüEintrag {; Prozessnr}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menüreferenz oder Menünummer
MenüEintrag	Lange Ganzzahl	➡	Nummer des Menüeintrags oder -1 für den zuletzt hinzugefügten Menüeintrag
Prozessnr	Lange Ganzzahl	➡	Prozessnummer
Funktionsergebnis	String	↩	Methodenname

Beschreibung

Die Funktion **Get menu item method** gibt den Namen der 4D Projektmethode zurück, die dem Menüeintrag, definiert durch die Parameter *Menü* und *MenüEintrag*, zugewiesen ist.
Sie können in *MenüEintrag* -1 übergeben, um den zuletzt im Menü hinzugefügten Eintrag zu spezifizieren.

In *Menü* übergeben Sie eine Menüreferenz (*MenüRef*) oder eine Menünummer.
Übergeben Sie eine Menüreferenz, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert, falls er übergeben ist.
Übergeben Sie eine Menünummer, berücksichtigt der Befehl das entsprechende Menü in der Hauptmenüleiste des aktuellen Prozesses. Wollen Sie einen anderen Prozess festlegen, übergeben Sie dessen Nummer im optionalen Parameter *Prozessnr*.

Die Funktion gibt den Namen der 4D Methode als Zeichenkette (Ausdruck) zurück. Ist dem Menüeintrag keine Methode zugewiesen, gibt die Funktion einen leeren String zurück.

Get menu item modifiers

Get menu item modifiers (Menü ; MenüEintrag {; Prozessnr}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menüreferenz oder Menünummer
MenüEintrag	Lange Ganzzahl	➡	Nummer des Menüeintrags oder -1 für den zuletzt hinzugefügten Menüeintrag
Prozessnr	Lange Ganzzahl	➡	Prozessnummer
Funktionsergebnis	Lange Ganzzahl	↩	Dem Menüeintrag hinzugefügte Zusatztasten

Beschreibung

Die Funktion **Get menu item modifiers** gibt zusätzliche Tasten zurück, die den standardmäßigen Tastenkürzeln des Menüeintrags, definiert durch *Menü* und *MenüEintrag*, zugewiesen sind.
Die Standardkürzel setzen sich zusammen aus den Sondertasten **Befehl** (Mac OS) oder **Strg** (Windows) plus Tastenkürzel. Sie werden über die Befehle **SET MENU ITEM SHORTCUT** und **Get menu item key** verwaltet.
Zusätzliche Tasten sind die Sondertasten **Shift** und **Option** (Mac OS) / **Alt** (Windows). Sie sind nur einsetzbar, wenn zuvor ein Standardkürzel festgelegt wurde.
Der zurückgegebene Wert entspricht dem Code der Zusatztasten. Sie lauten wie folgt:

- **Shift**= 512
- **Option** (Mac OS) oder **Alt** (Windows) = 2048
  Bei Verwendung beider Tasten werden die Werte addiert.

Hinweis: Sie können den Wert auch über die Konstanten Shift key mask und Option key mask unter dem Thema **Ereignisse (Zusatztasten)** berechnen.

Ist dem Menüeintrag keine Zusatztaste zugewiesen, gibt die Funktion 0 (Null) zurück.
Sie können in *MenüEintrag* -1 übergeben, um den zuletzt im Menü hinzugefügten Eintrag zu spezifizieren.
In *Menü* übergeben Sie eine Menüreferenz (*MenüRef*) oder eine Menünummer.
Übergeben Sie eine Menüreferenz, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert, falls er übergeben ist.
Übergeben Sie eine Menünummer, berücksichtigt die Funktion das entsprechende Menü in der Hauptmenüleiste des aktuellen Prozesses. Wollen Sie einen anderen Prozess festlegen, übergeben Sie dessen Nummer im optionalen Parameter *Prozessnr*.

Beispiel

Siehe Beispiel zur Funktion **Get menu item key**.

Get menu item parameter

Get menu item parameter (Menü ; MenüEintrag) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menüreferenz oder Menünummer
MenüEintrag	Lange Ganzzahl	➡	Nummer des Menüeintrags oder -1 für den zuletzt hinzugefügten Menüeintrag
Funktionsergebnis	String	⮌	Eigener Parameter des Menüeintrags

Beschreibung

Die Funktion **Get menu item parameter** gibt die eigene dem Menüeintrag zugewiesene Referenz (Text) zurück, definiert durch die Parameter *Menü* und *MenüEintrag*. Sie müssen die eigene Referenz zuvor über den Befehl **SET MENU ITEM PARAMETER** festlegen.

GET MENU ITEM PROPERTY

GET MENU ITEM PROPERTY (Menü ; MenüEintrag ; Eigenschaft ; Wert {; Prozessnr})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl	➡	Menüreferenz oder Menünummer
MenüEintrag	Lange Ganzzahl	➡	Nummer des Menüeintrags oder -1 für den zuletzt hinzugefügten Menüeintrag
Eigenschaft	String	➡	Eigenschaftstyp
Wert	Ausdruck	⬅	Eigenschaftswert
Prozessnr	Lange Ganzzahl	➡	Prozessznummer

Beschreibung

Der Befehl **GET MENU ITEM PROPERTY** gibt im Parameter *Wert* den aktuellen Wert der Eigenschaft des Menüeintrags zurück, definiert durch die Parameter *Menü* und *MenüEintrag*.
Sie können in *MenüEintrag* -1 übergeben, um den zuletzt im Menü hinzugefügten Eintrag anzugeben.

In *Menü* übergeben Sie eine Menüreferenz (*MenüRef*) oder eine Menünummer. Übergeben Sie eine Menüreferenz, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert, falls er übergeben ist. Übergeben Sie eine Menünummer, berücksichtigt der Befehl das entsprechende Menü in der Hauptmenüleiste des aktuellen Prozesses. Wollen Sie einen anderen Prozess festlegen, übergeben Sie dessen Nummer im optionalen Parameter *Prozessnr*.

Im Parameter *Eigenschaft* übergeben Sie die Eigenschaft, deren Wert Sie erhalten wollen. Dafür können Sie eine der Konstanten unter dem Thema **Menüzeilen Eigenschaften** übergeben, oder einen String, der einer selbst angelegten Eigenschaft entspricht. Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **SET MENU ITEM PROPERTY**.

Hinweis zur Kompatibilität: Ist die Variable *Wert* nicht explizit typisiert oder als Text deklariert, gibt der Befehl standardmäßig einen Namen von **Standardaktion** zurück. Um einen numerischen Wert zu erhalten, wie im (veralteten) Konstantenthema **Zugewiesene Standardaktion** definiert, müssen Sie die Variable *Wert* als Lange Ganzzahl deklarieren.

Get menu item style

Get menu item style (Menü ; MenüZeile {; Prozessnr}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüreferenz
MenüZeile	Lange Ganzzahl	➡	Nummer der Zeile oder -1 für zuletzt hinzugefügte Menüzeile
Prozessnr	Lange Ganzzahl	➡	Prozess Referenznummer
Funktionsergebnis	Lange Ganzzahl	↩	Stil des aktuellen Menüeintrags

Beschreibung

Die Funktion **Get menu item style** gibt den Schriftstil der in *MenüZeile* übergebenen Menüzeile aus der in *Menü* übergebenen Menünummer oder Menüreferenz zurück. Sie können in *MenüZeile* -1 übergeben, um die zuletzt im Menü hinzugefügte Zeile zu spezifizieren.

Prozessnr ist optional. Geben Sie den Parameter nicht an, gilt **Get menu item style** für die Menüleiste des aktuellen Prozesses. Sonst gilt die Funktion für den Prozess mit der in *Prozessnr* übergebenen Referenznummer.

Hinweis: Übergeben Sie in *Menü* den Parametertyp *MenüRef*, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert.

Get menu item style gibt eine oder die Kombination aus mehreren vordefinierten Konstanten unter dem Thema **Schriftstile** zurück:

Konstante	Typ	Wert
Bold	Lange Ganzzahl	1
Italic	Lange Ganzzahl	2
Plain	Lange Ganzzahl	0
Underline	Lange Ganzzahl	4

Beispiel

Mit folgender Zeile prüfen Sie, ob eine Menüzeile fett dargestellt wird:

```
If((Get menu item style($vlMenu;$vlItem)&Bold)#0)
  `...
End if
```

GET MENU ITEMS

GET MENU ITEMS (Menü ; MenüTitelArray ; MenüRefsArray)

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menüreferenz oder Menünummer
MenüTitelArray	Array String	⬅	Array mit Menütiteln
MenüRefsArray	Array String	⬅	Array mit Menüreferenzen

Beschreibung

Der Befehl **GET MENU ITEMS** gibt in den Arrays *MenüTitelArray* und *MenüRefsArray* Titel und Nummern für alle im Parameter *Menü* angegebenen Einträge des Menüs oder der Menüleiste zurück.

Im Parameter *Menü* können Sie eine Menüreferenz (*MenüRef*), eine Menüleistennummer oder eine Menüleistenreferenz übergeben. Letztere erhalten Sie über die Funktion **Get menu bar reference**.

Ist dem Eintrag keine Menüreferenz zugewiesen, wird im dazugehörigen Array Element ein leerer String zurückgegeben.

Beispiel

Sie möchten den Inhalt der Menüleiste des aktuellen Prozesses herausfinden:

```
ARRAY TEXT(menuTitlesArray;0)
ARRAY TEXT(menuRefsArray;0)
MenuBarRef:=Get menu bar reference(Frontmost process)
GET MENU ITEMS(MenuBarRef;menuTitlesArray;menuRefsArray)
```

Get menu title

Get menu title (Menü {; Prozessnr}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüreferenz
Prozessnr	Lange Ganzzahl	➡	Nummer des Prozesses
Funktionsergebnis	String	⮌	Menütitel

Beschreibung

Die Funktion **Get menu title** gibt den Titel der in *Menü* übergebenen Nummer oder Referenz zurück.

Prozessnr ist optional. Geben Sie den Parameter nicht an, gilt **Get menu title** für die Menüleiste des aktuellen Prozesses. Sonst gilt der Befehl für den Prozess mit der in *Prozessnr* übergebenen Referenznummer.

Hinweis: Übergeben Sie in *Menü* den Parametertyp *MenüRef*, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert.

Get selected menu item parameter

Get selected menu item parameter -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	String	Eigener Parameter des Menüeintrags

Beschreibung

Die Funktion **Get selected menu item parameter** gibt die eigene Referenz zurück, die dem gewählten Menüeintrag zugewiesen wurde. Sie müssen diesen Parameter zuvor über den Befehl **SET MENU ITEM PARAMETER** festgelegt haben.
Wurde kein Menüeintrag ausgewählt, gibt die Funktion einen leeren Text zurück ("").

INSERT MENU ITEM

INSERT MENU ITEM (Menü ; NachZeile ; NeuerText {; Untermenü {; Prozessnr}}{; *})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl	⇒	Menünummer oder Menüreferenz
NachZeile	Lange Ganzzahl	⇒	Nummer des Menüeintrags
NeuerText	String	⇒	Text für den einzufügenden Menüeintrag
Untermenü	MenüRef	⇒	Referenz des dem Eintrag zugeordneten Untermenüs
Prozessnr	Lange Ganzzahl	⇒	Referenznummer des Prozesses
*	Operator	⇒	Mit *: Metazeichen als Standardzeichen werten

Beschreibung

Der Befehl **INSERT MENU ITEM** fügt nach der vorhandenen Menüzeile mit der in *NachZeile* übergebenen Nummer neue Menüzeilen in das Menü mit der in *Menü* übergebenen Nummer oder Referenz ein.

Prozessnr ist optional. Geben Sie den Parameter nicht an, gilt **INSERT MENU ITEM** für die Menüleiste des aktuellen Prozesses. Sonst gilt der Befehl für die Menüleiste des Prozesses mit der in *Prozessnr* übergebenen Referenznummer.

Hinweis: Übergeben Sie in *Menü* den Parameter *MenüRef*, hat der Parameter *Prozessnr* keine Verwendung und wird ignoriert.

Übergeben Sie nicht den Parameter *, können Sie mit **INSERT MENU ITEM** in einem Aufruf eine oder mehrere Menüzeilen einfügen.

INSERT MENU ITEM arbeitet wie **APPEND MENU ITEM**, bis auf folgenden Unterschied:

- Mit **INSERT MENU ITEM** können Sie an beliebiger Stelle des Menüs Zeilen einfügen, mit **APPEND MENU ITEM** werden die Zeilen immer am Menüende angefügt.

Die Regeln zur Eingabe in *NeuerText* sowie die Funktionsweise mit dem optionalen Parameter * finden Sie unter dem Befehl **APPEND MENU ITEM**.

Hinweis: Ab 4D v16 R3 wird im Parameter *NeuerText* die Konstante ak standard action title unterstützt.

Mit dem optionalen Parameter *Untermenü* können Sie ein Menü wie die hinzugefügte Zeile angeben und so ein hierarchisches Untermenü anlegen. Sie müssen eine Menüreferenz (*MenüRef* vom Typ String) übergeben, das ein erstelltes Menü angibt, z.B. über die Funktion **Create menu**. Fügt der Befehl mehrere Menüzeilen hinzu, wird das Untermenü der ersten Zeile zugeordnet.

Wichtig: Den neuen Menüzeilen sind keine Methoden oder Aktionen zugeordnet. Diese müssen Sie den Menüzeilen über die Befehle **SET MENU ITEM PROPERTY** oder **SET MENU ITEM METHOD** zuordnen. Sie können die Zeilen auch über eine Formularmethode mit der Funktion **Menu selected** verwalten.

Beispiel

Nachfolgendes Beispiel erstellt ein Menü mit zwei Menüzeilen, denen Methoden zugewiesen werden:

```
menuRef:=Create menu
APPEND MENU ITEM(menuRef;"Zeichen")
SET MENU ITEM METHOD(menuRef;1;"CharMgmtDial")
INSERT MENU ITEM(menuRef;1;"Absätze")
SET MENU ITEM METHOD(menuRef;2;"ParaMgmtDial")
```

Menu selected

Menu selected {(Untermenü)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Untermenü	MenüRef	⇐	Menüreferenz mit der ausgewählten Zeile
Funktionsergebnis	Lange Ganzzahl	⇒	ausgewählter Menübefehl: Menünummer in höherwertigen 2 Byte, Menüzeilennummer in niederwertigen 2 Byte

Beschreibung

Die Funktion **Menu selected** wird nur beim Anzeigen von Formularen verwendet. Damit finden Sie heraus, welche Menüzeile der Anwender gewählt hat. Bei einem hierarchischen Untermenü wird die Referenz des Untermenüs zurückgegeben.

Tipp: Verwenden Sie vorzugsweise Methoden, die Menübefehlen in verknüpften Menüleisten (mit negativen Nummern) zugeordnet sind. Verknüpfte Menüleisten sind leichter zu verwalten, da Sie die Auswahl nicht prüfen müssen.

Menu selected lässt sich auch zum Arbeiten mit hierarchischen Untermenüs verwenden. Wird eine hierarchische Menüzeile außerhalb der ersten Ebene ausgewählt, gibt die Funktion im optionalen Parameter *Untermenü* die Referenz (Typ *MenüRef*, String mit 16 Zeichen) des Untermenüs zurück, zu der die gewählte Menüzeile gehört. Enthält die Menüzeile kein hierarchisches Untermenü, erhält dieser Parameter einen leeren String.

Menu selected gibt die Nummer des Menüs bzw. Menübefehls als Lange Ganzzahl zurück. Für die Menünummer dividieren Sie **Menu selected** durch 65.536 und konvertieren das Ergebnis in eine Ganzzahl. Für die Menüzeilennummer berechnen Sie den Restwert (Modulo) von **Menu selected** dividiert durch 65.536. Die jeweilige Formel lautet:

```
Menu:=Menu selected\ 65536
menu command:=Menu selected% 65536
```

Sie können Sie diese Werte auch über *Bit-Operatoren* entnehmen:

```
Menu:=(Menu selected  &  0xFFFF0000)>>16
menu command:=Menu selected  &  0xFFFF
```

Wurde kein Menübefehl ausgewählt, gibt **Menu selected** den Wert 0 zurück.

Beispiel

Folgende Formularmethode liefert über **Menu selected** die Argumente für Menü und Menübefehl an **SET MENU ITEM MARK**:

```
Case of
   :(Form event=On Menu Selected)
      C_STRING(16;$refMenuIncludingItem)
      C_LONGINT($ref;$MenuNum;$MenuItemNum)
      $ref:=Menu selected($refMenuIncludingItem)
      $MenuNum:=$ref\65536
      $MenuItemNum:=$ref%65536
      SET MENU ITEM MARK($refMenuIncludingItem;$MenuItemNum;Char(18))
End case
```

Hinweis: Das Formularereignis On Menu Selected wird nicht aktiviert, wenn kein Eintrag ausgewählt ist. *$refMenuIncludingItem* wird immer ein Wert gegeben und *$MenuNum* ist gleich 0, wenn das Menü kein Menü der Menüleiste ist.

RELEASE MENU

RELEASE MENU (Menü)

Parameter	Typ		Beschreibung
Menü	MenüRef	➡	Menüreferenz

Beschreibung

Der Befehl **RELEASE MENU** entfernt das Menü mit der im Parameter *Menü* übergebenen Referenznummer. Es muss mit der Funktion **Create menu** angelegt worden sein. Es gilt folgende Regel: Für jedes **Create menu** muss es ein entsprechendes **RELEASE MENU** geben.

Der Befehl entfernt die Instanzen des Menüs *Menü* aus allen Menüleisten und Prozessen. Gehört das Menü zu einer Menüleiste, die gerade verwendet wird, funktioniert dieses weiter, lässt sich jedoch nicht mehr verändern. Es wird erst dann aus dem Speicher entfernt, wenn die letzte Menüleiste, in der es vorkommt, nicht mehr in Gebrauch ist.

Dieser Befehl lässt sich auch mit Menüs verwenden, die als Menüleiste verwendet werden.

Alle von *Menü* verwendeten Untermenüs werden nicht entfernt, wenn sie direkt mit dem Befehl **Create menu** erstellt wurden. In diesem Fall müssen Sie jedes Untermenüs einzeln entfernen (siehe o.a. Regel). Wurden die Untermenü jedoch über Duplizieren eines vorhandenen Menüs erstellt, rufen Sie **RELEASE MENU** nicht auf, da 4D diese automatisch entfernt.

Beispiel

Dieses Beispiel zeigt verschiedene Anwendungsmöglichkeiten:

```
newMenu:=Create menu
APPEND MENU ITEM(newMenu;"Test1")
subMenu:=Create menu
APPEND MENU ITEM(subMenu;"SubTest1")
APPEND MENU ITEM(subMenu;"SubTest2")  // Einträge in Untermenü erstellen
APPEND MENU ITEM(newMenu;"Test2";subMenu)  // Untermenü an Menü anhängen

 //Hier wird das Untermenü an das Menü angehängt, wenn wir es später nicht mehr brauchen, ist dies der richtige Ort zum Entfernen.
 //Entfernen löscht subMenu nicht sofort, da es noch von newMenu verwendet wird, subMenu wird erst gelöscht, wenn newMenu gelöscht wurde.
 //Entfernen des Untermenüs an dieser Stelle spart Speicherplatz
RELEASE MENU(subMenu)

$result1:=Dynamic pop up menu(newMenu)  //Menü verwenden
copyMenu:=Create menu(newMenu)  // Menü durch Kopieren von newMenu erstellen (wodurch subMenu auch kopiert wird)
RELEASE MENU(newMenu)  //newMenu wird nicht mehr benötigt.

$result2:=Dynamic pop up menu(copyMenu)
RELEASE MENU(copyMenu)
 //Sie müssen sich nicht um das Löschen des Untermenüs von copyMenu kümmern, da es nicht direkt über Create menu erstellt wurde.
 //Es gilt folgende Regel: Jedes Create menu muss ein entsprechendes RELEASE MENU haben
```

SET MENU BAR

SET MENU BAR (Menüleiste {; Prozessnr}{; *})

Parameter	Typ		Beschreibung
Menüleiste	Lange Ganzzahl, String, MenüRef	➡	Nummer, Name oder Menüreferenz der zu zeigenden Menüleiste
Prozessnr	Lange Ganzzahl	➡	Referenznummer des Prozesses
*	Operator	➡	Status der Menüleiste sichern

Beschreibung

Der Befehl **SET MENU BAR** ersetzt die aktuelle Menüleiste durch *Menüleiste* nur für den aktuellen Prozess. In *Menüleiste* übergeben Sie Nummer oder Name der neuen Menüleiste. Sie können auch eine Menüreferenz vom Typ String mit 16 Zeichen übergeben. Arbeiten Sie mit Referenzen, können Sie Menüs als Menüleisten und umgekehrt verwenden. Weitere Informationen dazu finden Sie im Abschnitt **Menüs verwalten**.

Hinweis: Der Name einer Menüleiste muss einmalig sein und kann bis zu 31 Zeichen lang sein.

Existiert *Menüleiste* nicht, wird der Befehl **SET MENU BAR** nicht ausgeführt.

Hinweis: Übergeben Sie im Parameter *Menüleiste* den Parametertyp *MenüRef*, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert.

Der optionale Parameter *Prozessnr* ändert die Menüleiste des angegebenen Prozesses in *Menüleiste*.
Der optionale Parameter * behält den Zustand der Menüleiste bei. Wird er nicht angegeben, aktualisiert **SET MENU BAR** die aufgerufene Menüleiste.
Angenommen, nach Benutzung des Befehls **SET MENU BAR**(1) werden mehrere Menüzeilen mit **DISABLE MENU ITEM** deaktiviert. Bei erneuter Ausführung von **SET MENU BAR**(1), egal ob im selben oder einem anderen Prozess, kehren alle Menübefehle zum ursprünglichen Status aktiviert zurück.
Wird dagegen **SET MENU BAR**(1;*) ausgeführt, behält die Menüleiste den geänderten Status bei, die Menübefehle bleiben deaktiviert.

Hinweis: Übergeben Sie im Parameter *Menüleiste* den Parametertyp *MenüRef*, ist der Parameter * nicht erforderlich und wird ignoriert.

Öffnet ein Benutzer die Anwendungsumgebung, wird die erste Menüleiste angezeigt (Menüleiste #1). Sie können diese Menüleiste ändern, wenn Sie beim Öffnen einer Datenbank in der **Datenbankmethode On Startup** oder in der Startup Methode für den Benutzer eine spezifische Menüleiste angeben.

Beispiel 1

Folgendes Beispiel ändert die aktuelle Menüleiste in die Menüleiste mit Namen *FormMenuBar1* und versetzt die Menübefehle in den ursprünglichen Status:

```
SET MENU BAR(3)
```

Beispiel 2

Folgendes Beispiel ändert die aktuelle Menüleiste in die Menüleiste mit Namen *FormMenuBar1* und sichert die Menübefehle im aktuellen Status, d.h. deaktivierte Menüs bleiben deaktiviert.

```
SET MENU BAR("FormMenuBar1";*)
```

Beispiel 3

Folgendes Beispiel macht die aktuelle Menüleiste zu Menüleiste #3, während Datensätze geändert werden. Diese Menüleiste wird anschließend zu Menüleiste #2 mit dem ursprünglichen Status:

```
SET MENU BAR(3)
ALL RECORDS([Customers])
MODIFY SELECTION([Customers])
SET MENU BAR(2;*)
```

Beispiel 4

Dieses Beispiel erstellt per Programmierung eine Menüleiste mit den Menüs **File** und **Edit**:

```
  `Methode zum Erstellen des Menüs File
C_TEXT(16;FileMenu)  ` FileMenu enthält die Referenz auf Menü File
FileMenu:=Create menu
INSERT MENU ITEM(FileMenu;-1;"My Database "+Get indexed string(131;29))
SET MENU ITEM MARK(FileMenu;1;Char(18))
```

```
INSERT MENU ITEM(FileMenu;-1;"(-")
INSERT MENU ITEM(FileMenu;-1;"Quit the Test Application mode/Y")
SET MENU ITEM PROPERTY(FileMenu;3;Associated standard action;ak return to design mode)
INSERT MENU ITEM(FileMenu;-1;"(-")
INSERT MENU ITEM(FileMenu;-1;"Preferences")
SET MENU ITEM PROPERTY(FileMenu;5;Associated standard action;ak database settings)  `Einstellungen
INSERT MENU ITEM(FileMenu;-1;"(-")
INSERT MENU ITEM(FileMenu;-1;Get indexed string(131;30))
SET MENU ITEM PROPERTY(FileMenu;7;Associated standard action;ak quit)  `Beenden
SET MENU ITEM SHORTCUT(FileMenu;7;Character code("Q"))

  `Methode zum Erstellen des Untermenüs Find and Replace
C_TEXT(16;FindAndReplaceMenu)  `FindAndReplaceMenu enthält Referenz auf Menü Find and Replace
FindAndReplaceMenu:=Create menu
APPEND MENU ITEM(FindAndReplaceMenu;"Find;Find Next;Find Previous;(-;Replace;Replace Next;Replace Previous")
SET MENU ITEM SHORTCUT(FindAndReplaceMenu;1;Character code("F"))
SET MENU ITEM SHORTCUT(FindAndReplaceMenu;5;Character code("R"))
SET MENU ITEM METHOD(FindAndReplaceMenu;1;"MyFindMethod")

  `Methode zum Erstellen des Menüs Edit
C_TEXT(16;EditMenu)  `EditMenu enthält Referenz auf Menü Edit
EditMenu:=Create menu
APPEND MENU ITEM(EditMenu;"Cut;Copy;Paste")
SET MENU ITEM SHORTCUT(EditMenu;1;Character code("X"))
SET MENU ITEM PROPERTY(EditMenu;1;Associated standard action;ak cut)
SET MENU ITEM SHORTCUT(EditMenu;2;Character code("C"))
SET MENU ITEM PROPERTY(EditMenu;2;Associated standard action;ak copy)
SET MENU ITEM SHORTCUT(EditMenu;3;Character code("V"))
SET MENU ITEM PROPERTY(EditMenu;3;Associated standard action;ak paste)
INSERT MENU ITEM(EditMenu;-1;"(-")
INSERT MENU ITEM(EditMenu;-1;"Find and Replace";FindAndReplaceMenu)  ` Zeile mit Untermenü

main_Bar:=Create menu  ` Erstelle Menüleiste mit anderen Menüs
INSERT MENU ITEM(main_Bar;-1;Get indexed string(79;1);FileMenu)
APPEND MENU ITEM(main_Bar;"Edit";EditMenu)

SET MENU BAR(main_Bar)
```

SET MENU ITEM

SET MENU ITEM (Menü ; MenüZeile ; NeuerText {; Prozessnr}{; *})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüreferenz
MenüZeile	Lange Ganzzahl	➡	Nummer für Menüeintrag oder -1 für zuletzt hinzugefügten Eintrag
NeuerText	String	➡	Neuer Text für Menüeintrag
Prozessnr	Lange Ganzzahl	➡	Referenznummer des Prozesses
*	Operator	➡	Mit *: Metazeichen als Standardzeichen werten

Beschreibung

Der Befehl **SET MENU ITEM** ändert den Text, übergeben in *NeuerText* für die Zeile, übergeben in *MenüZeile* des Menüs mit der in *Menü* übergebenen Menünummer oder Menüreferenz. Sie können in *MenüZeile* -1 übergeben, um die zuletzt im Menü hinzugefügte Zeile zu definieren.

Übergeben Sie nicht den Parameter *, werden Sonderzeichen in *NeuerText*, wie z.B. "(", "!" oder ";", für Einträge als Steuerzeichen (Metazeichen) behandelt. Wollen Sie z.B. einen Menüeintrag als Trennungslinie setzen, fügen Sie in *NeuerText* das Zeichen "-" ein.
Übergeben Sie den Parameter *, werden Sonderzeichen als standardmäßige Zeichen gewertet, d.h. Sie können für Einträge Bezeichnungen wie "Kopieren (spezial)..." oder "Suchen/Ersetzen..." vergeben. Weitere Informationen dazu finden Sie unter dem Befehl **APPEND MENU ITEM**.

Geben Sie den Parameter *Prozessnr* nicht an, gilt **SET MENU ITEM** für die Menüleiste des aktuellen Prozesses. Sonst gilt der Befehl für die Menüleiste des Prozesses mit der in *Prozessnr* übergebenen Referenznummer.

Hinweis: Übergeben Sie in *Menü* den Parametertyp *MenüRef*, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert.

SET MENU ITEM ICON

SET MENU ITEM ICON (Menü ; MenüEintrag ; IconRef {; Prozessnr})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	⇒	Menüreferenz oder Menünummer
MenüEintrag	Lange Ganzzahl	⇒	Nummer des Menüeintrags oder -1 für den zuletzt hinzugefügten Menüeintrag
IconRef	Text, Lange Ganzzahl	⇒	Name oder Nummer des Bildes, das dem Menüeintrag zugewiesen werden soll
Prozessnr	Lange Ganzzahl	⇒	Prozessnummer

Beschreibung

Der Befehl **SET MENU ITEM ICON** ändert das dem Menüeintrag zugewiesene Icon, definiert durch die Parameter *Menü* und *MenüEintrag*.
Sie können in *MenüEintrag* -1 übergeben, um den zuletzt im Menü hinzugefügten Eintrag zu definieren.
In *Menü* übergeben Sie eine Menüreferenz (*MenüRef*) oder eine Menünummer.
Übergeben Sie eine Menüreferenz, gilt der Befehl für alle Menüinstanzen in allen Prozesse. Der Parameter *Prozessnr* wird dann ignoriert, falls er übergeben wurde.

Übergeben Sie eine Menünummer, berücksichtigt der Befehl das entsprechende Menü in der Hauptmenüleiste des aktuellen Prozesses. Wollen Sie einen anderen Prozess festlegen, übergeben Sie dessen Nummer im optionalen Parameter *Prozessnr*.

In *IconRef* übergeben Sie das Bild, das als Icon dienen soll. Sie können ein Bild aus der Bildbibliothek oder eine Bildreferenz verwenden.

- Bild der Bildbibliothek: Sie können entweder Name oder Nummer des Bildes zurückgeben. Es ist in der Regel besser, die Nummer zu verwenden, da Bildnummern im Gegensatz zu Bildnamen einmalig sind.
- Bildreferenz: Das Bild muss im Ordner **Resources** der Datenbank liegen und Sie müssen in *IconRef* die Syntax "File:{Pfadname}Dateiname" verwenden. Weitere Informationen zum Ordner **Resources** finden Sie im Abschnitt **Einführung in Ressourcen**.

Beispiel

Ein Bild aus dem Ordner **Resources** der Datenbank verwenden:

```
SET MENU ITEM ICON($MenuRef;2;"File:English.lproj/spot.png")
```

SET MENU ITEM MARK

SET MENU ITEM MARK (Menü ; MenüZeile ; Markierung {; Prozessnr})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüreferenz
MenüZeile	Lange Ganzzahl	➡	Nummer der Zeile oder -1 für zuletzt hinzugefügte Menüzeile
Markierung	String	➡	Neue Markierung für Menüzeile
Prozessnr	Lange Ganzzahl	➡	Referenznummer des Prozesses

Beschreibung

Der Befehl **SET MENU ITEM MARK** ändert die Markierung für die in *MenüZeile* übergebene Menüzeile aus der in *Menü* übergebenen Menünummer oder Menüreferenz. Sie können in *MenüZeile* -1 übergeben, um die zuletzt im Menü hinzugefügte Zeile zu spezifizieren. Diese Markierung besteht nur aus dem ersten Zeichen von *Markierung*.

Prozessnr ist optional. Geben Sie den Parameter nicht an, gilt **SET MENU ITEM MARK** für die Menüleiste des aktuellen Prozesses. Sonst gilt der Befehl für den Prozess mit der in *Prozessnr* übergebenen Referenznummer.

Hinweis: Übergeben Sie in *Menü* den Parametertyp *MenüRef*, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert.

Übergeben Sie einen leeren String, wird die Markierung vor einer Zeile gelöscht. Sonst gilt folgendes:

- Auf Macintosh wird das erste Zeichen des String zur Markierung der Menüzeile. Normalerweise übergeben Sie den Wert *Char (18)*, das ist das Markierungszeichen für Macintosh-Menüs.
- Unter Windows wird der Menüzeile die Standardmarkierung zugewiesen.

Beispiel

Siehe Beispiel zum Befehl **Get menu item mark**.

SET MENU ITEM METHOD

SET MENU ITEM METHOD (Menü ; MenüEintrag ; Methodenname {; Prozessnr})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menüreferenz oder Menünummer
MenüEintrag	Lange Ganzzahl	➡	Nummer des Menüeintrags oder -1 für den zuletzt hinzugefügten Menüeintrag
Methodenname	String	➡	Methodenname
Prozessnr	Lange Ganzzahl	➡	Prozessnummer

Beschreibung

Der Befehl **SET MENU ITEM METHOD** ändert die 4D Projektmethode, die dem Menüeintrag, definiert durch *Menü* und *MenüEintrag*, zugewiesen ist.
Sie können in *MenüEintrag* -1 übergeben, um den zuletzt im Menü hinzugefügten Eintrag zu spezifizieren.
In *Menü* übergeben Sie eine Menüreferenz (*MenüRef*) oder eine Menünummer.
Übergeben Sie eine Menüreferenz, gilt der Befehl für alle Menüinstanzen in allen Prozessen. Der Parameter *Prozessnr* wird dann ignoriert, falls er übergeben ist.

Übergeben Sie eine Menünummer, berücksichtigt der Befehl das entsprechende Menü in der Hauptmenüleiste des aktuellen Prozesses. Wollen Sie einen anderen Prozess festlegen, übergeben Sie dessen Nummer im optionalen Parameter *Prozessnr*.

In *Methodenname* übergeben Sie den Namen der 4D Methode als Zeichenkette (Ausdruck).

Hinweis: Entspricht der Menüeintrag dem Titel eines hierarchischen Untermenüs, wird die Methode nicht aufgerufen, wenn der Menüeintrag ausgewählt ist

Beispiel

Siehe Beispiel zum Befehl **SET MENU BAR**.

SET MENU ITEM PARAMETER

SET MENU ITEM PARAMETER (Menü ; MenüEintrag ; Parameter)

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menüreferenz oder Menünummer
MenüEintrag	Lange Ganzzahl	➡	Nummer der Menüzeile oder -1 für zuletzt hinzugefügte Menüzeile
Parameter	String	➡	Als Parameter zuzuordnender Text

Beschreibung

Der Befehl **SET MENU ITEM PARAMETER** weist dem Menüeintrag, definiert durch die Parameter *Menü* und *MenüEintrag*, eine eigene Referenz zu.
Die eigene Referenz wird hauptsächlich von der Funktion **Dynamic pop up menu** verwendet.

Beispiel

Dieser Code liefert ein Menü mit den Namen der offenen Fenster und ermöglicht, die Nummer des gewählten Fensters zu erhalten:

```
WINDOW LIST($alWindow)
$tMenuRef:=Create menu
For($i;1;Size of array($alWindow))
   APPEND MENU ITEM($tMenuRef;Get window title($alWindow{$i}))  // Titel des Menüeintrags
   SET MENU ITEM PARAMETER($tMenuRef;-1;String($alWindow{$i}))  // Vom Menüeintrag zurückgegebener Wert
End for
$tWindowRef:=Dynamic pop up menu($tMenuRef)
RELEASE MENU($tMenuRef)
```

SET MENU ITEM PROPERTY

SET MENU ITEM PROPERTY (Menü ; MenüEintrag ; Eigenschaft ; Wert {; Prozessnr})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menüreferenz oder Menünummer
MenüEintrag	Lange Ganzzahl	➡	Nummer des Menüeintrags oder -1 für den zuletzt hinzugefügten Menüeintrag
Eigenschaft	String	➡	Eigenschafttyp
Wert	Text, Zahl, Boolean	➡	Eigenschafttyp
Prozessnr	Lange Ganzzahl	➡	Prozessnummer

Beschreibung

Der Befehl **SET MENU ITEM PROPERTY** setzt den Wert der Eigenschaft für den Menüeintrag, definiert durch die Parameter *Menü* und *MenüEintrag*.

Sie können in *MenüEintrag* -1 übergeben, um den zuletzt im Menü hinzugefügten Eintrag anzugeben.

In *Menü* übergeben Sie eine Menüreferenz (*MenüRef*) oder eine Menünummer. Übergeben Sie eine Menüreferenz, gilt der Befehl für alle Menüinstanzen in allen Prozessen. Der Parameter *Prozessnr* ist dann nicht erforderlich und wird ignoriert, falls er übergeben ist.
Übergeben Sie eine Menünummer, berücksichtigt der Befehl das entsprechende Menü in der Hauptmenüleiste des aktuellen Prozesses. Wollen Sie einen anderen Prozess festlegen, übergeben Sie dessen Nummer im optionalen Parameter *Prozessnr*.

Im Parameter *Eigenschaft* übergeben Sie die Eigenschaft, deren Wert Sie ändern wollen, in *Wert* den neuen Wert.
Für *Eigenschaft* können Sie einen selbst erstellten Wert oder eine **Standardeigenschaft** (eine Konstante unter dem Thema **Menüzeilen Eigenschaften**) übergeben.

- **Standardeigenschaft:** Die Konstanten unter dem Thema **Menüzeilen Eigenschaften** und ihre möglichen Werte werden nachfolgend beschrieben.

Konstante	Typ	Wert	Kommentar
Access privileges	Zeichenkette	4D_access_group	Dem Befehl eine Zugriffsgruppe zuweisen 0 = Alle Gruppen >0 = Gruppennummer
Associated standard action	Zeichenkette	4D_standard_action	Einem Menüeintrag eine Standardaktion zuweisen Siehe Konstanten unter dem Thema **Standardaktion**.
Start a new process	Zeichenkette	4D_start_new_process	Dem Befehl eine Zugriffsgruppe zuordnen 0 = Ohne Einschränkung >0 = Gruppennummer

  Bei der Eigenschaft *Associated Standard Action* können Sie im Parameter *Wert* den Namen einer Standardaktion übergeben. Weitere Informationen dazu finden Sie im Abschnitt **Standardaktionen** des Handbuchs *4D Designmodus.* Die gängigsten Aktionen sind auch als Konstanten unter dem Thema **Standardaktion** verfügbar.
  Hinweis zur Kompatibilität: In bisherigen Releases wurden im Parameter *Wert* (Lange Ganzzahl) Konstanten aus dem Thema **Zugewiesene Standardaktion** verwendet. Diese sind ab 4D v16 R3 überholt, werden aber zur Wahrung der Kompatibilität noch unterstützt.

  Hinweis: Entspricht der Menüeintrag dem Titel eines hierarchischen Untermenüs, wird die Standardaktion nicht aufgerufen, wenn das Menü ausgewählt wird.

- **Selbst erstellte Eigenschaft:** In *Eigenschaft* können Sie auch einen eigenen Text übergeben und einen Wert vom Typ Text, Zahl oder Boolean zuweisen. Dieser Wert wird mit dem Eintrag gespeichert und lässt sich über den Befehl **GET MENU ITEM PROPERTY** ausfindig machen.
  Im Parameter *Eigenschaft* können Sie eine beliebige Zeichenkette verwenden. Sie müssen lediglich sicherstellen, dass es kein von 4D verwendeter Titel ist. Laut Konvention beginnen von 4D gesetzte Eigenschaften mit "4D_".

SET MENU ITEM SHORTCUT

SET MENU ITEM SHORTCUT (Menü ; MenüZeile ; Tastenkürzel ; Zusatztasten {; Prozessnr})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüeferenz
MenüZeile	Lange Ganzzahl	➡	Nummer der Zeile oder -1 für zuletzt hinzugefügte Menüzeile
Tastenkürzel	String, Lange Ganzzahl	➡	Buchstabe des Tastenkürzels oder ASCII-Code des Tastenkürzels (frühere Syntax)
Zusatztasten	Lange Ganzzahl	➡	Dem Tastenkürzel zuzuordnende Zusatztasten (wird bei ASCII-Code ignoriert)
Prozessnr	Lange Ganzzahl	➡	Referenznummer des Prozesses

Beschreibung

Der Befehl **SET MENU ITEM SHORTCUT** ersetzt das Tastaturkürzel für *MenüZeile* des Menüs *Menü* durch den Zeichencode oder Text, übergeben in *Tastenkürzel*. Sie können in *MenüZeile* -1 übergeben, um die zuletzt im Menü hinzugefügte Zeile zu spezifizieren. Dieses Kürzel wird unter Windows mit der **Strg-Taste**, auf Mac OS mit der **Befehlstaste** kombiniert.

Sie können den Buchstaben für das Kürzel direkt als String im Parameter *Tastenkürzel* übergeben, z.B. "U" für die Tastenkombination **Strg+U** unter Windows bzw. **Befehl+U** auf Mac OS. Mit dieser Syntax können Sie auch den optionalen Parameter *Zusatztasten* übergeben, um dem Tastenkürzel weitere Sondertasten zuzuweisen. So können Sie z.B. folgende Tastenkombinationen definieren: **Strg+Alt+Shift+Z** unter Windows bzw. **Befehl+Wahl+Shift+Z** auf Mac OS.
Dafür übergeben Sie in *Zusatztasten* folgende Werte:
- 256 für die **Strg-** bzw. die **Befehlstaste**
- 512 für die **Shift-Taste**
- 2048 für die **Alt-Taste** unter Windows bzw. **Wahltaste** auf Mac OS
- Um mehrere Tasten zuzuweisen, kombinieren Sie die einzelnen Werte.

Hinweis: Sie können die Werte auch über die Konstanten Command key mask, Shift key mask und Option key mask unter dem Thema **Ereignisse (Zusatztasten)** definieren.

4D weist die **Strg-Taste** unter Windows bzw. die **Befehlstaste** auf Mac OS automatisch den Tastenkürzeln zu, unabhängig, ob Sie diese explizit im Parameter *Zusatztasten* angeben. Sie müssen also in diesem Parameter nicht den Wert 256 hinzufügen, außer diese Taste ist die einzige Zusatztaste. In diesem Fall müssen Sie unter Zusatztasten entweder den Wert 256 oder die entsprechende Konstante übergeben.

Hinweis: Zur Wahrung der Kompatibilität erlaubt der Befehl im Parameter *Tastenkürzel* auch ASCII Code (frühere Syntax). In diesem Fall wird der Parameter *Zusatztasten* nicht berücksichtigt und kann weggelassen werden. Das Tastenkürzel wird nur der **Strg-Taste** unter Windows bzw. der **Befehlstaste** auf Mac OS zugewiesen.

Prozessnr ist optional. Geben Sie den Parameter nicht an, gilt **SET MENU ITEM SHORTCUT** für die Menüleiste des aktuellen Prozesses. Sonst gilt der Befehl für die Menüleiste des Prozesses mit der in *Prozessnr* übergebenen Referenznummer.

Hinweis: Übergeben Sie in *Menü* den Parametertyp *MenüRef*, ist der Parameter *Prozess* nicht erforderlich und wird ignoriert.

Übergeben Sie 0 (Null) in *Tastenkürzel*, werden alle Tastenkürzel aus der Menüzeile entfernt.

Beispiel 1

Definition des Tastenkürzel **Strg+Shift+U** unter Windows, **Befehl+Shift+U** auf Mac OS für die Menüzeile "Unterstrichen":

```
SET MENU ITEM(menuRef;1;"Underline")
SET MENU ITEM SHORTCUT(menuRef;1;"U";Shift key mask)
```

Beispiel 2

Definition des Tastenkürzel **Strg+R** (Windows) bzw. **Befehl+R** (Mac OS) für die Menüzeile "Restart" menu item:

```
INSERT MENU ITEM(FileMenu;-1;"Restart")
SET MENU ITEM SHORTCUT(FileMenu;-1;"R";Command key mask)
```

SET MENU ITEM STYLE

SET MENU ITEM STYLE (Menü ; MenüZeile ; MenüStil {; Prozessnr})

Parameter	Typ		Beschreibung
Menü	Lange Ganzzahl, MenüRef	➡	Menünummer oder Menüreferenz
MenüZeile	Lange Ganzzahl	➡	Nummer der Zeile oder -1 für zuletzt hinzugefügte Menüzeile
MenüStil	Lange Ganzzahl	➡	Neuer Stil der Menüzeile
Prozessnr	Lange Ganzzahl	➡	Referenznummer des Prozesses

Beschreibung

Der Befehl **SET MENU ITEM STYLE** ändert den Schriftstil der in *MenüZeile* übergebenen Menüzeile aus der in *Menü* übergebenen Menünummer oder Menüreferenz gemäß der Angabe in *MenüStil*. Sie können in *MenüZeile* -1 übergeben, um die zuletzt im Menü hinzugefügte Zeile zu spezifizieren.

Prozessnr ist optional. Geben Sie den Parameter nicht an, gilt **SET MENU ITEM STYLE** für die Menüleiste des aktuellen Prozesses. Sonst gilt der Befehl für den Prozess mit der in *Prozessnr* übergebenen Referenznummer.

Hinweis: Übergeben Sie in *Menü* den Parametertyp *MenüRef*, ist der Parameter *Prozessnr* nicht erforderlich und wird ignoriert.

SET MENU ITEM STYLE gibt eine oder die Kombination aus mehreren vordefinierten Konstanten unter dem Thema **Schriftstile** zurück:

Konstante	Typ	Wert
Bold	Lange Ganzzahl	1
Italic	Lange Ganzzahl	2
Plain	Lange Ganzzahl	0
Underline	Lange Ganzzahl	4

Objekte (Formulare)

4D Write Pro Areas

Die meisten Befehle im Kapitel **Objekte (Formulare)** unterstützen 4D Write Pro Areas. Weitere Informationen dazu finden Sie im Abschnitt **Befehle aus dem Kapitel Objekte (Formulare) verwenden** des *4D Write Pro Handbuchs*.

- Objekteigenschaften
- 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

Objekteigenschaften

Die Befehle in diesem Kapitel gelten für die Eigenschaften von Objekten in Formularen. Damit verändern Sie die Darstellung und das Verhalten von Objekten, wenn Sie Formulare in der Anwendungsumgebung verwenden.

Wichtig: Diese Befehle wirken sich nur auf das aktuelle Formular aus; die Änderungen werden wieder aufgehoben, sobald Sie das Formular verlassen.

Auf Objekte über ihre Objektnamen oder Namen ihrer Datenquelle zugreifen

Für Befehle der Objekteigenschaften gilt folgende generische Syntax:

Befehl NAME({*;} Objekt { ; zusätzliche Parameter je nach Befehl)

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an.

Wollen Sie mit einem Aufruf mehrere Objekte des Formulars ansprechen, können Sie innerhalb des Namens das Jokerzeichen @ verwenden. Folgende Tabelle zeigt einige Beispiele:

Objektname	Betroffene Objekte
HauptGruppenBox	Nur das Objekt HauptGruppenBox.
Haupt@	Alle Objekte, die mit "Haupt" beginnen.
@GruppenBox	Alle Objekte, die mit "GruppenBox" enden.
@Gruppe@	Alle Objekte, die "Gruppe" enthalten.
Haupt@Btn	Alle Objekte, die mit "Haupt" beginnen und mit "Btn" enden.
@	Alle Objekte im Formular.

Formularobjekte können bis zu 255 Bytes enthalten. So können Sie eigene Namensregeln definieren und anwenden, wie z.B. xxxx_Button oder xxx_Mac.

Hinweis: Sie können festlegen, wie das Jokerzeichen beim Einsetzen in eine Zeichenkette interpretiert wird. Diese Option beeinflusst die Funktionsweise der Befehle im Kapitel "Objekte (Formulare)". Weitere Informationen dazu finden Sie im Abschnitt **Text-Vergleiche** des Handbuchs *4D Designmodus*.

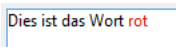
Lassen Sie den optionalen Parameter * weg, übergeben Sie in *Objekt* ein Datenfeld oder eine Variable. In diesem Fall geben Sie anstatt eines Strings eine Referenz auf das Datenfeld oder die Variable an (nur Objekte vom Typ Datenfeld oder Variable).

Generische Befehle in Bereichen mit Mehrfachstil

Ab 4D v14 gibt es neue Interaktionen zwischen generischen Befehlen wie **OBJECT SET RGB COLORS** oder **OBJECT SET FONT STYLE** und Textbereichen mit Mehrfachstil.

In bisherigen 4D Versionen hat das Ausführen solcher Befehle den Inhalt eigener Stilelemente im Bereich verändert. Jetzt werden nur standardmäßige Eigenschaften und über standardmäßige Tags gesicherte Eigenschaften durch diese Befehle beeinflusst. Eigene Stilelemente bleiben unverändert erhalten.

Nehmen wir z.B. einen Bereich mit Mehrfachstil, in dem Standard Tags gesichert wurden:



Die Textformatierung dafür lautet:

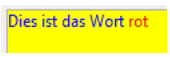
```
<span style="text-align:left;font-family:'Segoe UI';font-size:9pt;color:#009900">Dies ist das Wort <span
style="color:#D81E05">rot</span></span>
```

Führen Sie folgenden Code aus:

```
OBJECT SET COLOR(*;"myArea";-(Blue+(256*Yellow)))
```

bleibt in 4D v14 die rote Farbe erhalten:

4D v14




```
<span style="text-align:left;font-family:'Segoe
UI';font-size:9pt;color:#0000FF">Dies ist das
Wort <span style="color:#D81E05">rot</span>
</span>
```

bisherige Versionen


```
<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">Dies ist das Wort rot</span>
</span>
```

Das gilt für folgende generische Befehle:

OBJECT SET RGB COLORS
OBJECT SET COLOR
OBJECT SET FONT
OBJECT SET FONT STYLE
OBJECT SET FONT SIZE

Für Bereiche mit Mehrfachstil sollten generische Befehle nur zum Setzen von Standard Stilarten verwendet werden. Zum Verwalten von Stilarten während der Ausführung der Datenbank empfehlen wir, die o.a. Befehle zu verwenden (siehe **Mehrfachstil Text**).

GET STYLE SHEET INFO

GET STYLE SHEET INFO (StilvorlageName ; Schrift ; Größe ; Stile)

Parameter	Typ		Beschreibung
StilvorlageName	Text	➡	Name der Stilvorlage
Schrift	Text	⬅	Schriftart
Größe	Lange Ganzzahl	⬅	Schriftgröße
Stile	Lange Ganzzahl	⬅	Stilwerte

Beschreibung

Der Befehl **GET STYLE SHEET INFO** gibt die aktuelle Konfiguration der Stilvorlage zurück, die im Parameter *StilvorlageName* definiert ist.

In *StilvorlageName* übergeben Sie den Namen der Stilvorlage wie im Designmodus definiert.
Für eine automatische Stilvorlage können Sie eine Konstante unter dem Thema **Schriftstile** verwenden:

Konstante	Typ	Wert	Kommentar
Automatic style sheet	Zeichenkette	__automatic__	Wird standardmäßig für alle Objekte verwendet
Automatic style sheet_additional	Zeichenkette	__automatic_additional_text__	Gilt nur für statischen Text, Felder und Variablen für Zusatztext in Dialogfenstern.
Automatic style sheet_main	Zeichenkette	__automatic_main_text__	Gilt nur für statischen Text, Felder und Variablen für Haupttext in Dialogfenstern.

In *Schrift* gibt der Befehl den zugewiesenen Schriftnamen in der Stilvorlage der aktuellen Plattform zurück.
In *Größe* gibt der Befehl die zugewiesene Schriftgröße in der Stilvorlage der aktuellen Plattform zurück.
In *Stile* gibt der Befehl einen Wert der zugewiesenen Stilwerte in der Stilvorlage der aktuellen Plattform zurück. Sie können den empfangenen Wert mit den folgenden Konstanten unter dem Thema **Schriftstile** vergleichen:

Konstante	Typ	Wert
Bold	Lange Ganzzahl	1
Bold and Italic	Lange Ganzzahl	3
Bold and Underline	Lange Ganzzahl	5
Italic	Lange Ganzzahl	2
Italic and Underline	Lange Ganzzahl	6
Plain	Lange Ganzzahl	0
Underline	Lange Ganzzahl	4

Bei korrekt ausgeführtem Befehl wird die Systemvariable *OK* auf 1 gesetzt, sonst auf 0, z.B. wenn *StilvorlageName* nicht existiert.

Beispiel

Sie wollen die aktuelle Konfiguration der Stilvorlage "Automatic" herausfinden:

```
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 (arrStilvorlagen)

Parameter	Typ		Beschreibung
arrStilvorlagen	Array Text	⇐	Namen der Stilvorlagen, die in der Anwendung definiert sind

Beschreibung

Der Befehl **LIST OF STYLE SHEETS** gibt im Array *arrStilvorlagen* die Liste der Stilvorlagen der Anwendung an.

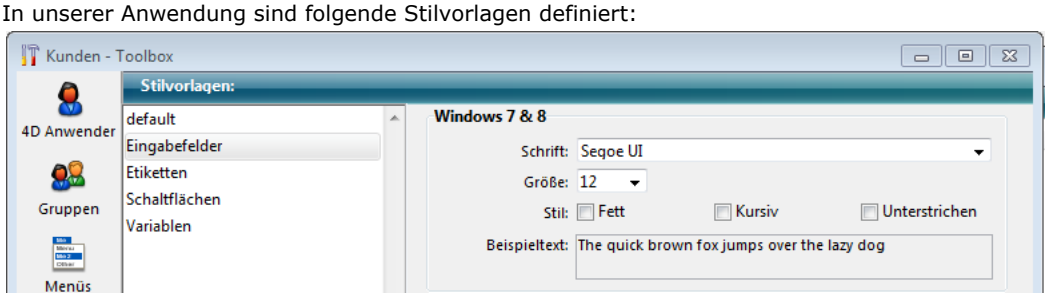
Ist das Array noch nicht definiert, erstellt der Befehl das Text Array *arrStilvorlagen*. Seine Größe richtet sich automatisch nach der Anzahl der definierten Stilvorlagen.

Nach Ausführen des Befehls enthält jedes Element im Array den Namen einer Stilvorlage. Die Namen werden wie im Editor für Stilvorlagen alphabetisch sortiert. Das erste Array Element enthält immer "default" für die Stilvorlage "Automatik".

Hinweis: Dieser Befehl gibt zur Wahrung der Kompatibilität **nicht** die automatischen Stilvorlagen "__automatic_main__text" und "__automatic_additional__text" zurück. Sie sind jedoch in den Formularen verfügbar.

Beispiel

In unserer Anwendung sind folgende Stilvorlagen definiert:



Führen Sie folgenden Code aus:

```
LIST OF STYLE SHEETS($arrStyles)
  // $arrStyles{1} enthält "default"
  // $arrStyles{2} enthält "Eingabefelder"
  // $arrStyles{3} enthält "Etiketten"
  // $arrStyles{4} enthält "Schaltflächen"
  // $arrStyles{5} enthält "Variablen"
```

OBJECT DUPLICATE

OBJECT DUPLICATE ({* ;} Objekt {; NeuerName {; NeueVar {; GehenZu {; BewegenH {; BewegenV {; AnpassenH {; AnpassenV}}}}}}} {; *})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable bzw. Feld (ohne *)
NeuerName	Text	➡	Name des neuen Objekts
NeueVar	Zeiger	➡	Zeiger auf Variable des neuen Objekts
GehenZu	Text	➡	Name des vorigen eingebbaren Objekts (oder Optionsfeld)
BewegenH	Lange Ganzzahl	➡	Horizontales Versetzen des neuen Objekts(>0 = nach rechts, <0 = nach links)
BewegenV	Lange Ganzzahl	➡	Vertikales Versetzen des neuen Objekts (>0 = nach unten, <0 = nach oben)
AnpassenH	Lange Ganzzahl	➡	Horizontale Anpassung des neuen Objekts
AnpassenV	Lange Ganzzahl	➡	Vertikale Anpassung des neuen Objekts
*	Operator	➡	Mit Stern = absolute Koordinaten Ohne Stern = relative Koordinaten

Beschreibung

Der Befehl **OBJECT DUPLICATE** erstellt eine Kopie des Objekts, definiert im Parameter *Objekt*. Die Kopie wird im Kontext des ausgeführten Formulars erstellt (Anwendungsmodus). Das Ausgangsformular im Designmodus wird nicht verändert. Standardmäßig werden alle Optionen, die für das Ausgangsdokument in der Eigenschaftenliste angegeben werden, auf die Kopie angewendet (Größe, Farbe, etc.), inkl. einer zugewiesenen Objektmethode.

Es gibt jedoch folgende Ausnahmen:

- **Standardschaltfläche:** Im Formular ist nur eine Standardschaltfläche möglich. Duplizieren Sie eine Schaltfläche mit der Eigenschaft "Standardschaltfläche", wird diese Eigenschaft der Kopie zugewiesen und aus dem Ausgangsdokument entfernt.
- **Tastenkombinationen:** Ist einem Ausgangsobjekt ein Tastenkürzel zugewiesen, wird es nicht dupliziert. Diese Eigenschaft bleibt in der Kopie leer.
- **Objektnamen:** In einem Formular kann es nicht mehrere Objekte mit demselben Namen geben. Übergeben Sie nicht den Parameter *NeuerName*, wird der Name des Ausgangsobjekts im neuen Objekt automatisch numeriert bzw. um 1 erhöht (siehe unten).

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie ein Feld oder eine Variablenreferenz (nur Objektfeld oder -variable) anstelle eines String.
Übergeben Sie eine Feld- oder Variablenreferenz und enthält das Formular mehrere Objekte, die dieselbe Referenz nutzen, wird das erste gefundene Auftreten verwendet. Um Zweideutigkeiten zu vermeiden, empfehlen wir, einmalige Objektnamen zu verwenden.

Im Parameter *NeuerName* übergeben Sie den Namen für die Kopie eines Objekts. Er muss die Namensregeln für Objekte beachten und im Formular einmalig sein. Ist er ungültig oder wird bereits von einem anderen Objekt verwendet, führt der Befehl nichts aus und die Variable OK gibt 0 (Null) zurück.
Lassen Sie diesen Parameter weg oder übergeben einen leeren String, wird der neue Name automatisch durch Nummerierung des Namens des Ausgangsdokuments erzeugt, außer er wird bereits verwendet.

Hier ein paar Beispiele:

Ausgangsname	Name der Kopie
Button	Button1
Button20	Button21
Button21	Button23 wenn Button22 bereits existiert

Übergeben Sie einen Zeiger auf die Variable, die dem neuen Objekt in *NeueVar* übergeben wird. Dabei müssen Sie beachten, dass Sie auf eine Variable verweisen, die vom gleichen Typ wie die des Ausgangsobjekts ist. Bestimmte Arten von "Neutypisierung" sind jedoch möglich. Der Befehl bietet eine automatische Vorgehensweise, um das Schreiben von generischem Code zu vereinfachen:

- In der Regel lassen sich alle eingebbaren Variablen neu typisieren; z.B. kann ein Objekt mit Datum oder Ganzzahl dupliziert und mit einer Variablen vom Typ Text verwendet werden. Alle kompatiblen Eigenschaften werden beibehalten. Der Befehl ermöglicht auch, die Typen zwischen Text- und Bildobjekten zu wechseln. Beachten Sie, dass ein Textobjekt, das dupliziert und einer Variablen bzw. Feld vom Typ Boolean zugewiesen wird, automatisch als Optionsfeld erscheint.
- Sie können in der Regel eine Variable dynamisch in ein Feld und umgekehrt umwandeln
  Dagegen lassen sich grafische Objekte, wie Schaltflächen, Optionsfelder, etc. nicht in andere Typen zur Steuerung umwandeln.

Ist der Variablentyp nicht kompatibel mit dem Objekt, führt der Befehl nichts aus und die Variable OK wird auf 0 (Null) gesetzt. Lassen Sie diesen Parameter weg, erstellt 4D die Variable dynamisch. Bei einigen Objekttypen wird diese Variable automatisch deklariert (sofern durch das Objekt bestimmt), bei anderen wird der Typ durch die erste Zuweisung bestimmt. Weitere Informationen dazu finden Sie im Abschnitt **Dynamische Variablen**. Duplizieren Sie ein statischen Objekt, z.B. Linie, Rechteck, statisches Bild, wird dieser Parameter ignoriert. Übergeben Sie einen Nil-Pointer (z. B. nicht initialisierte Pointer-Variable), wenn Sie die Möglichkeit haben wollen, die anderen Parameter zu benutzen.

Den Parameter *GehenZu* verwenden Sie in zwei Fällen:

- Update der Eingabereihenfolge: In diesem Fall übergeben Sie in *GehenZu* den Namen es eingebbaren Objekts just vor dem duplizierten Objekt. Soll das neue Objekt in der Eingabereihenfolge der Seite das erste Objekt werden, übergeben Sie die Konstante Object First in entry order (siehe Befehl **OBJECT Get pointer**).

- Zuweisung zu einer Gruppe Optionsfelder: Optionsfelder funktionieren in koordinierter Form, wenn sie gruppiert sind. Ist das duplizierte Objekt ein Optionsfeld, übergeben Sie in *GehenZu* den Namen des Optionsfeldes der Gruppe, dem das neue Objekt hinzugefügt werden soll.

Lassen Sie diesen Parameter weg oder übergeben einen leeren String, wird das neue Objekt das letzte eingebbare Objekt auf der Formularseite.

Über die Parameter *BewegenH*, *BewegenV* und *AnpassenH*, *AnpassenV* lässt sich das neue Objekt bewegen und in der Größe verändern. Wie beim Befehl **OBJECT MOVE** wird die Richtung zum Bewegen oder Anpassen durch die Vorzeichen der in *BewegenH* und *BewegenV* übergebenen Werte bestimmt:

- Bei einem positivem Wert wird das Objekt nach rechts oder unten bewegt bzw. angepasst.
- Bei einem negativen Wert wird das Objekt nach links oder unten bewegt bzw. angepasst.

Die Werte von *BewegenH*, *BewegenV* und *AnpassenH*, *AnpassenV* ändern die Koordinaten des Objekts in Bezug auf seine vorige Position. Wollen Sie für diese Parameter absolute Koordinaten angeben, übergeben Sie den letzten Parameter *.
Lassen Sie diese Parameter weg, wird das neue Objekt über dem Ausgangsobjekt angelegt.

Dieser Befehl muss im Rahmen einer Formularanzeige verwendet werden. Er wird in der Regel im Formularereignis On Load oder als Folge einer Benutzeraktion (Formularereignis On Clicked) aufgerufen.

- Version being modified

Hinweis: Ist das Formularereignis On Load dem Ausgangsobjekt zugewiesen, wird es beim Ausführen des Befehls für das duplizierte Objekt erzeugt. So lässt sich z.B. der Wert des Objekts initialisieren.

Aus technischen und logischen Gründen lässt sich **OBJECT DUPLICATE** in folgenden Formularereignissen nicht aufrufen:

- On Load in einer Objektmethode
- On Unload
- Ereignis im Kontext Drucken (On Header, On Printing Detail, etc.). Um ein Objekt mehrmals zu drucken, verwenden Sie besser die Funktion **Print object**.

Wird der Befehl in einem nicht unterstützten Kontext aufgerufen, wird das Objekt nicht dupliziert und die Systemvariable OK wird auf 0 gesetzt. Bei Aufrufen in einem Druckkontext wird auch der Fehler -10601 aufgerufen.

Wurde der Befehl korrekt ausgeführt, wird die Variable OK auf 1 gesetzt, andernfalls auf 0 (Null).

Beispiel 1

Eine neue Schaltfläche "CancelButton" über dem vorhandenen Objekt "OKButton" anlegen und der Variablen *vCancel* zuweisen:

```
OBJECT DUPLICATE(*;"OKButton";"CancelButton";vCancel)
```

Beispiel 2

Ein neues Optionsfeld "bRadio6" auf Basis des vorhandenen Optionsfeldes "bRadio5" anlegen. Diese Schaltfläche wird der Variablen <>r6 zugewiesen, in die Gruppe von "bRadio5" integriert und 20 Pixel höher gesetzt:

```
OBJECT DUPLICATE(*;"bRadio5";"bRadio6";<>r6;"bRadio5";0;20)
```

OBJECT Get action

OBJECT Get action ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	⇒	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt Feld oder Variable
Objekt	Formularobjekt	⇒	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Funktionsergebnis	Text	⇐	Name der zugewiesenen Standardaktion und (sofern vorhanden) Parameter String

Beschreibung

Die Funktion **OBJECT Get action** gibt den Namen und (sofern vorhanden) Parameter der Standardaktion zurück, die dem Objekt, definiert in den Parametern *Objekt* und *, zugewiesen ist.

Mit dem optionalen Parameter * ist *Objekt* ein Objektname (String), ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Referenz auf ein Feld oder eine Variable anstelle eines String (nur für Objekt Feld oder Variable).

Sie können eine Standardaktion für ein Objekt im Formulareditor über die Eigenschaftenliste oder über die Funktion **OBJECT SET ACTION** setzen. **OBJECT Get action** gibt einen String mit dem Namen der Standardaktion zurück, die dem Objekt zugewiesen ist (und sofern vorhanden seine Parameter).

Die komplette Übersicht der Standardaktionen finden Sie im Abschnitt **Standardaktionen** des Handbuchs *Designmodus*.

Beispiel

Die Aktion "Abbrechen" allen Objekten im Formular zuweisen, denen noch keine Aktion zugewiesen wurde:

```
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 ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname(String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	Boolean	⮌	Wahr= automatische Rechtschreibprüfung Falsch= keine automatische Rechtschreibprüfung

Beschreibung

Die Funktion **OBJECT Get auto spellcheck** gibt den Status der Option **Auto spellcheck** für das bzw. die Objekte zurück, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable oder ein Feld. In diesem Fall übergeben Sie eine Referenz anstelle eines Namens.

Die Funktion gibt **Wahr** zurück, wenn die automatische Rechtschreibprüfung für das Objekt aktiviert ist; **Falsch**, wenn sie nicht aktiviert ist.

OBJECT GET BEST SIZE

OBJECT GET BEST SIZE ({* ;} Objekt ; BesteBreite ; BesteHöhe {; maxBreite})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist Objektname (String), Ohne *: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable bzw. Feld (ohne *)
BesteBreite	Lange Ganzzahl	⬅	Optimale Objektbreite
BesteHöhe	Lange Ganzzahl	⬅	Optimale Objekthöhe
maxBreite	Lange Ganzzahl	➡	Maximale Objektbreite

Beschreibung

Der Befehl **OBJECT GET BEST SIZE** gibt für das Formularobjekt, definiert durch * und *Objekt*, in den Parametern *BesteBreite* und *BesteHöhe* die "optimale" Breite und Höhe zurück. Die Werte werden in Pixel angegeben. Dieser Befehl ist zusammen mit **OBJECT MOVE** besonders hilfreich zum Anzeigen oder Drucken von komplexen Berichten.

Die zurückgegebenen Werte geben die Mindestgröße des Objekts an, so dass sein aktueller Inhalt vollkommen innerhalb der Begrenzung liegt. Diese Werte haben sind in der Regel nur für Objekte mit Text von Bedeutung. Die Berechnung berücksichtigt Schriftart, -größe, -stil und Objektinhalt, aber auch Trennung und Zeilenumbruch. Ist das angegebene Objekt leer, gibt *BesteBreite* den Wert 0 (Null) zurück.

Die zurückgegebene Größe berücksichtigt weder Rahmen um das Objekt noch Rollbalken. Um die echte Größe eines Objekts auf dem Bildschirm zu erhalten, müssen Sie die Breite dieser Elemente hinzufügen.

Mit dem optionalen Parameter *MaxBreite* können Sie dem Objekt eine maximale Breite hinzufügen. Ist die optimale Breite größer als dieser Wert, gibt in **OBJECT GET BEST SIZE** *BesteBreite* den Wert von *MaxBreite* zurück und passt die optimale Höhe entsprechend an.

Dieser Befehl verwaltet folgende Objekte:

- Bereiche mit statischem Text
- Text, der als Referenz eingegeben wurde
- Felder und Variablen vom Typ Alpha, Text, Zahl, Ganzzahl, Lange Ganzzahl, Datum, Zeit, Boolean (Optionsfelder und Kontrollkästchen)
- Schaltflächen
- Angezeigte Spalten von Listboxen (nur sichtbare Zeilen werden berücksichtigt)

Für alle anderen Formularobjekte (gruppierte Bereiche, Registerkarten, Rechtecke, Linien, Kreise/Ovale, externe Bereiche, etc.) gibt **OBJECT GET BEST SIZE** die aktuelle Objektgröße zurück. Diese ist im Formulareditor definiert und verwendet möglicherweise den Befehl **OBJECT MOVE**.

Der Befehl eignet sich zusammen mit dem Befehl **OBJECT MOVE** besonders zum Anzeigen oder Drucken komplexer Berichte.

Beispiel

Siehe Beispiel zum Befehl **SET PRINT MARKER**.

OBJECT Get border style

OBJECT Get border style ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable bzw. Feld (ohne *)
Funktionsergebnis	Lange Ganzzahl	⮌	Rahmenstil

Beschreibung

Die Funktion **OBJECT Get border style** gibt den Rahmenstil für das bzw. die Objekte zurück, definiert über die Parameter *Objekt* und *.

Sie können den Rahmenstil über die Eigenschaftenliste im Designmodus setzen oder über den Befehl **OBJECT SET BORDER STYLE**.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Die Funktion gibt den Wert eines Rahmenstils zurück. Sie können den empfangenen Wert mit den folgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)** vergleichen:

Konstante	Typ	Wert	Kommentar
Border Dotted	Lange Ganzzahl	2	Objekte werden mit einer gepunkteten Linie von einem Punkt umrahmt
Border Double	Lange Ganzzahl	5	Objekte werden mit Doppellinie umrahmt, z.B. zwei kontinuierliche Linien von 1 Punkt mit 1 Pixel Abstand
Border None	Lange Ganzzahl	0	Objekte erscheinen ohne Rahmen
Border Plain	Lange Ganzzahl	1	Objekte werden mit einer kontinuierlichen Linie von 1 Punkt umrahmt
Border Raised	Lange Ganzzahl	3	Objekte werden mit einem erhobenen 3D Effekt umrahmt
Border Sunken	Lange Ganzzahl	4	Objekte werden mit einem vertieften 3D Effekt umrahmt
Border System	Lange Ganzzahl	6	Die Rahmenlinie wird gemäß der grafischen Spezifikation des Systems gezeichnet

OBJECT Get context menu

OBJECT Get context menu ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Funktionsergebnis	Boolean	⮌	Wahr = Kontextmenü aktiviert Falsch = Kontextmenü deaktiviert

Beschreibung

Die Funktion **OBJECT Get context menu** gibt den aktuellen Status der Option "Kontextmenü" für das bzw. die Objekte zurück, definiert über die Parameter *Objekt* und *.

Sie können die Option "Kontextmenü" über die Eigenschaftenliste im Designmodus setzen oder über den neuen Befehl **OBJECT SET CONTEXT MENU**.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Die Funktion gibt **Wahr** zurück, wenn das Kontextmenü für *Objekt* aktiviert wurde, sonst **Falsch**.

OBJECT GET COORDINATES

OBJECT GET COORDINATES ({* ;} Objekt ; Links ; Oben ; Rechts ; Unten)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist Name des Objekts (String), Ohne *: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable bzw. Feld (ohne *)
Links	Lange Ganzzahl	⬅	Linke Koordinate des Objekts
Oben	Lange Ganzzahl	⬅	Obere Koordinate des Objekts
Rechts	Lange Ganzzahl	⬅	Rechte Koordinate des Objekts
Unten	Lange Ganzzahl	⬅	Untere Koordinate des Objekts

Beschreibung

Der Befehl **OBJECT GET COORDINATES** gibt die Koordinaten *Links*, *Oben*, *Rechts* und *Unten* (in Punkten) in Variablen oder Feldern der Objekt(e) des aktuellen Formulars an, das mit den Parametern * und *Objekt* definiert ist.

Übergeben Sie den optionalen Parameter *, ist der Parameter *Objekt* ein Objektname (ein String). Übergeben Sie den optionalen Parameter * nicht, ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie keinen String, sondern ein Datenfeld oder eine Referenz auf eine Variable (nur Datenfeld oder Variable von Typ Objekt).

Übergeben Sie in *Objekt* einen Objektnamen mit Jokerzeichen ("@"), um mehrere Objekte auszuwählen, bezeichnen die zurückgegebenen Koordinaten das Rechteck, das aus allen betroffenen Objekten gebildet wird.

Hinweis: Sie können die Art der Interpretation für das Jokerzeichen ("@") festlegen, wenn es in einem String verwendet wird. Diese Option beeinflusst die Befehle zum Thema "Objekte". Weitere Informationen finden Sie im Abschnitt **Jokerzeichen (@)** des Handbuchs *4D Designmodus*.

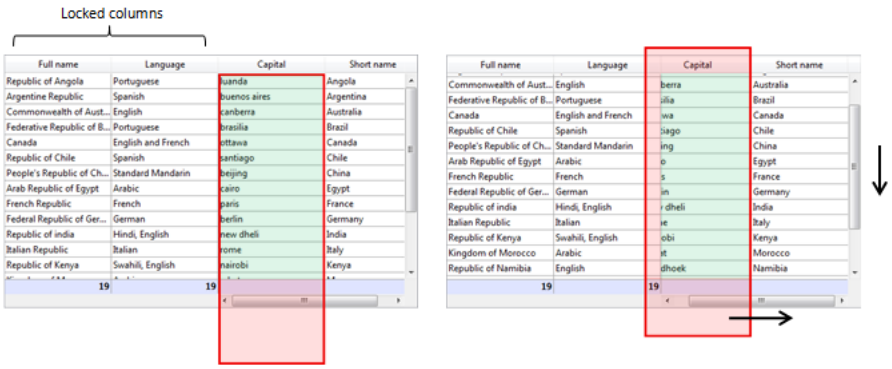
Gibt es ein Objekt nicht oder wird der Befehl nicht in einem Formular aufgerufen, erhalten Sie die Koordinaten (0;0;0;0).

Dieser Befehl kann im Rahmen von Listboxen die Koordinaten bestimmter Teile der Listbox zurückgeben, wie Spalten, Kopfteile oder Fußteile und nicht die Koordinaten der Listbox selbst. In 4D Versionen vor v14 R5 gab dieser Befehl immer die Koordinaten des Objekts Listbox zurück, egal auf welchen Teil er angewendet wurde. Ab jetzt werden die Koordinaten des jeweiligen Unterobjekts zurückgegeben, wenn die angegebene Referenz in *Objekt* ein Unterobjekt Kopfteil, Spalte oder Fußteil der Listbox ist. Auf diese Weise können Sie z.B. beim Klicken in eine Kopfteilzelle der Listbox ein Icon anzeigen, das der Benutzer anklicken kann, um ein Kontextmenü zu öffnen.

Zur Wahrung der Konsistenz bleibt der Referenzrahmen gleich, egal, ob das Objekt ein Unterobjekt der Listbox oder die Listbox selbst ist: Ausgangspunkt ist die obere linke Ecke des Formulars, welches das Objekt enthält. Für Unterobjekte der Listbox sind die zurückgegebenen Koordinaten theoretisch, d.h. sie berücksichtigen den Scrollen-Status der Listbox vor dem Zuschneiden. So kann es vorkommen, das das Unterobjekt nicht oder nur teilweise sichtbar ist, wenn die angegebenen Koordinaten außerhalb der Formulargrenzen oder negativ sind. Um herauszufinden, ob das Unterobjekt bzw. welcher Teil davon sichtbar ist, müssen Sie die zurückgegebenen Koordinaten mit den Koordinaten der Listbox vergleichen. Dabei gelten folgende Regeln:

- Alle Unterobjekte werden auf die Koordinaten der zugehörigen Listbox zugeschnitten, d.h. wie sie vom Befehl **OBJECT GET COORDINATES** für die Listbox zurückgegeben werden.
- Die Unterobjekte Kopfteil und Fußteil erscheinen vor dem Inhalt der Spalte: Überschneiden sich die Koordinaten einer Spalte mit den Koordinaten der Zeilen für Kopf- bzw. Fußteil, wird die Spalte an dieser Schnittstelle nicht angezeigt.
- Elemente gesperrter Spalten erscheinen vor den Elementen scrollbarer Spalten: Überschneiden sich die Koordinaten eines Elements in einer scrollbaren Spalte mit den Koordinaten eines Elements in einer gesperrten Spalte, erscheint es nicht an dieser Schnittstelle.

Nehmen wir als Beispiel folgende Darstellung eines Formulars:



Die Koordinaten von *Capital* werden als rotes Rechteck dargestellt. Wie Sie im ersten Bild sehen können, ist die Spalte länger als die Listbox, d.h. ihre Koordinaten liegen außerhalb der unteren Grenze der Listbox, inkl. Fußteil. Im zweiten Bild wurde die Listbox gescrollt, so dass die Spalte jetzt auch die Spalte *Language* und den Kopfteil überlagert. Um den tatsächlich sichtbaren Bereich (grüner Teil) zu berechnen, müssen Sie die roten Bereiche davon subtrahieren.

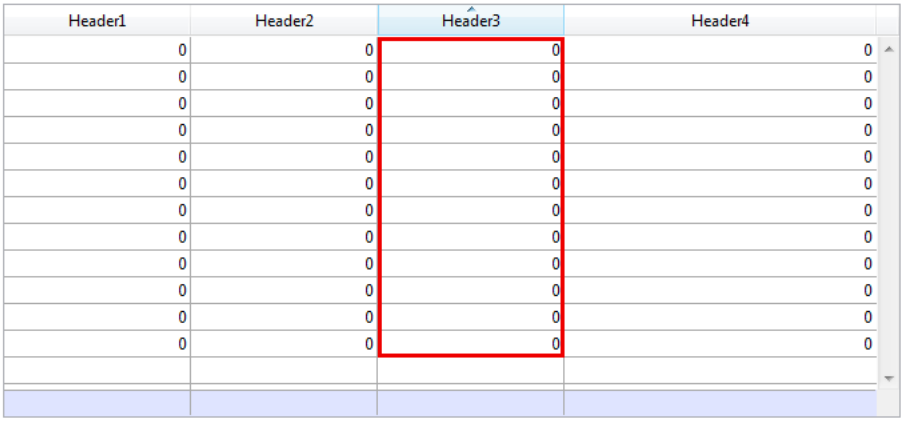
Beispiel 1

Sie möchten die Koordinaten des Rechtecks erhalten, das aus allen Objekten gebildet wird, die mit "Schaltfläche" beginnen:

```
OBJECT GET COORDINATES(*;"Schaltfläche@";Links;Oben;Rechts;Unten)
```

Beispiel 2

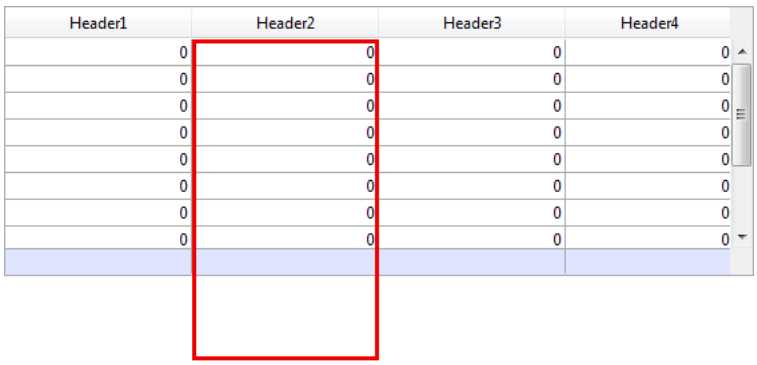
Für eine Oberflächenanzeige soll der angeklickte Bereich mit einem roten Rechteck eingerahmt werden:



Die Objektmethode der Listbox lautet wie folgt:

```
OBJECT SET VISIBLE(*;"rectangleInfo";False)  //Roten Rahmen für Rechteck setzen
$ptr:=OBJECT Get pointer(Object current)
OBJECT GET COORDINATES($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)  // liegt der angeklickte Bereich außerhalb der Listbox
   OBJECT SET VISIBLE(*;"Alert";True)  //eine Meldung zeigen
Else
   OBJECT SET VISIBLE(*;"Alert";False)
End if
```

Diese Methode gibt theoretische Koordinaten zurück. Wird die Größe der Listbox verändert, müssen Sie u.U. den Ausschnitt berechnen, um den sichtbaren Teil herauszufinden:



OBJECT Get corner radius

OBJECT Get corner radius ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable bzw. Feld (ohne *)
Funktionsergebnis	Lange Ganzzahl	⮌	Aktueller Radius der abgerundeten Ecken (in Pixel)

Beschreibung

Die Funktion **OBJECT Get corner radius** gibt den aktuellen Radius der gerundeten Ecken im Objekt "Rundes Viereck", definiert in *Objekt*, zurück. Dieser Wert wurde zuvor über die Eigenschaftenliste auf Formularebene (siehe **Neue Eigenschaft Eckradius für abgerundete Vierecke**) oder über den Befehl **OBJECT SET CORNER RADIUS** für den aktuellen Prozess gesetzt.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Hinweis: Da diese Funktion im aktuellen 4D Release nur auf abgerundete Vierecke, d.h. statische Objekte, angewandt wird, müssen Sie immer den Parameter * übergeben und die Syntax Objektname verwenden.

Die Funktion gibt den Wert (in Pixel) des Radius der gerundeten Ecken zurück. Der Wert ist standardmäßig 5 Pixel.

Beispiel

In der Methode einer Schaltfläche könnte folgender Code hinzugefügt werden:

```
C_LONGINT($radius)
$radius:=OBJECT Get corner radius(*;"GreenRect")  //den aktuellen Wert erhalten
OBJECT SET CORNER RADIUS(*;"GreenRect";$radius+1)  //den Radius erhöhen
 // Der maximale Wert wird automatisch verwaltet: Ist er erreicht,
 // führt die Schaltfläche nichts aus.
```

OBJECT Get data source

OBJECT Get data source ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Funktionsergebnis	Zeiger	⮌	Zeiger der aktuellen Datenquelle von Objekt

Beschreibung

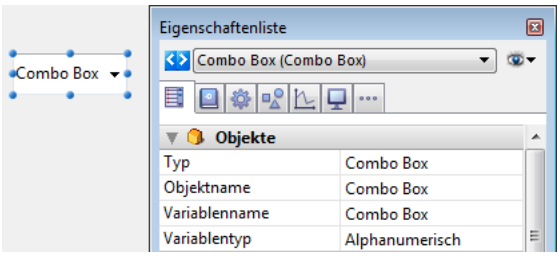
Die Funktion **OBJECT Get data source** gibt die aktuelle Datenquelle des bzw. der Objekte zurück, definiert über die Parameter *Objekt* und *.

Sie können die Datenquelle über die Eigenschaftenliste im Designmodus setzen oder über den Befehl **OBJECT SET DATA SOURCE**.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Beispiel

Wir nehmen ein Objekt Combo Box aus einem Formular:



und führen folgenden Code aus:

```
$vPtr :=OBJECT Get data source(*;"vCombo")
 // $vPtr enthält -> vCombo
```

OBJECT GET DRAG AND DROP OPTIONS

OBJECT GET DRAG AND DROP OPTIONS ({* ;} Objekt ; draggable ; autoDrag ; droppable ; autoDrop)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
draggable	Boolean	⬅	Draggable = Wahr; sonst Falsch
autoDrag	Boolean	⬅	autoDrag = Wahr; sonst Falsch
droppable	Boolean	⬅	droppable = Wahr; sonst Falsch
autoDrop	Boolean	⬅	autoDrop = Wahr; sonst Falsch

Beschreibung

Der Befehl **OBJECT GET DRAG AND DROP OPTIONS** gibt die Drag-and-Drop Optionen für das bzw. die Objekte zurück, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Der Befehl gibt die aktuellen Drag-and-Drop Optionen zurück, wie sie im Designmodus oder für den aktuellen Prozess, der den Befehl **OBJECT SET DRAG AND DROP OPTIONS** verwendet, definiert sind.

In jedem Parameter übergeben Sie einen Boolean Wert, der angibt, ob diese Option aktiviert oder deaktiviert ist.

- *draggable* = Wahr: Objekt lässt sich im programmierten Modus ziehen
- *autoDrag* = Wahr (nur bei Textfeldern und Variablen, Comboboxen und Listboxen): Objekt lässt sich im automatischen Modus ziehen
- *droppable* = Wahr: Objekt akzeptiert Drop im programmierten Modus
- *autoDrop* = Wahr (nur bei Bildfeldern und Variablen, Text, Comboboxen und Listboxen): Objekt akzeptiert Drop im automatischen Modus

OBJECT Get enabled

OBJECT Get enabled ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable Feld (ohne *)
Funktionsergebnis	Boolean	⮌	Wahr = Objekt(e) aktiviert; andernfalls Falsch

Beschreibung

Die Funktion **OBJECT Get enabled** gibt Wahr zurück, wenn das Objekt bzw. die Objektgruppe, definiert in *Objekt*, im Formular aktiviert ist; Falsch, wenn es nicht aktiviert ist.

Ein aktiviertes Objekt reagiert auf Mausklicks und Tastenkürzel.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Objekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz (nur Objektvariable) anstelle eines String.

Diese Funktion lässt sich auf folgende Objekttypen anwenden:

- Schaltfläche, Standard Schaltfläche, 3D Schaltfläche, Unsichtbare Schaltfläche, Hervorgehobene Schaltfläche
- Optionsfeld, 3D Optionsfeld, Optionsbild
- Kontrollkästchen, 3D Kontrollkästchen
- PopUp-Menü, DropDown-Liste, Combo Box, Menü/DropDown-Liste
- Thermometer, Lineal

OBJECT Get enterable

OBJECT Get enterable ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable bzw. Feld (ohne *)
Funktionsergebnis	Boolean	⮌	Wahr = Objekt(e) eingebbar; andernfalls falsch

Beschreibung

Die Funktion **OBJECT Get enterable** gibt wahr zurück, wenn das Objekt bzw. die Objektgruppe, definiert in *Objekt*, eingebbare Attribute hat; andernfalls gibt sie Falsch zurück.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie ein Feld oder eine Variablenreferenz (nur Objektfeld oder -variable) anstelle eines String.

OBJECT GET EVENTS

OBJECT GET EVENTS ({* ;} Objekt ; arrEreignisse)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable bzw. Feld (ohne *)
arrEreignisse	Array Lange Ganzzahl	⬅	Array der aktivierten Ereignisse

Beschreibung

Der Befehl **OBJECT GET EVENTS** erhält die aktuelle Konfiguration der Formularereignisse für das bzw. die Objekte, definiert über die Parameter *Objekt* und *.

Formularereignisse lassen sich entweder über die Eigenschaftenliste aktivieren/deaktivieren oder über den Befehl **OBJECT SET EVENTS** wenn er im aktuellen Prozess aufgerufen wurde.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt). Um die Konfiguration der Ereignisse für das Formular selbst zu erhalten, übergeben Sie in *Objekt* den optionalen Parameter * und einen leeren String "": in diesem Fall bestimmen Sie das aktuelle Formular.

Hinweis: Um Ereignisse für ein Unterformular zu einer Tabelle zu erhalten, können Sie nur die auf dem Objektnamen basierende Syntax verwenden.

Im Parameter *arrEreignisse* übergeben Sie ein Array Lange Ganzzahl. Beim Ausführen des Befehls wird das Array automatisch angepasst und empfängt alle vordefinierten oder eigenen Formularereignisse, die für *Objekt* oder das Formular aktiviert sind. Sie können die erhaltenen Werte mit den Konstanten unter dem Thema **Formularereignisse** vergleichen.

Beachten Sie, dass das Array *arrEreignisse* leer zurückgegeben wird, wenn dem Objekt keine Objektmethode oder dem Formular keine Formularmethode zugewiesen ist.

Beispiel

Zwei Ereignisse aktivieren und die Liste der Eigenschaften für ein Objekt erhalten:

```
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 ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable oder Feld
Funktionsergebnis	Text	⮌	Name des Filters

Beschreibung

Die Funktion **OBJECT Get filter** gibt den Namen der Filter zurück, die dem Objekt bzw. der Objektgruppe, definiert in *Objekt*, zugeordnet sind.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie ein Feld oder eine Variablenreferenz anstelle eines String.

OBJECT Get focus rectangle invisible

OBJECT Get focus rectangle invisible ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname(String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable
Funktionsergebnis	Boolean	⮌	Wahr = Fokusrahmen ausgeblendet, Falsch = Fokusrahmen angezeigt

Beschreibung

Die Funktion **OBJECT Get focus rectangle invisible** gibt den Status der Option sichtbar des Fokusrahmens für das bzw. die Objekte zurück, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess. Die Einstellung entspricht der Eigenschaft **Fokusrahmen ausblenden**, die in der Eigenschaftenliste des Designmodus für eingebbare Objekte verfügbar ist.
Diese Funktion gibt den aktuellen Status der Option zurück, wie er im Designmodus oder über den Befehl **OBJECT SET FOCUS RECTANGLE INVISIBLE** definiert wurde.

Hinweis: Diese Option können Sie nur auf Mac OS verwenden. Unter Windows hat sie keine Auswirkung.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.
Die Funktion gibt **Wahr** zurück, wenn der Fokusrahmen ausgeblendet ist; **Falsch**, wenn er angezeigt wird.

OBJECT Get font

OBJECT Get font ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable oder Feld
Funktionsergebnis	Text	⮌	Name der Schrift

Beschreibung

Die Funktion **OBJECT Get font** gibt den Namen der Schriftzeichen im Formularobjekt zurück, definiert in *Objekt*.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie ein Feld oder eine Variablenreferenz anstelle eines String.

OBJECT Get font size

OBJECT Get font size ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable oder Feld
Funktionsergebnis	Lange Ganzzahl	⮌	Schriftgröße in Punkt

Beschreibung

Die Funktion **OBJECT Get font size** gibt die Größe (in Punkten) der Schriftzeichen im Formularobjekt zurück, definiert in *Objekt*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie ein Feld oder eine Variablenreferenz (nur Objektfeld oder -variable) anstelle eines String.

OBJECT Get font style

OBJECT Get font style (* ; Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable oder Feld
Funktionsergebnis	Lange Ganzzahl	⮌	Schriftstil

Beschreibung

Die Funktion **OBJECT Get font style** gibt den aktuellen Stil der Schriftzeichen im Formularobjekt zurück, definiert in *Objekt*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie ein Feld oder eine Variablenreferenz (nur Objektfeld oder -variable) anstelle eines String.

Sie können den zurückgegebenen Wert mit dem Wert einer der vordefinierten Konstanten unter dem Thema **Schriftstile** vergleichen:

Konstante	Typ	Wert
Plain	Lange Ganzzahl	0
Bold	Lange Ganzzahl	1
Italic	Lange Ganzzahl	2
Underline	Lange Ganzzahl	4

OBJECT Get format

OBJECT Get format ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, Ohne *: Feld oder Variable
Funktionsergebnis	String	⮌	Anzeigeformat für Objekt

Beschreibung

Die Funktion **OBJECT Get format** gibt das aktuelle Anzeigeformat für das in *Objekt* angegebene Objekt zurück.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname ist. In diesem Fall übergeben Sie einen String. Ohne * geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie eine Referenz auf ein Feld oder eine Variable.

Diese Funktion gibt das aktuelle Anzeigeformat des Objekts zurück; das ist das Format, welches in der Designumgebung oder über den Befehl **OBJECT SET FORMAT** definiert wurde. **OBJECT Get format** arbeitet mit allen Arten von Formularobjekten (Felder oder Variablen), die ein Anzeigeformat akzeptieren: Boolean, Datum, Zeit, Bild, String, Zahl sowie Schaltflächengitter, Halbkreisskalen, Thermometer, Bild PopUp-Menüs, Bildschaltflächen, 3D Schaltflächen und Kopfteile von Listboxen.
Weitere Informationen zu den Anzeigeformaten finden Sie in der Dokumentation zum Befehl **OBJECT SET FORMAT**.

Hinweis: Wenden Sie diese Funktion auf einen Satz Objekte an, wird das Format des zuletzt gewählten Objekts zurückgegeben.

Wird **OBJECT Get format** auf Objekte vom Typ Datum, Zeit oder Bild (als Konstanten definierte Formate) angewandt, entspricht der zurückgegebene String dem Zeichen Code der Konstanten. Um den Wert zu erhalten, wenden Sie die Funktion **Character code** auf das Ergebnis an (siehe unten).

Beispiel 1

Über diesen Code erhalten Sie den Wert der Konstante **format**, angewandt auf die Bildvariable mit Namen "MeinFoto":

```
C_STRING(2;$format)
OBJECT SET FORMAT(*;"MeinFoto";Char(On background))
  `Hintergrundformat anwenden (Wert= 3)
$format:=OBJECT Get format(*;"MeinFoto")
ALERT("Formatnummer:"+String(Ascii($format)))
  `Anzeigewert "3"
```

Beispiel 2

Über diesen Code erhalten Sie das Format, angewandt auf das Boolean Feld [Members]Marital_status:

```
C_STRING(30;$format)
$format:=OBJECT Get format([Members]Marital_status)
ALERT($format)  `Anzeigeformat, zum Beispiel "verheiratet;ledig"
```

OBJECT Get help tip

OBJECT Get help tip ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable
Funktionsergebnis	Text	⮌	Hilfemeldung für Objekt

Beschreibung

Die Funktion **OBJECT Get help tip** gibt die Hilfemeldung für das bzw. die Objekte zurück, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Die Funktion gibt die aktuelle Hilfemeldung für das Objekt zurück, wie es im Designmodus definiert ist oder für den Prozess, der den Befehl **OBJECT SET HELP TIP** verwendet. Der zurückgegebene String zeigt die Meldung, wie sie beim Ausführen des Formulars erscheint. Enthält sie generische Einträge (xliff resname oder 4D Referenzen), werden sie gemäß dem Kontext interpretiert.

Beispiel

Der Titel einer Bildschaltfläche wird als Hilfemeldung gespeichert. Dieser Titel wird in einer xliff Datei gespeichert und lautet je nach der aktuellen Sprache des Programms anders:

```
OBJECT SET HELP TIP(*;"button1";":xliff:btn_discover")
$helpmessage:=OBJECT Get help tip(*;"button1")
 // $helpmessage enthält z.B. "Entdecken" mit einem deutschen 4D und "Discover" mit einem englischen 4D.
```

OBJECT Get horizontal alignment

OBJECT Get horizontal alignment ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Feld oder Variable
Funktionsergebnis	Lange Ganzzahl	⮌	Code für Ausrichtung

Beschreibung

Die Funktion **OBJECT Get horizontal alignment** gibt einen Code mit der Art der Ausrichtung für die Parameter *Objekt* und * zurück.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den Parameter * geben Sie in *Objekt* ein Feld oder eine Variable an. Sie geben dann nicht einen String, sondern eine Referenz auf ein Feld oder eine Variable an.

Hinweis: Wenden Sie die Funktion auf eine Gruppe von Objekten an, wird nur der Wert der Ausrichtung für das letzte Objekt zurückgegeben.

Der zurückgegebene Code entspricht einer der nachfolgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)**:

Konstante	Typ	Wert
Align center	Lange Ganzzahl	3
Align default	Lange Ganzzahl	1
Align left	Lange Ganzzahl	2
Align right	Lange Ganzzahl	4

OBJECT Get horizontal alignment ist für folgende Formularobjekte verwendbar:

- Rollbare Bereiche
- Combo Boxen
- Statischer Text
- Gruppenboxen
- PopUp-Menü/DropDown-Listen
- Felder
- Variablen
- Listboxen
- Spalten für Listboxen
- Kopfteile für Listboxen
- Fußteile für Listboxen
- 4D Write Pro Bereiche, siehe **4D Write Pro Handbuch**

OBJECT Get indicator type

OBJECT Get indicator type ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	Lange Ganzzahl	⮌	Typ der Anzeige

Beschreibung

Die Funktion **OBJECT Get indicator type** gibt den aktuellen Anzeigetyp für das bzw. die Thermometer zurück, definiert über die Parameter *Objekt* und *.

Sie können den Anzeigetyp über die Eigenschaftenliste im Designmodus setzen oder über die Funktion **OBJECT SET INDICATOR TYPE**.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Der zurückgegebene Wert entspricht einer der folgenden Konstanten unter dem Thema "**Formularobjekte (Eigenschaften)**":

Konstante	Typ	Wert	Kommentar
Asynchronous progress bar	Lange Ganzzahl	3	Drehendes Rad
Barber shop	Lange Ganzzahl	2	Animierter Balken
Progress bar	Lange Ganzzahl	1	Standard Ablaufbalken

OBJECT Get keyboard layout

OBJECT Get keyboard layout ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable
Funktionsergebnis	String	⮌	Sprachcode des Layout, "" = kein Layout

Beschreibung

Die Funktion **OBJECT Get keyboard layout** gibt die aktuelle Tastaturbelegung für das bzw. die Objekte zurück, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable oder ein Feld. In diesem Fall übergeben Sie eine Referenz anstelle eines Namens.

Diese Funktion gibt einen String mit dem verwendeten Sprachcode an, basierend auf RFC3066, ISO639 und ISO3166. Weitere Informationen dazu finden Sie unter dem Befehl **SET DATABASE LOCALIZATION**.

OBJECT Get list name

OBJECT Get list name ({* ;} Objekt {; ListeTyp}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable oder Feld
ListeTyp	Lange Ganzzahl	➡	Typ der Liste: Auswahlliste, Erforderlich-Liste oder Ausgenommen-Liste
Funktionsergebnis	Text	⮌	Name der Auswahlliste (im Designmodus angegeben)

Beschreibung

Die Funktion **OBJECT Get list name** gibt den Namen der Auswahlliste zurück, die dem Objekt bzw. der Objektgruppe, definiert in *Objekt*, zugeordnet ist. 4D ermöglicht, eine Auswahlliste (erstellt über den Auswahllisten-Editor im Designmodus) mit Formularobjekten über den Formulareditor oder den Befehl **OBJECT SET LIST BY NAME** zu erstellen.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie ein Feld oder eine Variablenreferenz anstelle eines String.

Mit dem optionalen Parameter *ListeTyp* bestimmen Sie den gewünschten Typ. Standardmäßig, d.h. ohne den Parameter *ListeTyp*, gibt die Funktion den Namen der Auswahlliste (Liste der Werte) zurück, die *Objekt* wie in bisherigen 4D Versionen zugeordnet war. Sie können die Namen der Erforderlich- oder Ausgenommen-Listen erhalten, wenn Sie in *ListeTyp* eine der folgenden Konstanten unter dem Thema "**Formularobjekte (Eigenschaften)**" übergeben:

Konstante	Typ	Wert	Kommentar
Choice list	Lange Ganzzahl	0	Liste zum Auswählen von Werten (Option "Auswahlliste" in der Eigenschaftenliste) (Standard)
Excluded list	Lange Ganzzahl	2	Liste mit ausgeschlossenen Werten für die Eingabe (Option "Ausgenommen-Liste" in der Eigenschaftenliste)
Required list	Lange Ganzzahl	1	Liste mit erforderlichen Werten für die Eingabe (Option "Erforderlich-Liste" in der Eigenschaftenliste)

Ist *Objekt* kein Listentyp zugewiesen, gibt die Funktion einen leeren String ("") zurück.

OBJECT Get list reference

OBJECT Get list reference ({* ;} Objekt {; ListeTyp}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
ListeTyp	Lange Ganzzahl	➡	Typ der Liste: Auswahlliste, Erforderlich-Liste oder Ausgenommen-Liste
Funktionsergebnis	ListRef	⮌	Referenznummer der Liste

Beschreibung

Die Funktion **OBJECT Get list reference**gibt die Referenznummer (ListRef) der hierarchischen Liste für das Objekt bzw. die Objekte zurück, definiert über die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Standardmäßig, d.h. ohne den Parameter *ListeTyp*, gibt die Funktion für das Objekt den Namen der Auswahlliste (Werte zum Auswählen) für *Objekt* zurück. Über den Parameter *ListeTyp* können Sie auch die Referenznummer der Erforderlich- oder Ausgenommen-Liste erhalten. Dazu übergeben Sie eine der folgenden Konstanten unter dem Thema "**Formularobjekte (Eigenschaften)**":

Konstante	Typ	Wert	Kommentar
Choice list	Lange Ganzzahl	0	Liste zum Auswählen von Werten (Option "Auswahlliste" in der Eigenschaftenliste) (Standard)
Excluded list	Lange Ganzzahl	2	Liste mit ausgeschlossenen Werten für die Eingabe (Option "Ausgenommen-Liste" in der Eigenschaftenliste)
Required list	Lange Ganzzahl	1	Liste mit erforderlichen Werten für die Eingabe (Option "Erforderlich-Liste" in der Eigenschaftenliste)

Wurde *Objekt* im Parameter *ListeTyp* keine hierarchische Liste zugewiesen, gibt die Funktion 0 zurück.

OBJECT GET MAXIMUM VALUE

OBJECT GET MAXIMUM VALUE ({* ;} Objekt ; maxWert)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
maxWert	Datum, Zeit, Zahl	⬅	Aktueller Maximumwert für Objekt

Beschreibung

Der Befehl **OBJECT GET MAXIMUM VALUE** gibt in der Variablen *maxWert* den aktuellen Maximumwert des Objekts bzw. der Objekte zurück, definiert über die Parameter *Objekt* und *.

Sie können die Eigenschaft "Maximum" über die Eigenschaftenliste im Designmodus setzen oder über den Befehl **OBJECT SET MAXIMUM VALUE**.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt].

OBJECT GET MINIMUM VALUE

OBJECT GET MINIMUM VALUE ({* ;} Objekt ; minWert)

Parameter	Typ		Beschreibung
*	Operator	⇒	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	⇒	Objektname (mit *) oder Feld oder Variable (ohne *)
minWert	Datum, Zeit, Zahl	⇐	aktueller Minimumwert für Objekt

Beschreibung

Der Befehl**OBJECT GET MINIMUM VALUE** gibt in der Variablen minWert den aktuellen Minimumwert für das bzw. die Objekte zurück, definiert über die Parameter *Objekt* und *.

Sie können die Eigenschaft "Minimum" über die Eigenschaftenliste im Designmodus setzen oder über den Befehl **OBJECT SET MINIMUM VALUE**.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

OBJECT Get multiline

OBJECT Get multiline ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
Funktionsergebnis	Lange Ganzzahl	⮌	Status Mehrzeilig des Objekts

Beschreibung

Die Funktion **OBJECT Get multiline** gibt den aktuellen Status der Option "Mehrzeilig" für das bzw. die Objekte zurück, definiert über die Parameter *Objekt* und *.

Sie können die Option "Mehrzeilig" für ein Objekt über die Eigenschaftenliste oder über den neuen Befehl **OBJECT SET MULTILINE** setzen.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter Objekt ein Objektname (String) ist. Ohne diesen Parameter ist Objekt ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Der zurückgegebene Wert entspricht einer der folgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)**:

Konstante	Typ	Wert	Kommentar
Multiline Auto	Lange Ganzzahl	0	In einzeiligen Bereichen werden Wörter am Zeilenende abgeschnitten und keine Zeilenumbrüche gesetzt. In mehrzeiligen Bereichen führt 4D automatisch Zeilenumbrüche aus.
Multiline No	Lange Ganzzahl	2	Es gibt nie Zeilenumbrüche: Der Text erscheint immer als eine Zeile. Bei Alpha oder Textfeldern bzw. Variablen mit Zeilenumbrüchen wird der Text nach dem ersten Zeilenumbruch entfernt, sobald der Text geändert wird.
Multiline Yes	Lange Ganzzahl	1	In einzeiligen Bereichen erscheint der Text bis zum ersten Zeilenumbruch oder bis zum letzten Wort, das sich ganz anzeigen lässt. 4D fügt Zeilenumbrüche ein; über die Pfeiltaste nach unten lässt sich im Inhalt scrollen. In mehrzeiligen Bereichen führt 4D automatische Zeilenumbrüche aus.

Hinweis: Weisen Sie **OBJECT Get multiline** einem Objekt zu, das die Option "Mehrzeilig" nicht unterstützt, gibt sie 0 zurück.

OBJECT Get name

OBJECT Get name {(Selector)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Selector	Lange Ganzzahl	⇒	Objektkategorie
Funktionsergebnis	Text	⇐	Name des Objekts

Beschreibung

Die Funktion **OBJECT Get name** gibt den Namen eines Formularobjekts zurück..

Die Funktion kann in Selector zwei Arten bezeichnen. Sie können eine der folgenden Konstanten unter dem Thema **Formularobjekte (Zugriff)** übergeben:

- Object current oder *Selector* weggelassen: Übergeben Sie diese Konstante bzw. lassen den Parameter *Selector* weg, gibt die Funktion den Namen des aufrufenden Objekts zurück (Objektmethode oder von der Objektmethode aufgerufene Untermethode). In diesem Fall muss die Funktion im Kontext eines Formularobjekts aufgerufen werden, andernfalls gibt sie einen leeren String zurück.
- Object with focus: Übergeben Sie diese Konstante, gibt die Funktion den Namen des Objekts zurück, das im Formular den Fokus hat.

Beispiel

Objektmethode für die Schaltfläche "bValidateForm":

```
$btnName:=OBJECT Get name(Object current)
```

Nach Ausführen dieser Objektmethode enthält die Variable *$btnName* den Wert "bValidateForm".

OBJECT Get placeholder

OBJECT Get placeholder ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
Funktionsergebnis	Text	⮌	Platzhaltertext für Objekt

Beschreibung

Die Funktion **OBJECT Get placeholder** gibt den Platzhaltertext für das bzw. die Objekte zurück, definiert über die Parameter *Objekt* und *. Ist dem Objekt kein Platzhaltertext zugewiesen, gibt die Funktion einen leeren String zurück.

Sie können den Text für Platzhalter über die Eigenschaftenliste im Designmodus setzen oder über den Befehl **OBJECT SET PLACEHOLDER**.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Ist der Platzhalter eine xliff Referenz, die über die Eigenschaftenliste definiert wurde, gibt die Funktion die ursprüngliche Referenz in Form von ":xliff:resname" zurück und nicht den berechneten Wert.

Beispiel

Den Platzhaltertext für ein Feld erhalten:

```
$txt:=OBJECT Get placeholder([People]LastName)
```

OBJECT Get pointer

OBJECT Get pointer {(Selector {; ObjektName {; UnterformularName}})} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Selector	Lange Ganzzahl	➡	Objektkategorie
ObjektName	Text	➡	Objektname
UnterformularName	Text	➡	Objektname des Unterformulars
Funktionsergebnis	Zeiger	⮌	Zeiger auf Objektvariable

Beschreibung

Die Funktion **OBJECT Get pointer** gibt einen Zeiger auf die Variable eines Formularobjekts zurück.

Mit dieser Funktion können Sie verschiedene Objekte nach dem Wert des Parameters *Selector* definieren. Sie können eine Konstante unter dem Thema **Formularobjekte (Zugriff)** übergeben:

- Object current oder kein *Selector*: Übergeben Sie diese Konstante oder lassen den Parameter *Selector* weg, gibt die Funktion einen Zeiger auf die dem aktuellen Objekt zugewiesene Variable zurück. Das ist das Objekt, dessen Formularmethode ausgeführt wird.
  Hinweis: Dies ist exakt identisch mit der bisherigen Funktionsweise von **Self**. Diese Funktion wird nur zur Wahrung der Kompatibilität beibehalten.
- Object with focus: Übergeben Sie diese Konstante, gibt die Funktion einen Zeiger auf die Variable zurück, die dem Objekt mit dem Fokus im Formular zugewiesen ist. Sind die beiden letzten optionalen Parameter übergeben, werden sie ignoriert.
  Hinweis: Dies ist identisch mit der Funktionsweise von **Focus object**. Diese Funktion ist in 4D v12 überholt.
- Object subform container: Übergeben Sie diesen Selektor, gibt die Funktion einen Zeiger auf die Variable zurück, die dem Unterformular Container zugewiesen ist. Sind die beiden letzten optionalen Parameter übergeben, werden sie ignoriert. Dieser Selektor ist folglich nur im Kontext eines Formulars verwendbar, das als Unterformular dient, um auf die Variable zuzugreifen, die dem Objekt Unterformular Container zugewiesen ist.
- Object named: Übergeben Sie diesen Selektor, müssen Sie auch den optionalen Parameter *ObjektName* übergeben. In diesem Fall gibt die Funktion einen Zeiger auf die Variable zurück, die dem in diesem Parameter übergebenen Objekt zugewiesen wurde.
  Hinweis: Entspricht *ObjektName* einem Unterformular und ist die Option "Ausgabe Unterformular" markiert, gibt die Funktion einen Zeiger auf die Tabelle des Unterformulars zurück, wenn eine Quelltabelle angegeben ist; andernfalls wird Nil zurückgegeben.

Hinweis: Im Rahmen einer Listbox gibt **OBJECT Get pointer** mit den Selektoren Object current oder Object with focus je nach Kontext einen Zeiger auf die Listbox, Spalte oder den Kopfteil zurück. Weitere Informationen dazu finden Sie im Abschnitt **Funktionsweise von OBJECT Get pointer**.

Mit dem optionalen Parameter *UnterformularName* können Sie einen Zeiger auf ein Objekt *ObjektName* wiederfinden, das nicht zum aktuellen Kontext gehört, d.h. im Hauptformular. Diesen Parameter können Sie nur nutzen, wenn der Selektor Object named übergeben wurde.
Wurde der Parameter *UnterformularName* übergeben, sucht die Funktion **OBJECT Get pointer** zuerst im aktuellen Formular nach dem Objekt Unterformular, genannt *UnterformularName*, dann innerhalb dieses Unterformulars nach einem Objekt, genannt *ObjektName*. Wird dieses Objekt gefunden, gibt es einen Zeiger auf die Variable dieses Objekts zurück.

Beispiel

Wir gehen von einem Formular "SF" aus, das im gleichen Elternformular zweimal als Unterformular verwendet wird. Die beiden Unterformulare lauten "SF1" und "SF2". Das Formular "SF" enthält ein Objekt mit Namen *AktuellerWert*. Im Formularereignis "On Load" der Formularmethode des Elternformulars wollen wir das Objekt *AktuellerWert* von SF1 auf "Januar" und das von SF2 auf "Februar" initialisieren:

```
C_POINTER($Ptr)
$Ptr:=OBJECT Get pointer(Object named;"AktuellerWert";"SF1")
$Ptr->:="Januar"
$Ptr:=OBJECT Get pointer(Object named;"AktuellerWert";"SF2")
$Ptr->:="Februar"
```

OBJECT GET PRINT VARIABLE FRAME

OBJECT GET PRINT VARIABLE FRAME ({* ;} Objekt ; variablerRahmen {; festesUnterformular})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
variablerRahmen	Boolean	⬅	Wahr = Variabler Druckrahmen, Falsch = Fester Druckrahmen
festesUnterformular	Lange Ganzzahl	⬅	Optionen zum Drucken von Unterformularen mit fester Größe

Beschreibung

Der Befehl **OBJECT GET PRINT VARIABLE FRAME** erhält die aktuelle Konfiguration der Druckoptionen für das bzw. die Objekte, definiert über die Parameter *Objekt* und *.

Die Eigenschaft *Variabler Druckrahmen* lässt sich über die Eigenschaftenliste oder über den Befehl **OBJECT SET PRINT VARIABLE FRAME** definieren.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Der Parameter *variablerRahmen* gibt der Befehl eine Boolean Variable zurück. Ihr Wert gibt den Status *Aktiviert* (Wahr) oder *Deaktiviert* (**Falsch**) für den variablen Druckrahmen zurück.

Ist *Objekt* ein Unterformular und der variable Druckrahmen deaktiviert (Falsch), gibt der Befehl im Parameter *festesUnterformular* auch die Druckoption fester Rahmen des Unterformulars zurück. Sie können den zurückgegebenen Wert mit den folgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)** vergleichen:

Konstante	Typ	Wert	Kommentar
Print Frame fixed with multiple records	Lange Ganzzahl	2	Der Rahmen behält die gleiche Größe, 4D druckt das Formular jedoch mehrmals, um alle Datensätze einzuschließen.
Print Frame fixed with truncation	Lange Ganzzahl	1	4D druckt nur die Datensätze, die in den Bereich des Unterformulars passen. Das Formular wird nur einmal gedruckt und die nicht gedruckten Datensätze werden ignoriert.

OBJECT GET RESIZING OPTIONS

OBJECT GET RESIZING OPTIONS ({* ;} Objekt ; horizontal ; vertikal)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable
horizontal	Lange Ganzzahl	⬅	Horizontale Anpassungsoption
vertikal	Lange Ganzzahl	⬅	Vertikale Anpassungsoption

Beschreibung

Der Befehl **OBJECT GET RESIZING OPTIONS** gibt die aktuellen Anpassungsoptionen für das bzw. die Objekte zurück, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Der Befehl gibt die aktuellen Anpassungsoptionen zurück, wie sie im Designmodus oder für den Prozess, der den Befehl **OBJECT SET RESIZING OPTIONS** verwendet, gesetzt wurden. Diese Optionen definieren die Anzeige des Objekts, wenn das Formularfenster in der Größe angepasst wird.
Der Parameter *horizontal* gibt einen Wert für die horizontale Anpassungsoption zurück, die für das Objekt gesetzt wurde. Sie können den empfangenen Wert mit einer der folgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)** vergleichen:

Konstante	Typ	Wert	Kommentar
Resize horizontal grow	Lange Ganzzahl	1	Wächst das Fenster um 50% in der Breite, wird das Objekt um 50% nach rechts verbreitert
Resize horizontal move	Lange Ganzzahl	2	Wächst das Fenster um 100 Pixel in der Breite, wird das Objekt um 100 Pixel nach rechts bewegt
Resize horizontal none	Lange Ganzzahl	0	Wird das Fenster verbreitert, bleiben Breite noch Position des Objekts unverändert

Der Parameter *vertikal* gibt einen Wert für die vertikale Anpassungsoption zurück, die für das Objekt gesetzt wurde. Sie können den empfangenen Wert mit einer der folgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)** vergleichen:

Konstante	Typ	Wert	Kommentar
Resize vertical grow	Lange Ganzzahl	1	Wächst das Fenster um 50% in der Höhe, wird das Objekt um 50% nach unten verlängert
Resize vertical move	Lange Ganzzahl	2	Wächst das Fenster um 100 Pixel in der Höhe, wird das Objekt um 100 Pixel nach unten verlängert
Resize vertical none	Lange Ganzzahl	0	Wird das Fenster verlängert, bleiben Höhe Position des Objekts unverändert

OBJECT GET RGB COLORS

OBJECT GET RGB COLORS ({* ;} Objekt ; Vordergrundfarbe {; Hintergrundfarbe {; altHintergrFarbe}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Feld oder Variable
Vordergrundfarbe	Lange Ganzzahl	⬅	Wert RGB Farbe für Vordergrund
Hintergrundfarbe	Lange Ganzzahl	⬅	Wert RGB Farbe für Hintergrund
altHintergrFarbe	Lange Ganzzahl	⬅	Wert RGB Farbe für wechselnden Hintergrund

Beschreibung

Der Befehl **OBJECT GET RGB COLORS** gibt die Vorder- und Hintergrundfarbe des Objekts bzw. der Objektgruppe zurück, definiert in *Objekt*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie ein Feld oder eine Variablenreferenz (nur Objektfeld oder -variable) anstelle eines String.

Wird der Befehl auf ein Objekt vom Typ Listbox angewandt, kann die wechselnde Hintergrundfarbe der Zeilen im Parameter *altHintergrFarbe* zurückgegeben werden. In diesem Fall wird der Wert von *Hintergrundfarbe* nur für den Hintergrund der Zeilen mit ungerader Nummer verwendet.

Die Werte der RGB Farbe, die in den Parametern *Vordergrundfarbe*, *Hintergrundfarbe* und *altHintergrFarbe* zurückgegeben wird, können 4-byte Lange Ganzzahlen im Format (0x00RRGGBB) oder negative Werte entsprechend der Systemfarben sein. Im zweiten Fall können Sie den erhaltenen Wert mit den Konstanten unter dem Thema **SET RGB COLORS** vergleichen:

Konstante	Typ	Wert	Kommentar
Background color	Lange Ganzzahl	-2	
Background color none	Lange Ganzzahl	-16	Diese Konstante lässt sich in den Parametern *Hintergrundfarbe* und *altHintergrFarbe* verwenden.
Dark shadow color	Lange Ganzzahl	-3	
Disable highlight item color	Lange Ganzzahl	-11	
Foreground color	Lange Ganzzahl	-1	
Highlight menu background color	Lange Ganzzahl	-9	
Highlight menu text color	Lange Ganzzahl	-10	
Highlight text background color	Lange Ganzzahl	-7	
Highlight text color	Lange Ganzzahl	-8	
Light shadow color	Lange Ganzzahl	-4	

Hinweis: Die Systemfarben werden über den Befehl **OBJECT SET RGB COLORS** angewendet.

Weitere Informationen über das Format der Parameter *Vordergrundfarbe*, *Hintergrundfarbe* und *altHintergrFarbe* finden Sie unter dem Befehl **OBJECT SET RGB COLORS**.

OBJECT GET SCROLL POSITION

OBJECT GET SCROLL POSITION ({* ;} Objekt ; vPosition {; hPosition})

Parameter	Typ		Beschreibung
*	Operator	⇒	Mit Stern: Objekt ist ein Objektname(String), Ohne Stern: Objekt ist Variable, Feld oder Tabelle
Objekt	Formularobjekt	⇒	Mit *: Objektname, ohne *: Variable, Feld oder Tabelle
vPosition	Lange Ganzzahl	⇐	Nummer der ersten angezeigten Zeile oder vertikales Scrollen in Pixel (Bilder)
hPosition	Lange Ganzzahl	⇐	Nummer der ersten angezeigten Spalte oder horizonales Scrollen in Pixel (Bilder)

Beschreibung

Der Befehl **OBJECT GET SCROLL POSITION** gibt in den Parametern *vPosition* und *hPosition* die Information zur Position der Rollbalken des Formularobjekts zurück, definiert durch die Parameter * und *Objekt*.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* der Name eines Objekts vom Typ Unterformular, rollbarer Bereich, hierarchische Liste, Listbox oder Bild (übergeben Sie hierfür einen String in *Objekt*) ist. Ohne diesen Parameter definieren Sie, dass *Objekt* eine Variable (*ListRef* einer hierarchischen Liste, Bild oder Listboxvariable) oder Feld ist.

Hinweis: Bei Objekten vom Typ Unterformular wird nur die Syntax mit * unterstützt.

Gibt *Objekt* ein Objekt vom Typ Liste (Unterformular, Listenformular, hierarchische Liste, rollbarer Bereich oder Listbox) an, gibt *vPosition* die Nummer der ersten im Objekt angezeigten Spalte und bei Listboxen zusätzlich in *hPosition* die Nummer der ersten im linken Teil der Listbox angezeigten Spalte zurück. Für die anderen Objekttypen gibt der zweite Parameter 0 (Null) zurück.

Gibt *Objekt* ein Bild vom Typ Variable oder Feld an, gibt *vPosition* die vertikale und *hPosition* die horizontale Bewegung des Bildes an. Diese Werte werden in Pixel in Bezug auf den Ursprung des Bildes im lokalen Koordinatensystem ausgedrückt.

OBJECT GET SCROLLBAR

OBJECT GET SCROLLBAR ({* ;} Objekt ; horizontal ; vertikal)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Feld oder Variable
horizontal	Boolean, Lange Ganzzahl	⬅	Sichtbarkeit des horizontalen Rollbalkens
vertikal	Boolean, Lange Ganzzahl	⬅	Sichtbarkeit des vertikalen Rollbalkens

Beschreibung

Der Befehl **OBJECT GET SCROLLBAR** gibt den Status ein-/ausgeblendet des horizontalen und vertikalen Rollbalkens des Objekts bzw. der Objektgruppe an, definiert in *Objekt*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie ein Feld oder eine Variablenreferenz (nur Objektfeld oder -variable) anstelle eines String.

In den Parametern *horizontal* und *vertikal* übergeben Sie Variablen vom Typ Boolean oder Lange Ganzzahl:

- Übergeben Sie eine Variable vom Typ Boolean, gibt der zurückgegebene Wert den **aktuellen** Status des Rollbalken an:
  - Wurde der Rollbalken als ausgeblendet definiert, empfängt der Parameter *False*,
  - Wurde der Rollbalken als eingeblendet definiert, empfängt der Parameter *True*,
  - Wurde der Rollbalken auf automatischen Modus gesetzt, empfängt der Parameter entweder *True* oder *False*, je nach dem aktuellen Anzeigestatus des Objekts.
- Übergeben Sie eine Variable vom Typ Lange Ganzzahl, gibt der zurückgegebene Wert die für den Rollbalken definierte Sichtbarkeit zurück:
  - Wurde der Rollbalken als ausgeblendet definiert, empfängt der Parameter 0,
  - Wurde der Rollbalken als eingeblendet definiert, empfängt der Parameter 1,
  - Wurde der Rollbalken auf automatischen Modus gesetzt, empfängt der Parameter 2.

Dieser Befehl lässt sich mit folgenden Formularobjekten verwenden:

- Listboxen
- Rollbare Bereiche
- hierarchische Listen
- Unterformulare

Weitere Informationen dazu finden Sie unter dem Befehl **OBJECT SET SCROLLBAR**.

OBJECT GET SHORTCUT

OBJECT GET SHORTCUT ({* ;} Objekt ; Taste ; Zusatztasten)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable
Taste	String	⬅	Dem Objekt zugewiesene Taste
Zusatztasten	Lange Ganzzahl	⬅	Ein oder mehrere kombinierte Kürzel für Zusatztasten

Beschreibung

Der Befehl **OBJECT GET SHORTCUT** gibt das zugewiesene Tastenkürzel für das bzw. die Objekte zurück, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Der Parameter *Taste* gibt das Zeichen an, das der Taste zugewiesen wurde (Standardtasten) oder einen String zwischen Klammern, der die entsprechende Taste angibt (Funktionstasten). Sie können diesen Wert mit den Konstanten des Themas **Tastaturabkürzungen** vergleichen (siehe Befehl **OBJECT SET SHORTCUT**).

Der Parameter *Zusatztasten* gibt einen Wert mit der bzw. den zugeordneten Zusatztasten zurück. Bei einer Kombination aus mehreren Zusatztasten gibt der Befehl die Summe der einzelnen Werte zurück. Sie können den zurückgegebenen Wert mit den folgenden Konstanten unter dem Thema **Ereignisse (Zusatztasten)** vergleichen:

Konstante	Typ	Wert	Kommentar
Command key mask	Lange Ganzzahl	256	Strg-Taste unter Windows, Befehlstaste auf OS X
Control key mask	Lange Ganzzahl	4096	Ctrl-Taste auf OS X, oder rechter Mausklick unter Windows und OS X
Option key mask	Lange Ganzzahl	2048	Windows = Alt-Taste, Mac OS = Wahltaste
Shift key mask	Lange Ganzzahl	512	Windows und Mac OS

Bei Tastenkürzeln ohne Zusatztasten gibt der Parameter *Zusatztasten* 0 zurück.

Hinweis: Bezeichnet der Parameter *Objekt* mehrere Objekte im Formular mit unterschiedlicher Einstellung, gibt der Befehl "" in *Taste* und 0 in *Zusatztasten* zurück.

OBJECT Get style sheet

OBJECT Get style sheet ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
Funktionsergebnis	Text	⮌	Name der Stilvorlage

Beschreibung

Die Funktion **OBJECT Get style sheet** gibt den Namen der Stilvorlage für das bzw. die Objekte zurück, definiert über die Parameter *Objekt* und *.

Stilvorlagen lassen sich über die Eigenschaftenliste im Designmodus oder über den Befehl **OBJECT SET STYLE SHEET** für den aktuellen Prozess zuweisen.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Die Funktion kann folgendes zurückgeben:

- Name der Stilvorlage
- Einen leeren String (""), wenn keine Stilvorlage vergeben wurde
- Wurde eine automatische Stilvorlage zugewiesen, eine Konstante aus dem Thema **Schriftstile**:

Konstante	Typ	Wert	Kommentar
Automatic style sheet	Zeichenkette	__automatic__	Wird standardmäßig für alle Objekte verwendet
Automatic style sheet_additional	Zeichenkette	__automatic_additional_text__	Gilt nur für statischen Text, Felder und Variablen für Zusatztext in Dialogfenstern.
Automatic style sheet_main	Zeichenkette	__automatic_main_text__	Gilt nur für statischen Text, Felder und Variablen für Haupttext in Dialogfenstern.

Bezeichnet die Funktion mehrere Objekte, ist die zurückgegebene Stilvorlage nur signifikant, wenn sie für alle Objekte übergeben wurde.

OBJECT GET SUBFORM

OBJECT GET SUBFORM ({* ;} Objekt ; TabelleZeiger ; SeitenUnterformular {; ListenUnterformular})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable
TabelleZeiger	Tabelle	⬅	Zeiger auf Tabelle des Formulars
SeitenUnterformular	Text	⬅	Name des Unterformulars als Seite
ListenUnterformular	Text	⬅	Name des Unterformulars als Liste (Tabellenformular)

Beschreibung

Der Befehl **OBJECT GET SUBFORM** erhält den/die Namen des bzw. der Formulare für das Objekt Unterformular, angegeben in den Parametern *Objekt* und * .

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Der Befehl gibt im Parameter *TabelleZeiger* einen Zeiger auf die Tabelle des bzw. der verwendeten Formulare zurück. Verwendet das Unterformular ein Projektformular, enthält dieser Parameter **Is nil pointer**.

In den Parametern *SeitenUnterformular* und (optional) *ListenUnterformular* gibt der Befehl folgendes zurück:

- Den Formularnamen, wenn das Unterformular im 4D Formulareditor erstellt wurde.
- Das Attribut "name" des Unterformulars, wenn das Unterformular über eine .json Datei oder ein 4D Objekt erstellt wurde. Ist das Attribut "name" undefiniert, gibt der Befehl folgendes zurück:
  - für eine .json Datei den Namen der .json Datei (ohne Endung)
  - für ein Objekt "untitled"

Gibt es kein Listenformular, wird ein leerer String zurückgegeben.

OBJECT GET SUBFORM CONTAINER SIZE

OBJECT GET SUBFORM CONTAINER SIZE (Breite ; Höhe)

Parameter	Typ		Beschreibung
Breite	Lange Ganzzahl	⇐	Breite des Objekts Unterformular
Höhe	Lange Ganzzahl	⇐	Höhe des Objekts Unterformular

Beschreibung

Der Befehl **OBJECT GET SUBFORM CONTAINER SIZE** gibt die *Breite* und *Höhe* (in Pixel) eines "aktuellen" Containers Unterformular zurück, das im Elternformular angezeigt wird.

Dieser Befehl muss in der Methode eines Formulars aufgerufen werden, das als Unterformular dient und in einem Unterformular Container angezeigt wird. Er gibt die *Breite* und *Höhe* des Containers zurück, welches das Unterformular enthält.
Dieser Befehl ist z.B. hilfreich, wenn Objekte im Unterformular gemäß den Einstellungen des Unterformular Containers bewegt oder angepasst werden müssen. Das Unterformular kann diesen Befehl im Formularereignis On Load aufrufen, um den verfügbaren Abstand zum Anzeigen seines Inhalts zu berechnen.

Hinweis: Das Ereignis On Resize lässt sich nicht direkt in der Methode des Unterformulars verwenden. Da es die Größenanpassung eines Fensters betrifft, wird es nur in der Methode des Elternformulars generiert. Sie können dagegen explizit das Unterformular von diesem Ereignis des Elternformulars aus aufrufen, z.B. über den Befehl **EXECUTE METHOD IN SUBFORM**.

- Wird der Befehl von einem Formular aufgerufen, das nicht als Unterformular verwendet wird, gibt er die aktuelle Größe des Formularfensters zurück.
- Wird der Befehl außerhalb des Kontexts Bildschirmanzeige aufgerufen, z.B. beim Drucken des Formulars, gibt er in *Breite* und *Höhe* 0 zurück.

OBJECT Get text orientation

OBJECT Get text orientation ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
Funktionsergebnis	Lange Ganzzahl	⮌	Winkel der Textrotation

Beschreibung

Die Funktion **OBJECT Get text orientation** gibt den aktuellen Wert der Rotation für den Text des bzw. der Objekte zurück, definiert über die Parameter *Objekt* und *.

Sie können die Option "Rotation" für ein Objekt über die Eigenschaftenliste im Designmodus setzen oder über den neuen Befehl **OBJECT SET TEXT ORIENTATION**.

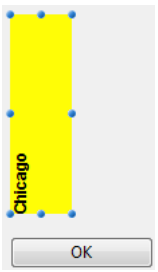
Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Der zurückgegebene Wert entspricht einer der folgenden Konstanten unter dem Thema "**Formularobjekte (Eigenschaften)**":

Konstante	Typ	Wert	Kommentar
Orientation 0°	Lange Ganzzahl	0	Keine Rotation (Standardwert)
Orientation 180°	Lange Ganzzahl	180	Text um 180° im Uhrzeigersinn drehen
Orientation 90° left	Lange Ganzzahl	270	Text um 90° gegen den Uhrzeigersinn drehen
Orientation 90° right	Lange Ganzzahl	90	Text um 90° im Uhrzeigersinn drehen

Beispiel

Nehmen wir folgendes Objekt (Mit der Option Rotation um 90° gegen den Uhrzeigersinn im Formulareditor:



Rufen Sie beim Ausführen des Formulars folgende Anweisung auf:

```
OBJECT SET TEXT ORIENTATION(*;"myText";Orientation 180°)
```

... erscheint das Objekt wie folgt:


```
$vOrt:=OBJECT Get text orientation(*;"myText") //$vOrt=180
```

OBJECT Get three states checkbox

OBJECT Get three states checkbox ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
Funktionsergebnis	Boolean	⮌	Wahr = Kontrollkästchen mit drei Zuständen Falsch = Standard Kontrollkästchen

Beschreibung

Die Funktion **OBJECT Get three states checkbox** gibt den aktuellen Status der Eigenschaft "Drei Zustände" für Kontrollkästchen zurück, definiert über die Parameter *Objekt* und *..

Sie können die Eigenschaft "Drei Zustände" entweder über die Eigenschaftenliste im Designmodus oder über den Befehl **OBJECT SET THREE STATES CHECKBOX**erhalten, wenn er im aktuellen Prozess aufgerufen wurde

OBJECT Get title

OBJECT Get title ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname(String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Feld oder Variable
Funktionsergebnis	Text	⮌	Titel der Schaltfläche

Beschreibung

Die Funktion **OBJECT Get title** gibt den Titel (Bezeichnung) des Formularobjekts zurück, definiert in *Objekt*. Dieser Befehl ist für alle Arten einfacher Objekte mit Textanzeige verwendbar:

- Schaltflächen
- Optionsfelder
- Kontrollkästchen
- Statische Texte
- Gruppenbereiche

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie ein Feld oder eine Variablenreferenz (nur Objektfeld oder -variable) anstelle eines String.

OBJECT Get type

OBJECT Get type ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	Lange Ganzzahl	⮌	Typ des Objekts

Beschreibung

Die Funktion **OBJECT Get type** gibt im aktuellen Formular den Typ des Objekts zurück, definiert über die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Diese Syntax ist zwingend, wenn Sie statische Objekte wie Linien oder Rechtecke bearbeiten.

Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Hinweis: Wenden Sie diese Funktion auf einen Satz Objekte an, wird der Typ des letzten Objekts zurückgegeben.

Der zurückgegebene Wert entspricht einer der folgenden Konstanten unter dem Thema "**Formular Objekttypen**":

Konstante	Typ	Wert
Object type 3D button	Lange Ganzzahl	16
Object type 3D checkbox	Lange Ganzzahl	26
Object type 3D radio button	Lange Ganzzahl	23
Object type button grid	Lange Ganzzahl	20
Object type checkbox	Lange Ganzzahl	25
Object type combobox	Lange Ganzzahl	11
Object type dial	Lange Ganzzahl	28
Object type group	Lange Ganzzahl	21
Object type groupbox	Lange Ganzzahl	30
Object type hierarchical list	Lange Ganzzahl	6
Object type hierarchical popup menu	Lange Ganzzahl	13
Object type highlight button	Lange Ganzzahl	17
Object type invisible button	Lange Ganzzahl	18
Object type line	Lange Ganzzahl	32
Object type listbox	Lange Ganzzahl	7
Object type listbox column	Lange Ganzzahl	9
Object type listbox footer	Lange Ganzzahl	10
Object type listbox header	Lange Ganzzahl	8
Object type matrix	Lange Ganzzahl	35
Object type oval	Lange Ganzzahl	34
Object type picture button	Lange Ganzzahl	19
Object type picture input	Lange Ganzzahl	4
Object type picture popup menu	Lange Ganzzahl	14
Object type picture radio button	Lange Ganzzahl	24
Object type plugin area	Lange Ganzzahl	38
Object type popup dropdown list	Lange Ganzzahl	12
Object type progress indicator	Lange Ganzzahl	27
Object type push button	Lange Ganzzahl	15
Object type radio button	Lange Ganzzahl	22
Object type radio button field	Lange Ganzzahl	5
Object type rectangle	Lange Ganzzahl	31
Object type rounded rectangle	Lange Ganzzahl	33
Object type ruler	Lange Ganzzahl	29
Object type splitter	Lange Ganzzahl	36
Object type static picture	Lange Ganzzahl	2
Object type static text	Lange Ganzzahl	1
Object type subform	Lange Ganzzahl	39
Object type tab control	Lange Ganzzahl	37
Object type text input	Lange Ganzzahl	3
Object type unknown	Lange Ganzzahl	0
Object type view pro area	Lange Ganzzahl	42
Object type web area	Lange Ganzzahl	40
Object type write pro area	Lange Ganzzahl	41

Beispiel

Ein Formular laden und eine Liste mit allen Objekten in der Listbox erhalten.

```
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 ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Variable
Funktionsergebnis	Lange Ganzzahl	⮌	Art der Ausrichtung

Beschreibung

Der Befehl **OBJECT Get vertical alignment** gibt einen Wert für die Art der vertikalen Ausrichtung des Objekts zurück, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Hinweis: Wenden Sie diese Funktion auf einen Satz Objekte an, wird nur der Wert der Ausrichtung für das letzte Objekt zurückgegeben.

Der zurückgegebene Wert entspricht einer der folgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)**:

Konstante	Typ	Wert
Align bottom	Lange Ganzzahl	4
Align center	Lange Ganzzahl	3
Align top	Lange Ganzzahl	2

Folgende Arten von Formularobjekten lassen sich vertikal ausrichten:

- Listboxen
- Spalten von Listboxen
- Kopf- bzw. Fußteile von Listboxen

OBJECT Get visible

OBJECT Get visible ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname(String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Feld oder Variable
Funktionsergebnis	Boolean	⮌	Wahr = Objekt(e) sichtbar; andernfalls Falsch

Beschreibung

Die Funktion **OBJECT Get visible** gibt Wahr zurück, wenn das Objekt bzw. die Objektgruppe, definiert in *Objekt*, das Attribut sichtbar hat, andernfalls Falsch.

Mit dem optionalen Parameter * geben Sie an, dass *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass Objekt ein Feld oder eine Variable ist. In diesem Fall übergeben Sie ein Feld oder eine Variablenreferenz (nur Objektfeld oder -variable) anstelle eines String.

OBJECT Is styled text

OBJECT Is styled text ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
Funktionsergebnis	Boolean	⮌	Wahr, wenn Objekt ein Text mit Mehrfachstil ist, sonst Falsch

Beschreibung

Die Funktion **OBJECT Is styled text** gibt Wahr zurück, wenn die Option "Mehrfachstil" für das Objekt, definiert über die Parameter Objekt und *, markiert ist.

Mit der Option "Mehrfachstil" können Sie Rich Text Bereiche mit individuellen Stilelementen verwenden. Weitere Informationen dazu finden Sie im Abschnitt **Text mit Stil (Rich Text)** des Handbuchs *4D Designmodus*.

Objekte mit Mehrfachstil lassen sich per Programmierung mit den Befehlen im Kapitel **Mehrfachstil Text** verwalten.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Hinweis: Bei Anwendung auf einen 4D Write Pro Bereich gibt die Funktion **Wahr** zurück.

Beispiel

Ein Formular enthält ein Feld, das mit zwei verschiedenen Objekten dargestellt wird; ein Objekt mit Mehrfachstil, das andere ohne. Sie können schreiben:

```
$Style:=OBJECT Is styled text(*;"Styled_text")
 // gibt Wahr zurück (Option "Mehrfachstil" ist markiert)

$Style:=OBJECT Is styled text(*;"Plain_text")
 //gibt Falsch zurück (Option "Mehrfachstil" ist nicht markiert)
```

OBJECT MOVE

OBJECT MOVE ({* ;} Objekt ; BewegenH ; BewegenV {; AnpassenH {; AnpassenV {; *}}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist Objektname (String) Ohne *: Objekt ist Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Feld oder Variable
BewegenH	Lange Ganzzahl	➡	Wert der horizontalen Verschiebung des Objekts (>0 = nach rechts, <0 = nach links)
BewegenV	Lange Ganzzahl	➡	Wert der vertikalen Verschiebung des Objekts (>0 = nach unten, <0 = nach oben)
AnpassenH	Lange Ganzzahl	➡	Wert der horizontalen Größe des Objekts
AnpassenV	Lange Ganzzahl	➡	Wert der vertikalen Größe des Objekts
*	Operator	➡	Mit Angabe = absolute Koordinaten Ohne Angabe = relative Koordinaten

Beschreibung

Der Befehl **OBJECT MOVE** verschiebt das/die Objekt/e im aktuellen Formular, definiert mit den Parametern * und *Objekt*. *BewegenH* verschiebt in Pixel-Schritten horizontal, *BewegenV* in Pixel-Schritten vertikal.
Bei Bedarf passen Sie das/die Objekt/e mit *AnpassenH* in Pixel-Schritten horizontal an, mit *AnpassenV* in Pixel-Schritten vertikal.
Die Richtung für Verschieben und Anpassen legen Sie mit Plus- und Minuswerten fest:

- Ist der Wert positiv, werden Objekte nach rechts verschoben bzw. nach rechts und nach unten angepasst.
- Ist der Wert negativ, werden Objekte nach links verschoben bzw. nach links und nach oben angepasst.

Übergeben Sie den ersten optionalen Parameter *, ist der Parameter *Objekt* ein Parametername (ein String). Übergeben Sie den ersten optionalen Parameter * nicht, ist der Parameter *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie keinen String, sondern ein Datenfeld oder eine Referenz auf eine Variable (nur Datenfeld oder Variable vom Typ Objekt).
Übergeben Sie in *Objekt* einen Objektnamen mit Jokerzeichen ("@"), um mehrere Objekte auszuwählen, werden alle betroffenen Objekte verschoben oder angepasst.
Hinweis: Sie können die Art der Interpretation für das Jokerzeichen ("@") festlegen, wenn es in einem String verwendet wird. Diese Option beeinflusst die Befehle zum Thema "Objekte". Weitere Informationen dazu finden Sie im Abschnitt **Jokerzeichen (@)** des Handbuchs *4D Designmodus*.
Die Werte *VerschiebenH*, *VerschiebenV*, *AnpassenH* und *AnpassenV* verändern die Koordinaten des Objekts standardmäßig in Bezug auf seine vorherige Position. Wollen Sie mit absoluten Parametern arbeiten, übergeben Sie den letzten optionalen Parameter *.
Dieser Befehl arbeitet nur in folgenden Kontexten:

- Eingabeformulare im Modus Dateneingabe,
- Formulare, die mit dem Befehl **DIALOG** angezeigt werden,
- Kopf- und Fußzeilen von Ausgabeformularen, die mit den Befehlen **MODIFY SELECTION** oder **DISPLAY SELECTION** angezeigt werden.
- Formularereignisse beim Drucken.

Beispiel 1

Folgende Anweisung verschiebt "Schaltfläche_1" um 10 Pixel nach rechts und um 20 Pixel nach oben und passt ihn auf 30 Pixel in der Breite und 40 Pixel in der Höhe an:

```
OBJECT MOVE(*;"Schaltfläche_1";10;-20;30;40)
```

Beispiel 2

Folgendes Statement verschiebt "Schaltfläche_1" auf die Koordinaten (10;20) (30;40):

```
OBJECT MOVE(*;"Schaltfläche_1";10;20;30;40;*)
```

OBJECT SET ACTION

OBJECT SET ACTION ({* ;} Objekt ; Aktion)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
Aktion	Text	➡	Aktion zum Zuweisen

Beschreibung

Der Befehl **OBJECT SET ACTION** ändert die Standardaktion für das bzw. die Objekte, definiert über die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Objekt Feld oder Variable).

Im Parameter *Aktion* übergeben Sie den Namen der Standardaktion für das Objekt oder eine Konstante unter dem Thema **Standardaktion**. Die Aktion kann optional auch Parameter haben.

Weitere Informationen dazu finden Sie im Abschnitt **Standardaktionen** des Handbuchs *4D Designmodus*.

Hinweis zur Kompatibilität: Bisherige Konstanten sind mit der Vorsilbe _o_ gekennzeichnet und ab 4D v16 R3 überholt. Sie werden zur Wahrung der Kompatibilität noch unterstützt.

Beispiel

Einer Schaltfläche die Standardaktion **Validate** zuordnen:

```
OBJECT SET ACTION(*;"bValidate";ak accept)
```

OBJECT SET AUTO SPELLCHECK

OBJECT SET AUTO SPELLCHECK ({* ;} Objekt ; autoSpellcheck)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Feld oder Variable
autoSpellcheck	Boolean	➡	Wahr= automatische Rechtschreibprüfung, Falsch= keine automatische Rechtschreibprüfung

Beschreibung

Der Befehl **OBJECT SET AUTO SPELLCHECK** setzt oder ändert dynamisch den Status der Option *autoSpellcheck* für das bzw. die Objekte, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess. Diese Option aktiviert oder deaktiviert die automatische Rechtschreibkontrolle bei der Dateneingabe in das Objekt (nur Objekte vom Typ Text).

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable oder ein Feld. In diesem Fall übergeben Sie eine Referenz anstelle eines Namens.

In *autoSpellcheck* übergeben Sie **Wahr**, um diese Funktion für *Objekt* zu aktivieren; **Falsch**, um sie zu deaktivieren.

OBJECT SET BORDER STYLE

OBJECT SET BORDER STYLE ({* ;} Objekt ; Rahmenstil)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Object name (if * is specified) or Field or variable (if * is omitted)
Rahmenstil	Lange Ganzzahl	➡	Rahmenstil

Beschreibung

Der Befehl **OBJECT SET BORDER STYLE** ändert den Rahmenstil des bzw. der Objekte, definiert über die Parameter *Objekt* und *.

Die Eigenschaft "Rahmenstil" ändert die Darstellung des Objektrahmens. Weitere Informationen dazu finden Sie im Abschnitt **Randstil** des Handbuchs *4D Designmodus*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Im Parameter *Rahmenstil* übergeben Sie den Wert des Rahmenstils für das Objekt. Sie können eine der folgenden Konstanten unter dem Thema "**Formularobjekte (Eigenschaften)**" übergeben:

Konstante	Typ	Wert	Kommentar
Border Dotted	Lange Ganzzahl	2	Objekte werden mit einer gepunkteten Linie von einem Punkt umrahmt
Border Double	Lange Ganzzahl	5	Objekte werden mit Doppellinie umrahmt, z.B. zwei kontinuierliche Linien von 1 Punkt mit 1 Pixel Abstand
Border None	Lange Ganzzahl	0	Objekte erscheinen ohne Rahmen
Border Plain	Lange Ganzzahl	1	Objekte werden mit einer kontinuierlichen Linie von 1 Punkt umrahmt
Border Raised	Lange Ganzzahl	3	Objekte werden mit einem erhobenen 3D Effekt umrahmt
Border Sunken	Lange Ganzzahl	4	Objekte werden mit einem vertieften 3D Effekt umrahmt
Border System	Lange Ganzzahl	6	Die Rahmenlinie wird gemäß der grafischen Spezifikation des Systems gezeichnet

OBJECT SET COLOR

OBJECT SET COLOR ({* ;} Objekt ; Farbe {; AltFarbe})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne *: Objekt ist Feld oder Variable
Objekt	Feld, Variable	➡	Mit *: Objektname, ohne *: Feld oder Variable
Farbe	Lange Ganzzahl	➡	Neue Farbe für Objekt
AltFarbe	Lange Ganzzahl	➡	Wechselnde Farbe für Listbox

Beschreibung

Der Befehl **OBJECT SET COLOR** setzt die Farben für Vordergrund und Hintergrund des Formularobjekts *Objekt*. Ist *Objekt* eine Listbox, können Sie einen weiteren Parameter für wechselne Vorder- und Hintergrundfarbe für Zeilen mit gerader Nummer verwenden.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den optionalen Parameter * geben Sie in *Objekt* ein Datenfeld oder eine Variable an. In diesem Fall geben Sie anstatt eines Strings eine Referenz auf das Datenfeld oder die Variable an (nur Datenfeld- oder Variablenobjekte). Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

Der Parameter *Farbe* gibt die Farben für Vordergund und Hintergrund an. Die Farbe wird folgendermaßen berechnet:

Farbe:=–(Vordergrund +(256 * Hintergrund))

Vordergrund und Hintergrund sind Farbnummern von 0 bis 255 der Farbpalette. *Farbe* ist immer eine negative Zahl. Ist zum Beispiel die Vordergrundfarbe 20 und die Hintergrundfarbe 10, ist die Farbe: –(20 + (256 * 10)) oder –2580.

Mit *AltFarbe* setzen Sie eine wechselnde Hintergrundfarbe für die Zeilen mit gerader Nummer einer Listbox oder Spalte einer Listbox. In *AltFarbe* geben Sie nur den Teil der Formel für Hintergrundfarbe an, wie z.B. **AltFarbe:=–(256 * Hintergrund)**. Ist dieser Parameter angegeben, wird der Parameter *Farbe* nur für die Zeilen mit ungerader Nummer verwendet. Wechselnde Farben machen Arrays leichter lesbar. Definiert *Objekt* die Listbox, gilt die wechselnde Farbe für das gesamte Objekt, definiert *Objekt* eine Spalte, gilt sie nur für die angegebene Spalte.

Hinweis: Die Farbpalette sehen Sie in der Eigenschaftenliste des Formulareditors.

4D bietet für häufig gebrauchte Farben folgende vordefinierten Konstanten unter dem Thema **Farben**:

Konstante	Typ	Wert
Black	Lange Ganzzahl	15
Blue	Lange Ganzzahl	6
Brown	Lange Ganzzahl	13
Dark blue	Lange Ganzzahl	5
Dark brown	Lange Ganzzahl	10
Dark green	Lange Ganzzahl	9
Dark grey	Lange Ganzzahl	11
Green	Lange Ganzzahl	8
Grey	Lange Ganzzahl	14
Light blue	Lange Ganzzahl	7
Light grey	Lange Ganzzahl	12
Orange	Lange Ganzzahl	2
Purple	Lange Ganzzahl	4
Red	Lange Ganzzahl	3
White	Lange Ganzzahl	0
Yellow	Lange Ganzzahl	1

Hinweis: OBJECT SET COLOR arbeitet mit indizierten Farben, die in der Standardfarbtabelle von 4D enthalten sind. **OBJECT SET RGB COLORS** erlaubt jede beliebige RGB Farbe. Um für ein Objekt die automatischen Farben wiederherzustellen, verwenden Sie **OBJECT SET RGB COLORS** mit den Konstanten Default foreground color und Default background color.

Beispiel 1

Folgendes Beispiel legt die Farbe des Textbereichs fest, die im Formulareditor angezeigt wird:

Nach Ausführen der Anweisung:

```
OBJECT SET COLOR(*;"Meintext";-(Yellow+(256*Red)))
```

... erscheint der Bereich im Anwendungsmodus folgendermaßen:

Beispiel 2

Eine wechselnde Hintergrundfarbe für eine Spalte in der Listbox setzen:

```
OBJECT SET COLOR(*;"countryCol";-(Dark blue+(256*Red));-(256*Orange))
```

OBJECT SET CONTEXT MENU

OBJECT SET CONTEXT MENU ({* ;} Objekt ; Kontextmenü)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Kontextmenü	Boolean	➡	Wahr = Kontextmenü aktivieren Falsch = Kontextmenü deaktivieren

Beschreibung

Der Befehl **OBJECT SET CONTEXT MENU** aktiviert oder deaktiviert für den aktuellen Prozess die Zuweisung eines standardmäßigen Kontextmenüs für das bzw. die Objekte, definiert über die Parameter *Objekt* und *.

Die Option "Kontextmenü" ist für Eingabebereiche vom Typ Text, Web Areas und Bilder verfügbar. Damit können Sie diesen Objekten, abhängig vom Typ, ein Menü mit Standardaktionen zuweisen, z.B. Kopieren/Einfügen für Textobjekte. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Übergeben Sie **Wahr** in *Kontextmenü*, um das Kontextmenü zu aktivieren; **Falsch**, um es zu deaktivieren.

OBJECT SET COORDINATES

OBJECT SET COORDINATES ({* ;} Objekt ; links ; oben {; rechts ; unten})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable
Objekt	Lange Ganzzahl	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
links	Lange Ganzzahl	➡	Linke Koordinate des Objekts in Pixel
oben	Lange Ganzzahl	➡	Obere Koordinate des Objekts in Pixel
rechts	Lange Ganzzahl	➡	Rechte Koordinate des Objekts in Pixel
unten	Lange Ganzzahl	➡	Untere Koordinate des Objekts in Pixel

Beschreibung

Der Befehl **OBJECT SET COORDINATES** ändert für den aktuellen Prozess die Position und optional die Größe des bzw. der Objekte, definiert über die Parameter *Objekt* und *.

Hinweis: Dieser Befehl führt dasselbe aus wie der Befehl **OBJECT MOVE** mit übergebenem Parameter *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

In den Parametern *links* und *oben* übergeben Sie die neuen absoluten Koordinaten von *Objekt* im Formular. Sie müssen in Pixel angegeben werden, ausgehend von der linken oberen Ecke im Formular.

In den Parametern *rechts* und *unten* können Sie auch absolute Koordinatenwerte für die untere rechte Ecke des Objekts angeben. Entspricht diese Ecke nach Anwendung der Parameter *links* und *oben* nicht der Ecke des Objekts, wird *Objekt* entsprechend angepasst.

Hinweis: Wollen Sie ein Objekt in bezug auf seine Ausgangsposition bewegen, empfehlen wir, den Befehl **OBJECT MOVE** zu verwenden.

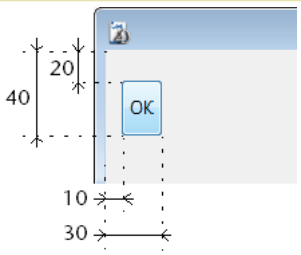
Dieser Befehl funktioniert nur in folgendem Kontext:

- Eingabeformulare im Eingabemodus
- Formulare, die über den Befehl **DIALOG** angezeigt werden,
- Kopfteile und Fußteile von Ausgabeformularen, die über die Befehle **MODIFY SELECTION** oder **DISPLAY SELECTION** angezeigt werden
- Formulare zum Drucken

Beispiel

Folgende Anweisung setzt das Objekt "button_1" mit den Koordinaten (10,20) (30,40):

```
OBJECT SET COORDINATES(*;"button_1";10;20;30;40)
```



OBJECT SET CORNER RADIUS

OBJECT SET CORNER RADIUS ({* ;} Objekt ; Radius)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
Radius	Lange Ganzzahl	➡	Neuer Radius für die gerundeten Ecken (in Pixel)

Beschreibung

Der Befehl **OBJECT SET CORNER RADIUS** ändert den Radius der gerundeten Ecken im Objekt "Rundes Viereck", definiert in *Objekt*. Der neue Radius gilt nur für den Prozess; er wird nicht im Formular gesichert.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

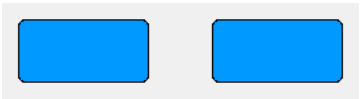
Hinweis: Da dieser Befehl in aktuellen 4D Release nur auf abgerundete Vierecke, d.h. statische Objekte, angewandt wird, müssen Sie immer den Parameter * übergeben und die Syntax Objektname verwenden.

In *Radius* übergeben Sie den Wert (in Pixel) des neuen Radius für die abgerundeten Ecken des Objekts. Der Wert ist standardmäßig 5 Pixel.

Hinweis: Dieser Wert lässt sich auch über die Eigenschaftenliste auf Formularebene setzen (siehe **Neue Eigenschaft Eckradius für abgerundete Vierecke**).

Beispiel

Ein Formular enthält zwei Vierecke mit Namen "Rect1" und "Rect2":



Sie können die Rundung der Ecken verändern und z.B. folgenden Code ausführen:

```
OBJECT SET CORNER RADIUS(*;"Rect@";20)
```

OBJECT SET DATA SOURCE

OBJECT SET DATA SOURCE ({* ;} Objekt ; Datenquelle)

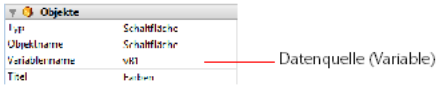
Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Datenquelle	Zeiger	➡	Zeiger auf die neue Datenquelle für Objekt

Beschreibung

Der Befehl **OBJECT SET DATA SOURCE** ändert die Datenquelle des bzw. der Objekte, definiert über die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Die Datenquelle ist ein Feld oder eine Variable, deren Wert beim Ausführen des Formulars vom Objekt dargestellt wird.
Im Designmodus wird die Datenquelle in der Eigenschaftenliste über die Einträge Quelle und Quellfeld oder Variablenname definiert:



Dieser Befehl kann alle Datenquellen des Formulars verändern, mit Ausnahme von Listboxen (siehe unten). Der Entwickler muss bei Änderungen selbst für die Wahrung der Konsistenz sorgen.

Bei Listboxen ist folgendes zu beachten:

- Das Ändern der Datenquelle ist abhängig vom Typ der Listbox: Zum Beispiel lässt sich ein Feld nicht als Datenquelle für eine Spalte in einer Listbox vom Typ Array verwenden
- Bei Listboxen vom Typ Auswahl lässt sich die Datenquelle der Listbox selbst weder ändern noch lesen, denn dies ist eine interne Referenz und keine Datenquelle
- Dieser Befehl wird hauptsächlich für Listboxen vom Typ Array verwendet. Für Listboxen vom Typ Auswahl können Sie stattdessen **LISTBOX SET COLUMN FORMULA** verwenden.

Wird dieser Befehl auf eine Datenquelle angewandt, die nicht änderbar ist, führt er nichts aus.

Beispiel

Die Datenquelle für einen Eingabebereich ändern:

```
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 ({* ;} Objekt ; draggable ; autoDrag ; droppable ; autoDrop)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname(String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
draggable	Boolean	➡	draggable = Wahr; sonst Falsch
autoDrag	Boolean	➡	autoDrag = Wahr; sonst Falsch
droppable	Boolean	➡	droppable = Wahr; sonst Falsch
autoDrop	Boolean	➡	autoDrop = Wahr; sonst Falsch

Beschreibung

Der Befehl **OBJECT SET DRAG AND DROP OPTIONS** setzt oder ändert dynamisch die Drag-and-Drop Optionen für das bzw. die Objekte, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

In jedem Parameter übergeben Sie einen Wert, der angibt, ob diese Option aktiviert oder deaktiviert ist.

- *draggable* = Wahr: Objekt lässt sich im programmierten Modus ziehen
- *autoDrag* = Wahr (nur bei Textfeldern und Variablen, Comboboxen und Listboxen): Objekt lässt sich im automatischen Modus ziehen
- *droppable* = Wahr: Objekt akzeptiert Drop im programmierten Modus
- *autoDrop* = Wahr (nur bei Bildfeldern und Variablen, Text, Comboboxen und Listboxen): Objekt akzeptiert Drop im automatischen Modus

Beispiel

Einen Textbereich auf automatisches Drag-and-Drop setzen:

```
OBJECT SET DRAG AND DROP OPTIONS(*;"Kommentare";False;True;False;True)
```

OBJECT SET ENABLED

OBJECT SET ENABLED ({* ;} Objekt ; aktiv)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Variable oder Feld
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
aktiv	Boolean	➡	Wahr = Objekt(e) aktiviert; sonst Falsch

Beschreibung

Der Befehl **OBJECT SET ENABLED** setzt das Objekt bzw. die Objektgruppe, angegeben durch *Objekt* im aktuellen Formular, auf aktiv oder inaktiv.
Ein aktiviertes Objekt reagiert auf Mausklicks und auf Tastenkürzel.

Mit dem optionalen Parameter * ist *Objekt* ein Objektname (String), ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz (nur Objektvariable) anstelle eines String.

Dieser Befehl lässt sich auf folgende Objekttypen anwenden:

- Schaltfläche, Standardschaltfläche, 3D Schaltfläche, Unsichtbare Schaltfläche, markierte Schaltfläche
- Optionsfeld, 3D Optionsfeld, Bildschaltfläche
- Kontrollkästchen, 3D Kontrollkästchen
- PopUp-Menü, DropDown-Liste, Combo Box, Menü/DropDown-Liste
- Ablaufbalken, Lineal

Hinweis: Dieser Befehl hat keine Auswirkung auf ein Objekt, dem eine Standardaktion zugewiesen wurde, außer im Falle der Aktionen **Bestätigen** und **Abbrechen**. 4D schaut bei Bedarf nach dem Ändern des Status dieses Objekts.

OBJECT SET ENTERABLE

OBJECT SET ENTERABLE ({* ;} Objekt ; Eingebbar)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Eingebbar	Boolean	➡	Wahr für eingebbar; Falsch für nicht eingebbar

Beschreibung

Der Befehl **OBJECT SET ENTERABLE** macht das Formularobjekt *Objekt* eingebbar oder nicht eingebbar.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den optionalen Parameter * geben Sie in *Objekt* eine Tabelle, ein Datenfeld oder eine Variable an. In diesem Fall geben Sie anstatt eines Strings eine Referenz auf die Tabelle, das Datenfeld oder die Variable an (nur Datenfeld- oder Variablenobjekte). Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

Dieser Befehl hat dieselbe Wirkung wie das Auswählen des Attributs Eingebbar für ein Datenfeld oder eine Variable in der Eigenschaftenliste des Formulareditors. In Unterformularen funktioniert dieser Befehl nur, wenn er in der Formularmethode des Unterformulars liegt.

Mit **OBJECT SET ENTERABLE** können Sie per Programmierung die Option "Eingabe in Liste" für Unterformulare und Listenformulare aktivieren, die über die Befehle **MODIFY SELECTION** und **DISPLAY SELECTION** angezeigt werden.

- Für Unterformulare übergeben Sie in *Objekt* entweder den Namen der Tabelle des Unterformulars oder den Namen des Unterformularobjekts, z.B. **OBJECT SET ENTERABLE***(*;"Unterformular";Wahr)*.
- Für Listenformulare müssen Sie in *Objekt* den Namen der Tabelle des Formulars übergeben; z.B. **OBJECT SET ENTERABLE**([MeineTabelle];Wahr).

Ist *Objekt* eingebbar (WAHR), kann der Benutzer den Cursor in den Bereich setzen und Daten eingeben. Ist *Objekt* nicht eingebbar (FALSCH), kann der Benutzer den Cursor nicht in den Bereich setzen und keine Daten eingeben.

Bitte beachten Sie, dass sich auch bei nicht eingebbaren Objekten die Werte weiterhin über Programmierung ändern lassen.

Hinweis: Wollen Sie eine bestimmte Zelle der Listbox nicht eingebbar machen, übergeben Sie im Ereignis On Before Data Entry in $0 den Wert -1, siehe **Eingabe verwalten**.

Beispiel 1

Folgendes Beispiel setzt ein Datenfeld abhängig vom Frachtgewicht. Ist das Gewicht kleiner oder gleich 1 kg, wird dem Feld Versandart automatisch der Wert Post zugewiesen und dieses anschließend auf nicht eingebbar gesetzt.

```
If([Shipments]Weight<=1)
   [Shipments]Shipper:="Post"
   OBJECT SET ENTERABLE([Shipments]Shipper;False)
Else
   OBJECT SET ENTERABLE([Shipments]Shipper;True)
End if
```

Beispiel 2

Nachfolgende Objektmethode setzt ein Kontrollkästchen in den Kopfteil einer Liste, um *Eingebbar* im Listenmodus zu verwalten:

```
C_BOOLEAN(bEnterable)
OBJECT SET ENTERABLE([Table1];bEnterable)
```

OBJECT SET EVENTS

OBJECT SET EVENTS ({* ;} Objekt ; arrEreignisse ; Modus)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname oder "" für Formular (mit *) oder Feld bzw. Variable (ohne *)
arrEreignisse	Array Lange Ganzzahl	➡	Array der Ereignisse zum Setzen
Modus	Lange Ganzzahl	➡	Aktivierungsmodus für Ereignisse, definiert in arrEreignisse

Beschreibung

Der Befehl **OBJECT SET EVENTS** ändert für den aktuellen Prozess die Konfiguration von Formularereignissen des Formulars oder Objekts, definiert über die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).
Um die Einstellung der Ereignisse für das Formular selbst zu definieren, übergeben Sie in *Objekt* den optionalen Parameter * und einen leeren String "": So bestimmen Sie das aktuelle Formular.

Hinweis: Wollen Sie die Ereignisse eines Unterformulars zu einer Tabelle ändern, können Sie die Syntax verwenden, die auf dem Objektnamen basiert.

Im Parameter *arrEreignisse* übergeben Sie ein Array Lange Ganzzahl mit der Liste der vordefinierten oder eigenen Formularereignisse, die Sie ändern wollen. Mit dem Parameter *Modus* können Sie Aktivieren oder Deaktivieren der Ereignisse angeben.
Um ein vordefiniertes Ereignis zum Ändern zu bestimmen, können Sie in jedem Element des Array *arrEreignisse* eine der folgenden Konstanten unter dem Thema **Formularereignisse**:

Konstante	Typ	Wert	Kommentar
On Activate	Lange Ganzzahl	11	Das Formularfenster wird zum vordersten Fenster.
On After Edit	Lange Ganzzahl	45	Der Inhalt des eingebbaren Objekts mit Fokus wurde gerade geändert.
On After Keystroke	Lange Ganzzahl	28	Ein Zeichen wird gerade in das Objekt mit Fokus eingegeben. **Get edited text** gibt den Text im Objekt inkl. diesem Zeichen zurück
On After Sort	Lange Ganzzahl	30	(*nur Listbox*) In einer Spalte der Listbox wurde gerade eine Standard-Sortierung ausgeführt.
On Arrow Click	Lange Ganzzahl	38	(*nur 3D buttons*) Der Pfeilbereich einer 3D Schaltfläche ist angeklickt
On Before Data Entry	Lange Ganzzahl	41	(*nur Listbox*) Eine Zelle der Listbox wechselt gerade in den Editiermodus
On Before Keystroke	Lange Ganzzahl	17	Ein Zeichen wird gerade in das Objekt mit Fokus eingegeben. **Get edited text** gibt den Text im Objekt ohne dieses Zeichen zurück.
On Begin Drag Over	Lange Ganzzahl	46	Ein Objekt wird gerade bewegt (Drag)
On Begin URL Loading	Lange Ganzzahl	47	(*nur Web Areas*) Eine neue URL wird in den Webbereich geladen.
On bound variable change	Lange Ganzzahl	54	Die dem Unterformular zugewiesene Variable wird geändert.
On Clicked	Lange Ganzzahl	4	Das Objekt wurde angeklickt.
On Close Box	Lange Ganzzahl	22	Die Schließbox des Fensters wurde angeklickt.
On Close Detail	Lange Ganzzahl	26	Sie haben das Eingabeformular verlassen und gehen zurück zum Ausgabeformular.
On Collapse	Lange Ganzzahl	44	(*hierarchische Listen und hierarchische Listboxen*) Ein Element der hierarchischen Liste bzw. Listbox wurde über Mausklick oder Tastenanschlag zugeklappt.
On Column Moved	Lange Ganzzahl	32	(*nur Listbox*) Der Benutzer hat eine Spalte der Listbox per Drag and Drop bewegt.
On Column Resize	Lange Ganzzahl	33	(*nur Listbox*) Der Benutzer hat die Breite einer Spalte der Listbox mit der Maus geändert.
On Data Change	Lange Ganzzahl	20	Daten im Objekt wurden geändert.
On Deactivate	Lange Ganzzahl	12	Das Formularfenster ist nicht mehr das vorderste Fenster.
On Delete Action	Lange Ganzzahl	58	(*nur hierarchische Listen und Listboxen*) Ein Benutzer möchte ein Element löschen
On Display Detail	Lange Ganzzahl	8	Ein Datensatz wird gleich in einer Liste bzw. eine Zeile in einer Listbox angezeigt.
On Double Clicked	Lange Ganzzahl	13	Auf ein Objekt wurde ein Doppelklick ausgeführt.
On Drag Over	Lange Ganzzahl	21	Daten werden in ein Objekt gezogen.
On Drop	Lange Ganzzahl	16	Daten werden in ein Objekt gezogen.
On End URL Loading	Lange Ganzzahl	49	(*nur Web Areas*) Alle Ressourcen des URL wurden geladen.
On Expand	Lange Ganzzahl	43	(*hierarchische Listen und hierarchische Listboxen*) Ein Element der hierarchischen Liste bzw. Listbox wurde per Mausklick oder Tastenanschlag aufgeklappt.
On Footer Click	Lange Ganzzahl	57	(*nur Listboxen*) Der Fußteil einer Listbox oder einer Spalte der Listbox ist angeklickt
On Getting Focus	Lange Ganzzahl	15	Ein Formularobjekt erhält den Fokus.
On Header	Lange Ganzzahl	5	Der Kopfteil des Formulars wird gleich gedruckt oder angezeigt.
On Header Click	Lange Ganzzahl	42	(*nur Listbox*) Ein Spaltentitel der Listbox wird angeklickt.
On Load Record	Lange Ganzzahl	40	Bei der Eingabe in die Liste, wird ein Datensatz während der Änderung geladen (Der Benutzer klickt auf eine Zeile im Datensatz und ein Feld wechselt in den Editiermodus).
On Long Click	Lange Ganzzahl	39	(*nur 3D buttons*) Eine 3D Schaltfläche wird angeklickt und die Maustaste bleibt für eine gewisse Zeit gedrückt.
On Losing Focus	Lange Ganzzahl	14	Ein Formularobjekt verliert den Fokus.
On Mac toolbar button	Lange Ganzzahl	55	Der Benutzer klickt auf die Schaltfläche Toolbar Management unter Mac OS.
On Menu Selected	Lange Ganzzahl	18	Ein Menüeintrag wurde ausgewählt.
On Mouse Enter	Lange Ganzzahl	35	Der Mauszeiger geht in den grafischen Bereich eines Objekts.

Konstante	Typ	Wert	Kommentar
On Mouse Leave	Lange Ganzzahl	36	Der Mauszeiger verlässt den grafischen Bereich eines Objekts.
On Mouse Move	Lange Ganzzahl	37	Der Mauszeiger bewegt sich (mindestens 1 Pixel) ODER eine Modifier-Taste (Shift, Alt, Shift Lock) wurde gedrückt. Wurde das Ereignis nur für ein Objekt markiert, wird es nur generiert, wenn der Cursor im grafischen Bereich eines Objekts liegt.
On Open Detail	Lange Ganzzahl	25	Ein dem Ausgabeformular oder der Listbox zugeordnetes Eingabeformular wird gerade geöffnet.
On Open External Link	Lange Ganzzahl	52	(*nur Web Areas*) Im Browser wurde eine externe URL geöffnet.
On Outside Call	Lange Ganzzahl	10	Das Formular hat einen Aufruf **POST OUTSIDE CALL** erhalten.
On Picture Scroll	Lange Ganzzahl	59	Der Benutzer scrollt den Inhalt eines Feldes vom Typ Bild oder Variable mit der Maus oder Tastatur.
On Plug in Area	Lange Ganzzahl	19	Ein externes Objekt hat angefragt, seine Objektmethode auszuführen.
On Printing Break	Lange Ganzzahl	6	Ein Umbruchbereich im Formular wird gleich gedruckt.
On Printing Detail	Lange Ganzzahl	23	Der Detailbereich des Formulars wird gleich gedruckt.
On Printing Footer	Lange Ganzzahl	7	Der Fußteil des Formulars wird gleich gedruckt.
On Resize	Lange Ganzzahl	29	Das Formularfenster wird angepasst.
On Row Moved	Lange Ganzzahl	34	(*nur Listbox*) Der Benutzer hat eine Zeile der Listbox per Drag-and-Drop bewegt.
On Selection Change	Lange Ganzzahl	31	• *Listbox*: Die aktuelle Auswahl der Zeilen oder Spalten wurde geändert. • *Datensätze in Liste*: Der aktuelle Datensatz oder die aktuelle Auswahl der Zeilen in einem Listen- bzw. Unterformular wurde geändert. • *Hierarchische Liste*: Die Auswahl in der Liste wurde nach einem Mausklick oder Tastenanschlag geändert. • *Eingebbares Feld oder Variable*: Die Textauswahl oder Position des Cursors im Bereich wurde nach einem Mausklick oder Tastenanschlag geändert.
On Timer	Lange Ganzzahl	27	Die Anzahl der durch **SET TIMER** definierten Ticks wurde überschritten.
On Unload	Lange Ganzzahl	24	Das Formular wird gerade verlassen oder erneuert.
On URL Filtering	Lange Ganzzahl	51	(*nur Web Areas*) Der Web Bereich hat eine URL geblockt.
On URL Loading Error	Lange Ganzzahl	50	(*nur Web Areas*) Beim Laden der URL ist ein Fehler aufgetreten.
On URL Resource Loading	Lange Ganzzahl	48	(*nur Web Areas*) Eine neue Ressource wird in den Web Bereich geladen.
On Validate	Lange Ganzzahl	3	Die Eingabe in den Datensatz wurde bestätigt.
On Window Opening Denied	Lange Ganzzahl	53	(*nur Web Areas*) Ein PopUp-Fenster wurde blockiert.

Bitte beachten Sie, dass das Ereignis On Load in dieser Liste nicht enthalten ist: Dieses Ereignis lässt sich nicht definieren, da es bereits während der Ausführung des Befehls generiert wurde.

In *arrEreignisse* können Sie auch einen Wert für ein eigenes Ereignis übergeben. Wir empfehlen dafür, negative Werte zu verwenden. Weitere Informationen dazu finden Sie unter dem Befehl **CALL SUBFORM CONTAINER**.

Im Parameter *Modus* setzen Sie die generelle Handhabung für Array Elemente. Dazu können Sie eine der folgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)** verwenden:

Konstante	Typ	Wert	Kommentar
Disable events others unchanged	Lange Ganzzahl	2	Alle im *arrEreignisse* aufgeführten Ereignisse werden deaktiviert; der Status anderer Ereignisse bleibt unverändert
Enable events disable others	Lange Ganzzahl	0	Alle im *arrEreignisse* aufgeführten Ereignisse werden aktiviert; alle anderen werden deaktiviert
Enable events others unchanged	Lange Ganzzahl	1	Alle im *arrEreignisse* aufgeführten Ereignisse werden aktiviert; der Status anderer Ereignisse bleibt unverändert

Der Befehl **OBJECT SET EVENTS** kann u.U. Ereignisse aktivieren, die Objekt (abhängig vom Typ) nicht unterstützt. In diesem Fall werden die Ereignisse einfach ignoriert.

Wird ein Objekt nach Aufrufen von **OBJECT SET EVENTS** dupliziert, wird auch die daraus resultierende Konfiguration *aktiviert/deaktiviert* dupliziert.

Beispiel 1

Für einen Satz Listboxobjekte drei Formularereignisse aktivieren und die anderen deaktivieren:

```
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)
 // aktiviert 3 Ereignisse und deaktiviert alle anderen
```

Beispiel 2

Für einen Satz Listboxobjekte drei Formularereignisse deaktivieren, ohne die anderen zu verändern:

```
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)
 // deaktiviert nur diese 3 Ereignisse
```

Beispiel 3

Ein Formularereignis für ein Objekt aktivieren, ohne die anderen zu verändern:

```
ARRAY LONGINT($MyEventsOnLB;1)
$MyEventsOnLB {1}:=On Column Moved
OBJECT SET EVENTS(*;"Col1";$MyEventsOnLB;Enable events others unchanged)
 // aktiviert nur dieses Ereignis
```

Beispiel 4

Alle Ereignisse des Formulars deaktivieren:

```
ARRAY LONGINT($MyFormEvents;0)
OBJECT SET EVENTS(*;"";$MyFormEvents;Enable events disable others)
 // deaktiviert alle Ereignisse
```

Beispiel 5

Deaktiviert ein einzelnes Ereignis des Formulars, ohne die anderen zu verändern:

```
ARRAY LONGINT($MyFormEvents;1)
$MyFormEvents{1}:=On Timer
OBJECT SET EVENTS(*;"";$MyFormEvents;Disable events others unchanged)
 // deaktiviert nur dieses Ereignis
```

OBJECT SET FILTER

OBJECT SET FILTER ({* ;} Objekt ; Filter)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Filter	String	➡	Neuer Filter für eingebbaren Bereich

Beschreibung

Der Befehl **OBJECT SET FILTER** ersetzt den Eingabefilter für *Objekt* durch *Filter*.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den optionalen Parameter * geben Sie in *Objekt* ein Datenfeld oder eine Variable an. In diesem Fall geben Sie anstatt eines Strings eine Referenz auf das Datenfeld oder die Variable an (nur Datenfeld- oder Variablenobjekte). Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

Sie können **OBJECT SET FILTER** in Eingabe- und Ausgabeformularen einsetzen und auf Datenfelder und eingebbare Variablen anwenden, die in der Designumgebung einen Eingabefilter zulassen.

Übergeben Sie als Filter einen leeren Text, erhält Objekt keinen Zeichenfilter.

Hinweise:

- Dieser Befehl lässt sich nicht auf Datenfelder in Unterformularen anwenden.
- Wollen Sie in *Filter* Anzeigeformate verwenden, die Sie im Dialogfenster Datenbank-Eigenschaften vordefiniert haben, setzen Sie vor den Formatnamen einen senkrechten Strich (|).

Beispiel 1

Folgendes Beispiel legt den Eingabefilter für das Datenfeld Postleitzahl fest. Ist die Adresse in Deutschland, wird der Filter auf Nur Ziffern gesetzt. Sonst ist jede Eingabe erlaubt:

```
If([Companies]Land="D") ` Setze Filter auf Nur Ziffern
   OBJECT SET FILTER([Companies]PLZ;"&9#####")
Else ` Filter soll Werte vom Typ alpha und numerisch anerkennen und alpha-Werte groß schreiben
   OBJECT SET FILTER([Companies]PLZ;"~@")
End if
```

Beispiel 2

Folgendes Beispiel erlaubt an zwei Stellen des Datenfeldes Feld nur die Buchstaben “a,” “b,” “c,” oder “g”:

```
OBJECT SET FILTER([Table]Field ;"&"+Char(Double quote)+"a;b;c;g"+Char(Double quote)+"##")
```

Hinweis: Dieses Beispiel setzt den Eingabefilter auf *"a;b;c;g"##*.

OBJECT SET FOCUS RECTANGLE INVISIBLE

OBJECT SET FOCUS RECTANGLE INVISIBLE ({* ;} Objekt ; unsichtbar)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
unsichtbar	Boolean	➡	Wahr= Fokusrahmen ausgeblendet, Falsch= Fokusrahmen angezeigt

Beschreibung

Der Befehl **OBJECT SET FOCUS RECTANGLE INVISIBLE** setzt oder ändert dynamisch die Option sichtbar des Fokusrahmens für das bzw. die Objekte, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess. Diese Einstellung entspricht der Eigenschaft *Fokusrahmen ausblenden*, die in der Eigenschaftenliste des Designmodus für eingebbare Objekte verfügbar ist.

Hinweis: Sie können diese Option nur unter Mac OS verwenden. Auf Windows hat sie keine Auswirkung.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Übergeben Sie **Wahr** im Parameter *unsichtbar*, um den Fokusrahmen auszublenden; **Falsch**, um ihn sichtbar zu lassen.

OBJECT SET FONT

OBJECT SET FONT ({* ;} Objekt ; Schrift)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Schrift	String	➡	Name der Schrift

Beschreibung

Der Befehl **OBJECT SET FONT** zeigt das *Objekt* mit dem in *Schrift* definierten Schriftnamen. Der Parameter *Schrift* muss einen gültigen Schriftnamen enthalten.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den optionalen Parameter * geben Sie in *Objekt* ein Datenfeld oder eine Variable an. In diesem Fall geben Sie anstatt eines Strings eine Referenz auf das Datenfeld oder die Variable an (nur Datenfeld- oder Variablenobjekte).

Beispiel 1

Folgendes Beispiel legt die Schrift für die Schaltfläche *bOK* fest:

```
OBJECT SET FONT(bOK;"Arial")
```

Beispiel 2

Folgendes Beispiel legt die Schrift für alle Formularobjekte fest, die "info" enthalten:

```
OBJECT SET FONT(*;"@info@";"Times")
```

Beispiel 3

Folgendes Beispiel verwendet die Spezialschrift *%password* zur Eingabe und Anzeige in Feldern vom Typ "Kennwort". Übergeben Sie im Parameter *Schrift %password,* gilt folgendes:

- Jedes im Objekt eingegebene Zeichen erscheint mit dem gleichen Symbol
- Die Aktionen "Kopieren" und "Ausschneiden" im Objekt sind deaktiviert

Hinweis: Sie können die Option *%password* für Objekte vom Typ Feld, Variable und Combo-Box verwenden.

```
OBJECT SET FONT([Users]Password;"%password")
```

OBJECT SET FONT SIZE

OBJECT SET FONT SIZE ({* ;} Objekt ; Größe)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Größe	Lange Ganzzahl	➡	Größe in Punkten

Beschreibung

Der Befehl **OBJECT SET FONT SIZE** ändert die Schriftgröße des Formularobjekts *Objekt* auf *NeueGröße* am Bildschirm oder beim Ausdruck.
Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den optionalen Parameter * geben Sie in *Objekt* ein Datenfeld oder eine Variable an. In diesem Fall geben Sie anstatt eines Strings eine Referenz auf das Datenfeld oder die Variable an (nur Datenfeld- oder Variablenobjekte). Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

NeueGröße kann eine Ganzzahl zwischen 1 und 255 sein. Existiert die exakte Schriftgröße nicht, werden die Zeichen entsprechend skaliert. Reicht der Bereich des Objekts im Formular für die neue Schriftgröße nicht aus, wird der Text abgeschnitten oder im Extremfall gar nicht angezeigt.

Beispiel 1

Folgendes Beispiel legt die Schriftgröße für die Variable *vtInfo* fest:

```
OBJECT SET FONT SIZE(vtInfo;14)
```

Beispiel 2

Folgendes Beispiel legt die Schriftgröße für alle Formularobjekte fest, die mit "hl" beginnen:

```
OBJECT SET FONT SIZE(*;"hl@";14)
```

OBJECT SET FONT STYLE

OBJECT SET FONT STYLE ({* ;} Objekt ; NeuerStil)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
NeuerStil	Lange Ganzzahl	➡	Schriftstil

Beschreibung

Der Befehl **OBJECT SET FONT STYLE** ändert den Schriftstil der Formularobjekte *Objekt* in *NeuerStil* um. Die für *NeuerStil* möglichen Werte sind unten aufgeführt. Kombinationen verschiedener Stile lassen sich durch Addition der einzelnen Stilwerte zusammenstellen.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den optionalen Parameter * geben Sie in *Objekt* ein Datenfeld oder eine Variable an. In diesem Fall übergeben Sie anstatt eines Strings eine Referenz auf das Datenfeld oder die Variable (nur Datenfeld- oder Variablenobjekte). Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

4D bietet folgende vordefinierte Konstanten:

Konstante	Typ	Wert
Bold	Lange Ganzzahl	1
Italic	Lange Ganzzahl	2
Plain	Lange Ganzzahl	0
Underline	Lange Ganzzahl	4

Beispiel 1

Dieses Beispiel legt für die Schaltfläche *bAddNew* den Schriftstil Fett Kursiv fest:

```
OBJECT SET FONT STYLE(bAddNew;Bold+Italic)
```

Beispiel 2

Dieses Beispiel legt für Formularobjekte, die mit "vt" beginnen den Schriftstil Fett fest:

```
OBJECT SET FONT STYLE(*;"vt@";Plain)
```

OBJECT SET FORMAT

OBJECT SET FORMAT ({* ;} Objekt ; Format)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Format	String	➡	Neues Anzeigeformat des Objekts

Beschreibung

Der Befehl **OBJECT SET FORMAT** ersetzt das Anzeigeformat von *Objekt* durch das neue Anzeigeformat *Format*. Das neue Format wird nur für die aktuelle Anzeige verwendet; es wird nicht mit dem Formular gespeichert.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den optionalen Parameter * geben Sie in *Objekt* ein Datenfeld oder eine Variable an. In diesem Fall geben Sie anstatt eines Strings eine Referenz auf das Datenfeld oder die Variable an (nur Datenfeld- oder Variablenobjekte). Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

OBJECT SET FORMAT können Sie für Eingabe- und Ausgabeformulare einsetzen und auf Datenfelder oder eingebbare/nicht eingebbare Variablen anwenden.
Sie müssen natürlich ein Format verwenden, das zum Objekt bzw. zum im Objekt verwendeten Datentyp passt.

Boolean

Für Datenfelder vom Typ Boolean gibt es zwei Möglichkeiten:

- Sie übergeben in *Format* einen Wert. In diesem Fall erscheint das Feld als Kontrollkästchen, seine Bezeichnung ist der angegebene Wert.
- Sie übergeben in *Format* zwei Werte, getrennt durch Strichpunkt (*;*). In diesem Fall erscheint das Feld mit zwei Optionsfeldern.

Datum

Für Datenfelder oder Variablen vom Typ Datum übergeben Sie in *Format* das ASCII-Zeichen **Char(n)**. *n* ist eine der folgenden Konstanten, die von 4D vordefiniert sind:

Konstante	Typ	Wert	Kommentar
Blank if null date	Lange Ganzzahl	100	"" statt 0
Date RFC 1123	Lange Ganzzahl	10	Fri, 10 Sep 2010 13:07:20 GMT
Internal date abbreviated	Lange Ganzzahl	6	29. Dez. 2006
Internal date long	Lange Ganzzahl	5	29. Dezember 2006
Internal date short	Lange Ganzzahl	7	29.12.2006
Internal date short special	Lange Ganzzahl	4	29.12.06 (aber 29.12.1896 oder 29.12.2096)
ISO Date	Lange Ganzzahl	8	2006-29-12T00:00:00 (überholt)
ISO Date GMT	Lange Ganzzahl	9	2010-09-13T16:11:53Z
System date abbreviated	Lange Ganzzahl	2	So, 29. Dez. 2006
System date long	Lange Ganzzahl	3	Sonntag, 29. Dezember 2006
System date short	Lange Ganzzahl	1	29.12.2006

Hinweis: Die Konstante Blank if null muss an das Format angehängt werden; damit zeigt 4D bei einem Nullwert einen leeren Bereich anstelle von Nullen.

Zeit

Für Datenfelder oder Variablen vom Typ Zeit übergeben Sie in *Format* das ASCII-Zeichen **Char(n)**. *n* ist eine der folgenden Konstanten, die von 4D vordefiniert sind:

Konstante	Typ	Wert	Kommentar
Blank if null time	Lange Ganzzahl	100	"" statt 0
HH MM	Lange Ganzzahl	2	
HH MM AM PM	Lange Ganzzahl	5	
HH MM SS	Lange Ganzzahl	1	
Hour min	Lange Ganzzahl	4	1 Stunde 2 Minuten
Hour min sec	Lange Ganzzahl	3	1 Stunde 2 Minuten 3 Sekunden
ISO time	Lange Ganzzahl	8	
Min sec	Lange Ganzzahl	7	62 Minuten 3 Sekunden
MM SS	Lange Ganzzahl	6	
System time long	Lange Ganzzahl	11	1:02:03 AM HNEC (nur Mac OS)
System time long abbreviated	Lange Ganzzahl	10	1•02•03 AM (nur Mac OS)
System time short	Lange Ganzzahl	9	

Hinweis: Die Konstante Blank if null muss an das Format angehängt werden; damit zeigt 4D bei einem Nullwert einen leeren Bereich anstelle von Nullen.

Bild

Für Datenfelder oder Variablen vom Typ Bild übergeben Sie in *Format* das Zeichen **Char(n)**. *n* ist eine der folgenden Konstanten, die von 4D vordefiniert sind:

Konstante	Typ	Wert
On background	Lange Ganzzahl	3
Replicated	Lange Ganzzahl	7
Scaled to fit	Lange Ganzzahl	2
Scaled to fit prop centered	Lange Ganzzahl	6
Scaled to fit proportional	Lange Ganzzahl	5
Truncated centered	Lange Ganzzahl	1
Truncated non centered	Lange Ganzzahl	4

Alphanumerisch und Zahl

Für Datenfelder oder Variablen vom Typ Alphanumerisch oder Zahl übergeben Sie die Bezeichnung direkt in *Format*.

Weitere Informationen zu Anzeigeformaten finden Sie im Handbuch *4D Designmodus* in den Abschnitten **Zahlenformate** und **Alpha Formate**.

Hinweis: Wollen Sie in *Format* Anzeigeformate verwenden, die Sie in der Toolbox definiert haben, setzen Sie vor den Formatnamen einen senkrechten Strich (|).

Bildschaltflächen

Für Bildschaltflächen übergeben Sie in *Format* eine Zeichenkette mit folgender Syntax:
Spalten;Zeilen;Bild;Flags{;ticks}

- *Spalten* = Anzahl Spalten im Bild
- *Zeilen* = Anzahl Zeilen im Bild
- *Bild* = Bild aus der Bildbibliothek oder Bildvariable
  - Bei einem Bild aus der Bildbibliothek geben Sie seine Nummer mit vorangestelltem Fragezeichen ein, z.B. ?250.
  - Bei einem Bild aus einer Bildvariablen geben Sie den Variablennamen ein.
- *Flags* = Anzeige und Ausführung einer Bildschaltfläche. Dieser Parameter kann folgende Werte annehmen: 0, 1, 2, 16, 32, 64 und 128. Jeder dieser Werte steht für einen Anzeige- oder Ausführungsmodus. Die Werte lassen sich auch kumulieren: Wollen Sie z.B. Modus 1 und 64 aktivieren, übergeben Sie als Parameter 65. Es gibt folgende Flags:
  - *Flags* = 0 (Keine Option)
  - Zeigt das nächste Bild in der Serie, wenn der Benutzer auf das Bild klickt. Zeigt das vorige Bild in der Serie, wenn der Benutzer bei gedrückter Umschalttaste auf das Bild klickt. Ist das letzte Bild der Serie erreicht, ändert sich das Bild nicht bei erneutem Anklicken, d.h. es geht nicht weiter zum ersten Bild.
  - *Flags* = 1 (kontinuierlich wechseln)
  - Ähnlich wie die vorige Option. Der Benutzer kann jedoch mit gedrückter Maustaste die Bilder kontinuierlich anzeigen, z.B. als Animation. Ist das letzte Bild der Serie erreicht, ändert sich das Bild nicht bei erneutem Anklicken, d.h. es geht nicht weiter zum ersten Bild.
  - *Flags* = 2 (Schleife zurück zum ersten Bild)
  - Ähnlich wie die vorige Option. Die Bilder erscheinen hier als kontinuierliche Schleife. Hat der Benutzer das letzte Bild erreicht und klickt erneut, erscheint das erste Bild, dann das zweite, etc.
  - *Flags* = 16 (Wechsel bei Darüberziehen)
  - Der Inhalt der Bildschaltfläche ändert sich, wenn der Mauszeiger darübergezogen wird. Sobald der Cursor den Schaltflächenbereich verlässt, wird das ursprüngliche Bild wieder angezeigt. Dieser Modus wird häufig in Multimedia Anwendungen oder HTML Dokumenten verwendet. Das dann gezeigte Bild ist das letzte Bild der Thumbnail-Tabelle, ausser die Option 128 ist aktiv (Verwende letztes Bild, wenn die Option Deaktiviert gewählt wird). In diesem Fall wird das Bild angezeigt, das dem letzten Thumbnail am nächsten liegt.
  - *Flags* = 32 (Springe zurück bei Loslassen)
  - Dieser Modus arbeitet mit zwei Bildern. Er zeigt immer das erste Bild. Klickt der Benutzer auf die Schaltfläche, erscheint das zweite Bild, bis die Maustaste losgelassen wird. Dann erscheint wieder das erste Bild. Mit diesem Modus können Sie eine Aktionsschaltfläche erstellen, die den jeweiligen Status angibt (angeklickt/nicht angeklickt). Sie können einen 3D Effekt erstellen oder jede andere Art verwenden, welche die Aktion darstellt.
  - *Flags* = 64 (Transparent)
  - Damit machen Sie das Hintergrundbild transparent.
  - *Flags* = 128 (Verwende letztes Bild bei Deaktivierung)
  - Damit legen Sie fest, dass das letzte Thumbnail erscheint, wenn die Schaltfläche deaktiviert wird. Verwenden Sie diesen Modus zusammen mit den Modi 0, 1 und 2, wird das letzte Thumbnail nicht berücksichtigt. Es erscheint nur bei deaktivierter Schaltfläche.
- *Ticks*: Aktiviert den Modus "Wechsle jedes nte Tick" und setzt das Zeitintervall zwischen die Anzeige jedes Bilds. Dieser optionale Parameter ermöglicht, den Inhalt der Bildschaltfläche mit der angegebenen Geschwindigkeit zu durchlaufen. Geben Sie z.B. ein "2;3;?,16807;0;10", zeigt die Schaltfläche alle 10 Ticks ein anderes Bild an. Ist dieser Modus aktiv, lässt sich nur der Modus 64 (Transparent) verwenden.

Bild-PopUp-Menüs

Für Bild-PopUp-Menüs übergeben Sie in *Format* eine Zeichenkette mit folgender Syntax: *Spalten;Zeilen;Bild;hRand;vRand;Flags*

- *Spalten* = Anzahl Spalten im Bild
- *Zeilen* = Anzahl Zeilen im Bild
- *Bild* = Bild aus der Bildbibliothek oder Bildvariable
  - Bei einem Bild aus der Bildbibliothek geben Sie seine Nummer mit vorangestelltem Fragezeichen ein, z.B. ?250.
  - Bei einem Bild aus einer Bildvariablen geben Sie den Variablennamen ein.
- *hRand* = Rand in Pixel zwischen horizontaler Begrenzung von Menü und Bild
- *vRand* = Rand in Pixel zwischen vertikaler Begrenzung von Menü und Bild
- *Flags* = Transparenter Modus für Bild-PopUp-Menüs. Akzeptiert die Werte 0 und 64.
  - *Flags* = 0: Bild-PopUp-Menü ist nicht transparent
  - *Flags* = 64: Bild-PopUp-Menüs ist transparent

Thermometer und Lineale

Für Objekte vom Typ Thermometer oder Lineal übergeben Sie in *Format* eine Zeichenkette mit folgender Syntax:
min;max;Einheit;Schritt;Flags{;Format{;Anzeige}}

- *Min* = Wert für erste Einteilung des Indikators
- *Max* = Wert für letzte Einteilung des Indikators
- *Einheit* = Intervall zwischen den Einteilungen
- *Schritt* = Mindestintervall für Cursor-Bewegung auf dem Indikator
- *Flags* = Anzeige und Ausführung des Indikators. Dieser Parameter kann die Werte 0, 2, 3, 16, 32 und 128 annehmen. Die Werte lassen sich auch kumulieren, um mehrere Optionen zu setzen (außer 128). Es gibt folgende Flags:
  - *Flags* = 0: Einheiten erscheinen nicht
  - *Flags* = 2: Einheiten erscheinen rechts oder unter dem Indikator
  - *Flags* = 3: Einheiten erscheinen links oder über dem Indikator
  - *Flags* = 16: Einteilung erscheint neben den Einheiten
  - *Flags* = 32: On Data Change wird ausgeführt, während der Benutzer die Skala anpasst. Wird dieser Wert nicht verwendet, tritt On Data Change erst ein, nachdem der Benutzer die Skala angepasst hat.
  - *Flags* = 128: Aktiviert den Modus unbestimmt (animiert). Dieser Wert ist nicht mit anderen kombinierbar. Ist er aktiviert, werden die anderen Parameter ignoriert. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.
- *Format* = Anzeigeformat für die Einteilung auf der Skala.
  Beachten Sie, dass Einheiten und Einteilung automatisch ausgeblendet sind, wenn die Größe des Indikators für eine korrekte Anzeige nicht ausreicht.
- *Anzeige* = spezifische Anzeigeoptionen. Bei Thermometern wird dieser Parameter nur berücksichtigt, wenn der Unterparameter *Flags* 128 ist.
  - *Anzeige* = 0 (oder weggelassen): Zeigt das Standardlineal an / Zeigt Thermometer mit kontinuierlicher Animation an.
  - *Anzeige* = 1 : aktiviert den Modus "Stepper" für ein Lineal / aktiviert den Modus "Asynchrones Fortschreiten".
    Weitere Informationen zu diesen Optionen finden Sie im Handbuch *Designmodus*.

Halbkreisskala

Für Halbkreisskalen übergeben Sie in *Format* eine Zeichenkette mit folgender Syntax:
Min;Max;Einheit;Schritt{ ;Flags}

- *Min* = Wert für erste Einteilung des Indikators
- *Max* = Wert für letzte Einteilung des Indikators
- *Einheit* = Intervall zwischen den Einteilungen
- *Schritt* = Mindestintervall für Cursor-Bewegung auf dem Indikator
- *Flags* = Ausführung des Indikators (optional). Dieser Parameter akzeptiert nur den Wert 32: On Data Change wird ausgeführt, während der Benutzer den Indikator anpasst. Wird dieser Wert nicht verwendet, tritt On Data Change erst ein, nachdem der Benutzer den Indikator angepasst hat

Schaltflächengitter

Für Schaltflächengitter übergeben Sie in *Format* eine Zeichenkette mit folgender Syntax: *Spalten;Zeilen*

- *Spalten* = Anzahl Spalten im Gitter
- *Zeilen* = Anzahl Zeilen im Gitter

Hinweis: Weitere Informationen zu Anzeigeformaten für Formularobjekte finden Sie im Handbuch *4D Designmodus*.

3D Schaltflächen

Um 3D Schaltflächen zu formatieren, übergeben Sie im Parameter *Format* einen String mit folgender Syntax:
title;picture;background;titlePos;titleVisible;iconVisible;style;horMargin;vertMargin;iconOffset;popupMenu;hyperlink;numStates

- *title* = Titel der Schaltfläche. Dieser Wert kann ein Text oder eine Ressourcennummer sein (z.B.: ":16800,1")
- *picture* = Mit einer Schaltfläche verknüpftes Bild aus der Bildbibliothek oder einer Bildvariablen
  - Bei einem Bild aus der Bildbibliothek geben Sie seine Nummer mit vorangestelltem Fragezeichen ein, z.B. ?250.
  - Bei einem Bild aus einer Datei, die im Ordner Resources der Datenbank liegt, geben Sie eine URL vom Typ "# {Ordner/}Bildname" oder "Datei:{Ordner/}Bildname" ein.
- *background* = Mit einer Schaltfläche verknüpftes Hintergrundbild aus einer Bildbibliothek, einer Bildvariablen oder einer Datei, die im Ordner Resources liegt (siehe oben).
- *titlePos* = Position des Titels der Schaltfläche. Fünf Werte sind möglich:
  - *titlePos* = 1: Links
  - *titlePos* = 2: Oben
  - *titlePos* = 3: Rechts
  - *titlePos* = 4: Unten
  - *titlePos* = 5: Mitte
- *titleVisible* = Definiert, ob der Titel sichtbar ist, zwei Werte sind möglich:
  - *titleVisible* = 0: Titel ist ausgeblendet
  - *titleVisible* = 1: Titel wird angezeigt
- *iconVisible* = Definiert, ob der Icon sichtbar ist, zwei Werte sind möglich:
  - *iconVisible* = 0 : Icon ist ausgeblendet
  - *iconVisible* = 1 : Icon wird angezeigt
- *style* = Stil der Schaltfläche. Der Wert dieser Option bestimmt, ob weitere Optionen wie Hintergrund, o.ä. berücksichtigt werden. Zehn Werte sind möglich:
  - *style* = 0: Keine
  - *style* = 1: Hintergrund Versatz
  - *style* = 2: Push Button
  - *style* = 3: Toolbar Schaltfläche
  - *style* = 4: Eigene
  - *style* = 5: Kreis
  - *style* = 6: Kleines Systemviereck
  - *style* = 7: Office XP
  - *style* = 8: Bevel
  - *style* = 9: Abgerundeter Bevel

- *horMargin* = Horizontaler Rand. Anzahl Pixel für inneren linken und rechten Rand der Schaltfläche (Bereiche, über die Icon oder Text nicht hinausgehen dürfen)
- *vertMargin* = Vertikaler Rand. Anzahl Pixel für inneren oberen und unteren Rand der Schaltfläche (Bereiche, über die Icon oder Text nicht hinausgehen dürfen)
- *iconOffset* = Icon versetzt nach rechts und unten. Dieser Wert in Pixel gibt den Versatz des Icons nach rechts und nach unten an, wenn auf die Schaltfläche geklickt wird (derselbe Wert wird für beide Richtungen verwendet).
- *popupMenu* = Der Schaltfläche zugewiesenes PopUp-Menü. Drei Werte sind möglich:
  - *popupMenu* = 0: Kein PopUp-Menü
  - *popupMenu* = 1: Mit verknüpftem PopUp-Menüu
  - *popupMenu* = 2: Mit separatem PopUp-Menü
- *hyperlink* = Titel wird bei Maus-Darüberziehen unterstrichen, um einem Hyperlink zu ähneln (bisheriger Mechanismus). Zwei Werte sind möglich:
  - *hyperlink* = 0: Titel wird bei Maus-Darüberziehen nicht unterstrichen
  - *hyperlink* = 1: Titel wird bei Maus-Darüberziehen unterstrichen
- *numStates* = Anzahl der Stadien im Bild, die als Icon für die 3D Schaltfläche verwendet werden und zur Darstellung der einzelnen Stadien (von 0 bis 4) dienen.

Manche Optionen gelten nicht für alle Stilarten von 3D Schaltflächen. In bestimmten Fällen wollen Sie auch nicht alle Optionen verändern. Um eine Option nicht zu vergeben, lassen Sie einfach den entsprechenden Wert weg. Wollen Sie z.B. nicht die Optionen *titleVisible* und *vertMargin* übergeben, schreiben Sie:

```
OBJECT SET FORMAT(myVar;"NiceButton;?256;:562;1;;1;4;5;;5;0")
```

Kopfteile der Listbox

Zum Einrichten des Icon im Kopfteil übergeben Sie im Parameter *Format* einen Zeichenstring mit der Syntax *picture;iconPos*

- *picture* = Bild für Kopfteil aus der Bildbibliothek, eine Bildvariable oder eine Bilddatei:
  - Bei einem Bild aus der Bildbibliothek geben Sie seine Nummer mit vorangestellten Fragezeichen ein (z.B.: "?250").
  - Bei einer Bildvariablen geben Sie den Variablennamen ein.
  - Bei einer Datei aus dem Ordner *Resources* der Anwendung geben Sie eine URL in Form von "#{Ordner/}Bildname" oder "Datei:{Ordner/}Bildname".
- *iconPos* = Position des Icon im Kopfteil. Es gibt zwei Werte:
  - *iconPos* = 1: Links
  - *iconPos* = 2: Rechts

Dieses Feature ist z.B. hilfreich, wenn Sie einen eigenen Icon zum Sortieren verwenden wollen.

Beispiel 1

Folgender Code formatiert das Datenfeld *[Angestellte]Einstellungsdatum* mit der Konstanten Internal date short

```
OBJECT SET FORMAT([Angestellte]Einstellungsdatum;Char(Internal date short))
```

Beispiel 2

Folgendes Beispiel ändert das Format des Datenfelds *[Firma]Bankleitzahl* je nach der Länge des Wertes:

```
If(Length([Firma]BLZ)=9)
   OBJECT SET FORMAT([Firma]BLZ;"###-###-##")
Else
   OBJECT SET FORMAT([Firma]BLZ;"################")
End if
```

Beispiel 3

Folgendes Beispiel formatiert den Wert des Feldes [Stats]Results abhängig davon, ob die Zahl positiv, negativ oder Null ist:

```
OBJECT SET FORMAT([Stats]Results;"### ##0.00;(### ##0.00);")
```

Beispiel 4

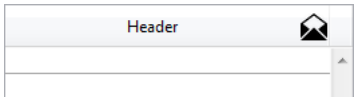
Folgendes Beispiel setzt für das Datenfeld vom Typ Boolean anstatt der Standardwerte Ja und Nein die Werte Verheiratet und Ledig ein:

```
OBJECT SET FORMAT([Angestellte]Familienstand;"Verheiratet;Ledig")
```

Beispiel 5

Im Ordner *Resources* der Anwendung ist eine Bilddatei mit Namen "envelope_open.png" gespeichert. Sie schreiben folgenden Code:

```
vIcon:="#envelope_open.png"
vPos:="2"  // Rechts
OBJECT SET FORMAT(*;"Header1";vIcon+";"+vPos)
```



Beispiel 6

Folgendes Beispiel setzt für das Datenfeld vom Typ Boolean ein Kontrollfeld mit den Bezeichnung "Klassifiziert" ein:

```
OBJECT SET FORMAT([Ordner]Klassifizierung;"Klassifiziert")
```

Beispiel 7

Sie haben eine Tabelle mit Thumbnails mit einer Zeile und vier Spalten, um eine Bildschaltfläche anzuzeigen ("Standard", "Angeklickt", "Darüberziehen" und "Deaktiviert"). Es soll folgendes ausgeführt werden: Wechseln bei Darüberziehen, Zurückwechseln bei Loslassen und Letztes Bild verwenden bei Deaktiviert. Der Code dafür lautet:

```
OBJECT SET FORMAT(*;"Bildschaltfläche";"4;1;?15000;176")
```

Beispiel 8

Thermometer in animierten Modus setzen:

```
OBJECT SET FORMAT($Mythermo;";;;;128")
$Mythermo:=1  `Starte Animation
```

OBJECT SET HELP TIP

OBJECT SET HELP TIP ({* ;} Objekt ; HilfeTipp)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
HilfeTipp	Text	➡	Inhalt der Hilfemeldung

Beschreibung

Der Befehl **OBJECT SET HELP TIP** setzt oder ändert dynamisch den Hilfetipp für das bzw. die Objekte, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Im Parameter *HilfeTipp* übergeben Sie eine Zeichenkette für den Inhalt der Meldung. Übergeben Sie einen leeren String "", wird der Hilfetipp entfernt.

Beim Ausführen des Formulars erscheinen Hilfemeldungen als Hilfetipps, wenn der Cursor über das Feld oder Objekt geht. Die verzögerte Anzeige und maximale Dauer der Anzeige von Hilfemeldungen lässt sich über die Selektoren Tips delay und Tips duration des Befehls **SET DATABASE PARAMETER** steuern.

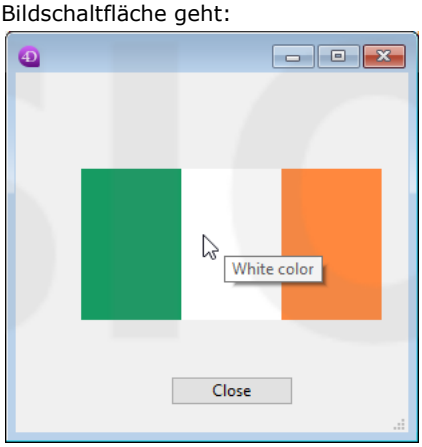
Mit dem Befehl **OBJECT SET HELP TIP** lassen sich Hilfetipps dynamisch steuern: Er schließt den angezeigten Hilftetipp, öffnet an der Mausposition einen neuen Tipp und startet den Zähler Tips duration neu.

Hinweise:

- Der Inhalt von Hilfetipps lässt sich auch im Formulareditor (siehe **Eingabekontrollen**) und im Strukturediteor (siehe **Feldeigenschaften**) des Designmodus setzen.
- Mit dem Selektor Tips enabled des Befehls **SET DATABASE PARAMETER** lassen sich Hilfetipps global deaktivieren.

Beispiel 1

In diesem Formular erscheint ein Hilfetipp und ändert sich automatisch, wenn die Maus über verschiedene Teile der Bildschaltfläche geht:




```
//Objektmethode "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) //Aktivierung von Tipps sicherstellen
      SET DATABASE PARAMETER(Tips delay;0) // Tipp wird unmittelbar bei Mausstopp angezeigt
      SET DATABASE PARAMETER(Tips duration;60*10) // max. Anzeigedauer 10 Sekunden
   :(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 //jeder Teil der Flagge ist 76 Pixel
```

```
            :($x<76)
                $tip:="Green color"
            :($x<152)
                $tip:="White color"
            Else
                $tip:="Orange color"
        End case

        $doRefresh:=($tip#oldtip)  //wahr wenn anderer Tipp
        If(Not($doRefresh))  //gleicher Inhalt
            $doRefresh:=((Abs($x-oldX)>30)|(Abs($y-oldY)>30))  //wahr wenn Cursor bewegt wurde
        End if

        If($doRefresh)  //anderen Tipp anzeigen
            OBJECT SET HELP TIP(*;"myFlag";$tip)
            oldX:=$x
            oldY:=$y
            oldTip:=$tip
        End if

    End case
```

Beispiel 2

In der Listbox "Commands List" einen Hilfetipp setzen, um die Beschreibung zum Eintrag anzuzeigen. Die Beschreibung liegt in der Tabelle [Documentation].

```
    C_REAL($mouseX;$mouseY;$mouseZ)
    C_LONGINT($col;$row)

    Case of

        :(Form event=On Mouse Enter)

            SET DATABASE PARAMETER(Tips delay;1)  // zeigt den Tipp rasch an

        :(Form event=On Mouse Move)

      //#1 : herausfinden, über welche Zeile die Maus zieht

            GET MOUSE($mouseX;$mouseY;$mouseZ)
            LISTBOX GET CELL POSITION(*;"Commands List";$mouseX;$mouseY;$col;$row)

      //#2 : den passenden Hilfetipp setzen

            If($row#0)
                GOTO SELECTED RECORD([Documentation];$row)
                OBJECT SET HELP TIP(*;"Commands List";[Documentation]Description)  // Die Beschreibung erscheint im Hilfetipp, wenn die Maus
stoppt.
            End if

        :(Form event=On Mouse Leave)

            SET DATABASE PARAMETER(Tips delay;3)  // zeigt den Tipp normal an

    End case
```

Das Ergebnis ist...

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 ({* ;} Objekt ; Ausrichtung)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Ausrichtung	Lange Ganzzahl	➡	Code für Ausrichtung

Beschreibung

Der Befehl **OBJECT SET HORIZONTAL ALIGNMENT** bestimmt die Art der Ausrichtung für die Parameter *Objekt* und *.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den Parameter * geben Sie in *Objekt* ein Feld oder eine Variable an. Sie geben dann nicht einen String, sondern eine Referenz auf ein Feld oder eine Variable an.

Im Parameter *Ausrichtung* übergeben Sie eine Konstante unter dem Thema **Formularobjekte (Eigenschaften)**:

Konstante	Typ	Wert
Align center	Lange Ganzzahl	3
Align default	Lange Ganzzahl	1
Align left	Lange Ganzzahl	2
Align right	Lange Ganzzahl	4

OBJECT SET HORIZONTAL ALIGNMENT ist für folgende Formularobjekte verwendbar:

- Rollbare Bereiche
- Combo Boxen
- Statischer Text
- Gruppenboxen
- PopUp-Menüs/DropDown-Listen
- Felder
- Variablen
- Listboxen
- Spalten von Listboxen
- Kopfteile von Listboxen
- Fußteile von Listboxen
- 4D Write Pro Bereiche, siehe **4D Write Pro Handbuch**

OBJECT SET INDICATOR TYPE

OBJECT SET INDICATOR TYPE ({* ;} Objekt ; Indikator)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Indikator	Lange Ganzzahl	➡	Anzeigetyp

Beschreibung

Der Befehl **OBJECT SET INDICATOR TYPE** ändert im aktuellen Prozess die Art der Ablaufanzeige für die Thermometer, definiert über die Parameter *Objekt* und *.

Der Anzeigetyp bestimmt die Darstellung des Thermometers. Weitere Informationen dazu finden Sie im Abschnitt **Indikatoren** des Handbuchs *4D Designmodus*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Im Parameter *Indikator* übergeben Sie den gewünschten Typ für die Anzeige. Sie können eine der folgenden Konstanten unter dem Thema "**Formularobjekte (Eigenschaften)**":

Konstante	Typ	Wert	Kommentar
Asynchronous progress bar	Lange Ganzzahl	3	Drehendes Rad
Barber shop	Lange Ganzzahl	2	Animierter Balken
Progress bar	Lange Ganzzahl	1	Standard Ablaufbalken

OBJECT SET KEYBOARD LAYOUT

OBJECT SET KEYBOARD LAYOUT ({* ;} Objekt ; SprachCode)

Parameter	Typ		Beschreibung
*	Operator	⇒	Mit Stern: Objekt ist ein Objektname(String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	⇒	Objektname (mit *) oder Feld bzw. Variable (ohne *)
SprachCode	String	⇒	Sprachcode RFC3066 ISO639 und ISO3166, "" = keine Änderung

Beschreibung

Der Befehl **OBJECT SET KEYBOARD LAYOUT**setzt oder ändert dynamisch das dem bzw. den Objekten zugeordnete Tastatur-Layout, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable oder ein Feld. In diesem Fall übergeben Sie eine Referenz anstelle eines Namens.

In *SprachCode* übergeben Sie einen String mit dem Code der gewünschten Sprache, basierend auf RFC3066, ISO639 und ISO3166. Weitere Informationen dazu finden Sie unter dem Befehl **SET DATABASE LOCALIZATION**.

OBJECT SET LIST BY NAME

OBJECT SET LIST BY NAME ({* ;} Objekt {; ListeTyp}; Liste)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
ListeTyp	Lange Ganzzahl	➡	Typ der Liste: Auswahlliste, Erforderlich-Liste oder Ausgenommen-Liste
Liste	String	➡	Name der Auswahlliste (wie in Designumgebung definiert)

Beschreibung

Der Befehl **OBJECT SET LIST BY NAME** setzt bzw. ersetzt die in *Objekt* angegebene Auswahlliste, die einem Objekt oder einer Objektgruppe zugewiesen ist, durch die in *Liste* übergebene Auswahlliste, wie sie im Listeneditor der Designumgebung definiert wurde.

OBJECT SET LIST BY NAME können Sie für Eingabe- und Ausgabeformulare einsetzen und auf Datenfelder und eingebbare Variablen für Text anwenden. Die Auswahlliste erscheint bei der Dateneingabe, wenn der Benutzer den Textbereich auswählt.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den optionalen Parameter * geben Sie in *Objekt* ein Datenfeld oder eine Variable an. In diesem Fall geben Sie anstatt eines Strings eine Referenz auf das Datenfeld oder die Variable an (nur Datenfeld- oder Variablenobjekte). Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

Hinweis: Dieser Befehl lässt sich nicht auf Datenfelder in Unterformularen anwenden.

Der Befehl **OBJECT SET LIST BY NAME** kann alle Listentypen setzen oder ersetzen, die dem bzw. den Objekten zugeordnet sind, definiert durch die Parameter *Objekt* und *. Das sind Auswahllisten, Listen mit erforderlichen Werten und Listen mit ausgenommenen Werten. Dazu übergeben Sie in *ListeTyp* eine der folgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)**:

Konstante	Typ	Wert	Kommentar
Choice list	Lange Ganzzahl	0	Liste zum Auswählen von Werten (Option "Auswahlliste" in der Eigenschaftenliste) (Standard)
Excluded list	Lange Ganzzahl	2	Liste mit ausgeschlossenen Werten für die Eingabe (Option "Ausgenommen-Liste" in der Eigenschaftenliste)
Required list	Lange Ganzzahl	1	Liste mit erforderlichen Werten für die Eingabe (Option "Erforderlich-Liste" in der Eigenschaftenliste)

Lassen Sie diesen Parameter weg, wird standardmäßig 0 (Auswahlliste) verwendet.

Um im aktuellen Prozess die Zuweisung einer Liste für *Objekt* aufzuheben, übergeben Sie im Parameter *Liste* für den entsprechenden Listentyp einen leeren String ("").

Beispiel 1

Folgendes Beispiel legt eine Auswahlliste für das Datenfeld Lieferanten fest. Bei Übernachtlieferung erscheint die Auswahlliste der Expresslieferanten. Sonst wird die Standardliste angezeigt:

```
If([Shipments]Overnight)
    OBJECT SET LIST BY NAME([Shipments]Shipper;"Expresslieferanten")
Else
    OBJECT SET LIST BY NAME([Shipments]Shipper;"Normale Lieferanten")
End if
```

Beispiel 2

Die Liste "Farbwahl" als einfache PopUp/DropDown-Liste mit Namen "Türfarbe" zuweisen:

```
OBJECT SET LIST BY NAME(*;"Türfarbe";Choice list;"Farbwahl")
  // in diesem Fall können Sie den 3. Parameter (Konstante) weglassen
```

Beispiel 3

Sie wollen die Liste "Farbwahl" einer Combo Box "Wandfarbe" zuordnen. Da sie eingebbar ist, sollen bestimmte Farben wie "schwarz", "purpur" etc., von der Eingabe ausgeschlossen sein. Diese Farben setzen Sie in die Liste "ausgeschl_Farben":

```
OBJECT SET LIST BY NAME(*;"Wandfarbe";Choice list;"Farbwahl")
OBJECT SET LIST BY NAME(*;"Wandfarbe";Excluded list;"ausgeschl_Farben")
```

Beispiel 4

Sie wollen die zugewiesenen Listen entfernen:

```
// Die Auswahlliste entfernen
OBJECT SET LIST BY NAME(*;"Türfarbe";Choice list;"")
// Die Liste mit nicht-erlaubten Werten entfernen
OBJECT SET LIST BY NAME(*;"Wandfarbe";Excluded list;"")
```

OBJECT SET LIST BY REFERENCE

OBJECT SET LIST BY REFERENCE ({* ;} Objekt {; ListeTyp}; Liste)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
ListeTyp	Lange Ganzzahl	➡	Typ der Liste: Auswahlliste, Erforderlich-Liste oder Ausgenommen-Liste
Liste	ListRef	➡	Referenznummer der Liste

Beschreibung

Der Befehl **OBJECT SET LIST BY REFERENCE** definiert oder ersetzt die Liste für das bzw. die Objekte, definiert durch die Parameter *Objekt* und *, mit der hierarchischen Liste, die im Parameter *Liste* angegeben ist.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Standardmäßig, d.h. ohne den Parameter *ListeTyp*, definiert der Befehl für das Objekt als Ursprung eine Auswahlliste (Werte zum Auswählen). Im Parameter *ListeTyp* bestimmen Sie einen Listentyp. Dazu übergeben Sie eine der folgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)**:

Konstante	Typ	Wert	Kommentar
Choice list	Lange Ganzzahl	0	Liste zum Auswählen von Werten (Option "Auswahlliste" in der Eigenschaftenliste) (Standard)
Excluded list	Lange Ganzzahl	2	Liste mit ausgeschlossenen Werten für die Eingabe (Option "Ausgenommen-Liste" in der Eigenschaftenliste)
Required list	Lange Ganzzahl	1	Liste mit erforderlichen Werten für die Eingabe (Option "Erforderlich-Liste" in der Eigenschaftenliste)

In *Liste* übergeben Sie die Referenznummer der hierarchischen Liste, die Sie dem Objekt zuordnen wollen. Diese Liste muss über die Funktion **Copy list**, **Load list** oder **New list** erstellt worden sein.

Wollen Sie die Zuordnung der Liste zu *Objekt* beenden, übergeben Sie im Parameter *Liste* für den betreffenden Listentyp einfach den Wert 0.

Entfernen der Zuordnung löscht nicht die Listenreferenz im Speicher. Denken Sie daran, **CLEAR LIST** aufzurufen, wenn Sie die Liste nicht mehr benötigen.
Dieser Befehl ist besonders bei PopUp-Menüs oder Comboboxen interessant, die einer Variable oder einem Feld zugeordnet sind (siehe Handbuch *4D Designmodus*). Hier ist die Zuweisung dynamisch und jede Änderung in der Liste wird in das Formular kopiert. Ist das Objekt einem Array zugewiesen, wird die Liste in das Array kopiert und Änderungen in der Liste sind nicht automatisch verfügbar (siehe Beispiel 5).

Beispiel 1

Einem Feld vom Typ Text eine einfache Auswahlliste zuweisen (standardmäßiger Listentyp):

```
vCountriesList:=New list
APPEND TO LIST(vCountriesList;"Spanien";1)
APPEND TO LIST(vCountriesList;"Portugal";2)
APPEND TO LIST(vCountriesList;"Griechenland";3)
OBJECT SET LIST BY REFERENCE([Contact]Country;vCountriesList)
```

Beispiel 2

Die Liste "vColor" als einfache Auswahlliste mit der PopUp/DropDown Liste "Türfarbe" zuweisen:

```
vColor:=New list
APPEND TO LIST(vColor;"Blau";1)
APPEND TO LIST(vColor;"Grün";2)
APPEND TO LIST(vColor;"Rot";3)
APPEND TO LIST(vColor;"Gelb";4)
OBJECT SET LIST BY REFERENCE(*;"Türfarbe";Choice list;vColor)
```

Beispiel 3

Jetzt wollen Sie die Liste "vColor" der Combobox mit Namen "Wandfarbe" zuordnen. Da sich in die Combobox Werte eingeben lassen, möchten Sie sichergehen, dass bestimmte Farben wie "schwarz," "purpur," etc., ausgeschlossen sind.
Diese Farben werden in die Liste "vReject" gesetzt:

```
OBJECT SET LIST BY REFERENCE(*;"Wandfarbe";Choice list;vColor)
vReject:=New list
APPEND TO LIST(vReject;"schwarz";1)
APPEND TO LIST(vReject;"grau";2)
APPEND TO LIST(vReject;"purpur";3)
OBJECT SET LIST BY REFERENCE(*;"Wandfarbe";Excluded list;vReject)
```

Beispiel 4

Sie wollen die Listenzuweisungen entfernen:

```
OBJECT SET LIST BY REFERENCE(*;"Wandfarbe";Choice list;0)
OBJECT SET LIST BY REFERENCE(*;"Wandfarbe";Required list;0)
OBJECT SET LIST BY REFERENCE(*;"Wandfarbe";Excluded list;0)
```

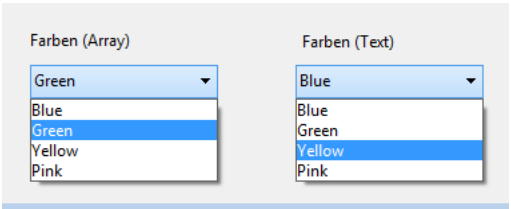
Beispiel 5

ieses Beispiel zeigt die unterschiedliche Arbeitsweise des Befehls für ein PopUp-Menü, das einem Array Text oder einer Textvariablen zugewiesen ist. Hier die beiden PopUp-Menüs im Formular

Der Inhalt dieser PopUp-Menüs wird über die Liste <>vColor gesetzt, die Farbwerte enthält. Beim Laden des Formulars wird folgender Code ausgeführt:

```
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)
```

Während der Ausführung bieten beide Menüs dieselben Werte an:

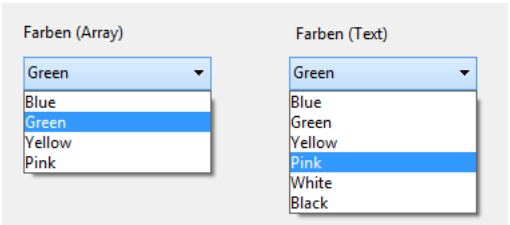


(*Diese Ansicht zeigt den Inhalt beider Menüs gleichzeitig an*)

Dann führen Sie folgenden Code aus, z.B. über eine Schaltfläche:

```
APPEND TO LIST(<>vColor;"White";5)
APPEND TO LIST(<>vColor;"Black";6)
```

Als Ergebnis sehen Sie, dass über die dynamische Referenz nur das Menü aktualisiert wird, das dem Textfeld zugeordnet ist



Die Liste, die dem PopUp-Menü per Array zugewiesen wird, wird aktualisiert, wenn Sie erneut den Befehl **OBJECT SET LIST BY REFERENCE** aufrufen und den Inhalt der Liste kopieren.

OBJECT SET MAXIMUM VALUE

OBJECT SET MAXIMUM VALUE ({* ;} Objekt ; maxWert)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
maxWert	Datum, Zeit, Zahl	➡	Maximumwert für Objekt

Beschreibung

Der Befehl **OBJECT SET MAXIMUM VALUE** ändert für den aktuellen Prozess den Maximumwert für das bzw. die Objekte, definiert über die Parameter *Objekt* und *.

Die Eigenschaft "Maximum" lässt sich für Daten vom Typ Zahl, Datum oder Zeit anwenden. Weitere Informationen dazu finden Sie im Abschnitt **Maximum- und Minimumwerte festlegen** des Handbuchs *4D Designmodus*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

In *maxWert* übergeben Sie den neuen Maximumwert, den Sie *Objekt* für den aktuellen Prozess zuweisen wollen. Dieser Wert muss zum Objekttyp passen, sonst wird Fehler Nr. 18 "Die Datenfeldtypen sind nicht kompatibel" zurückgegeben.

OBJECT SET MINIMUM VALUE

OBJECT SET MINIMUM VALUE ({* ;} Objekt ; minWert)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
minWert	Datum, Zeit, Zahl	➡	Minimumwert für Objekt

Beschreibung

Der Befehl **OBJECT SET MINIMUM VALUE** ändert für den aktuellen Prozess den Minimumwert für das bzw. die Objekte, definiert über die Parameter *Objekt* und *.

Die Eigenschaft "Minimum" lässt sich für Daten vom Typ Zahl, Datum oder Zeit anwenden. Weitere Informationen dazu finden Sie im Abschnitt **Maximum- und Minimumwerte festlegen** des Handbuchs *4D Designmodus*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

In *minWert* übergeben Sie den neuen Minimumwert, den Sie *Objekt* für den aktuellen Prozess zuweisen wollen. Dieser Wert muss zum Objekttyp passen, sonst wird Fehler Nr. 18 "Die Datenfeldtypen sind nicht kompatibel." zurückgegeben.

OBJECT SET MULTILINE

OBJECT SET MULTILINE ({* ;} Objekt ; Mehrzeilig)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld oder Variable (ohne *)
Mehrzeilig	Lange Ganzzahl	➡	Status der Eigenschaft mehrzeilig

Beschreibung

Der Befehl **OBJECT SET MULTILINE** ändert die Eigenschaft "Mehrzeilig" des bzw. der Objekte, definiert über die Parameter *Objekt* und *.

Die Eigenschaft "Mehrzeilig" steuert zwei Parameter zum Anzeigen und Drucken von Textbereichen: Anzeige der Wörter am Ende der Zeile für einzeilige Bereiche und automatisches Einfügen von Zeilenumbrüchen. Weitere Informationen dazu finden Sie im Abschnitt **Mehrzeilige Textbereiche** des Handbuchs *4D Designmodus*. Wenden Sie diesen Befehl auf ein Objekt an, das diese Eigenschaft nicht unterstützt, führt der Befehl nichts aus.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Im Parameter *Mehrzeilig* übergeben Sie den neuen Wert der gewünschten Option. Sie können folgende Konstanten unter dem Thema **Formularobjekte (Eigenschaften)** verwenden:

Konstante	Typ	Wert	Kommentar
Multiline Auto	Lange Ganzzahl	0	In einzeiligen Bereichen werden Wörter am Zeilenende abgeschnitten und keine Zeilenumbrüche gesetzt. In mehrzeiligen Bereichen führt 4D automatisch Zeilenumbrüche aus.
Multiline No	Lange Ganzzahl	2	Es gibt nie Zeilenumbrüche: Der Text erscheint immer als eine Zeile. Bei Alpha oder Textfeldern bzw. Variablen mit Zeilenumbrüchen wird der Text nach dem ersten Zeilenumbruch entfernt, sobald der Text geändert wird.
Multiline Yes	Lange Ganzzahl	1	In einzeiligen Bereichen erscheint der Text bis zum ersten Zeilenumbruch oder bis zum letzten Wort, das sich ganz anzeigen lässt. 4D fügt Zeilenumbrüche ein; über die Pfeiltaste nach unten lässt sich im Inhalt scrollen. In mehrzeiligen Bereichen führt 4D automatische Zeilenumbrüche aus.

Beispiel

In einem Eingabebereich mehrzeiligen Text ausschließen:

```
OBJECT SET MULTILINE(*;"vEntry";Multiline No)
```

OBJECT SET PLACEHOLDER

OBJECT SET PLACEHOLDER ({* ;} Objekt ; Platzhaltertext)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Platzhaltertext	Text	➡	Platzhaltertext für Objekt

Beschreibung

Der Befehl **OBJECT SET PLACEHOLDER** weist Platzhaltertext dem bzw. den Objekten zu, definiert mit den Parametern *Objekt* und *.

Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Wurde dem Objekt bereits über die Eigenschaftenliste Platzhaltertext zugewiesen, wird dieser Text im aktuellen Prozess durch den Inhalt des Parameters *PlatzhalterText* ersetzt

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

In *PlatzhalterText* übergeben Sie den Hilfstext oder die Anweisung, die bei einem leeren Objekt erscheinen soll

Hinweis: Der Befehl **OBJECT SET PLACEHOLDER** unterstützt keine xliff Referenzen im Platzhaltertext. Das ist nur für Platzhaltertext möglich, der über die Eigenschaftenliste definiert wurde.

Dieser Befehl lässt sich nur mit Formularobjekten vom Typ Variable, Feld oder Combo Box verwenden. Platzhaltertext lässt sich für Werte vom Typ Text oder alphanumerisch aber auch für Datum und Zeit verwenden, wenn für das Formularobjekt die Eigenschaft **Leer wenn Null** aktiv ist

Beispiel

In einer Combo Box den Platzhaltertext "Search" setzen:

```
OBJECT SET PLACEHOLDER(*;"search_combo";"Search")
```

OBJECT SET PRINT VARIABLE FRAME

OBJECT SET PRINT VARIABLE FRAME ({* ;} Objekt ; variablerRahmen {; festesUnterformular})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
variablerRahmen	Boolean	➡	Wahr = Variabler Druckrahmen, Falsch = Fester Druckrahmen
festesUnterformular	Lange Ganzzahl	➡	Optionen zum Drucken von Unterformularen mit fester Größe

Beschreibung

Der Befehl **OBJECT SET PRINT VARIABLE FRAME** ändert die Eigenschaft Variabler Druckrahmen für das bzw. die Objekte, definiert über die Parameter *Objekt* und *.

Diese Eigenschaft ist für folgende Objekte verfügbar:

- Variablen und Felder vom Typ Text oder Bild (siehe **Drucke variable Größe** im Handbuch *4D Designmodus*)
- 4D Write Pro Areas (siehe **4D Write Pro Bereich verwenden** im Handbuch *4D Write Pro*).
- Unterformulare. Unterformulare haben eine weitere Option zum Drucken mit fester Größe (siehe **Drucken** im Handbuch *4D Designmodus*); das lässt sich mit diesem Befehl mit dem Parameter *festesUnterformular* konfigurieren.

Wenden Sie diesen Befehl auf ein Objekt an, das diese Eigenschaft nicht unterstützt, führt der Befehl nichts aus.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Im Parameter *variablerRahmen* übergeben Sie ein Boolean: Bei **Wahr** wird das Objekt mit variablem Rahmen gedruckt, bei **Falsch** mit festem Rahmen.

Über den optionalen Parameter *festesUnterformular* können Sie eine weitere Option setzen, wenn Sie im Parameter *variablerRahmen* **Falsch** übergeben haben und *Objekt* ein Unterformular ist (in allen anderen Fällen wird das ignoriert). Dann können Sie für das Unterformular einen festen Druckrahmen übergeben. Sie können eine der folgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)** übergeben:

Konstante	Typ	Wert	Kommentar
Print Frame fixed with multiple records	Lange Ganzzahl	2	Der Rahmen behält die gleiche Größe, 4D druckt das Formular jedoch mehrmals, um alle Datensätze einzuschließen.
Print Frame fixed with truncation	Lange Ganzzahl	1	4D druckt nur die Datensätze, die in den Bereich des Unterformulars passen. Das Formular wird nur einmal gedruckt und die nicht gedruckten Datensätze werden ignoriert.

OBJECT SET RESIZING OPTIONS

OBJECT SET RESIZING OPTIONS ({* ;} Objekt ; horizontal ; vertikal)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
horizontal	Lange Ganzzahl	➡	Horizontale Anpassungsoption
vertikal	Lange Ganzzahl	➡	Vertikale Anpassungsoption

Beschreibung

Der Befehl **OBJECT SET RESIZING OPTIONS** setzt oder ändert dynamisch die Anpassungsoptionen für das bzw. die Objekte, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess. Diese Optionen geben an, wie das Objekt angezeigt wird, wenn das Formularfenster in der Größe angepasst wird.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Im Parameter *horizontal* übergeben Sie einen Wert für die horizontale Anpassungsoption, die Sie für *Objekt* definieren wollen. Sie können folgende Konstanten unter dem Thema **Formularobjekte (Eigenschaften)** verwenden:

Konstante	Typ	Wert	Kommentar
Resize horizontal grow	Lange Ganzzahl	1	Wächst das Fenster um 50% in der Breite, wird das Objekt um 50% nach rechts verbreitert
Resize horizontal move	Lange Ganzzahl	2	Wächst das Fenster um 100 Pixel in der Breite, wird das Objekt um 100 Pixel nach rechts bewegt
Resize horizontal none	Lange Ganzzahl	0	Wird das Fenster verbreitert, bleiben Breite noch Position des Objekts unverändert

Im Parameter *vertikal* übergeben Sie einen Wert für die vertikale Anpassungsoption, die Sie für *Objekt* definieren wollen. Sie können folgende Konstanten unter dem Thema **Formularobjekte (Eigenschaften)** verwenden:

Konstante	Typ	Wert	Kommentar
Resize vertical grow	Lange Ganzzahl	1	Wächst das Fenster um 50% in der Höhe, wird das Objekt um 50% nach unten verlängert
Resize vertical move	Lange Ganzzahl	2	Wächst das Fenster um 100 Pixel in der Höhe, wird das Objekt um 100 Pixel nach unten verlängert
Resize vertical none	Lange Ganzzahl	0	Wird das Fenster verlängert, bleiben Höhe Position des Objekts unverändert

OBJECT SET RGB COLORS

OBJECT SET RGB COLORS ({* ;} Objekt ; Vordergrundfarbe ; Hintergrundfarbe {; altHintergrFarbe})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit *: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Mit *: Objektname, ohne *: Feld oder Variable
Vordergrundfarbe	Lange Ganzzahl	➡	RGB Wert für Vordergrundfarbe
Hintergrundfarbe	Lange Ganzzahl	➡	RGB Wert für Hintergrundfarbe
altHintergrFarbe	Lange Ganzzahl	➡	RGB Farbe für wechselnde Hintergrundfarbe

Beschreibung

Der Befehl **OBJECT SET RGB COLORS** ändert die Vordergrund- und Hintergrundfarben der in *Objekt* angegebenen Objekte. Ist *Objekt* eine Listbox, können Sie einen weiteren Parameter für wechselnde Vorder- und Hintergrundfarbe für Zeilen mit gerader Nummer verwenden.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den optionalen Parameter * geben Sie in *Objekt* ein Datenfeld oder eine Variable an. In diesem Fall geben Sie anstatt eines Strings eine Referenz auf das Datenfeld oder die Variable an (nur Datenfeld- oder Variablenobjekte). Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

Mit *altHintergrFarbe* setzen Sie eine wechselnde Hintergrundfarbe für die Zeilen mit gerader Nummer. Dann wird *altHintergrFarbe* nur für die Hintergrundfarbe der Zeilen mit ungerader Nummer verwendet. Wechselnde Farben machen Arrays leichter lesbar.
Definiert *Objekt* die Listbox, gilt die wechselnde Farbe für das gesamte Objekt, definiert *Objekt* eine Spalte, gilt sie nur für die angegebene Spalte.

In *Vordergrundfarbe*, *Hintergrundfarbe* und *altHintergrFarbe* geben Sie die RGB-Werte an. Ein RGB Wert ist eine lange Ganzzahl mit 4-byte im Format (*0x00RRGGBB*). Bytes werden von 0 bis 3 von rechts nach links numeriert:

Byte	Beschreibung
3	Muss bei absoluter RGB Farbe Null sein
2	Rote Komponente (0..255)
1	Grüne Komponente (0..255)
0	Blaue Komponente (0..255)

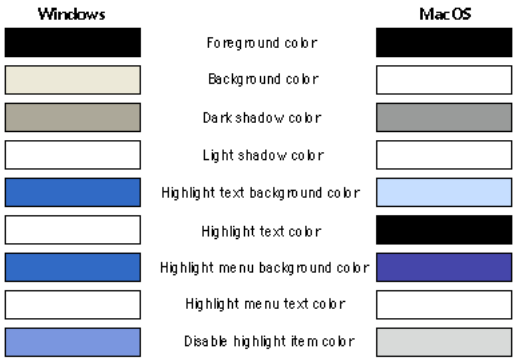
Folgende Tabelle zeigt einige Beispiele für RGB Farben:

Wert	Beschreibung
0x00000000	Schwarz
0x00FF0000	Hellrot
0x0000FF00	Hellgrün
0x000000FF	Hellblau
0x007F7F7F	Grau
0x00FFFF00	Hellgelb
0x00FF7F7F	Pastellrot
0x00FFFFFF	Weiß

Als Alternative können Sie aber auch eine der Farben verwenden, die beim Zeichnen von Objekten automatisch gesetzt werden. 4D bietet folgende vordefinierte Konstanten unter dem Thema **Farben**:

Konstante	Typ	Wert	Kommentar
Background color	Lange Ganzzahl	-2	
Background color none	Lange Ganzzahl	-16	Diese Konstante lässt sich in den Parametern *Hintergrundfarbe* und *altHintergrFarbe* verwenden.
Dark shadow color	Lange Ganzzahl	-3	
Disable highlight item color	Lange Ganzzahl	-11	
Foreground color	Lange Ganzzahl	-1	
Highlight menu background color	Lange Ganzzahl	-9	
Highlight menu text color	Lange Ganzzahl	-10	
Highlight text background color	Lange Ganzzahl	-7	
Highlight text color	Lange Ganzzahl	-8	
Light shadow color	Lange Ganzzahl	-4	

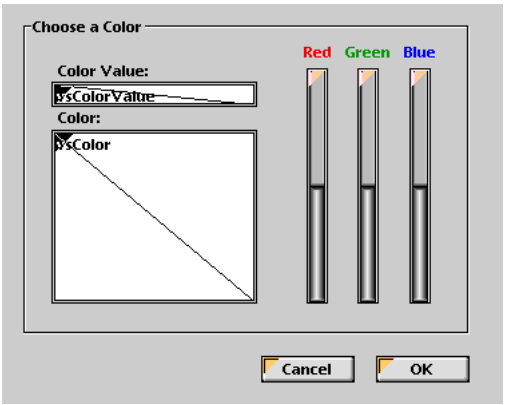
Auf einem Standardsystem können Sie z.B. folgende Farben für Objekte vom Typ eingebbares Feld oder Variable erhalten:



WARNUNG: Diese automatischen Farben hängen vom System ab sowie vom Objekttyp, dem sie zugewiesen sind. Je nach Version des Betriebssystems oder bei eigenen Anpassungen der Systemfarben passt 4D die automatischen Farben entsprechend an. Verwenden Sie die Werte für automatische Farben, um Objekte mit Systemfarben und nicht mit den oben aufgeführten Beispielfarben zu versehen.

Beispiel 1

Dieses Formular enthält zwei nicht eingebbare Variablen *vsColorValue* und *vsColor* sowie drei Thermometer: *thRed*, *thGreen* und *thBlue*.



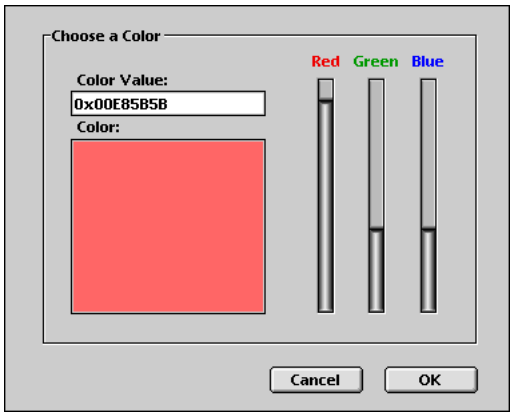
Für diese Objekte gelten folgende Methoden:

```
` Objektmethode für nicht eingebbaren Wert vsColorValue
Case of
   :(Form event=On Load)
      vsColorValue:="0x00000000"
End case
` Objektmethode für nicht eingebbare Variable vsColor
Case of
   :(Form event=On Load)
      vsColor:=""
      OBJECT SET RGB COLORS(vsColor;0x00FFFFFF;0x0000)
End case

` Objektmethode für Thermometer thRed
CLICK IN COLOR THERMOMETER

` Objektmethode für Thermometer thGreen
CLICK IN COLOR THERMOMETER

` Objektmethode für Thermometer thBlue
CLICK IN COLOR THERMOMETER
```

Die Thermometer rufen folgende Projektmethode auf:

```
` Projektmethode CLICK IN COLOR THERMOMETER
OBJECT SET RGB COLORS(vsColor;0x00FFFFFF;(thRed<<16)+(thGreen<<8)+thBlue)
vsColorValue:=String((thRed<<16)+(thGreen<<8)+thBlue;"&x")
If(thRed=0)
   vsColorValue:=Substring(vsColorValue;1;2)+"0000"+Substring(vsColorValue;3)
End if
```

Beachten Sie, dass zum Berechnen der Farbwerte aus dem Thermometer **Bit Operatoren** verwendet werden.
In der Design- oder Anwendungsumgebung sieht das Formular folgendermaßen aus:



Beispiel 2

In transparenten Hintergrund mit heller Schriftfarbe wechseln:


```
OBJECT SET RGB COLORS(*;"myVar";Light shadow color;Background color none)
```

OBJECT SET SCROLL POSITION

OBJECT SET SCROLL POSITION (* ; Objekt {; vPosition {; hPosition}}{; *})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (string) Ohne *: Objekt ist Tabelle oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Tabelle (ohne *)
vPosition	Lange Ganzzahl	➡	Anzuzeigende Zeilennummer (Listbox) oder vertikales Scrollen in Pixel (Bilder)
hPosition	Lange Ganzzahl	➡	Anzuzeigende Spaltennummer (Listbox) oder horizontales Scrollen in Pixel (Bilder)
*	Operator	➡	Anzeige der Zeile (und Spalte, wenn der Parameter hPosition übergeben wurde) an der ersten Position nach dem Scrollen

Beschreibung

Der Befehl **OBJECT SET SCROLL POSITION** scrollt den Inhalt verschiedener Objekttypen: Zeilen in einem Unterformular, einem Listenformular, das über die Befehle **MODIFY SELECTION**, **DISPLAY SELECTION** angezeigt wird, oder eine hierarchische Liste Zeilen oder Spalten einer Listbox, und sogar Pixel eines Bildes.

Hinweis: Das Scrollen per Programmierung bleibt auch bei ausgeblendeten Rollbalken im Formular möglich.

Geben Sie den ersten optionalen Parameter * an, definieren Sie, dass der Parameter *Objekt* der Name eines Unterformulars, einer hierarchischen Liste, Listbox oder Variablen/ Feld vom Typ Bild ist. In diesem Fall übergeben Sie einen String in *Objekt*. Ohne Angabe definieren Sie, dass der Parameter *Objekt* eine Tabelle (Listenformular oder Untertabelle) eine Variable (ListRef einer hierarchischen Liste, Listbox oder Bild) oder ein Feld ist.

Mit dem Parameter *vPosition* können Sie die Nummer der anzuzeigenden Zeile angeben oder bei einem Bild die vertikale Koordinate der anzuzeigenden Pixel.
Geben Sie den Parameter *vPosition* nicht an, löst der Befehl vertikales Scrollen der Zeilen in der Liste aus, so dass die erste hervorgehobene Zeile in der Liste sichtbar ist. Wurde keine Zeile ausgewählt oder ist mindestens eine ausgewählte Zeile bereits sichtbar, wird kein vertikales Scrollen ausgeführt.
Geben Sie diesen Parameter an, löst der Befehl vertikales Scrollen der Zeilen in der Liste aus, so dass die gesetzte Zeile (hervorgehoben oder nicht) in der Liste sichtbar ist. Ist die Zeile bereits sichtbar, führt der Befehl nichts aus, außer der zweite Parameter * ist übergeben.

- Für Listen- und Unterformulare ist diese Nummer die Zeilennummer in der aktuellen Auswahl, d.h. ihre Position.
- Bei hierarchischen Listen berücksichtigt der Befehl den Status auf-/zugeklappt der Einträge.
- Für Listboxen ist es die Zeilennummer unter allen Objektzeilen, inkl. ausgeblendeter Zeilen. Gehört die in *vPosition* übergebene Nummer zu einer ausgeblendeten Zeile in der Listbox, zeigt der Befehl die erste sichtbare Zeile an.
  Hinweis: Beachten Sie, dass dieser Befehl immer auf der Standardanzeige (nicht-hierarchisch) der Listbox basiert, selbst wenn diese im hierarchischen Modus angezeigt wird. Deshalb kann das Ergebnis unterschiedlich sein, je nachdem ob die Listbox im standardmäßigen oder hierarchischen Modus angezeigt wird (siehe Beispiel).
- Für Bilder, die in einem Formular angezeigt werden, gibt *vPosition* den vertikalen Koordinatenpunkt für das Bild an. Übergeben Sie 0 in *vPosition*, wenn das Bild nicht vertikal gescrollt werden soll. Der Wert muss in Pixel in Relation zum Ursprung des Bildes angegeben werden. Erscheint der vertikale Koordinatenpunkt bereits im Objekt, führt der Befehl nichts aus (außer, der 2. Parameter * ist übergeben - siehe unten). Das Bild muss im Format "Abgeschnitten (nicht-zentriert)" erscheinen.

Der Parameter *hPosition* wird für Listboxen oder Bilder verwendet.

- Für Listboxen übergeben Sie in *hPosition* eine Spaltennummer. Dann wird in der Listbox horizontal gescrollt, bis diese Spalte erscheintt. Ist die Spalte bereits sichtbar, führt der Befehl nichts aus. Übergeben Sie den zweiten optionalen Parameter *, wird die sichtbar gemachte Spalte an erste Stelle gesetzt (siehe unten).
- Für Bilder, die in einem Formular angezeigt werden, gibt *hPosition* den horizontalen Koordinatenpunkt für das Bild an. Der Wert muss in Pixel in Relation zum Ursprung des Bildes angegeben werden. Erscheint der horizontale Koordinatenpunkt bereits im Objekt, führt der Befehl nichts aus (außer, der 2. Parameter * ist übergeben - siehe unten).

Geben Sie den zweiten optionalen Parameter * an, gilt folgendes:

- wird die über den Befehl sichtbar gemachte Zeile – sofern die Liste gescrollt wurde – an die erste Position der Liste gesetzt. Liegt die Zeile am Ende der Liste, hat diese Option keine Auswirkung.
- Bilder werden in Bezug auf ihre aktuelle Position und nicht in Bezug auf ihren Ursprung gescrollt.

Hinweis: Der Befehl **HIGHLIGHT RECORDS** enthält den optionalen Parameter *, über den Sie das Verwalten von Scrollen an den Befehl **OBJECT SET SCROLL POSITION** abgeben können.

Beispiel 1

Dieses Beispiel zeigt die unterschiedliche Funktionsweise des Befehls, je nachdem ob die Listbox im standardmäßigen oder hierarchischen Modus angezeigt wird:

```
OBJECT SET SCROLL POSITION(*;"mylistbox";4;2;*)  // zeigt die 4. Zeile der 2. Spalte der Listbox in der ersten Position an.
```

Wird diese Anweisung auf eine Listbox im Standardmodus angewandt:

Deutschland	Bayern	München	1300000	...
Deutschland	Bayern	Freising	45000	...
Deutschland	Bayern	Rosenheim	65000	...
Deutschland	**Hessen**	Frankfurt	675000	...
Deutschland	Hessen	Darmstadt	138000	...
...	...	...	...	...

... scrollen die Zeilen und Spalten der Listbox folgendermaßen:

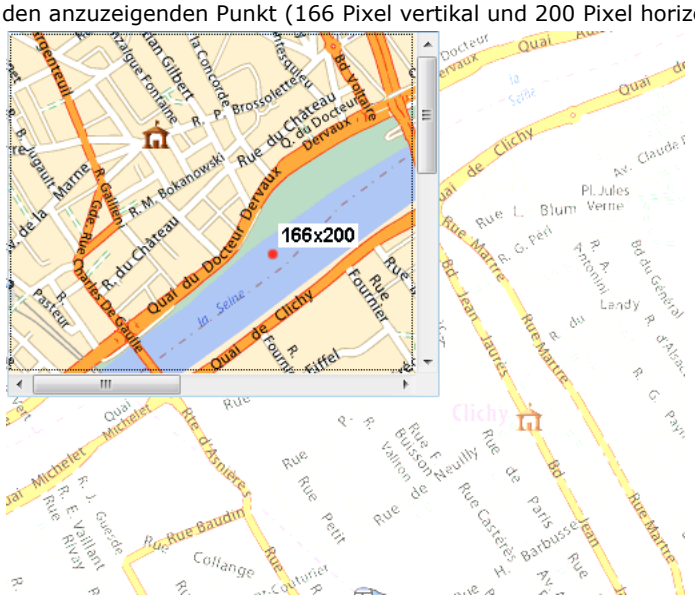
Hessen	Frankfurt	675000	...	...
Hessen	Darmstadt	138000	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...

Wird dieselbe Anweisung auf eine Listbox im hierarchischen Modus angewandt, scrollen die Zeilen, aber nicht die Spalten, da die 2. Spalte zur Hierarchie gehört:

V Hessen	
Frankfurt	675000
Darmstadt	138000
...	

Beispiel 2

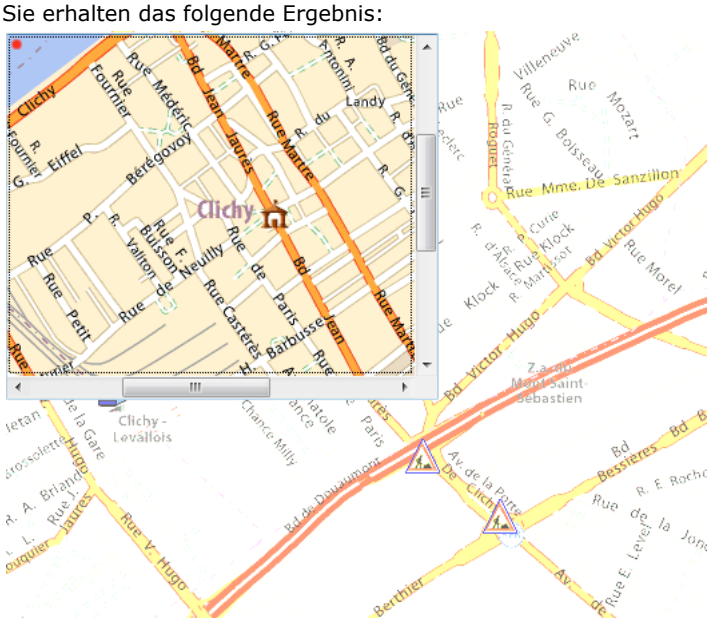
Sie wollen ein Bild scrollen, das in einer Formularvariable enthalten ist. Diese Montage zeigt den sichtbaren Teil des Bildes, sowie den anzuzeigenden Punkt (166 Pixel vertikal und 200 Pixel horizontal):



Um den sichtbaren Teil zu scrollen und die Position des roten Punkts beizubehalten, schreiben Sie folgende Anweisung:

```
OBJECT SET SCROLL POSITION(*;"myVar";166;200;*)
```

Sie erhalten das folgende Ergebnis:



Achten Sie darauf, dass in diesem Fall der zweite Parameter * übergeben ist, da sonst das Bild nicht scrollt, weil der definierte Punkt bereits angezeigt wird.

OBJECT SET SCROLLBAR

OBJECT SET SCROLLBAR ({* ;} Objekt ; Horizontal ; Vertikal)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Horizontal	Boolean, Lange Ganzzahl	➡	True = Anzeigen, False = Ausblenden
Vertikal	Boolean, Lange Ganzzahl	➡	True = Anzeigen, False = Ausblenden

Beschreibung

Der Befehl **OBJECT SET SCROLLBAR** zeigt in der Listbox, definiert durch *Objekt* und den Parameter *, den horizontalen/vertikalen Rollbalken an oder blendet ihn aus.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Objekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie ein Feld oder eine Variablenreferenz (nur Objektfeld oder -variable) anstelle eines String. Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

In *horizontal* und *vertikal* übergeben Sie Werte zum Anzeigen der passenden Rollbalken. Sie können Boolean Werte (*True* = angezeigt, *False* = ausgeblendet) oder eine Zahl (0=ausgeblendet, 1=angezeigt, 2=automatisch) übergeben. Mit dem Wert *2=automatisch* werden Rollbalken nur bei Bedarf angezeigt.

Nachfolgende Tabelle zeigt die möglichen Werte für *horizontal* und *vertikal*. Der automatische Modus ist nicht für alle Objekte verfügbar:

Objekte mit Rollbalken	Rollbalken ausblenden	Rollbalken einblenden	Automatischer Modus
Textobjekt Feld und Variable	False oder 0	True oder 1	*nicht verfügbar*
Bildobjekt Feld und Variable	False oder 0	True oder 1	2
Listboxen	False oder 0	True oder 1	2
Hierarchische Listen	False oder 0	True oder 1	2
Unterformulare	False oder 0	True oder 1	*nicht verfügbar*

Rollbalken werden standardmäßig angezeigt.

Hinweis: Weitere Informationen zum automatischen Modus finden Sie im Abschnitt **Rollbalken**

OBJECT SET SHORTCUT

OBJECT SET SHORTCUT ({* ;} Objekt ; Taste {; Zusatztasten})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Taste	String	➡	dem Objekt zugewiesene Taste
Zusatztasten	Lange Ganzzahl	➡	ein oder mehrere kombinierte Kürzel für Zusatztasten

Beschreibung

Der Befehl **OBJECT SET SHORTCUT** setzt oder ändert dynamisch das Tastenkürzel für das bzw. die Objekte, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable oder ein Feld. In diesem Fall übergeben Sie eine Referenz anstelle eines Namens.

Im Parameter *Taste* übergeben Sie einen String für das Tastenkürzel, das dem Objekt zugeordnet werden soll. Sie können folgendes übergeben:

- ein Standardkürzel, zum Beispiel "B"
- eine Konstante (oder ihren Wert) unter dem Thema **Tastaturabkürzungen**:

Konstante	Typ	Wert
Shortcut with Backspace	Zeichenkette	[backspace]
Shortcut with Carriage Return	Zeichenkette	[return]
Shortcut with Delete	Zeichenkette	[del]
Shortcut with Down arrow	Zeichenkette	[down arrow]
Shortcut with End	Zeichenkette	[end]
Shortcut with Enter	Zeichenkette	[enter]
Shortcut with Escape	Zeichenkette	[esc]
Shortcut with F1	Zeichenkette	[F1]
Shortcut with F10	Zeichenkette	[F10]
Shortcut with F11	Zeichenkette	[F11]
Shortcut with F12	Zeichenkette	[F12]
Shortcut with F13	Zeichenkette	[F13]
Shortcut with F14	Zeichenkette	[F14]
Shortcut with F15	Zeichenkette	[F15]
Shortcut with F2	Zeichenkette	[F2]
Shortcut with F3	Zeichenkette	[F3]
Shortcut with F4	Zeichenkette	[F4]
Shortcut with F5	Zeichenkette	[F5]
Shortcut with F6	Zeichenkette	[F6]
Shortcut with F7	Zeichenkette	[F7]
Shortcut with F8	Zeichenkette	[F8]
Shortcut with F9	Zeichenkette	[F9]
Shortcut with Help	Zeichenkette	[help]
Shortcut with Home	Zeichenkette	[home]
Shortcut with Left arrow	Zeichenkette	[left arrow]
Shortcut with Page down	Zeichenkette	[page down]
Shortcut with Page up	Zeichenkette	[page up]
Shortcut with Right arrow	Zeichenkette	[right arrow]
Shortcut with Tabulation	Zeichenkette	[tab]
Shortcut with Up arrow	Zeichenkette	[up arrow]

Im Parameter *Zusatztasten* übergeben Sie ein oder mehrere Kürzel, die dem Tastenkürzel zugeordnet werden soll. Im Parameter *Zusatztasten* übergeben Sie eine oder mehrere Konstanten vom Typ "Mask" unter dem Thema **Ereignisse (Zusatztasten)**:

Konstante	Typ	Wert	Kommentar
Command key mask	Lange Ganzzahl	256	Strg-Taste unter Windows, Befehlstaste auf OS X
Control key mask	Lange Ganzzahl	4096	Ctrl-Taste auf OS X, oder rechter Mausklick unter Windows und OS X
Option key mask	Lange Ganzzahl	2048	Windows = Alt-Taste, Mac OS = Wahltaste
Shift key mask	Lange Ganzzahl	512	Windows und Mac OS

Hinweis: Lassen Sie den Parameter *Zusatztasten* weg, wird das Objekt aktiviert, sobald Sie auf die definierte Taste drücken. Haben Sie z.B. die Taste "H" einer Schaltfläche zugewiesen, wird diese aktiviert, sobald Sie auf die Taste H drücken. Diese Funktionsweise sollte für spezifische Oberflächen reserviert sein.

Beispiel

Sie wollen je nach der aktuellen Sprache der Anwendung ein anderes Tastenkürzel zuordnen. Dazu schreiben Sie im Formulareereignis On Load form:

```
Case of
	vLang="FR"
	OBJECT SET SHORTCUT(*;"SortButton";"T";Command key mask+Shift key mask)  // Ctrl+Shift+T in Französisch
	vLang="US"
	OBJECT SET SHORTCUT(*;"SortButton";"O";Command key mask+Shift key mask)  // Ctrl+Shift+O in Englisch
End case
```

OBJECT SET STYLE SHEET

OBJECT SET STYLE SHEET ({* ;} Objekt ; StilvorlageName)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
StilvorlageName	Text	➡	Name der Stilvorlage

Beschreibung

Der Befehl **OBJECT SET STYLE SHEET** ändert für den aktuellen Prozess die zugewiesene Stilvorlage für das bzw. die Objekte, definiert über die Parameter Objekt und *. Eine Stilvorlage ändert Schriftart, Schriftgröße und mit Ausnahme von automatischen Stilvorlagen auch den Schriftstil.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Im Parameter *StilvorlageName* übergeben Sie den Namen der Stilvorlage für *Objekt*. Sie können folgendes übergeben:

- Name einer vorhandenen Stilvorlage. Falls sie nicht existiert, wird ein Fehler zurückgegeben, den sie über eine Methode abfangen können, die mit **ON ERR CALL** installiert wird.
- Einen leeren String (""), so dass in *Objekt* keine Stilvorlage vergeben wird
- Eine der folgenden Konstanten aus dem Thema **Schriftstile**, um eine automatische Stilvorlage anzuwenden:

Konstante	Typ	Wert	Kommentar
Automatic style sheet	Zeichenkette	__automatic__	Wird standardmäßig für alle Objekte verwendet
Automatic style sheet_additional	Zeichenkette	__automatic_additional_text__	Gilt nur für statischen Text, Felder und Variablen für Zusatztext in Dialogfenstern.
Automatic style sheet_main	Zeichenkette	__automatic_main_text__	Gilt nur für statischen Text, Felder und Variablen für Haupttext in Dialogfenstern.

Wurde für das Objekt bereits eine Stilvorlage im Designmodus zugewiesen, ersetzt sie dieser Befehl für den aktuellen Prozess.

Verwenden Sie während der Sitzung in *Objekt* die Befehle **ST SET ATTRIBUTES**, **ST SET TEXT** oder **OBJECT SET FONT** um die Schrift oder Schriftgröße zu verändern, wird die Referenz auf die Stilvorlage automatisch vom Objekt gelöscht -- auch wenn Sie die gleichen Attribute wie in der Stilvorlage zuweisen.
Verändern Sie dagegen den Stil, also fett, kursiv, etc., z.B. über die Befehle **ST SET ATTRIBUTES** oder **OBJECT SET FONT STYLE**, werden die neuen Eigenschaften für die Dauer der Sitzung der Stilvorlage zugewiesen.

OBJECT SET SUBFORM

OBJECT SET SUBFORM ({* ;} Objekt {; Tabellenname}; SeitenUnterformular {; ListenUnterformular})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Tabellenname	Tabelle	➡	Tabelle des Formulars (wenn Tabellenformular)
SeitenUnterformular	Text, Objekt	➡	Name (Text) des Seitenformulars oder POSIX Pfad (Text) zu einer .json Datei mit Beschreibung des Seitenformulars, oder Objekt mit Beschreibung des Seitenformulars des Unterformulars.
ListenUnterformular	Text, Objekt	➡	Name (Text) des Listenformulars oder POSIX Pfad (Text) zu einer .json Datei mit Beschreibung des Listenformulars oder Objekt mit Beschreibung des Listenformulars des Unterformulars (Tabellenformular).

Beschreibung

Der Befehl **OBJECT SET SUBFORM** ändert dynamisch das Seitenformular sowie optional, das Listenformular auf dem Bildschirm für das Objekt Unterformular, angegeben in den Parametern *Objekt* und * für den aktuellen Prozess.

Hinweis: Dieser Befehl kann nicht die Art des Unterformulars (Liste oder Seite) ändern. Diese Eigenschaft lässt sich nur im Designmodus setzen.

Übergeben Sie den optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines String.

Im Parameter *Tabellenname* übergeben Sie die Tabelle des entsprechenden Formulars. Dieser Parameter ist optional, Sie können ihn weglassen, wenn Sie ein Projektformular als Seitenunterformular verwenden.

In den Parametern *SeitenUnterformular* und *ListenUnterformular* übergeben Sie entweder:

- Den Namen des Formulars oder
- Den Pfad * (in POSIX Syntax) zu einer gültigen .json Datei mit der Beschreibung des Formulars. Siehe **Dateipfade für Formulare** oder
- Ein Objekt mit der Beschreibung des Formulars. Weitere Informationen dazu finden Sie unter **Dynamische Formulare**.

* Dateipfade in **OBJECT SET SUBFORM** sind, im Gegensatz zu anderen Befehlen für dynamische Formulare, relativ zum Elternformular des Unterformulars**.**

Hinweis: Der Parameter *ListenUnterformular* lässt sich nur übergeben, wenn Sie ein Unterformular vom Typ Liste ändern.

Ändern Sie ein Unterformular vom Typ Seite, lässt sich der Befehl jederzeit ausführen; aktuelle Auswahlen werden nicht geändert. Ein Unterformular vom Typ Liste lässt sich dagegen nur ändern, wenn es die Liste anzeigt. Wird der Befehl ausgeführt, wenn nach einem Doppelklick in die Liste das Seitenformular erscheint, wird ein Fehler generiert.

OBJECT SET TEXT ORIENTATION

OBJECT SET TEXT ORIENTATION ({* ;} Objekt ; Rotation)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Rotation	Lange Ganzzahl	➡	Wert der Objektrotation

Beschreibung

Der Befehl **OBJECT SET TEXT ORIENTATION** ändert für den aktuellen Prozess die Rotation des bzw. der Objekte, definiert über die Parameter *Objekt* und *.

Im Gegensatz zur Eigenschaft "Rotation" im Formulareditor dreht dieser Befehl nur den Text selbst, und nicht das Objekt, in dem der Text liegt. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).

Nur statischer Text sowie nicht-eingebbare Variablen und Felder lassen sich drehen. Wenden Sie diesen Befehl auf ein Objekt an, das keine Textrotation unterstützt, führt der Befehl nichts aus.

Im Parameter *Rotation* übergeben Sie die absolute Drehung für *Objekt*. Sie müssen eine der folgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)** übergeben:

Konstante	Typ	Wert	Kommentar
Orientation 0°	Lange Ganzzahl	0	Keine Rotation (Standardwert)
Orientation 180°	Lange Ganzzahl	180	Text um 180° im Uhrzeigersinn drehen
Orientation 90° left	Lange Ganzzahl	270	Text um 90° gegen den Uhrzeigersinn drehen
Orientation 90° right	Lange Ganzzahl	90	Text um 90° im Uhrzeigersinn drehen

Hinweis: Nur Winkel mit diesen Werten werden unterstützt. Andere Werte werden ignoriert.

Beispiel

Eine Variable im Formular um 270° drehen:

```
OBJECT SET ENTERABLE(*;"myVar";False)
 // mandatory if variable is enterable
OBJECT SET TEXT ORIENTATION(*;"myVar";Orientation 90° left)
```

OBJECT SET THREE STATES CHECKBOX

OBJECT SET THREE STATES CHECKBOX ({* ;} Objekt ; dreiZustände)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
dreiZustände	Boolean	➡	Wahr = Kontrollkästchen mit drei Zuständen, Falsch = Standard Kontrollkästchen

Beschreibung

Der Befehl **OBJECT SET THREE STATES CHECKBOX** ändert im aktuellen Prozess die Eigenschaft "Drei Zustände" für Kontrollkästchen, definiert über die Parameter *Objekt* und *.

Die Option "Drei Zustände" ermöglicht für standardmäßige Kontrollkästchen auch einen alternativen Zustand. Weitere Informationen dazu finden Sie im Abschnitt **Kontrollkästchen mit drei Zuständen** des Handbuchs *4D Designmodus*.
Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Objekt* ein Objektname (String) ist. Ohne diesen Parameter ist *Objekt* ein Feld oder eine Variable. In diesem Fall übergeben Sie eine Feld- oder Variablenreferenz anstelle eines String (nur Feld oder Variablenobjekt).
Dieser Befehl ist nur für Kontrollkästchen mit numerischen Variablen möglich, und nicht für Boolean Felder, die als Kontrollkästchen dargestellt werden.

Im Parameter *dreiZustände* übergeben Sie **Wahr**, um diesen Modus zu aktivieren, **Falsch** zum Deaktivieren.

OBJECT SET TITLE

OBJECT SET TITLE ({* ;} Objekt ; Titel)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Titel	String	➡	Neuer Titel für das Objekt

Beschreibung

Der Befehl **OBJECT SET TITLE** ersetzt den Titel des oder der Objekte, angegeben in *Objekt,* durch den Wert in *Titel*.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den optionalen Parameter * geben Sie in *Objekt* ein Datenfeld oder eine Variable an. In diesem Fall geben Sie anstatt eines Strings eine Referenz auf das Datenfeld oder die Variable an (nur Datenfeld- oder Variablenobjekte). Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

OBJECT SET TITLE gilt nur für einfache Objekte, die einen Titel anzeigen:

- Schaltflächen und 3D Schaltflächen
- Kontrollkästchen und 3D Kontrollkästchen
- Optionsfelder und 3D Optionsfelder
- Statische Textbereiche
- Gruppenboxen

Normalerweise wenden Sie diesen Befehl jeweils nur auf ein Objekt an. Der Titelbereich des Objekts muss für den Text ausreichen, sonst wird der Text abgeschnitten.

Verwenden Sie keine Zeilenschaltung in *Titel*. Wollen Sie mehrzeilige Titel setzen, verwenden Sie als Zeilenschaltung das Zeichen "\" ("\\" im Code-Editor). Diese Möglichkeit besteht für 3D Schaltflächen, 3D Kontrollkästchen, 3D Optionsfelder und Kopfteile in Listboxen.

Hinweis: Wird im Titel das Zeichen "\" verwendet, übergeben Sie im Titel "\\\".

Beispiel 1

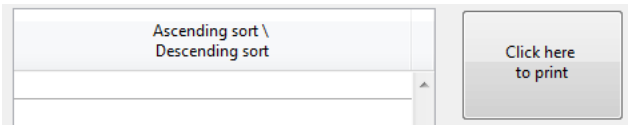
Folgendes Beispiel ist die Objektmethode einer Schaltfläche **Suchen** in einem Ausgabeformular, das mit **MODIFY SELECTION** angezeigt wird. Die Methode sucht eine Tabelle; die Schaltfläche *bDelete* wird je nach Suchergebnis aktiviert oder deaktiviert und der Titel entsprechend geändert:

```
QUERY([People];[People]Name=vName)
Case of
   :(Records in selection([People])=0)  // Keine Person gefunden
      OBJECT SET TITLE(bDelete;" Delete")
      OBJECT SET ENABLED(bDelete;False)
   :(Records in selection([People])=1)  // Eine Person gefunden
      OBJECT SET TITLE(bDelete;"Delete Person")
      OBJECT SET ENABLED(bDelete;True)
   :(Records in selection([People])>1)  // Viele Personen gefunden
      OBJECT SET TITLE(bDelete;"Delete People")
      OBJECT SET ENABLED(bDelete;True)
End case
```

Beispiel 2

Zweizeilige Titel einfügen:

```
OBJECT SET TITLE(*;"header1";"Ascending sort \\\ \\Descending sort")
OBJECT SET TITLE(*;"button1";"Click here \\to print")
```



OBJECT SET VERTICAL ALIGNMENT

OBJECT SET VERTICAL ALIGNMENT ({* ;} Objekt ; Ausrichtung)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Ausrichtung	Lange Ganzzahl	➡	Ausrichtungscode

Beschreibung

Der Befehl **OBJECT SET VERTICAL ALIGNMENT** ändert die Art der vertikalen Ausrichtung für das/die Objekte, angegeben in den Parametern *Objekt* und *.

Übergeben Sie den ersten optionalen Parameter *, ist *Objekt* ein Objektname (String). Ohne diesen Parameter ist *Objekt* eine Variable. In diesem Fall übergeben Sie eine Variablenreferenz anstelle eines Strings.
In *Ausrichtung* übergeben Sie eine der folgenden Konstanten unter dem Thema **Formularobjekte (Eigenschaften)**:

Konstante	Typ	Wert
Align bottom	Lange Ganzzahl	4
Align center	Lange Ganzzahl	3
Align default	Lange Ganzzahl	1
Align top	Lange Ganzzahl	2

Folgende Formularobjekte lassen sich vertikal ausrichten:

- Listboxen
- Spalten von Listboxen
- Kopf- und Fußteile von Listboxen

OBJECT SET VISIBLE

OBJECT SET VISIBLE ({* ;} Objekt ; Sichtbar)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String), Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Feld bzw. Variable (ohne *)
Sichtbar	Boolean	➡	Wahr = sichtbar, Falsch = unsichtbar

Beschreibung

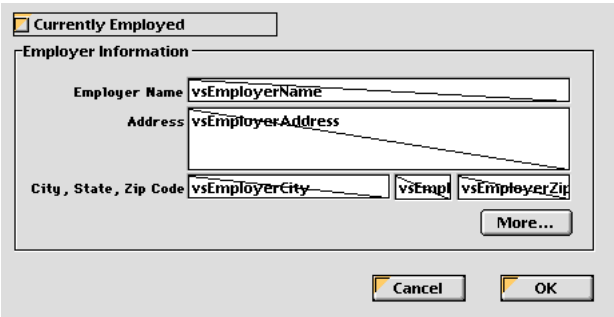
Der Befehl **OBJECT SET VISIBLE** blendet die in *Objekt* angegebenen Formularobjekte ein oder aus.

Mit dem optionalen Parameter * geben Sie in *Objekt* einen Objektnamen (String) an. Ohne den optionalen Parameter * geben Sie in *Objekt* ein Datenfeld oder eine Variable an. In diesem Fall geben Sie anstatt eines Strings eine Referenz auf das Datenfeld oder die Variable an (nur Datenfeld- oder Variablenobjekte). Weitere Informationen dazu finden Sie im Abschnitt **Objekteigenschaften**.

Hat *Sichtbar* den Wert *Wahr*, werden die Objekte eingeblendet. Hat *Sichtbar* den Wert *Falsch*, werden die Objekte ausgeblendet.

Beispiel

Hier sehen Sie ein typisches Formular in der Designumgebung:

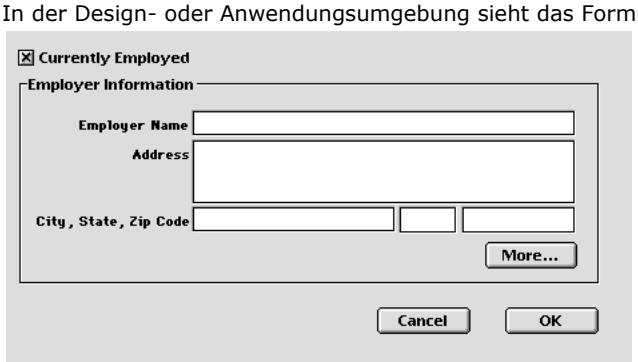


Alle Objekte im Bereich **Employer Information** enthalten den Ausdruck “employer”. Die Objekte sollen sichtbar sein, wenn das Kontrollkästchen **Currently Employed** angekreuzt ist, und nicht sichtbar, wenn das Kontrollkästchen **Currently Employed** nicht angekreuzt ist.

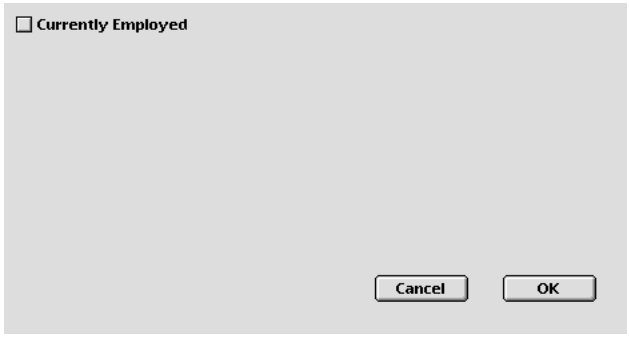
Die Objektmethode für das Kontrollfeld lautet:

```
  ` Objektmethode Kontrollfeld cbCurrentlyEmployed
Case of
   :(Form event=On Load)
      cbCurrentlyEmployed:=1

   :(Form event=On Clicked)
   ` Blende alle Objekte mit der Silbe "emp" ein oder aus
      OBJECT SET VISIBLE(*;"@emp@";cbCurrentlyEmployed # 0)
   ` Das Kontrollfeld soll jedoch immer sichtbar sein
      OBJECT SET VISIBLE(cbCurrentlyEmployed;True)
End case
```

In der Design- oder Anwendungsumgebung sieht das Formular folgendermaßen aus:



oder:


Currently Employed
Cancel
OK

_o_DISABLE BUTTON

_o_DISABLE BUTTON ({* ;} Objekt)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)

Hinweis zur Kompatibilität

Der Befehl **_o_DISABLE BUTTON** ist seit 4D v12 überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Seine allgemeingültige Reichweite, d.h. alle Instanzen der definierten Variable und nicht nur die des aktuellen Formulars, passt nicht zur Reichweite für die Befehle im Kapitel "Objekte".

Wir empfehlen, diesen Befehl durch den Befehl **OBJECT SET ENABLED** zu ersetzen.

_o_ENABLE BUTTON

_o_ENABLE BUTTON ({* ;} Objekt)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist ein Objektname (String) Ohne Stern: Objekt ist eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)

Hinweis zur Kompatibilität

Der Befehl **_o_ENABLE BUTTON** ist in 4D v12 überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Seine allgemeingültige Reichweite, d.h. alle Instanzen der definierten Variable und nicht nur des aktuellen Formulars, passt nicht zur Reichweite für die Befehle im Kapitel "Objekte".

Wir empfehlen, die Befehle **_o_ENABLE BUTTON** und **_o_DISABLE BUTTON** durch die neuen Befehle **OBJECT SET ENABLED** und **OBJECT Get enabled** zu ersetzen.

_o_OBJECT Get action

_o_OBJECT Get action ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Objekt ist Objektname (String) Ohne Stern: Objekt ist Feld oder Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable bzw. Feld (ohne *)
Funktionsergebnis	Text	⮌	Zugewiesene Standardaktion

Hinweis zur Kompatibilität

Diese Funktion ist ab 4D v16 R3 überholt. Sie wird durch die aktualisierte Funktion **OBJECT Get action** ersetzt, die den erweiterten Satz der Namen von Standardaktionen verwendet, der auf Text basiert.

Beschreibung

Der Befehl **_o_OBJECT Get action** gibt den Code der Standardaktion für das bzw. die Objekte zurück, definiert über die Parameter *Objekt* und *.
Hinweis: Konstanten für Aktionen, die auf Code basieren, sind ab 4D v16 R3 überholt. Weitere Informationen dazu finden Sie unter dem Befehl **OBJECT Get action** (aktualisiert).

Objekte (Sprache)

- Struktur der 4D Programmiersprache Objekte
- Objektnotation verwenden
- Shared Objects und Shared Collections
- 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

Struktur der 4D Programmiersprache Objekte

Die Befehle im Kapitel **Objekte (Sprache)** erstellen Daten in Objektformularen und arbeiten damit. Das erweitert die Austauschmöglichkeiten zwischen 4D und jeder Anwendung, die strukturierte Daten unterstützt.

Alle Befehle dieses Kapitels unterstützen folgende 4D Objekte:

- Objektvariablen oder Objektarrays, die mit den Befehlen **C_OBJECT** (Kapitel **Compiler**) oder **ARRAY OBJECT** (Kapitel **Arrays**) erstellt und initialisiert wurden.
- Objektfelder aus der 4D Datenbank. Weitere Informationen dazu finden Sie im Abschnitt **4D Datenfeldtypen**.

Weitere Informationen zur Struktur von "native" 4D Objekten finden Sie im Abschnitt **Objekt** des Kapitels **Datentypen**.

Objektnotation verwenden

Überblick

Sie können 4D Programmiersprache Objekte mit der Objektnotation verwalten, um ihre Werte zu erhalten oder zu setzen. Zur Wahrung der Kompatibilität müssen Sie explizit eine Option zur Kompatibilität aktivieren. Dann können Sie dieses Feature überall in 4D verwenden, wo Ausdrücke erwartet werden.

Wo wird die Objektnotation verwendet?

Jeder Eigenschaftswert, auf den über Objektnotation zugegriffen wird, wird als ein *Ausdruck* gewertet. Ist Objektnotation in Ihrer Anwendung aktiviert (siehe unten), können Sie solche Werte überall verwenden, wo 4D *Ausdrücke* erwartet werden:

- In 4D Code, entweder in den Methoden (Methodeneditor) oder extern geschrieben (Formeln, Dateien mit 4D Tags, die mit **PROCESS 4D TAGS** oder dem Web Server bearbeitet werden, Exportdateien, 4D Write Pro Dokumente...),
- In den Bereichen Ausdruck des Debugger und des Runtime Explorer,
- In der Eigenschaftenliste des Formulareditors für Formularobjekte: Felder vom Typ Variable oder Ausdruck, sowie verschiedene Auswahl Listboxen und Spalten Ausdrücke (Datenquelle, Hintergrundfarbe, Stil oder Schriftfarbe).

Initialisierung

Über Objektnotation verwaltete Objekte müssen initialisiert sein, z.B. über die Funktion **New object**. Sonst wird beim Lesen oder Ändern ihrer Eigenschaften ein Syntaxfehler erzeugt.

Beispiele:

```
C_OBJECT($obVar)  //Ein Objekt vom Typ 4D Variable erstellen.
$obVar:=New object  //Objekt initialisieren und der 4D Variablen zuweisen.
```

Das gleiche Prinzip gilt für Felder vom Typ **Objekt**:

```
CREATE RECORD([Person])  //Einen neuen Datensatz mit einem Objektfeld in der Tabelle hinzufügen.
[Person]Data_o:=New object  //Objekt initialisieren und dem 4D Feld zuweisen.
```

Überblick

Sie können Objekte der 4D Programmiersprache mit Standard **Objektnotation** verwenden, um ihre Werte zu erhalten oder setzen. Mit Objektnotation lässt sich:

- auf Objekteigenschaften über das Symbol "Punkt" (Beispiel: *employee.name*) oder als String in eckigen Klammern (Beispiel: *employee["name"]*) zugreifen,
- auf Elemente einer Collection (siehe **C_COLLECTION**) über eckige Klammern zugreifen (Beispiel: *colRooms[2]*),
- auf eingebundene Unterobjekte und Eigenschaften von Unterobjekten über eine Sequenz von Symbolen zugreifen (Beispiel: *employee.children[2].age*).

Jeder Attributwert, der über Objektnotation zugänglich ist, wird als ein *Ausdruck* gewertet. Ist in Ihrer Anwendung die Objektnotation aktiviert (siehe unten), können Sie solche Werte überall nutzen, wo 4D Ausdrücke erwartet werden:

- In 4D Code, entweder in den Methoden geschrieben (Methodeneditor) oder extern gesetzt (Formeln, Dateien mit 4D Tags, die über den Befehl **PROCESS 4D TAGS** oder den Web Server bearbeitet werden, Exportdateien, 4D Write Pro Dokumente...),
- in Bereichen mit Ausdrücken im Debugger und im Runtime Explorer,
- in der Eigenschaftenliste des Formulareditors für Formularobjekte: Feld vom Typ Variable oder Ausdruck, sowie verschiedene Ausdrücke in Listbox und Spalten vom Typ Auswahl (Datenquelle, Hintergrundfarbe, Stil oder Schriftfarbe).

Hinweis zur Kompatibilität: Da die Objektnotation zum Erkennen von Tokens spezifische Zeichen (".", "[" and "]") verwendet, kann sie nicht in 4D Anwendungen benutzt werden, die diese Symbole in Variablen-, Feldnamen, etc. enthalten. Das führt zu Falschinterpretation des Code und macht ihn ungültig. Aus diesem Grund wurde eine spezifische Option eingerichtet, die Aktivieren der Objektnotation in Ihren Anwendungen verwaltet (siehe unten).

Objekteigenschaften

Objektnotation bietet zwei Wege, um auf Objekteigenschaften zuzugreifen:

- Über das Zeichen "Punkt":

```
object.propertyName
```

Beispiel:

```
employee.name:="Smith"
```

- Über eine Zeichenkette in eckigen Klammern:

```
object["propertyName"]
```

Beispiel:

```
$vName:=employee["name"]
```

Da der Wert einer Objekteigenschaft ein Objekt oder eine Collection sein kann, akzeptiert Objektnotation eine Folge von Symbolen, um auf Untereigenschaften zuzugreifen, wie zum Beispiel:

```
$vAge:=employee.children[2].age
```

Objektnotation ist in allen Elementen der Programmiersprache verfügbar, die ein Objekt enthalten oder zurückgeben, wie z.B.:

- **Die Objekte selbst** (gespeichert in Variablen, Feldern, Objekteigenschaften, Objekt Arrays oder Collection Elemente).
Beispiele:

```
$age:=$myObjVar.employee.age  //Variable
$addr:=[Emp]data_obj.address  //Feld
$city:=$addr.city  //Eigenschaft eines Objekts
$pop:=$aObjCountries{2}.population  //Objekt Array
$val:=$myCollection[3].subvalue  //Collection Element
```

- **4D Befehle**, die Objekte zurückgeben.
Beispiel:

```
$measures:=Get database measures.DB.tables
```

- **Projektmethoden**, die Objekte zurückgeben.
Beispiel:

```
 // MyMethod1
C_OBJECT($0)
$0:=New object("a";10;"b";20)

 //myMethod2
$result:=MyMethod1.a  //10
```

- **Collections**
Beispiel:

```
myColl.length  //Größe der Collection
maxSal:=myColl.max("salary")
```

Collection Elemente und Eigenschaft length

Um auf ein Collection Element zuzugreifen, müssen Sie eine Elementnummer in eckigen Klammern übergeben:

```
collectionName[expression]
```

Hinweis: Weitere Informationen dazu finden Sie im Abschnitt **Collection**.
Sie können jeden gültigen 4D Ausdruck übergeben, der im Parameter *Ausdruck* eine positive Ganzzahl übergibt. Beispiele:

```
myCollection[5]  //auf das 6. Element der Collection zugreifen
myCollection[$var]
```

Hinweis: Beachten Sie, dass Collection Elemente ab 0 nummeriert werden.
Über die Objektnotation können Sie einem Collection Element einen Wert zuweisen:

```
myCol[10]:="My new element"
```

Geht die Elementnummer über das letzte existierende Element der Collection hinaus, wird die Collection automatisch angepasst und alle dazwischenliegenden Elemente erhalten den Wert **null**:

```
C_COLLECTION(myCol)
myCol:=New collection("A";"B")
myCol[5]:="Z"
 //myCol[2]=null
```

```
//myCol[3]=null
//myCol[4]=null
```

Eigenschaft length

Die Eigenschaft **length** ist automatisch für alle Collection verfügbar und gibt die Größe der Collection zurück, z.B. die Anzahl der enthaltenen Elemente. Es gibt zwei Wege, um auf diese Eigenschaft zuzugreifen:

- Über das Zeichen "Punkt", z.B.:

```
$vSize:=myCollection.length
```

- Über eine Zeichenkette in eckigen Klammern, z.B.:

```
$vSize:=myCollection["length"]
```

Beachten Sie, dass sich die Eigenschaft **length** nur lesen lässt, sie kann nicht verändert werden.

Zeiger

Werte von Eigenschaften sind über Zeiger zugänglich. Die Objektnotation mit Zeigern ist ähnlich wie die Objektnotation direkt mit Objekten, der Unterschied ist Weglassen des Zeichens "Punkt".

- Direkter Zugriff:

```
pointerOnObject->propertyName
```

- Zugriff über Name:

```
pointerOnObject->["propertyName"]
```

Beispiel:

```
C_OBJECT(vObj)
C_POINTER(vPtr)
vObj:=New object
vObj.a:=10
vPtr:=->vObj
x:=vPtr->a  //x=10
```

Wert Null

Die Objektnotation unterstützt über die Funktion **Null** den **Nullwert**. Damit können Sie Objekteigenschaften oder Collection Elementen den Nullwert zuweisen oder vergleichen, zum Beispiel:

```
myObject.address.zip:=Null
If(myColl[2]=Null)
```

Weitere Informationen dazu finden Sie unter der Funktion **Null**.

Wert Undefiniert

Das Bewerten einer Objekteigenschaft kann manchmal einen *undefinierten* Wert ergeben. Wenn Sie versuchen, undefinierte Ausdrücke zu lesen oder zuzuweisen, generiert 4D normalerweise Fehler. Das passiert jedoch nicht in folgenden Fällen:

- Die Eigenschaft eines undefinierten Objekts oder Werts lesen gibt undefiniert zurück; Variablen (außer Arrays) einen undefinierten Wert zuweisen hat dieselbe Wirkung, wie **CLEAR VARIABLE** aufrufen:

```
C_OBJECT($o)
C_LONGINT($val)
$val:=10  //$val=10
$val:=$o.a  //$o.a ist undefiniert (kein Fehler), Zuweisen dieses Werts entfernt die Variable
 //$val=0
```

- Die Eigenschaft **length** einer undefinierten Collection lesen ergibt 0:

```
C_COLLECTION($c)  //Variable erstellt, aber keine Collection definiert
$size:=$c.length  //$size = 0
```

- Ein undefinierter Wert in einer Projektmethode wird automatisch in 0 oder "" konvertiert, je nach dem deklarierten Parametertyp.

```
C_OBJECT($o)
mymethod($o.a)  //einen undefinierten Parameter übergeben

 //In der Methode mymethod
```

```
C_TEXT($1) //Parametertyp ist Text
 // $1 enthält ""
```

- Ein bedingter Ausdruck wird automatisch in Falsch konvertiert, wenn er über **If** und **Case of** als undefiniert gewertet wird:

```
C_OBJECT($o)
If($o.a) // falsch
End if
Case of
   :($o.a) // falsch
End case
```

- Einen undefinierten Wert einer vorhandenen Objekteigenschaft zuweisen initialisiert je nach Typ ihren Wert neu oder hebt ihn auf:
  - Objekt, Collection, Zeiger: Null
  - Bild: Leeres Bild
  - Boolean: Falsch
  - String: ""
  - Zahl: 0
  - Datum: !00-00-00! wenn die Einstellung "Verwende Datumstyp anstelle von ISO Datumsformat in Objekten" aktiviert ist, sonst ""
  - Zeit: 0 (Anzahl ms)
  - Undefiniert, Null: keine Änderung

```
C_OBJECT($o)
$o:=New object("a";2)
$o.a:=$o.b //$o.a=0
```

- Einen undefinierten Wert einer nicht vorhandenen Objekteigenschaft zuweisen führt nichts aus.

Erwartet Ihr 4D Code Ausdrücke eines bestimmten Typs, können Sie sicherstellen, dass sie den korrekten Typ haben, selbst wenn sie als undefiniert bewertet werden. Dazu übergeben Sie die passenden 4D Befehle **String**, **Num**, **Time**, **Date**, **Bool**. Sie geben einen leeren Wert des angegebenen Typs zurück, wenn der Ausdruck als undefiniert bewertet wird. Zum Beispiel:

```
$myString:=Lowercase(String($o.a.b)) //Sicherstellen, dass Sie einen Stringwert erhalten, selbst wenn er undefiniert ist
 //um Fehler im Code zu vermeiden
```

Identifier für Objekteigenschaft

Die Namensvergabe für Token, z.B. Namen von Objekteigenschaften, auf die über Objektnotation zugegriffen wird, ist restriktiver als für standardmäßige 4D Objektnamen. Hierfür müssen die Schreibregeln für JavaScript angewandt werden (siehe ECMA Script standard):

- Das erste Zeichen muss ein Buchstabe, ein Unterstrich (_) oder ein Dollarzeichen sein ($).
- Nachfolgende Zeichen können ein Buchstabe, Digit, Unterstrich oder Dollarzeichen sein. Leerzeichen sind NICHT erlaubt.
- Groß- und Kleinschreibung wird berücksichtigt.

Hinweise:

- Die Verwendung eines Tabellenfeldes als Collection Index, wie z.B. a.b[[Table1]Id], ist nicht erlaubt. Sie müssen eine Variable dazwischen setzen.
- Bei Objektattributen, die als String in eckige Klammern gesetzt sind, müssen Sie die ECMA Schreibregeln nicht beachten. Beispiel: Das Attribut $o["My Att"] ist in 4D auch trotz Leerzeichen gültig. Sie können dann jedoch mit diesem Attribut keine Objektnotation verwenden.

Objektnotation aktivieren

Bisher hat 4D in jeder Version in tokenisierten Objektnamen der Anwendung immer Punkte (.) und eckige Klammern ([and]) akzeptiert (Tabellen, Felder, Variablen und Methoden).

Diese Zeichen werden jedoch zum Erkennen von Tokens der Programmiersprache in der standardmäßigen Objektnotation verwendet. Deshalb sind Anwendungen, die in Namen Punkte oder eckige Klammern enthalten, nicht kompatibel mit der Standard Objektnotation, da Falschinterpretation den Code unterbrechen kann. Wird z.B. folgender Code geschrieben:

a.b
a.b:=c[1]

... kann 4D nicht wissen, ob *a.b* und *c[1]* standardmäßige Variablennamen darstellt oder ob *b* eine Eigenschaft von Objekt *a* ist und *c* das zweite Element einer Objekt Collection *c*.

Deshalb gilt folgendes:

- In Anwendungen, die aus Versionen vor 4D v17 konvertiert werden, müssen Sie eine spezifische Option aktivieren, wenn Sie die Objektnotation verwenden wollen. Dadurch deklarieren Sie, dass sich Ihr Code für Objektnotation eignet, da er keine Namen mit den Zeichen "." oder "[]" verwendet.
- Ein spezifisches Feature in MSC hilft Ihnen beim Suchen der Namen, die mit Objektnotation nicht kompatibel sind. Es wird dringend empfohlen, dieses Feature vor Aktivieren der Objektnotation einzusetzen. Weitere Informationen dazu finden Sie

im Abschnitt **Seite Prüfen** des Kapitels "MSC". Und zur Sicherheit sollten Sie generell mit einer Kopie der Strukurdatei arbeiten.
- Die Objektnotation lässt sich nicht rückgängig machen, da dafür Retokenisieren des Code erforderlich ist. Ist sie einmal aktiviert, lässt sie sich nicht aufheben und die Anwendung nicht mehr mit einer vorigen Version öffnen.
- In 4D v17 (ab v16 R4) sind die Zeichen "." und "[]" in Namen von tokenisierten Objekten nicht mehr erlaubt.

Einstellungen zur Kompatibilität

Um Objektnotation in Ihrem Code zu nutzen, müssen Sie in den Datenbank-Eigenschaften auf der Seite **Kompatibilität** die Option **Verwende Objektnotation, um auf Objekteigenschaften zuzugreifen** auswählen:

Diese Einstellung ändert den internen Status Ihrer Strukturdatei und lässt sich nicht rückgängig machen. Weitere Informationen dazu finden Sie auf der **Seite Kompatibilität**.

Hinweis: Komponenten haben eine andere Einstellung von der Host Datenbank.

Beispiele

Der Einsatz von Objektnotation im 4D Code vereinfacht die Verwaltung von Objekten. Beachten Sie, dass die Befehlsnotation weiterhin voll unterstützt wird.

- Objekte schreiben und lesen (dieses Beispiel vergleicht Objektnotation und Befehlsnotation miteinander):

```
 // Objektnotation verwenden
C_OBJECT($myObj) //deklariert eine 4D Variable Objekt
$myObj:=New object //erstellt ein Objekt und weist es der Variablen zu
$myObj.age:=56
$age:=$myObj.age //56

 // Befehlsnotation verwenden
C_OBJECT($myObj2) //deklariert eine 4D Variable Objekt
OB SET($myObj2;"age";42) //erstellt ein Objekt und fügt die Eigenschaft "age" hinzu
$age:=OB Get($myObj2;"age") //42</p><p>// Sie können auch beide Notationen mischen
C_OBJECT($myObj3)
OB SET($myObj3;"age";10)
$age:=$myObj3.age //10
```

- Eine Eigenschaft erstellen und Werte, inkl. Objekte, zuweisen:

```
C_OBJECT($Emp)
$Emp:=New object
$Emp.city:="London" //erstellt die Eigenschaft "city" und setzt ihren Wert auf "London"
$Emp.city:="Paris" //ändert die Eigenschaft "city"
$Emp.phone:=New object("office";"123456789";"home";"0011223344")
 //erstellt die Eigenschaft "phone" und setzt ihren Wert auf ein Objekt
```

- Mit Objektnotation ist es ganz einfach, einen Wert in einem Unterobjekt zu erhalten:

```
$vCity:=$Emp.city //"Paris"
$vPhone:=$Emp.phone.home //"0011223344"
```

- Mit dem Operator [ ] können Sie auf Eigenschaften als String zugreifen:

```
$Emp["city"]:="Berlin" //ändert die Eigenschaft "city"
 //das ist hilfreich, um Eigenschaften über Variablen zu erstellen
C_TEXT($addr)
$addr:="address"
For($i;1;4)
   $Emp[$addr+String($i)]:=""
End for
 // erstellt im Objekt $Emp 4 leere Eigenschaften "address1...address4"
```

Shared Objects und Shared Collections

Definition

Shared objects und **shared collections** sind spezifische Objekte und Collections, deren Inhalt zwischen Prozessen geteilt wird. Im Gegensatz zu **Interprozessvariablen** haben "shared objects" und "shared collections" den Vorteil, dass sie mit **Preemptive 4D Prozesse** kompatibel sind: Sie können per Referenz als Parameter an Befehle wie **New process** oder **CALL WORKER** übergeben werden.

"Shared objects/collections" lassen sich in Variablen speichern, die mit Standardbefehlen **C_OBJECT** und **C_COLLECTION** deklariert wurden, müssen aber über spezifische Befehle eine Instanz erhalten:

- Ein "shared object" erstellen Sie mit der Funktion **New shared object**,
- Eine "shared collection" erstellen Sie mit der Funktion **New shared collection**.

Hinweis: "Shared objects/collections" lassen sich als Eigenschaften von standardmäßigen Objekten (not-shared) oder Collections verwenden.

Zum Ändern von "shared object/collection" muss die Struktur **Use...End use** aufgerufen werden. Wird ein Wert von "shared object/collection" nur gelesen, ist **Use...End use** nicht erforderlich.

Ein einmaliger, globaler Katalog, der von der Funktion **Storage** zurückgegeben wird, ist immer in der gesamten Anwendung und ihren Komponenten verfügbar. Darin lassen sich alle "shared objects/collections" speichern.

Shared Objects oder Collections verwenden

Ist mit den Funktionen **New shared object** oder **New shared collection** eine Instanz von *shared object/collection* erstellt, lassen sich ihre jeweiligen Eigenschaften und Elemente in jedem Prozess ändern oder lesen.

Ändern

Sie können *shared objects/collections* folgendermaßen bearbeiten:

- Objekteigenschaften ändern oder entfernen
- In *shared objects* unterstützte Werte hinzufügen oder bearbeiten, inkl. andere *shared objects/collections* (was eine *shared group* erstellt, siehe unten).

Beachten Sie, dass alle Anweisungen zum Ändern in *shared object* oder *collection* in die Struktur **Use...End use** eingebettet sein müssen, sonst wird ein Fehler erzeugt.

```
$s_obj:=New shared object("prop1";"alpha")
Use($s_obj)
    $s_obj.prop1:="omega"
End Use
```

Shared object/collection lässt sich zur selben Zeit immer nur von einem Prozess verändern. **Use** sperrt *shared object/collection* für andere Threads, während das letzte **End use** alle Objekte und Collections entsperrt, die den gleichen Sperrschlüssel teilen (siehe unten **Sperrschlüssel (Locking Identifier)**). Versuchen Sie, ein *shared object/collection* ohne mindest ein **Use...End use** zu ändern, wird ein Fehler generiert.

Ruft ein Prozess **Use...End use** in *shared object/collection* auf, das bereits von einem anderen Prozess benutzt wird, wird er bis zum Entsperren durch **End use** in Wartestellung gesetzt (es wird kein Fehler generiert). Deshalb sollten Anweisungen innerhalb der Struktur **Use...End use** rasch ablaufen und die Elemente so bald wie möglich entsperren und Sie sollten ein *shared object/collection* nicht direkt auf der Oberfläche ändern, also z.B. über ein Dialogfenster.

Shared objects/collections lassen sich auch Eigenschaften oder Elementen von anderen *shared objects/collections* zuweisen. Das erstellt **shared groups**. Eine *shared group* wird automatisch erstellt, wenn ein *shared object/collection* als Eigenschaftswert oder Element eines anderen *shared object/collection* gesetzt wird. Shared groups erlauben das Einbinden von *shared objects/collections*. Dafür gelten jedoch zusätzliche Regeln:

- Der Aufruf von **Use** in *shared object/collection* innerhalb einer Gruppe sperrt die Eigenschaften/Elemente aller *shared objects/collections*, die zur gleichen Gruppe gehören.
- Eine *shared object/collection* kann nur zu einer *shared group* gehören. Versuchen Sie, eine zu einer Gruppe gehörende *shared object/collection* in eine andere Gruppe zu setzen, wird ein Fehler generiert.
- Die Gruppierung von *shared objects/collections* lässt sich nicht wieder auflösen. Gehören sie zu einer *shared group*, bleibt diese Zuordnung während der gesamten Sitzung erhalten. Selbst wenn alle Referenzen eines Objekts bzw. einer Collection aus dem übergeordneten Objekt bzw. der Collection entfernt werden, bleibt diese Gruppierung erhalten.

In Beispiel 2 sehen Sie die Anwendung der Regeln für *shared groups*.

Hinweis: *Shared groups* werden über die interne Eigenschaft *locking identifier* verwaltet. Weitere Informationen dazu finden Sie unten im Abschnitt **Sperrschlüssel (Locking Identifier)**.

Lesen

Eigenschaften oder Elemente von *shared object/collection* lassen sich ohne die Struktur **Use...End use** lesen, selbst wenn *shared object/collection* von einem anderen Prozess benutzt wird.

Sind dagegen mehrere Werte logisch miteinander verbunden, sollte *shared object/collection* aus Konsistenzgründen in der Struktur **Use...End use** gelesen werden.

Duplizieren

Sie können **OB Copy** mit einem *shared object* oder mit einem Objekt aufrufen, das *shared object(s)* als Eigenschaften enthält. Die Funktion gibt jedoch ein standardmäßiges Objekt (not shared) mit den darin enthaltenen Objekten (falls vorhanden) zurück.

Storage

Storage ist ein einmaliges *shared object*, das automatisch in jeder Anwendung und auf jedem Rechner verfügbar ist. Es wird von der Funktion **Storage** zurückgegeben. Sie können es verwenden, um auf alle während der Sitzung definierten *shared objects/collections* zu verweisen, die über jeden preemptive oder standardmäßige Prozesse verfügbar sein sollen.

Beachten Sie, dass das Storage Objekt, im Gegensatz zu den standardmäßigen *shared objects*, keine *shared group* erstellt, wenn *shared objects/collections* als Eigenschaft hinzugefügt werden. Auf diese Weise lässt sich das Storage Objekt ohne Sperren aller verbundenen *shared objects/collections* verwenden.

Weitere Informationen dazu finden Sie unter der Funktion **Storage**.

Beispiel 1

Mehrere Prozesse starten, die eine Inventur von verschiedenen Produkten durchführen und das gleiche *shared object* aktualisieren. Der Hauptprozess erstellt eine Instanz von einem leeren *shared object,* startet dann die anderen Prozesse und übergibt das *shared object* und die zu zählenden Produkte als Parameter:

```
ARRAY TEXT($_items;0)
... //das Array mit den zu zählenden Einträgen füllen
$nbItems:=Size of array($_items)
C_OBJECT($inventory)
$inventory:=New shared object
Use($inventory)
   $inventory.nbItems:=$nbItems
End use

 //Prozesse erstellen
For($i;1;$nbItems)
   $ps:=New process("HowMany";0;"HowMany_"+$_items{$i};$_items{$i};$inventory)
 //per Referenz gesendetes Objekt $inventory
End for
```

In der Methode "HowMany" ist "inventory" ausgeführt und das *shared object* $inventory wird sobald wie möglich aktualisiert:

```
C_TEXT($1)
C_TEXT($what)
C_OBJECT($2)
C_OBJECT($inventory)
$what:=$1  //zur besseren Lesbarkeit
$inventory:=$2

$count:=CountMethod($what)  //Methode zum Zählen der Produkte
Use($inventory)  //shared object verwenden
   $inventory[$what]:=$count   //Ergebnis für diesen Eintrag sichern
End use
```

Beispiel 2

Nachfolgende Beispiele zeigen spezifische Regeln beim Verwalten von **shared groups**:

```
$ob1:=New shared object
$ob2:=New shared object
Use($ob1)
   $ob1.a:=$ob2   //Gruppe 1 wird erstellt
End use

$ob3:=New shared object
$ob4:=New shared object
Use($ob3)
   $ob3.a:=$ob4   //Gruppe 2 wird erstellt
End use

Use($ob1)  //Ein Objekt aus Gruppe 1 verwenden
   $ob1.b:=$ob4   //ERROR
 //$ob4 gehört bereits zu einer anderen Gruppe
 //Die Zuweisung ist nicht erlaubt
End use

Use($ob3)
```

```
    $ob3.a:=Null  //Jede Referenz auf $ob4 aus Gruppe 2 entfernen
End use

Use($ob1)  //Ein Objekt aus Gruppe 1 verwenden
    $ob1.b:=$ob4   //ERROR
  //$ob4 gehört immer noch zu Gruppe 2
  //Die Zuweisung ist nicht erlaubt
End use
```

Sperrschlüssel (Locking Identifier)

Hinweis: Dieser Abschnitt gibt weitere Informationen über interne Mechanismen und ist nur in ganz bestimmten Fällen nützlich

Jedes neue *shared object/collection* wird mit einem einmaligen **Sperrschlüssel** vom Typ lange Ganzzahl erstellt. Der Anfangswert ist immer negativ, d.h. das ist "single". Wird *shared object/collection* mit "single" Status als Eigenschaft oder Element eines anderen *shared object/collection* gesetzt, werden beide "multiple" und es wird eine shared group erstellt. Sie teilen dann den gleichen Sperrschlüssel: Hinzugefügte Eigenschaften/Elemente erben den Sperrschlüssel des übergeordneten Objekts bzw. der Collection.

Wird ein *shared object/collection* aus dem übergeordneten Objekt bzw. der Collection entfernt:

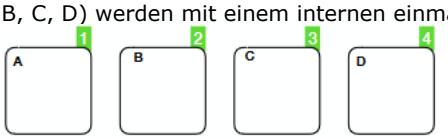
- gelten sie weiterhin als "multiple"
- behalten sie den gleichen Sperrschlüssel bei

Für *shared objects/collections*, die als Eigenschaften oder Elemente anderer *shared objects/collections* gesetzt werden, gelten bestimmte Regeln:

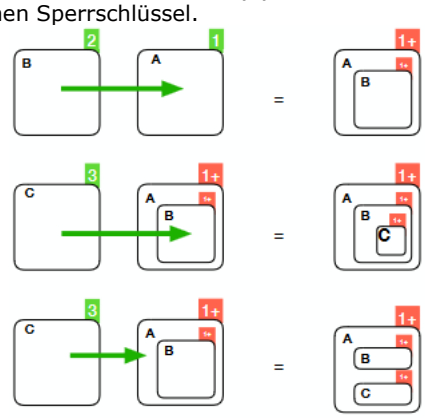
- Ein "single" Objekt/Collection lässt sich an ein anderes "single" oder "multiple" Objekt/Collection anfügen,
- Ein "multiple" Objekt/Collection lässt sich an ein anderes "multiple" Objekt/Collection mit dem gleichen Sperrschlüssel anfügen,
- Versuchen Sie, ein "multiple" Objekt/Collection an ein anderes "single" oder "multiple" Objekt/Collection mit einem anderen Sperrschlüssel anzufügen, tritt ein Fehler auf.

Die folgenden Abbildungen zeigen die verschiedenen Szenarien:

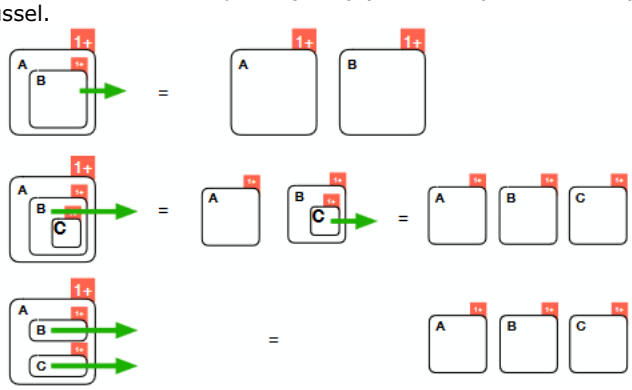
1. *Shared objects* und *shared collections* (A, B, C, D) werden mit einem internen einmaligen Sperrschlüssel (1, 2, 3, 4) erstellt.



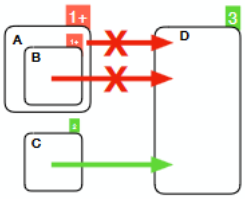
2. Wird innerhalb eines single Objekts (B) auf ein single Objekt (A) verwiesen, werden beide multiple und teilen den gleichen Sperrschlüssel. Weitere single Objekte lassen sich entweder bei (A) oder innerhalb von (B) hinzufügen. Alle Objekte werden als multiple gewertet und teilen den gleichen Sperrschlüssel.



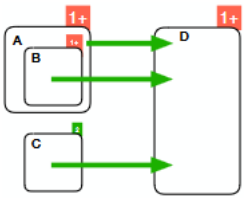
3. Wird ein multiple Objekt (B) von einem anderen multiple Objekt (A) entfernt (dereferenced), bleiben beide multiple und behalten den gleichen Sperrschlüssel.



4. Multiple Objekte (A, B) sind nicht als Eigenschaft eines single Objekts (D) verwendbar, single Objekte (C) lassen sich dagegen verwenden.

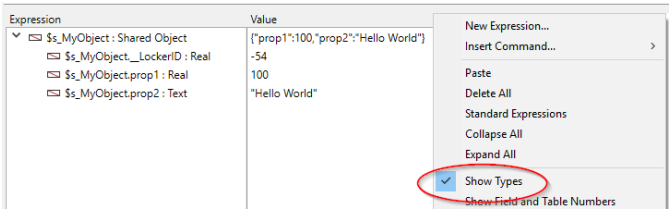


5. Multiple Objekte (A, B) sind als Eigenschaften des Objekts (D) verwendbar, wenn sie den gleichen Sperrschlüssel teilen. Single Objekte (C) lassen sich auch verwenden.



Debugger

Der Sperrschlüssel von *shared objects* und *shared collections* erscheint im Debugger als private Eigenschaft *__LockerID*. Um den Typ "Shared Object" im Debugger anzuzeigen, wählen Sie im Kontextmenü den Eintrag **Show Types**:



Beachten Sie, dass *shared objects* und *shared collections* im Debugger eingebbar sind, solange sie nicht anderswo verwendet werden. In diesem Fall lassen sie sich nicht verändern.

New object

New object {(Eigenschaft ; Wert {; Eigenschaft2 ; Wert2 ; ... ; EigenschaftN ; WertN})} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Eigenschaft	Text	➡	Name der anzulegenden Eigenschaft
Wert	Text, Datum, Boolean, Zeiger, Zahl, Objekt	➡	Wert der Eigenschaft
Funktionsergebnis	Objekt	⮌	Neues Objekt der Programmiersprache

Beschreibung

Die Funktion **New object** erstellt ein neues leeres oder vorab gefülltes Objekt und gibt seine Referenz zurück.

Übergeben Sie keine Parameter, erstellt **New object** ein leeres Objekt und gibt seine Referenz zurück. Sie müssen diese Referenz einer 4D Variablen übergeben, die mit **C_OBJECT** oder über ein 4D Objektfeld deklariert ist.

Hinweis: **C_OBJECT** deklariert eine Variable vom Typ *Objekt*, erstellt aber kein Objekt.

Optional können Sie das neue Objekt vorab füllen, indem Sie ein oder mehrere *Eigenschaft/Wert* Paare als Parameter übergeben:

- Im Parameter *Eigenschaft* übergeben Sie die Bezeichnung der zu erstellenden Eigenschaft. Beachten Sie, dass dieser Parameter zwischen Groß- und Kleinschreibung unterscheidet.
- Im Parameter *Wert* übergeben Sie den Wert, den Sie für die Eigenschaft setzen wollen. Es werden verschiedene Datentypen unterstützt. Es gilt folgendes:
  - ein Zeiger wird unverändert beibehalten; er wird über die Funktion **JSON Stringify** bewertet,
  - Datumsangaben werden als Datumsformat "yyyy-mm-dd" oder als Zeichenkette "YYYY-MM-DDTHH:mm:ss.SSSZ" gespeichert, gemäß der aktuellen Datenbank-Eigenschaft für Datum innerhalb von Objekten (siehe **Seite Kompatibilität**). Werden 4D Datumsangaben vor dem Speichern im Objekt in Text umgewandelt, berücksichtigt das Programm standardmäßig die lokale Zeitzone. Dieses Verhalten können Sie über den Selektor Dates inside objects des Befehls **SET DATABASE PARAMETER** ändern.
  - eine Zeitangabe wird als Anzahl in Millisekunden gespeichert (Zahl).

Beispiel 1

Dieser Code kann leere oder gefüllte Objekte erstellen:

```
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}
```

Beispiel 2

Ein neues Objekt mit einem Objekt als Parameter Wert erstellen:

```
C_OBJECT($Children;$Contact)

 //Ein Objekt Array erstellen
ARRAY TEXT($arrChildren;3)
$arrChildren{1}:="Richard"
$arrChildren{2}:="Susan"
$arrChildren{3}:="James"
OB SET ARRAY($Children;"Children";$arrChildren)

 //Das Objekt initialisieren
$Contact:=New object("FirstName";"Alan";"LastName";"Parker";"age";30;"Children";$Children)
 // $Contact = {FirstName:Alan,LastName:Parker,age:30,Children:{Children:[Richard,Susan,James]}}
```

Beispiel 3

Dieser Code übergibt Objekte als Parameter:

```
C_OBJECT($measures)
$measures:=Get database measures(New object("path";"DB.cacheReadBytes";"withHistory";True;"historyLength";120))
```

Beispiel 4

Mit diesem Code können Sie Objekte in Schleifen einfach verwalten:

```
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 {(Eigenschaft ; Wert {; Eigenschaft2 ; Wert2 ; ... ; EigenschaftN ; WertN})} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Eigenschaft	Text	➡	Name der zu erstellenden Eigenschaft
Wert	Text, Datum, Boolean, Zeiger, Zahl, Objekt	➡	Wert der Eigenschaft
Funktionsergebnis	Objekt	⮌	Neues shared object

Beschreibung

Die Funktion **New shared object** erstellt ein neues leeres oder vorab gefülltes *shared object* und gibt seine Referenz zurück. Ändern oder bearbeiten Sie dieses Objekt, muss es in die Struktur **Use...End use** eingebettet werden, sonst wird ein Fehler erzeugt. Es ist dagegen möglich, eine Eigenschaft ohne Struktur **Use...End use** zu lesen.

Hinweis: Weitere Informationen dazu finden Sie auf der Seite **Shared Objects und Shared Collections**.

Übergeben Sie keine Parameter, erstellt **New shared object** ein leeres Objekt und gibt seine Referenz zurück. Sie müssen diese Referenz einer 4D Variablen zuweisen, die mit dem Befehl **C_OBJECT** deklariert wurde.

Hinweis: Beachten Sie, dass **C_OBJECT** eine Variable vom Typ *Objekt* deklariert, aber kein Objekt erstellt.

Optional können Sie das neue Objekt vorab füllen, indem Sie ein oder mehrere *Eigenschaft*/*Wert* Paare als Parameter übergeben:

- Im Parameter *Eigenschaft* übergeben Sie den Namen der zu erstellenden Eigenschaft (bis zu 255 Zeichen). Beachten Sie, dass *Eigenschaft* zwischen Groß- und Kleinschreibung unterscheidet.
- Im Parameter *Wert* übergeben Sie den Wert, den Sie für die Eigenschaft setzen wollen. Für *Shared objects* werden Werte mit folgendem Typ unterstützt:
  - Numerisch (Zahl, Lange Ganzzahl...) Zahlenwerte werden immer als Zahl gespeichert
  - Text
  - Boolean
  - Datum
  - Zeit (gespeichert als Anzahl Millisekunden - Zahl)
  - Null
  - shared object(*)
  - shared collection(*)

  Hinweis: Im Gegensatz zu standardmäßigen Objekten unterstützen *shared objects* weder Bilder oder Zeiger, noch *not-shared objects/collections*.

  (*) Wird ein *shared object/collection* zu einem anderen *shared object* hinzugefügt, teilen sie sich denselben Sperrschlüssel. Weitere informationen dazu finden Sie im Abschnitt **Sperrschlüssel (Locking Identifier)**.

Beispiel 1

Ein neues vorab gefülltes *shared object* erstellen:

```
C_OBJECT($contact)
$contact:=New shared object("name";"Smith";"firstname";"John")
```

Beispiel 2

Ein *shared object* erstellen und ändern. Dieses Objekt benötigt die Struktur **:**

```
C_OBJECT($s_obj)
$s_obj:=New shared object("prop1";"alpha")
Use($s_obj)
   $s_obj.prop1:="omega"
End use
```

OB Copy

OB Copy (Objekt {; ZeigerAuflösen}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Objekt	Objekt, Objektfeld	➡	Strukturiertes Objekt
ZeigerAuflösen	Boolean	➡	Wahr = Zeiger auflösen Falsch oder weggelassen = Zeiger nicht auflösen
Funktionsergebnis	Objekt	⬅	Kopie von Objekt

Beschreibung

Die Funktion **OB Copy** gibt ein Objekt zurück, das die vollständige Kopie der Eigenschaften, Unterobjekte und Werte für *Objekt* enthält.

Objekt muss zuvor über den Befehl **C_OBJECT** definiert werden oder ein 4D Objektfeld angeben.

Enthält *Objekt* Werte vom Typ Zeiger, enthält die Kopie standardmäßig auch die Zeiger. Sie können jedoch Zeiger beim Kopieren auflösen, wenn Sie im Parameter *ZeigerAuflösen* Wahr übergeben. In diesem Fall wird jeder als Wert in *Objekt* verfügbare Zeiger beim Kopieren bewertet und sein dereferenzierter Wert verwendet.

Hinweis: Sind Eigenschaften von *Objekt* *shared objects* oder *collections*, werden sie in der zurückgegebenen Kopie zu standardmäßigen Objekten oder Collections (not shared).

Beispiel 1

Ein Objekt mit einfachen Werten duplizieren:

```
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)  //direkt kopieren
   APPEND TO ARRAY($arraySel;$ref_value)
 //der Inhalt von $ref_value ist identisch mit dem Inhalt von $Object
   NEXT RECORD([Product])
End while
 //Inhalt von $ref_value
$JsonString:=JSON Stringify array($arraySel)
```

Beispiel 2

Ein Objekt mit Zeigern duplizieren:

```
C_OBJECT($ref)

OB SET($ref;"name";->[Company]name)  //Objekt mit Zeigern
OB SET($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)  //ohne Bewertung der Zeiger kopieren
 // $ref_comp={"name":"->[Company]name","country":"->[Company]Country"}
   $ref_comp2:=OB Copy($ref;True)  //mit Bewertung der Zeiger kopieren
 // $ref_comp={"name":"4D SAS","country":"France"}
   APPEND TO ARRAY($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 (Objekt ; Eigenschaft {; Typ}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Objekt	Objekt, Objektfeld	➡	Strukturiertes Objekt
Eigenschaft	Text	➡	Name der zu lesenden Eigenschaft
Typ	Lange Ganzzahl	➡	Typ, in den der Wert konvertiert werden soll
Funktionsergebnis	Ausdruck	⬅	Aktueller Wert der Eigenschaft

Beschreibung

Die Funktion **OB Get** gibt den aktuellen Wert der Eigenschaft von *Objekt* zurück, das optional in den angegebenen Typ konvertiert wird.

Objekt muss zuvor über den Befehl **C_OBJECT** definiert werden oder ein 4D Objektfeld angeben.

Hinweis: Dieser Befehl unterstützt Attributdefinitionen in 4D Write Pro *Objekten*, wie mit dem Befehl **WP GET ATTRIBUTES** (siehe Beispiel 9).

Im Parameter *Eigenschaft* übergeben Sie die Bezeichnung der einzulesenden Eigenschaft. Beachten Sie, dass *Eigenschaft* Groß- und Kleinschreibung berücksichtigt.

4D gibt den Wert der Eigenschaft standardmäßig im ursprünglichen Typ zurück. Über den Parameter *Typ* können Sie den Typ des zurückgegebenen Werts erzwingen. Dazu übergeben Sie eine der folgenden Konstanten unter dem Thema **Feld und Variablentypen**:

Konstante	Typ	Wert
Is Boolean	Lange Ganzzahl	6
Is collection	Lange Ganzzahl	42
Is date	Lange Ganzzahl	4
Is longint	Lange Ganzzahl	9
Is null	Lange Ganzzahl	255
Is object	Lange Ganzzahl	38
Is picture	Lange Ganzzahl	3
Is pointer	Lange Ganzzahl	23
Is real	Lange Ganzzahl	1
Is text	Lange Ganzzahl	2
Is time	Lange Ganzzahl	11

Die Funktion gibt den Wert von *Eigenschaft* zurück. Es werden mehrere Datentypen unterstützt. Dabei müssen Sie folgendes beachten:

- Übergeben Sie einen Zeiger, bleibt er genauso erhalten; er wird über die Funktion **JSON Stringify** bewertet
- Je nach der aktuellen Einstellung in Ihrer Datenbank wird ein Datum in Objektattributen als Datumstyp oder als Text im ISO Format gespeichert (ab 4D v16 R6). Weitere Informationen dazu finden Sie auf der **Seite Kompatibilität**, Option *Verwende Datumstyp statt ISO Datumsformat in Objekten*. Damit **OB Get** ein Datum korrekt als Text gespeichert interpretiert, müssen Sie die Konstante Is date verwenden (siehe Beispiel 5).
- In Werten vom Typ Zahl ist der Dezimaltrenner immer ein Punkt "."
- Zeit wird als Zahl zurückgegeben. In Objekten wird Zeit standardmäßig in Sekunden gespeichert (siehe nachfolgende Hinweise zur Kompatibilität). Für eine Zeitangabe im 4D Format verwenden Sie die Konstante Is time.

Hinweise zur Kompatibilität:

- In Versionen vor v17 wurden Zeitangaben in Objekten in Millisekunden gespeichert. Zur Wahrung der Kompatibilität lässt sich über den Selector Times inside objects des Befehls **SET DATABASE PARAMETER** das frühere Verhalten einstellen. Das Ergebnis ist unabhängig von der Einstellung korrekt, wenn die Konstante Is time übergeben ist.

- *4D Write Pro*: Definiert *Eigenschaft* ein 4D Write Pro Bildattribut (wie z.B. wk image), wird in Versionen vor v16 R6 immer ein Textwert mit einer Daten URI zurückgegeben. Ab 4D v16 R6 werden 4D Write Pro Bildattribute immer als Bildwerte zurückgegeben. Sie müssen eine spezifische *Eigenschaft* wie wk image url angeben, um eine Daten URI zu erhalten.

- In Versionen vor v16 R4 gibt 4D einen leeren String zurück, wenn *Eigenschaft* einen Nullwert enthält und der Parameter *Typ* nicht verwendet wird. In 4D v16 R4 und neueren Versionen wird in diesem Fall die Konstante Is null zurückgegeben. Zur Wahrung der Kompatibilität tritt diese Änderung nur ein, wenn in den Datenbank-Eigenschaften die Option **Verwende Objektnotation, um auf Objekteigenschaften zuzugreifen** markiert ist. Weitere Informationen dazu finden Sie auf der **Seite Kompatibilität**.

Beispiel 1

Einen Wert vom Typ Text zurückgeben:

```
C_OBJECT($ref)
C_TEXT($FirstName)
OB SET($ref;"FirstName";"Harry")
$FirstName:=OB Get($ref;"FirstName") // $FirstName = "Harry" (text)
```

Beispiel 2

Einen Wert vom Typ Zahl, konvertiert in Ganzzahl, wiederfinden:

```
OB SET($ref ;"age";42)
$age:=OB Get($ref ;"age")  // $ageist eine Ziffer vom Typ Zahl (Standard)
$age:=OB Get($ref ;"age";ls longint)  // $age ist eine Lange Ganzzahl
```

Beispiel 3

Die Werte eines Objekts wiederfinden:

```
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)
```

Beispiel 4

Das Alter eines Angestellten zweimal ändern:

```
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)  //Ein Objekt Array erstellen
APPEND TO ARRAY($myArray;$ref_jim)
 // Das Alter für John von 35 auf 25 setzen
OB SET($myArray{1};"age";25)
 // Das Alter von "John" im Array ersetzen
For($i;1;Size of array($myArray))
   If(OB Get($myArray{$i};"name")="John")
      OB SET($myArray{$i};"age";36)  // instead of 25
 // $ref_john={"name":"John","age":36}
   End if
End for
```

Beispiel 5

Wird ein Datum gefunden, richtet sich der Ergebnisstring nach der aktuellen Datumseinstellung der Datenbank.

- Ist die Option "Verwende Datumstyp statt ISO Datumsformat in Objekten" nicht markiert:

```
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 Zeitzone)
```

- Ist die Option "Verwende Datumstyp statt ISO Datumsformat in Objekten" markiert:

```
C_OBJECT($object)
C_DATE($birthday)
OB SET($object;"Birthday";!30/01/2010!)
$birthday:=OB Get($object;"Birthday")  //30/01/10, ls date ist nicht erforderlich
```

Hinweis: Weitere Informationen dazu finden Sie auf der **Seite Kompatibilität**.

Beispiel 6

Verschachtelte Objekte verwenden:

```
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)
 //or
$childName:=OB Get(OB Get($children;"children");"lastname")
 // $childName = "Monroe" (text)
```

Beispiel 7

In einem Objekt gespeicherte Zeit in 4D wiedergeben:

```
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}
 // Zeit wird in Sekunden gespeichert
$get_h:=OB Get($obj_o;"myHour";Is time)
 // $get_h == ?01:00:01?
```

Beispiel 8

4D Objektfelder verwenden:

```
 // Einen Wert definieren
OB SET([People]Identity_OB;"First name";$firstName)
OB SET([People]Identity_OB;"Last name";$lastName)

 // Einen Wert erhalten
$firstName:=OB Get([People]Identity_OB;"First name")
$lastName:=OB Get([People]Identity_OB;"Last name")
```

Beispiel 9

In der Methode eines Formulars mit einem 4D Write Pro Bereich können Sie schreiben:

```
If(Form event=On Validate)
   OB SET([MyDocuments]My4DWP;"myatt_Last edition by";Current user)
   OB SET([MyDocuments]My4DWP;"myatt_Category";"Memo")
End if
```

Sie können auch eigene Attribute des Dokuments lesen:

```
vAttrib:=OB Get([MyDocuments]My4DWP;"myatt_Last edition by")
```

Beispiel 10

Die Größe eines Bildes erfahren, das in einem Objekt Attribut gespeichert ist:

```
C_LONGINT($vSize)
$vSize:=Picture size(OB Get($object;"photo";Is picture))
```

Hinweis: Weisen Sie das Funktionsergebnis einer Bildvariablen zu, ist die Konstante Is picture nicht notwendig. Beispiel:

```
C_PICTURE($vPict)
$vPict:=OB Get($object;"photo")  //In diesem Fall ist "is picture" nicht nötig.
```

OB GET ARRAY

OB GET ARRAY (Objekt ; Eigenschaft ; Array)

Parameter	Typ		Beschreibung
Objekt	Objekt, Objektfeld	➡	Strukturiertes Objekt
Eigenschaft	Text	➡	Name der zu lesenden Eigenschaft
Array	Array Text, Array Zahl, Array Boolean, Array Objekt, Array Zeiger, Array Lange Ganzzahl	⬅	Wert des Array von Eigenschaft

Beschreibung

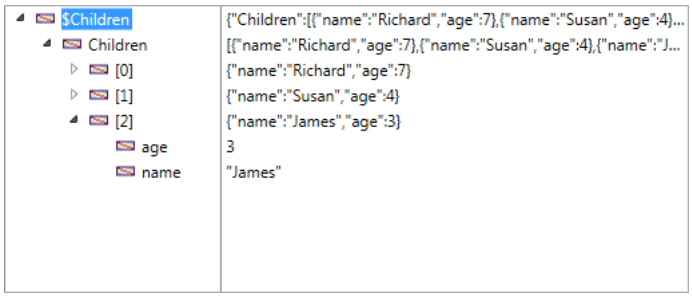
Der Befehl **OB GET ARRAY** findet in *Array* das Array der Werte, gespeichert in der Eigenschaft des Objekts, das im Parameter *Objekt* definiert ist.

Objekt muss zuvor über den Befehl **C_OBJECT** definiert werden oder ein 4D Objektfeld angeben.

Im Parameter *Eigenschaft* übergeben Sie die Bezeichnung der zu lesenden Eigenschaft.Beachten Sie, dass *Eigenschaft* Groß- und Kleinschreibung berücksichtigt.

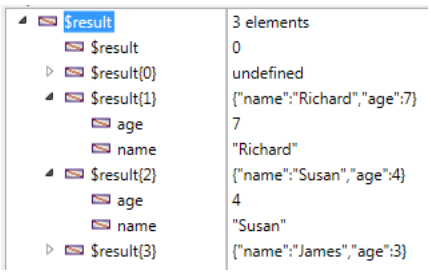
Beispiel 1

Wir nehmen das Array Objekt aus dem Beispiel zum Befehl **OB SET ARRAY**:



Wir wollen diese Werte wiederfinden:

```
ARRAY OBJECT($result;0)
OB GET ARRAY($Children;"Children";$result)
```



Beispiel 2

Wir wollen einen Wert im ersten Element des Array ändern:

```
// Den Wert von "age" ändern:
ARRAY OBJECT($refs)
OB GET ARRAY($refEmployees;"__ENTITIES";$refs)
OB SET($refs{1};"age";25)
```

OB GET PROPERTY NAMES

OB GET PROPERTY NAMES (Objekt ; arrEigenschaften {; arrTypen})

Parameter	Typ		Beschreibung
Objekt	Objekt	➡	Strukturiertes Objekt
arrEigenschaften	Array Text	⬅	Namen der Eigenschaft
arrTypen	Array Lange Ganzzahl	⬅	Typen der Eigenschaft

Beschreibung

Der Befehl **OB GET PROPERTY NAMES** gibt in *arrEigenschaften* die Namen der Eigenschaften im Objekt zurück, das im Parameter *Objekt* definiert ist.

Objekt muss zuvor über den Befehl **C_OBJECT** definiert werden oder ein 4D Objektfeld angeben.

Im Parameter *arrEigenschaften* übergeben Sie ein Array Text. Ist das Array nicht vorhanden, erstellt es der Befehl und passt die Größe automatisch an.

In *arrTypen* können Sie optional auch ein Array Lange Ganzzahl übergeben. Für jedes Element von *arrEigenschaften* gibt der Befehl in *arrTypen* den Typ des Werts zurück, der in der Eigenschaft gespeichert ist. Sie können die empfangenen Werte mit den folgenden Konstanten unter dem Thema **Feld und Variablentypen** vergleichen:

Konstante	Typ	Wert
Is Boolean	Lange Ganzzahl	6
Is collection	Lange Ganzzahl	42
Is null	Lange Ganzzahl	255
Is object	Lange Ganzzahl	38
Is real	Lange Ganzzahl	1
Is text	Lange Ganzzahl	2
Object array	Lange Ganzzahl	39

Hinweis: Für Array Eigenschaften gibt der Befehl Is collection zurück.

Beispiel 1

Testen, ob ein Objekt nicht leer ist:

```
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
```

Beispiel 2

Das Element eines Objekt Array verwenden:

```
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) //zusätzliches Attribut
APPEND TO ARRAY($arrayChildren;$ref_susan)
OB SET($ref_james;"name";"James")
OB SET NULL($ref_james;"age") //Attribut Null
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 (Objekt ; Eigenschaft) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Objekt	Objekt	➡	Strukturiertes Objekt
Eigenschaft	Text	➡	Name der Eigenschaft
Funktionsergebnis	Lange Ganzzahl	⮌	Werttyp der Eigenschaft

Beschreibung

Die Funktion **OB Get type** gibt den Typ des Werts zurück, der *Eigenschaft* von *Objekt* zugewiesen ist.

Objekt muss zuvor über den Befehl **C_OBJECT** definiert werden oder ein 4D Objektfeld angeben.

Im Parameter *Eigenschaft* übergeben Sie die Bezeichnung der Eigenschaft, deren Typ Sie herausfinden wollen. Beachten Sie, dass *Eigenschaft* Groß- und Kleinschreibung berücksichtigt.

Die Funktion gibt eine Lange Ganzzahl zurück, die den Typ des Werts angibt. Sie können die gefundenen Werte mit den folgenden Konstanten unter dem Thema **Feld und Variablentypen** vergleichen:

Konstante	Typ	Wert
Is Boolean	Lange Ganzzahl	6
Is collection	Lange Ganzzahl	42
Is date	Lange Ganzzahl	4
Is null	Lange Ganzzahl	255
Is object	Lange Ganzzahl	38
Is real	Lange Ganzzahl	1
Is text	Lange Ganzzahl	2
Is undefined	Lange Ganzzahl	5

Hinweis: Für Bildattribute gibt die Funktion Is object zurück.

Beispiel

Den Typ der Standardwerte erhalten:

```
C_OBJECT($ref)
OB SET($ref;"name";"smith";"age";42)
$type:=OB Get type($ref;"name")  //$type gibt 2 zurück
$type2:=OB Get type($ref;"age")  //$type2 gibt 1 zurück
```

OB Is defined

OB Is defined (Objekt {; Eigenschaft}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Objekt	Objekt, Objektfeld	➡	Strukturiertes Objekt
Eigenschaft	Text	➡	Wenn übergeben = Eigenschaft prüfen Wenn weggelassen = Objekt prüfen
Funktionsergebnis	Boolean	⮌	Ohne Eigenschaft: Wahr, wenn Objekt definiert ist, sonst Falsch. Mit Eigenschaft: Wahr, wenn Eigenschaft definiert, sonst Falsch

Beschreibung

Die Funktion **OB Is defined** gibt **Wahr** zurück, wenn *Objekt* oder *Eigenschaft* definiert ist, sonst **Falsch**.

Objekt muss zuvor über den Befehl **C_OBJECT** definiert werden oder ein 4D Objektfeld angeben.

Standardmäßig, d.h. ohne den Parameter *Eigenschaft*, prüft die Funktion, ob *Objekt* definiert ist. Ein Objekt ist definiert, wenn sein Inhalt initialisiert ist

Hinweis: Ein Objekt kann definiert sein, jedoch leer. Über die Funktion **OB Is empty** können Sie herausfinden, ob ein Objekt undefiniert oder leer ist.

Übergeben Sie den Parameter *Eigenschaft*, prüft der Befehl, ob diese Eigenschaft in *Objekt* vorhanden ist. Beachten Sie, dass *Eigenschaft* Groß- und Kleinschreibung berücksichtigt.

Beispiel 1

Die Initialisierung des Objekts testen:

```
C_OBJECT($object)
$def:=OB Is defined($object)  // $def=falsch, da $object nicht initialisiert ist

OB SET($object;"Name";"Martin")
OB REMOVE($object;"Name")
$def2:=OB Is defined($object)  // $def2=wahr, da $object leer {}, aber initialisiert ist
```

Beispiel 2

Testen, ob eine Eigenschaft vorhanden ist:

```
C_OBJECT($ref)
OB SET($ref;"name";"smith";"age";42)
 //...
If(OB Is defined($ref;"age"))
 //...
End if
```

Dieser Test ist dasselbe wie:

```
If(OB Get type($Object;"name")#Is undefined)
```

OB Is empty

OB Is empty (Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Objekt	Objekt, Objektfeld	➡	Strukturiertes Objekt
Funktionsergebnis	Boolean	⮌	Wahr, wenn Objekt leer oder undefiniert ist, sonst Falsch

Beschreibung

Die Funktion **OB Is empty** gibt *Wahr* zurück, wenn *Objekt* undefiniert oder leer ist. *Falsch*, wenn *Objekt* definiert (initialisiert) ist und mindestens eine Eigenschaft enthält.

Objekt muss zuvor über den Befehl **C_OBJECT** definiert werden oder ein 4D Objektfeld angeben.

Beispiel

Hier die Ergebnisse dieser Funktion und **OB Is defined**, die je nach Kontext unterschiedlich sind:

```
C_OBJECT($ref)
$empty:=OB Is empty($ref)  // Wahr
$def:=OB Is defined($ref)  // Falsch

OB SET($ref;"name";"Susie";"age";4)
 // $ref="{"name":"Susie","age":4}"
$empty:=OB Is empty($ref)  // Falsch
$def:=OB Is defined($ref)  // Wahr

OB REMOVE($ref;"name")
OB REMOVE($ref;"age")
$empty:=OB Is empty($ref)  // Wahr
$def:=OB Is defined($ref)  // Wahr
```

OB REMOVE

OB REMOVE (Objekt ; Eigenschaft)

Parameter	Typ		Beschreibung
Objekt	Objekt, Objektfeld	➡	Strukturiertes Objekt
Eigenschaft	Text	➡	Name der zu entfernenden Eigenschaft

Beschreibung

Der Befehl **OB REMOVE** entfernt die Eigenschaft des Objekts, definiert im Parameter *Objekt*. Er entfernt die Eigenschaft und den aktuellen Wert.

Objekt muss zuvor über den Befehl **C_OBJECT** definiert oder als 4D Objektfeld zugewiesen werden.

Im Parameter *Eigenschaft* übergeben Sie die Bezeichnung der entsprechenden Eigenschaft. Beachten Sie, dass *Eigenschaft* Groß- und Kleinschreibung berücksichtigt.

Beispiel

Die Eigenschaft "age" eines Objekts entfernen:

```
C_OBJECT($Object)
OB SET($Object;"name";"smith";"age";42;"client";True)
 // $Object={"name":"smith","age":42,"client":true}
OB REMOVE($Object;"age")
 // $Object={"name":"smith","client":true}
```

OB SET

OB SET (Objekt ; Eigenschaft ; Wert {; Eigenschaft2 ; Wert2 ; ... ; EigenschaftN ; WertN})

Parameter	Typ		Beschreibung
Objekt	Objektfeld, Objekt	➡	Strukturiertes Objekt
Eigenschaft	Text	➡	Name der zu setzenden Eigenschaft
Wert	Ausdruck	➡	Neuer Wert der Eigenschaft

Beschreibung

Der Befehl **OB SET** erstellt oder ändert ein oder mehrere Paare Eigenschaft/Wert im Programmiersprache-Objekt, das im Parameter *Objekt* definiert ist.

Objekt muss zuvor über den Befehl **C_OBJECT** definiert werden oder ein 4D Objektfeld angeben.

Im Parameter *Eigenschaft* übergeben Sie die Bezeichnung der Eigenschaft zum Erstellen oder Ändern. Ist die Eigenschaft bereits in *Objekt* vorhanden, wird ihr Wert aktualisiert. Ist sie noch nicht vorhanden, wird sie angelegt.
Beachten Sie, dass *Eigenschaft* Groß- und Kleinschreibung berücksichtigt.

Im Parameter *Wert* übergeben Sie den passenden Wert für die Eigenschaft. Es werden mehrere Datentypen unterstützt. Dabei müssen Sie folgendes beachten:

- Übergeben Sie einen Zeiger, bleibt er genauso erhalten; er wird über die Funktion **JSON Stringify** bewertet
- Übergeben Sie ein Datum, hängt es von der aktuellen Einstellung in Ihrer Datenbank ab, ob es als Datumstyp oder als Text im ISO Format gespeichert wird. Weitere Informationen dazu finden Sie auf der **Seite Kompatibilität**, Option *Verwende Datumstyp statt ISO Datumsformat in Objekten*.
- Übergeben Sie eine Zeit, wird sie in *Objekt* als Zahl in Sekunden (Typ *Zahl*) gespeichert.
- Übergeben Sie ein Programmiersprache-Objekt oder eine Collection, verwendet der Befehl nicht eine Kopie, sondern die Referenz. Jede Änderung im Objekt oder der Collection wird an alle Referenzen weitergegeben.
- Ab 4D v16 R4 können Sie ein Bild jedes unterstützten Typs übergeben. Weitere Informationen dazu finden Sie im Abschnitt **Native Unterstützung von Formaten** .

Beispiel 1

Ein Objekt erstellen und eine Eigenschaft vom Typ Text hinzufügen:

```
C_OBJECT($Object)
OB SET($Object ;"FirstName";"John";"LastName";"Smith")
 // $Object = {"FirstName":"John","LastName":"Smith"}
```

Beispiel 2

Ein Objekt erstellen und eine Eigenschaft vom Typ Boolean hinzufügen:

```
C_OBJECT($Object)
OB SET($Object ;"LastName";"smith";"age";42;"client";True)
 // $Object = {"LastName":"smith","age":42,"client":true}
```

Beispiel 3

Eine Eigenschaft ändern:

```
 // $Object = {"FirstName":"John","LastName":"Smith"}
OB SET($Object ;"FirstName";"Paul")
 // $Object = {"FirstName":"Paul","LastName":"Smith"}
```

Beispiel 4

Eine Eigenschaft hinzufügen:

```
 // $Object = {"FirstName":"John","LastName":"Smith"}
OB SET($Object ;"department";"Accounting")
 // $Object = {"FirstName":"Paul","LastName":"Smith","department":"Accounting"}
```

Beispiel 5

Eine Eigenschaft umbenennen:

```
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}
```

Beispiel 6

Zeiger verwenden:

```
 // $Object = {"FirstName":"Paul","LastName":"Smith"}
C_TEXT($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"}
```

Beispiel 7

Ein Objekt verwenden:

```
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)
```

Sie können einen Wert auch direkt ändern:

```
OB SET($ref_smith ;"name";"Smyth")
 // $ref_smith = {"employee":{"name":"Smyth"}}
$string:=JSON Stringify($ref_emp)
 // $string = "{"employee":{"name":"Smyth"}}"
```

Beispiel 8

Haben Sie das Feld [Rect]Desc als Objektfeld definiert, können Sie schreiben:

```
CREATE RECORD([Rect])
[Rect]Name:="Blue square"
OB SET([Rect]Desc;"x";"50";"y";"50";"color";"blue")
SAVE RECORD([Rect])
```

Beispiel 9

Sie wollen Daten in JSON exportieren. Sie enthalten ein 4D Datum, das in einen String ohne Angabe der Zeitzone konvertiert werden soll. Beachten Sie, dass die Konvertierung passiert, wenn das Datum im Objekt gesichert wird. Deshalb müssen Sie den Befehl **SET DATABASE PARAMETER** vor **OB SET** aufrufen:

```
C_OBJECT($o)
$vDateSetting:=Get database parameter(Dates inside objects)  // Die aktuelle Einstellung sichern
SET DATABASE PARAMETER(Dates inside objects;String type without time zone)
OB SET($o ;"myDate";Current date)  // In JSON konvertieren
$json:=JSON Stringify($o)
SET DATABASE PARAMETER(Dates inside objects;$vDateSetting)
```

Beispiel 10

In der Methode eines Formulars mit einem 4D Write Pro Bereich können Sie schreiben:

```
If(Form event=On Validate)
    OB SET([MyDocuments]My4DWP;"myatt_Last edition by";Current user)
    OB SET([MyDocuments]My4DWP;"myatt_Category";"Memo")
End if
```

Sie können auch eigene Attribute des Dokuments lesen:

```
vAttrib:=OB Get([MyDocuments]My4DWP;"myatt_Last edition by")
```

Beispiel 11

Eine Collection als einen Eigenschaftswert setzen:

```
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)
```

Beispiel 12

Ein Bild in einem Objektfeld speichern:

```
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 (Objekt ; Eigenschaft ; Array)

Parameter	Typ		Beschreibung
Objekt	Objektfeld, Objekt	➡	Strukturiertes Objekt
Eigenschaft	Text	➡	Name der zu setzenden Eigenschaft
Array	Array	➡	Array zum Speichern in Eigenschaft

Beschreibung

Der Befehl **OB SET ARRAY** definiert das Array, das der Eigenschaft im Objekt, definiert im Parameter *Objekt*, zugeordnet ist.

Objekt muss zuvor über den Befehl **C_OBJECT** definiert werden oder ein 4D Objektfeld angeben.

Im Parameter *Eigenschaft* übergeben Sie die Bezeichnung der Eigenschaft zum Erstellen oder Ändern. Ist die Eigenschaft bereits in *Objekt* vorhanden, wird ihr Wert aktualisiert. Ist sie noch nicht vorhanden, wird sie angelegt. Beachten Sie, dass *Eigenschaft* Groß- und Kleinschreibung berücksichtigt.

Im Parameter *Array* übergeben Sie das Array, das als Wert der *Eigenschaft* übergeben werden muss. Es werden mehrere Array-Typen unterstützt, nämlich Zahl, Lange Ganzzahl, Text, Boolean , Objekt, Zeiger. Ab 4D v16 R4 werden Arrays vom Typ Bild unterstützt.

Hinweis: Zweidimensionale Arrays lassen sich hier nicht verwenden.

Beispiel 1

Ein Array Text verwenden:

```
C_OBJECT($Children)
ARRAY TEXT($arrChildren;3)
$arrChildren{1}:="Richard"
$arrChildren{2}:="Susan"
$arrChildren{3}:="James"

OB SET ARRAY($Children;"Children";$arrChildren)
 // Value of $Children = {"Children":["Richard","Susan","James"]}
```

Beispiel 2

In einem Array ein Element hinzufügen:

```
ARRAY TEXT($arrText;2)
$arrText{1}:="Smith"
$arrText{2}:="White"
C_OBJECT($Employees)
OB SET ARRAY($Employees;"Employees";$arrText)
APPEND TO ARRAY($arrText;"Brown")  // Add to the 4D array
 // $Employees = {"Employees":["Smith","White"]}

OB SET ARRAY($Employees;"Employees";$arrText)
 // $Employees = {"Employees":["Smith","White","Brown"]}
```

Beispiel 3

Ein Array Text mit Auswahl eines Elements verwenden:

```
 // $Employees = {"Employees":["Smith","White","Brown"]}
OB SET ARRAY($Employees ;"Manager";$arrText{1})
 // $Employees = {"Employees":["Smith","White","Brown"],"Manager":["Smith"]}
```

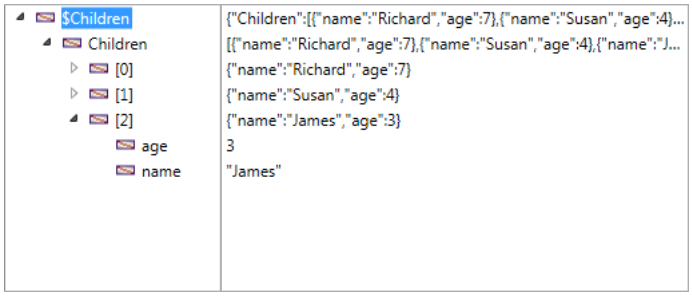
Beispiel 4

Ein Array Objekt verwenden:

```
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}]}
```

Hier sehen Sie, wie das Objekt im Debugger erscheint:



Beispiel 5

Ein Objektfeld verwenden

```
ARRAY TEXT($arrGirls;3)
$arrGirls{1}:="Emma"
$arrGirls{2}:="Susan"
$arrGirls{3}:="Jamie"
OB SET ARRAY([People]Children;"Girls";$arrGirls)
```

Beispiel 6

Ein Array vom Typ Bild verwenden:

```
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 (Objekt ; Eigenschaft)

Parameter	Typ		Beschreibung
Objekt	Objekt, Objektfeld	➡	Strukturiertes Objekt
Eigenschaft	Text	➡	Name der Eigenschaft, für die der Wert Null gültig sein soll

Beschreibung

Der Befehl **OB SET NULL** speichert den Wert **Null** im Programmiersprache-Objekt, das im Parameter *Objekt* definiert ist.

Objekt muss zuvor über den Befehl **C_OBJECT** definiert werden oder ein 4D Objektfeld angeben.

Im Parameter *Eigenschaft* übergeben Sie die Bezeichnung der Eigenschaft, in der Sie den Wert **Null** speichern wollen. Ist die Eigenschaft bereits in *Objekt* vorhanden, wird ihr Wert aktualisiert. Ist sie noch nicht vorhanden, wird sie angelegt.
Beachten Sie, dass *Eigenschaft* Groß- und Kleinschreibung berücksichtigt!

Beispiel

Den Wert Null in der Eigenschaft "age" für Lea setzen:

```
C_OBJECT($ref)
OB SET($ref;"name";"Lea";"age";4)
 // $ref = {"name":"Lea","age":4}
...
OB SET NULL($ref ;"age")
 // $ref = {"name":"Lea","age":null}
```

Storage

Storage -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Objekt		Katalog der eingetragenen shared objects und shared collections

Beschreibung

Die Funktion **Storage** gibt den Katalog der eingetragenen *shared objects* oder *shared collections* zurück, die im Objekt *Storage* auf dem aktuellen Rechner oder der Komponente eingetragen sind.

Der von **Storage** zurückgegebene Katalog wird von 4D automatisch erstellt und ist für alle Prozesse der Anwendung, inkl. preemptive Prozesse verfügbar. Es gibt einen Katalog **Storage** pro Rechner und pro Komponente: In einer Client/Server Anwendung gibt es ein *shared object* **Storage** auf dem Server und ein *shared object* **Storage** auf jeder remote 4D Applikation; verwendet die Anwendung Komponenten, hat jede Komponente ein eigenes *shared object* **Storage**.

Der Katalog **Storage** dient als Referenz auf *shared objects/collections*, die ein preemptive oder standardmäßiger Prozess verwenden soll. Um ein *shared object/collection* im Katalog einzutragen, fügen Sie die jeweilige Referenz ein, die von **Storage** zurückgegeben wird.

Da der von **Storage** zurückgegebene Katalog ein *shared object* ist, folgt er den auf der Seite **Shared Objects und Shared Collections** beschriebenen Regeln. Es gibt jedoch folgende Besonderheiten:

- Dieses Objekt akzeptiert nur *shared objects* und *shared collections*. Versuchen Sie, andere Werte wie standardmäßige Objekte oder Collections, skalare Werte oder Nullwerte hinzuzufügen, wird ein Fehler erzeugt.
- Zum Schreiben muss die Struktur **Use...End use** verwendet werden, sonst wird ein Fehler erzeugt. Lesen ist dagegen ohne Struktur **Use...End use** möglich.
- **Storage** sperrt nur die Eigenschaft auf oberster Ebene für andere Prozesse. Die Eigenschaften selbst erfordern eigene **Use...End use** (siehe Beispiel 2).
- Das von **Storage** zurückgegebene Objekt teilt seinen Sperrschlüssel NICHT mit den als Eigenschaften hinzugefügten *shared objects/collections*. Weitere Informationen dazu finden Sie im Abschnitt **Sperrschlüssel (Locking Identifier)**.

Beispiel 1

Ein gängiges Beispiel ist, das Objekt **Storage** in der **Datenbankmethode On Startup** zu initialisieren:

```
Use(Storage)
   Storage.counters:=New shared object("customers";0;"invoices";0)
End use
```

Beispiel 2

Dieses Beispiel zeigt einen Standardweg zum Setzen von **Storage** Werten:

```
Use(Storage)
   Storage.mydata:=New shared object
   Use(Storage.mydata)
      Storage.mydata.prop1:="Smith"
      Storage.mydata.prop2:=100
   End use
End use
```

Beispiel 3

Storage ermöglicht, ein *Singleton* (Entwurfsmuster) mit einer *lazy initialization* zu implementieren:
HInweis: Weitere Informationen dazu finden Sie auf folgender Wikipedia Seite.

```
C_LONGINT($0)
C_LONGINT($counterValue)
C_OBJECT(counter)  //Eine Referenz auf "counter" für den Prozess erzeugen

If(counter=Null)  //Ist diese Referenz Null, diese von Storage erhalten
   Use(Storage)  // Storage wird nur einmal benötigt!
      If(Storage.counter=Null)
         Storage.counter:=New shared object("operation";0)
      End if
      counter:=Storage.counter  //Referenz von shared object "counter" erhalten
   End use
End if
```

```
Use(counter)  //das shared object "counter" direkt verwenden (Storage wird nicht benötigt!)
    counter.operation:=counter.operation+1
    $counterValue:=counter.operation
End use

$0:=$counterValue
```

Operatoren

Operatoren

Ein Operator ist ein Symbol für Berechnungen bei Ausdrücken. 4D ermöglicht:

- Berechnungen mit Zahlen, Datum und Zeit.
- Berechnungen mit Zeichenketten, Boolean Ausdrücken sowie spezielle Operationen in Bildern.
- Kombination einfacher Ausdrücke zum Erstellen eines neuen Ausdrucks.

Rangfolge

Die Reihenfolge, in der ein Ausdruck berechnet wird, heißt Rangfolge. In 4D gibt es keine Priorität für Operatoren. Der Interpreter geht immer von links nach rechts vor, die algebraische Rangfolge wird nicht berücksichtigt. Beispiel:

```
3+4*5
```

ergibt 35, da der Ausdruck von links nach rechts berechnet wird: 3 + 4 ist gleich 7, multipliziert mit 5 ergibt als Endergebnis 35.
Wollen Sie die Rangfolge von links nach rechts ändern, MÜSSEN Sie Klammern setzen. Beispiel:

```
3+(4*5)
```

ergibt 23, da der Ausdruck (4 * 5) aufgrund der Klammer zuerst berechnet wird. Zum Ergebnis 20 wird dann 3 hinzugezählt. Das ergibt als Endergebnis 23.
Sie können Klammern auch innerhalb anderer Klammern setzen. Achten Sie darauf, dass eine geöffnete Klammer auch immer wieder geschlossen wird. Fehlende oder falsch gesetzte Klammern können ungültige Ausdrücke oder unerwartete Ergebnisse hervorrufen. Sie können Ihre Anwendungen nur kompilieren, wenn die Klammern vollständig sind. Bei fehlenden Klammern meldet 4D Compiler einen Syntaxfehler.

Zuordnungsoperator

Der Zuordnungsoperator *:=* unterscheidet sich von den anderen Operatoren. Er kombiniert nicht Ausdrücke zu einem neuen, sondern weist einem Ausdruck (Datenfeld oder Variable) einen Wert zu. Links vom Zuordnungsoperator steht der Ausdruck, der den Wert aufnimmt, rechts davon der Wert, der zugeordnet wird. Beispiel:

```
MyVar:=Length("Acme")
```

setzt den Wert 4 (die Anzahl der Zeichen des Worts Acme) in die Variable *MyVar*. Diese wird dann als Zahl gewertet.
Wichtig: Verwechseln Sie bitte nicht die Operatoren = für Prüfung auf Gleichheit und := für die Zuordnung eines Wertes.
Die anderen Operatoren der 4D Programmiersprache werden in folgenden Abschnitten beschrieben:

String Operatoren

Siehe Abschnitt **String Operatoren**.

Numerische Operatoren

Siehe Abschnitt **Numerische Operatoren**.

Datumsoperatoren

Siehe Abschnitt **Datumsoperatoren**.

Zeitoperatoren

Siehe Abschnitt **Zeitoperatoren**.

Vergleichsoperatoren

Siehe Abschnitt **Vergleichsoperatoren**.

Boolean Operatoren

Siehe Abschnitt **Boolean Operatoren**.

Bildoperatoren

Siehe Abschnitt **Bildoperatoren**.

Bit Operatoren

Siehe Abschnitt **Bit Operatoren**.

Bit Operatoren

Bit Operatoren arbeiten mit Ausdrücken oder Werten vom Typ *Lange Ganzzahl*.

Hinweis: Übergeben Sie einem Bit Operator einen Wert vom Typ Ganzzahl oder Zahl, wertet 4D diesen vor der Berechnung mit einem Bit Operator als *Lange Ganzzahl*.

Beim Arbeiten mit Bit Operatoren müssen Sie sich den Wert vom Typ Lange Ganzzahl als Array mit 32 Bits vorstellen. Bits sind von rechts nach links von 0 bis 31 durchnumeriert.

Da jedes Bit entweder 0 oder 1 ist, können Sie sich einen Wert vom Typ Lange Ganzzahl auch als Wert vorstellen, der 32 Boolean Werte speichern kann. Ein Bit gleich 1 bedeutet Wahr, ein Bit gleich 0 bedeutet Falsch.

Ein Ausdruck mit Bit Operator gibt einen Wert von Typ *Lange Ganzzahl* zurück, mit Ausnahme des Operators *Bit Test*. Hier wird ein Wert vom Typ Boolean zurückgegeben. Es gibt folgende Bit Operatoren:

Operation	Operator	Syntax	Ergibt
Bitweises UND	&	Lang & Lang	Lang
Bitweises ODER (inklusiv)	|	Lang | Lang	Lang
Bitweises ODER (exklusiv)	^|	Lang ^| Lang	Lang
Left Bit Shift	<<	Lang << Lang	Lang (1)
Right Bit Shift	>>	Lang >> Lang	Lang (1)
Bit setzen	?+	Lang ?+ Lang	Lang (2)
Bit löschen	?-	Lang ?- Lang	Lang (2)
Bit Test	??	Lang ?? Lang	Boolean (2)

Hinweise

1. Für die Operationen *Left Bit Shift* und *Right Bit Shift* gibt der zweite Operand die Stellen an, um die die Bits des ersten Operanden verschoben werden. Deshalb sollte dieser zweite Operand zwischen 0 und 31 liegen. Beachten Sie jedoch, dass das Verschieben von 0 einen unveränderten Wert zurückgibt und mehr als 31 Bits 0x00000000 zurückgibt, da hier alle Bits verloren gehen. Übergeben Sie als zweiten Operanden einen anderen Wert, ist das Ergebnis nicht signifikant.
2. Für die Operationen *Bit setzen*, *Bit löschen* und *Bit Test* gibt der zweite Operand die Stelle des entsprechenden Bit an. Deshalb muss dieser zweite Operand zwischen 0 und 31 liegen. Sonst ist das Ergebnis des Ausdrucks nicht signifikant.

Folgende Tabelle zeigt die Bit Operatoren und ihre Auswirkung:

Operation	Beschreibung
Bitweises UND	Jedes resultierende Bit ist das logische UND der Bits in den beiden Operanden. Hier ist die logische UND Tabelle: 1 & 1 --> 1 0 & 1 --> 0 1 & 0 --> 0 0 & 0 --> 0 Das resultierende Bit ist 1, wenn beide Bits der Operanden 1 sind; sonst ist das resultierende Bit 0.
Bitweises ODER (inkl.)	Jedes resultierende Bit ist das logische ODER der Bits in den beiden Operanden. Hier ist die logische ODER Tabelle: 1 | 1 --> 1 0 | 1 --> 1 1 | 0 --> 1 0 | 0 --> 0 Das resultierende Bit ist 1, wenn mindestens ein Bit der beiden Operanden 1 ist; sonst ist das resultierende Bit 0.
Bitweises ODER (exkl.)	Jedes resultierende Bit ist das logische ODER (exkl.) der Bits in den beiden Operanden. Hier ist die logische ODER (exkl.) Tabelle: 1 ^| 1 --> 0 0 ^| 1 --> 1 1 ^| 0 --> 1 0 ^| 0 --> 0 Das resultierende Bit ist 1, wenn nur ein Bit der beiden Operanden 1 ist; sonst ist das resultierende Bit 0.
Left Bit Shift	Der resultierende Wert wird auf den ersten Operanden gesetzt, dann werden die Bits um die im zweiten Operanden angegebenen Stellen nach links verschoben. Die Bits auf der linken Seite gehen verloren, die neuen Bits auf der rechten Seite werden auf 0 gesetzt. **Hinweis:** Werden nur die positiven Werte berücksichtigt, ist das Verschieben um N Bits nach links dasselbe wie das Multiplizieren mit 2^N.
Right Bit Shift	Der resultierende Wert wird auf den ersten Operanden gesetzt, dann werden die Bits um die angegebenen Stellen nach rechts verschoben. Die Bits auf der rechten Seite gehen verloren, die neuen Bits auf der linken Seite werden auf 0 gesetzt. **Hinweis:** Werden nur die positiven Werte berücksichtigt, ist das Verschieben um N Bits nach rechts dasselbe wie das Dividieren durch 2^N.
Bit setzen	Der resultierende Wert wird auf den ersten Operanden gesetzt, dann wird das Bit, dessen Stelle im zweiten Operanden angegeben ist, auf 1 gesetzt. Die anderen Bits bleiben unverändert.
Bit Löschen	Der resultierende Wert wird auf den ersten Operanden gesetzt, dann wird das Bit, dessen Stelle im zweiten Operanden angegeben ist, auf 0 gesetzt. Die anderen Bits bleiben unverändert.
Bit Test	Ergibt wahr, wenn im ersten Operanden das Bit, dessen Stelle im zweiten Operanden angegeben ist, gleich 1 ist. Ergibt Falsch, wenn im ersten Operanden das Bit, dessen Stelle im zweiten Operanden angegeben ist, gleich 0 ist.

Beispiel

1) Folgende Tabelle zeigt für jeden Bit Operator ein Beispiel:

Operation	Beispiel	Ergebnis
Bitweises UND	0x0000FFFF & 0xFF00FF00	0x0000FF00
Bitweises ODER (incl.)	0x0000FFFF | 0xFF00FF00	0xFF00FFFF
Bitweises ODER (excl.)	0x0000FFFF ^| 0xFF00FF00	0xFF0000FF
Left Bit Shift	0x0000FFFF << 8	0x000FFFF0
Right Bit Shift	0x0000FFFF >> 8	0x00000FFF
Bit setzen	0x00000000 ?+ 16	0x00010000
Bit löschen	0x00010000 ?- 16	0x00000000
Bit Test	0x00010000 ?? 16	True

2) 4D bietet viele vordefinierte Konstanten. Ihre Bezeichnung endet in einigen Fällen mit “bit” oder “mask.” Das gilt zum Beispiel für die Konstanten unter dem Thema **Ressourcen Eigenschaften**:

Konstante	Typ	Wert
Changed resource bit	Lange Ganzzahl	1
Changed resource mask	Lange Ganzzahl	2
Locked resource bit	Lange Ganzzahl	4
Locked resource mask	Lange Ganzzahl	16
Preloaded resource bit	Lange Ganzzahl	2
Preloaded resource mask	Lange Ganzzahl	4
Protected resource bit	Lange Ganzzahl	3
Protected resource mask	Lange Ganzzahl	8
Purgeable resource bit	Lange Ganzzahl	5
Purgeable resource mask	Lange Ganzzahl	32
System heap resource bit	Lange Ganzzahl	6
System heap resource mask	Lange Ganzzahl	64

Mit diesen Konstanten können Sie den Wert testen, den die Funktion **Get resource properties** zurückgibt oder den Wert erstellen, der im Befehl **_o_SET RESOURCE PROPERTIES** übergeben wurde. Konstanten mit der Endung "bit" liefern die Position des Bit, das Sie testen, löschen oder setzen wollen. Konstanten mit der Endung "mask" liefern einen Wert vom Typ lange Ganzzahl, in der nur das Bit, das Sie testen, löschen oder setzen wollen, den Wert 1 hat.

Sie wollen zum Beispiel prüfen, ob eine Ressource (deren Eigenschaften Sie aus der Variablen *$vlResAttr* erhalten haben) aus dem Speicher entfernbar ist:

```
If($vlResAttr ?? Purgeable resource bit) ` Ist die Ressource entfernbar?
```

oder:

```
If(($vlResAttr & Purgeable resource mask)#0)Ist die Ressource entfernbar?
```

Umgekehrt können Sie mit diesen Konstanten dasselbe Bit setzen. Schreiben Sie:

```
$vlResAttr:=$vlResAttr ?+Entfernbares Ressourcen Bit
```

oder:

```
$vlResAttr:=$vlResAttr |Entfernbares Ressourcen Bit
```

3) Dieses Beispiel speichert zwei Werte vom Typ Zahl in einem Wert vom Typ Lange Ganzzahl:

```
$vlLong:=($viIntA<<16)|$viIntB ` Speichere zwei Zahlen in Langer Ganzzahl

$vlIntA:=$vlLong>>16 ` Berechne Zahl aus dem high-word-Bereich

$viIntB:=$vlLong & 0xFFFF ` Berechne Zahl aus dem low-word-Bereich
```

Tipp: Seien Sie vorsichtig beim Bearbeiten von Werten vom Typ Ganzzahl oder Zahl in Ausdrücken, die sowohl numerische als auch Bit Operatoren einsetzen. Das hohe Bit (Bit 31 für Lange Ganzzahl, Bit 15 für Zahl) bestimmt das Vorzeichen des Wertes—positiv, wenn es gelöscht wird, negativ, wenn es gesetzt wird. Numerische Operatoren finden anhand dieses Bits das Vorzeichen des Wertes heraus, für Bit Operatoren ist dieses Bit nicht relevant.

Vergleichsoperatoren

Die Tabellen in diesem Abschnitt zeigen die Vergleichsoperatoren bei Ausdrücken vom Typ alphanumerisch, numerisch, Datum, Zeit, Zeiger und Bilder mit Metadaten (nicht verwendbar bei Ausdrücken vom Typ Array oder BLOB). Ein Ausdruck mit Vergleichsoperator vergleicht zwei Werte miteinander und gibt einen Boolean Wert zurück, entweder *TRUE* oder *FALSE*.
Hinweis: Sie können zwei Bilder über die Funktion **Equal pictures** miteinander vergleichen.

Alphanumerisch

Operation	Syntax	Ergibt	Ausdruck	Wert
Gleichheit	String = String	Boolean	"abc" = "abc"	True
			"abc" = "abd"	False
Ungleichheit	String # String	Boolean	"abc" # "abd"	True
			"abc" # "abc"	False
Größer als	String > String	Boolean	"abd" > "abc"	True
			"abc" > "abc"	False
Kleiner als	String < String	Boolean	"abc" < "abd"	True
			"abc" < "abc"	False
Größer als oder gleich	String >= String	Boolean	"abd" >= "abc"	True
			"abc" >= "abd"	False
Kleiner als oder gleich	String <= String	Boolean	"abc" <= "abd"	True
			"abd" <= "abc"	False
Enthält Schlüsselwort	String % String	Boolean	"Alpha Bravo" % "Bravo"	True
			"Alpha Bravo" % "ravo"	False
	Picture % String	Boolean	*Picture_expr* % "Mer"	True (*)

(*) Wenn das Schlüsselwort "Mer" dem Bild zugewiesen ist, das in einem Ausdruck Bild (Feld oder Variable) gespeichert ist. Weitere Informationen dazu finden Sie am Ende dieses Abschnitts unter **Vergleiche vom Typ alphanumerisch** .

Numerisch

Operation	Syntax	Ergibt	Ausdruck	Wert
Gleichheit	Zahl = Zahl	Boolean	10 = 10	True
			10 = 11	False
Ungleichheit	Zahl # Zahl	Boolean	10 #11	True
			10 # 10	False
Größer als	Zahl > Zahl	Boolean	11 > 10	True
			10 > 11	False
Kleiner als	Zahl < Zahl	Boolean	10 < 11	True
			11 < 10	False
Größer als oder gleich	Zahl >= Zahl	Boolean	11 >= 10	True
			10 >= 11	False
Kleiner als oder gleich	Zahl <= Zahl	Boolean	10 <= 11	True
			11 <= 10	False

Weitere Informationen zur Genauigkeit beim Vergleichen von Zahlen im Bezug auf Gleichheit finden Sie unter dem Befehl **SET REAL COMPARISON LEVEL**.

Datum

Operation	Syntax	Ergibt	Ausdruck	Wert
Gleichheit	Datum = Datum	Boolean	!1.1.97! =!1.1.97!	True*
			!20.1.97! =!1.1.97!	False
Ungleichheit	Datum # Datum	Boolean	!20.1.97! # !1.1.97!	True
			!1.1.97! # !1.1.97!	False
Größer als	Datum > Datum	Boolean	!20.1.97! > !1.1.97!	True
			!1.1.97! > !1.1.97!	False
Kleiner als	Datum < Datum	Boolean	!1.1.97! < !20.1.97!	True
			!1.1.97! < !1.1.97!	False
Größer als oder gleich	Datum >= Datum	Boolean	!20.1.97! >=!1.1.97!	True
			!1.1.97!>=!20.1.97!	False
Kleiner als oder gleich	Datum <= Datum	Boolean	!1.1.97!<=!20.1.97!	True
			!20.1.97!<=!1.1.97!	False

Zeit

Operation	Syntax	Ergibt	Ausdruck	Wert
Gleichheit	Zeit = Zeit	Boolean	?01:02:03? = ?01:02:03?	True
			?01:02:03? = ?01:02:04?	False
Ungleichheit	Zeit # Zeit	Boolean	?01:02:03? # ?01:02:04?	True
			?01:02:03? # ?01:02:03?	False
Größer als	Zeit > Zeit	Boolean	?01:02:04? > ?01:02:03?	True
			?01:02:03? > ?01:02:03?	False
Kleiner als	Zeit < Zeit	Boolean	?01:02:03? < ?01:02:04?	True
			?01:02:03? < ?01:02:03?	False
Größer als oder gleich	Zeit >= Zeit	Boolean	?01:02:03? >=?01:02:03?	True
			?01:02:03? >=?01:02:04?	False
Kleiner als oder gleich	Zeit <= Zeit	Boolean	?01:02:03? <=?01:02:03?	True
			?01:02:04? <=?01:02:03?	False

Zeiger

Bei Zeigern sind nur die Operatoren gleich und ungleich möglich: Wenn gilt

```
  ` vPtrA und vPtrB zeigen auf das gleiche Objekt
vPtrA:=->EinObjekt
vPtrB:=->EinObjekt
  ` vPtrC zeigt auf ein anderes Objekt
vPtrC:=->ein anderes Objekt
```

ergibt sich folgendes:

Operation	Syntax	Ergibt	Ausdruck	Wert
Gleichheit	Zeiger = Zeiger	Boolean	vPtrA = vPtrB	True
			vPtrA = vPtrC	False
Ungleichheit	Zeiger # Zeiger	Boolean	vPtrA # vPtrC	True
			vPtrA # vPtrB	False

Vergleiche vom Typ alphanumerisch

Beachten Sie folgende Regeln:

- Strings werden Zeichen für Zeichen miteinander verglichen (außer bei Suchen nach Schlüsselwort).
- Die Groß- und Kleinschreibung wird nicht berücksichtigt. So gibt "a"="A" TRUE zurück. Wollen Sie die Schreibweise von zwei Zeichen überprüfen, vergleichen Sie deren Zeichen Codes. So ergibt z.B. folgender Ausdruck FALSE:

```
Character code("A")=Character code("a")  // da 65 ungleich 97 ist
```

- Diakritische Zeichen werden nicht berücksichtigt. 4D verwendet beim Vergleich die Zeichen des Systems auf Ihrem Rechner. So ergibt z.B. folgender Ausdruck TRUE:

```
"n"="ñ"
"n"="Ñ"
"A"="å"
  // usw
```

- Bei der Suche nach Schlüsselwörtern werden Wörter nur im ganzen berücksichtigt. Der Operator % gibt immer Falsch zurück, wenn die Suche mehr als ein Wort oder nur einen Teil davon betrifft, z.B. eine Vorsilbe. Wörter werden definiert als Zeichenketten, getrennt durch „Trenner", also Leerzeichen, Bindestrich, Gedankenstrich, o.ä. Ein Apostroph, z.B. Today's gilt normalerweise als Teil des Wortes, wird aber in bestimmten Fällen ignoriert (siehe Regeln unten). Sie können auch nach Nummern suchen, da sie inkl. Trennzeichen für Tausend oder Dezimalstellen (. ,) als Ganzes gewertet werden. Andere Zeichen (Währungssymbole, Temperatur, usw.) werden dabei ignoriert.

```
"Alpha Bravo Charlie"%"Bravo"  // Gibt Wahr zurück
"Alpha Bravo Charlie"%"vo"  // Gibt Falsch zurück
"Alpha Bravo Charlie"%"Alpha Bravo"  // Gibt Falsch zurück
"Alpha,Bravo,Charlie"%"Alpha"  // Gibt Wahr zurück
"Software and Computers"%"comput@"  // Gibt Wahr zurück
```

Hinweise:
- 4D verwendet zum Suchen der Schlüsselwörter die ICU library. Weitere Informationen zu den Regeln bei Schlüsselwörtern finden Sie unter http://www.unicode.org/unicode/reports/tr29/#Word_Boundaries
- In der japanischen Version verwendet 4D zum Suchen von Schlüsselwörtern anstatt ICU standardmäßig *Mecab*. Weitere Informationen dazu finden Sie im Abschnitt **Unterstützung von Mecab (japanische Version)**.

- Das Jokerzeichen (@) wird berücksichtigt. @ kann für beliebig viele Zeichen stehen. So ergibt z.B. folgender Ausdruck TRUE:

```
"abc@"="abcdefghij"
```

  Sie können das Jokerzeichen nur für den zweiten Operanden (der String auf der rechten Seite) einsetzen. Demnach ergibt folgender Ausdruck FALSE, da @ im ersten Operanden lediglich als 1 Zeichen gewertet wird: "abc@" = "abcdefghij"
  Das Jokerzeichen kann ein Zeichen, viele Zeichen oder gar kein Zeichen ersetzen. Demnach ergeben folgende Ausdrücke TRUE:

```
"abcdefghij"="abcdefghij@"
"abcdefghij"="@abcdefghij"
"abcdefghij"="abcd@efghij"
"abcdefghij"="@abcdefghij@"
"abcdefghij"="@abcde@fghij@"
```

  Zwei aufeinanderfolgende Jokerzeichen werden dagegen nicht erkannt. Hier gibt der Ausdruck FALSE zurück:

```
"abcdefghij"="abc@@fg"
```

  Enthält der Vergleichsoperator ein < oder > Zeichen, wird nur der Vergleich mit einem Joker am Ende des Operanden unterstützt:

```
"abcd"<="abc@"  // Valid comparison
"abcd"<="abc@ef"/Not a valid comparison/
```

Hinweise:
Wollen Sie Vergleiche oder Suchläufe mit @ als Zeichen, also nicht als Joker durchführen, gibt es zwei Möglichkeiten:

- Sie verwenden die Anweisung **Character code** (At sign).
  Angenommen, Sie wollen wissen, ob ein String mit dem Zeichen @ endet.
  - der folgende Ausdruck ist immer WAHR, außer *$vsValue* ist leer:

```
($vsValue[[Length($vsValue)]]="@")
```

  - der folgende Ausdruck wird korrekt interpretiert:

```
(Character code($vsValue[[Length($vsValue)]])#64)
```

- Sie aktivieren im Dialogfenster Datenbank-Eigenschaften die Option "@ nur am Anfang oder Ende eines Textes als Joker betrachten". Dann gilt @ als normales Zeichen, wenn es innerhalb einer Zeichenkette gefunden wird. Mit dieser Option können Sie beeinflussen, wie Vergleichsoperatoren in Such- und Sortierläufen verwendet werden. Weitere Informationen dazu finden Sie im Abschnitt **Text-Vergleiche** des Handbuchs *4D Designmodus*.

Datumsoperatoren

Ein Ausdruck mit einem Datumsoperator gibt ein Datum oder eine Zahl zurück. Eine Datumskonstante muss immer zwischen Ausrufezeichen geschrieben werden. 4D berechnet die Daten ab dem Jahr 100 und bis zum Jahr 32 000. Auch das Schaltjahr wird berücksichtigt.

Es gibt folgende Datumsoperatoren:

Operation	Syntax	Ergibt	Ausdruck	Wert
Datumsdifferenz	Datum – Datum	Zahl	!1998-01-20! – !1998-01-01!	19
Datum vorausrechnen	Datum + Zahl	Datum	!1998-01-20! + 9	!1998-01-29!
Datum zurückrechnen	Datum – Zahl	Datum	!1998-01-20! – 9	!1998-01-11!

Boolean Operatoren

4D unterstützt die Boolean Operatoren UND (&) und ODER (|). Ein logisches UND ergibt TRUE, wenn beide Ausdrücke wahr sind. Ein logisches ODER ergibt TRUE, wenn mindestens einer der Ausdrücke wahr ist.

Hinweis: Das Zeichen | erhalten Sie unter Windows mit der Tastenkombination alt gr + Zeichen <, auf Macintosh mit der Wahltaste + Ziffer 7.

4D bietet auch die Boolean Funktionen *True*, *False* und *Not*. Weitere Informationen dazu finden Sie in der Beschreibung zu diesen Funktionen.

Es gibt folgende Boolean Operatoren:

Operation	Syntax	Ergibt	Ausdruck	Wert
UND	Boolean & Boolean	Boolean	("A" = "A") & (15 # 3)	True
			("A" = "B") & (15 # 3)	False
			("A" = "B") & (15 = 3)	False
ODER	Boolean | Boolean	Boolean	("A" = "A") | (15 # 3)	True
			("A" = "B") | (15 # 3)	True
			("A" = "B") | (15 = 3)	False

Für das logische UND gilt:

Expr1	Expr2	Expr1 & Expr2
True	True	True
True	False	False
False	True	False
False	False	False

Für das logische ODER gilt:

Expr1	Expr2	Expr1 | Expr2
True	True	True
True	False	True
False	True	True
False	False	False

Tipp

Ein ausschließenden ODER berechnen Sie mit der Funktion *Not*. Schreiben Sie folgenden Code:

```
(Expr1|Expr2) & Not(Expr1 & Expr2)
```

Numerische Operatoren

Ein Ausdruck mit einem numerischen Operator gibt eine Zahl zurück. Es gibt folgende numerische Operatoren:

Operation	Syntax	Ergibt	Ausdruck	Wert
Addition	Zahl + Zahl	Zahl	2 + 3	5
Subtraktion	Zahl – Zahl	Zahl	3 – 2	1
Multiplikation	Zahl * Zahl	Zahl	5 * 2	10
Division	Zahl / Zahl	Zahl	5 / 2	2,5
Ganzzahlige Division	Zahl \ Zahl	Zahl	5 \ 2	2
Modulo	Zahl % Zahl	Zahl	5 % 2	1
Exponent	Zahl ^ Zahl	Zahl	2 ^ 3	8

Der Modulo Operator % dividiert die erste Zahl durch die zweite und übergibt den ganzzahligen Restwert. Beispiele:

- 10 % 2 ergibt 0, da kein Restwert übrigbleibt.
- 10 % 3 ergibt 1, da der Restwert 1 ist.
- 10,5 % 2 ergibt 0, da der Restwert keine Ganzzahl ist.

WARNUNG:

- Der Modulo Operator % gibt signifikante Werte mit Zahlen aus dem Bereich Lange Ganzzahl (von - 2^31 bis + 2^31 - 1) zurück. Verwenden Sie bei Restwertberechnungen mit Zahlen außerhalb dieses Bereich die Funktion **Mod**.
- Der Operator für ganzzahlige Division gibt nur für Ganzzahlen signifikante Werte zurück.

Bildoperatoren

Ein Ausdruck mit einem Bildoperator gibt ein Bild zurück. Es gibt folgende Bildoperatoren.

Operation	Syntax	Aktion
Horizontales Anfügen	Bild1 + Bild2	Setzt Bild2 rechts neben Bild1
Vertikales Anfügen	Bild1 / Bild2	Setzt Bild2 linksbündig über Bild1
Exklusives Aufeinandersetzen(*)	Bild1 & Bild2	Setzt Bild2 vor Bild1 Bild2 ist vorne (exklusiv ODER)
Inklusives Aufeinandersetzen(*)	Bild1 | Bild2 zurück, wenn beide Bildern dieselbe Größe haben	Setzt Bild2 auf Bild1 und gibt Ergebnisbild
Horizontales Verschieben	Bild + Zahl	Verschiebt Bild horizontal um n Pixel
Vertikales Verschieben	Bild / Zahl	Verschiebt Bild vertikal um n Pixel
Zoomen	Bild * Zahl	Verändert Bildgröße gemäß Faktor n
Horizontales Verzerren	Bild *+ Zahl	Verzerrt Bild horizontal gemäß Faktor n
Vertikales Verzerren	Bild */ Zahl	Verzerrt Bild vertikal gemäß Faktor n

(*) Die Funktionsweise für exklusives Aufeinandersetzen (&) und inklusives Aufeinandersetzen (|) hat sich ab 4D v14 geändert, weil 4D andere Libraries zur Bildausgabe verwendet:
Pict3 := Pict1 & Pict2 ergibt dasselbe wie die Anweisung:

```
COMBINE PICTURES(pict3;pict1;Superimposition;pict2)
```

Pict3 := Pict1 | Pict2 ergibt dasselbe wie die Anweisung:

```
$equal:=Equal pictures(Pict1;Pict2;Pict3)
```

Beachten Sie, dass Pict1 und Pict2 für den Operator **|** exakt dieselben Ausmaße haben müssen. Haben die Bilder unterschiedliche Ausmaße, ergibt die Operation Pict1 | Pict2 ein leeres Bild.

Hinweis: Mit dem Befehl **COMBINE PICTURES** können Sie Bilder übereinanderlegen und die Merkmale jedes Quellbildes im Ergebnisbild beibehalten.

Die beiden Operatoren & und | geben immer ein Bild als Bitmap zurück, egal welches Format die Ausgangsbilder haben. Der Grund hierfür ist, dass 4D die Bilder zuerst im Speicher in Bitmaps zeichnet, und dann das Ergebnisbild anhand der grafischen exklusiv oder inklusiv ODER auf die Pixel der Bitmaps berechnet.

Die Bildoperatoren geben Vektorbilder zurück, wenn beide Ausgangsbilder Vektor-Bilder sind. Beachten Sie jedoch, dass Bilder im Anzeigeformat **Auf Hintergrund** als Bitmap gedruckt werden.

Beispiel

Folgende Beispiele zeigen alle Bilder im Format **Auf Hintergrund**.
Bild 1 ist ein Kreis:

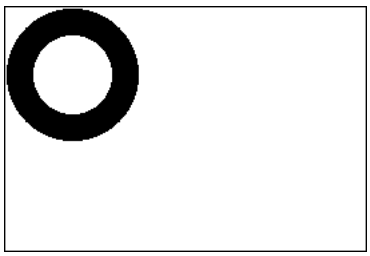
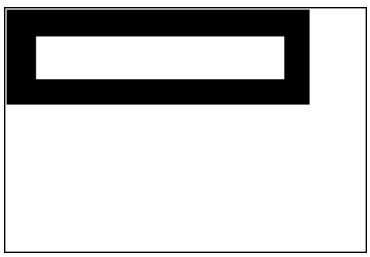


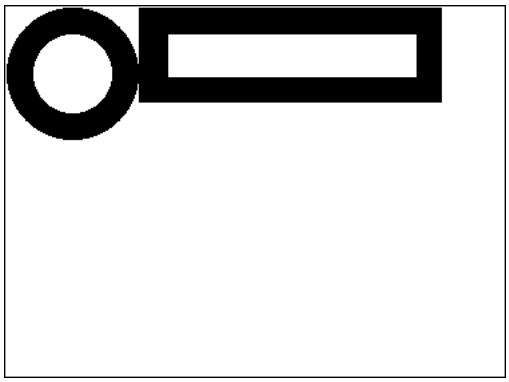
Bild 2 ist ein Rechteck:



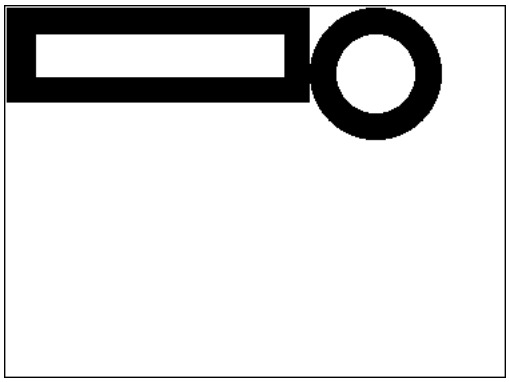
Nachfolgend sehen Sie die Syntax für die jeweilige Operation und die entsprechende grafische Darstellung.

- Horizontales Anfügen

```
Kreis+Rechteck ` Setzt Rechteck rechts neben Kreis
```

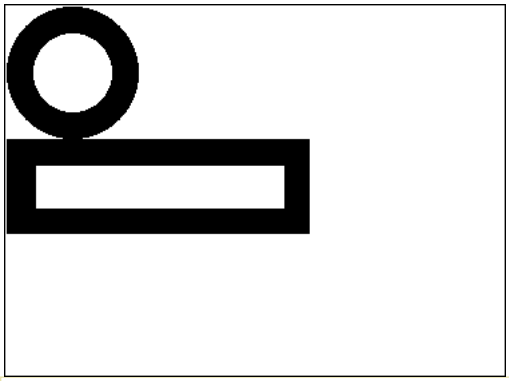


Rechteck + Kreis ` Setzt Kreis rechts neben Rechteck

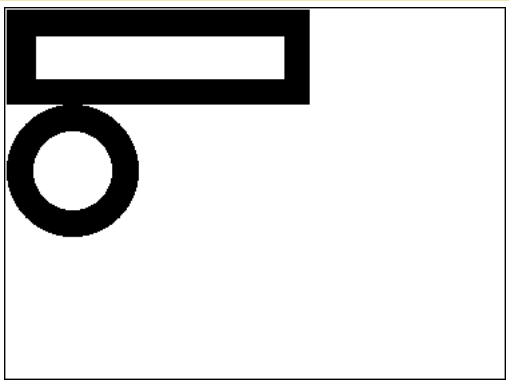


- Vertikales Anfügen

```
Kreis/Rechteck ` Setzt Rechteck unter Kreis
```

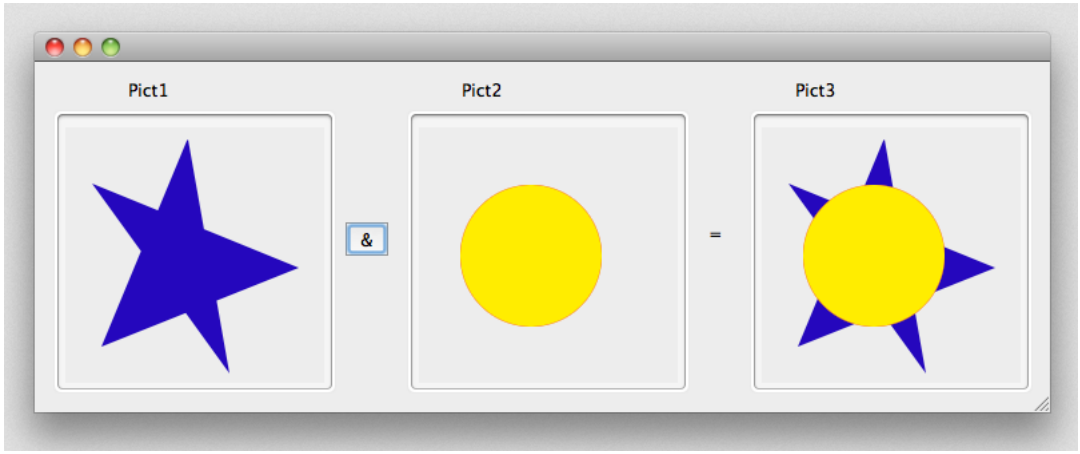


```
Rechteck/Kreis ` Setzt Kreis unter Rechteck
```



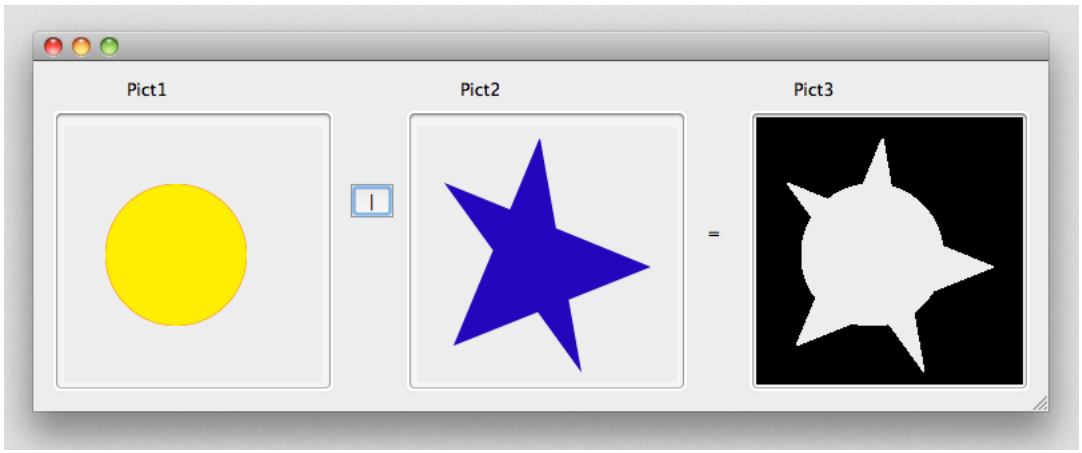
- Exklusives Aufeinandersetzen (Exklusiv ODER)

```
Pict3:=Pict1 & Pict2 // Setzt Pict2 über Pict1
```



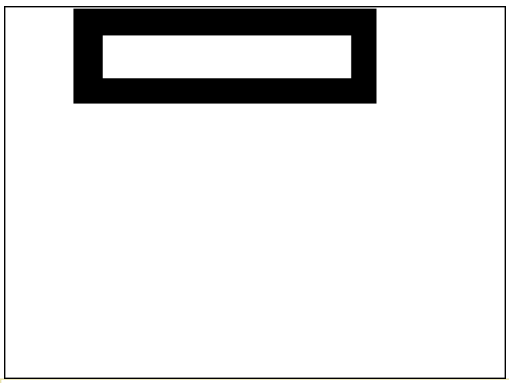
- Inklusives Aufeinandersetzen

```
Pict3:=Pict1|Pict2 // Gibt die Ergebnismaske aus dem Aufeinandersetzen von zwei Bildern mit derselben Größe zurück
```

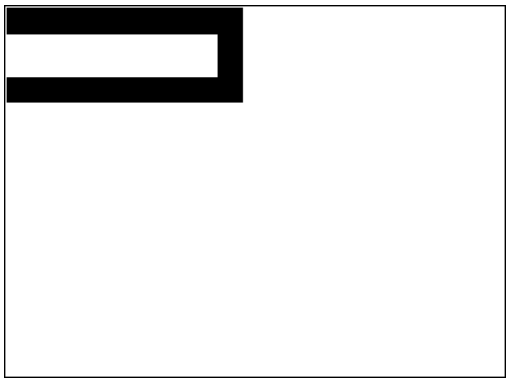


- Horizontales Verschieben

```
Rechteck+50 ` Verschiebt das Rechteck 50 Pixel nach rechts
```

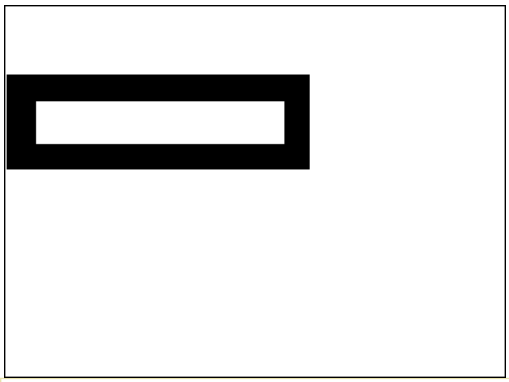


```
Rechteck-50 ` Verschiebt das Rechteck 50 Pixel nach links
```

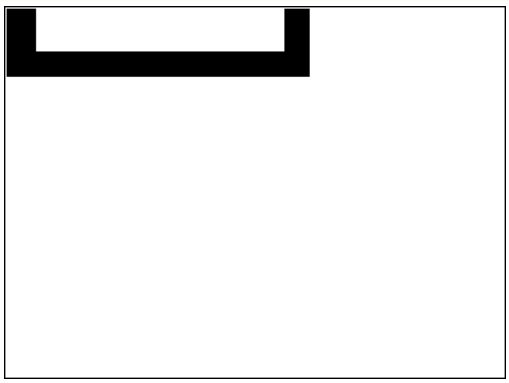


- Vertikales Verschieben

Rechteck/50 ` Verschiebt das Rechteck 50 Pixel nach unten



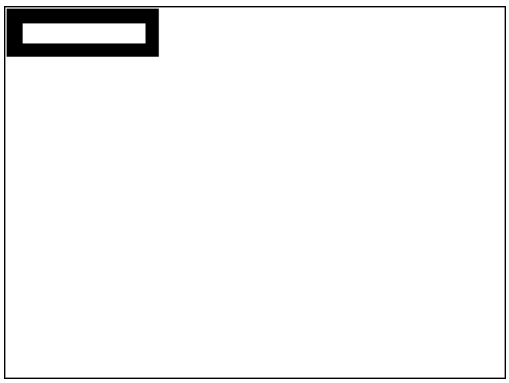
Rechteck/-20 ` Verschiebt das Rechteck 20 Pixel nach oben



- Zoomen

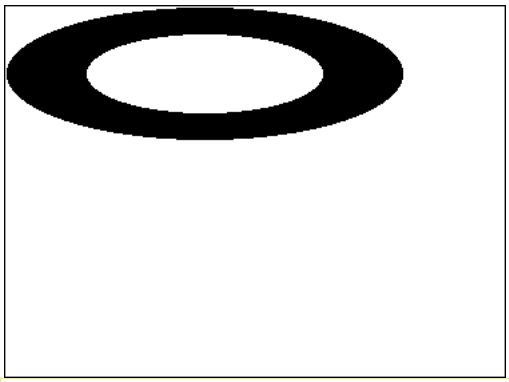
Rechteck*1.5 ` Vergrößert das Rechteck um 50%

Rechteck*0.5 ` Verkleinert das Rechteck um 50%

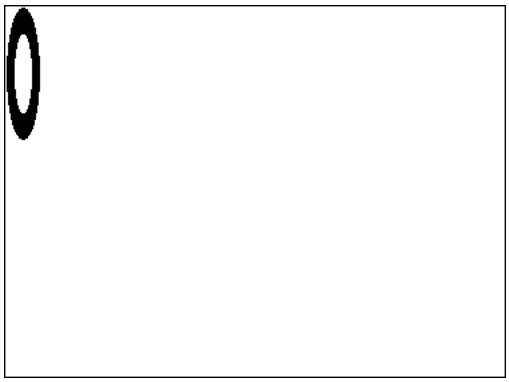


- Horizontales Verzerren

Kreis*+3 ` Der Kreis wird dreimal breiter

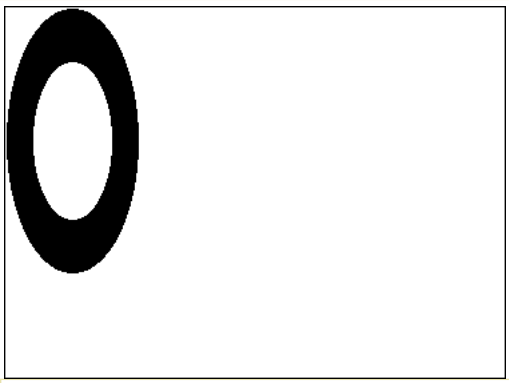


Kreis*+0,25 ` Die Kreisbreite wird viermal kleiner

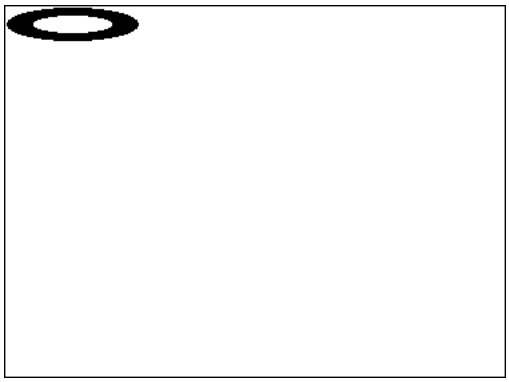


- Vertikales Verzerren

Kreis*/2 ` Der Kreis wird doppelt so hoch



Kreis*/0,25 ` Die Kreishöhe wird viermal kleiner



String Operatoren

Ein Ausdruck mit einem String Operator gibt einen String (Zeichenkette) zurück.
Es gibt folgende String Operatoren:

Operation	Syntax	Ergibt	Ausdruck	Wert
Verbindung	String + String	String	"abc" + "def"	"abcdef"
Wiederholung	String * Numerisch	String	"ab" * 3	"ababab"

Zeitoperatoren

Ein Ausdruck mit einem Zeitoperator gibt eine Zeit oder eine Zahl zurück. Eine Zeitkonstante muss immer zwischen Fragezeichen geschrieben werden. Es gibt folgende Zeitoperatoren:

Operation	Syntax	Ergibt	Ausdruck	Wert
Addition	Zeit + Zeit	Zeit	?02:03:04? + ?01:02:03?	?03:05:07?
Subtraktion	Zeit – Zeit	Zeit	?02:03:04? – ?01:02:03?	?01:01:01?
Addition	Zeit + Zahl	Zahl	?02:03:04? + 65	7449
Subtraktion	Zeit – Zahl	Zahl	?02:03:04? – 65	7319
Multiplikation	Zeit * Zahl	Zahl	?02:03:04? * 2	14768
Division	Zeit / Zahl	Zahl	?02:03:04? / 2	3692
Ganzzahlige Division	Zeit \ Zahl	Zahl	?02:03:04? \ 2	3692
Modulo	Zeit % Zeit	Zeit	?20:10:00? % ?04:20:00?	?02:50:00?
Modulo	Zeit % Zahl	Zahl	?02:03:04? % 2	0

Beispiel 1

Mit den Funktionen **Time** und **Current time** können Sie Ausdrücke vom Typ Zeit und Zahl kombinieren. 4D rechnet die Zeit in Sekunden nach Mitternacht um und führt dann die Berechnung aus. Zum Beispiel:

```
   ` Folgende Zeile weist $vlSeconds die Anzahl Sekunden zu, die zwischen
   ` Mitternacht und einer Stunde nach der aktuellen Zeit vergangen sind.
$vlSeconds:=Current time+3600

   ` Folgende Zeile weist $vHSoon die Zeit zu, die in einer Stunde sein wird
$vhSoon:=Time(Time string(Current time+3600))
```

Die 2. Zeile lässt sich auch einfacher schreiben:

```
   ` Folgende Zeile weist $vHSoon die Zeit zu, die in einer Stunde sein wird
$vhSoon:=Current time+?01:00:00?
```

Beispiel 2

Beim Entwickeln Ihrer Anwendung werden Sie öfters Zeitausdrücke als numerische Werte benötigen. Sie öffnen z.B. ein Dokument mit der Funktion **Open document**. Sie erhalten eine Dokumentreferenz (*DocRef*), die als Zeitausdruck formatiert ist. Übergeben Sie diese *DocRef* später in einer externen Routine von 4D, benötigen Sie einen numerischen Wert. 4D bietet hier die Addition mit Null (0). Um eine Zeitangabe in eine Zahl umzurechnen, müssen Sie nur eine Null hinzuaddieren. Beispiel:

```
   ` Wähle und öffne ein Dokument
$vhDocRef:=Open document("")
If(OK=1)
   ` Übergib DocRef, ausgedrückt in Zeit als Zahl an die externe Routine von 4D
   DO SOMETHING SPECIAL(0+$vhDocRef)
End if
```

Beispiel 3

Der Operator Modulo dient insbesondere dazu, Zeiten im 24 Stunden Format hinzuzufügen:

```
$t1:=?23:00:00?  // Es ist 23:00 p.m.
 // Wir wollen 2 1/2 Stunden hinzufügen
$t2:=$t1 +?02:30:00?  // Mit einer einfachen Addition gilt: $t2 ist ?25:30:00?
$t2:=($t1 +?02:30:00?)%?24:00:00?  // $t2 ist ?01:30:00? und es ist 1:30 a.m. des nächsten Tages
```

ORDA - DataClass

Ausführliche Informationen zu Dataclasses und Code-Beispiele finden Sie auf der Seite **Dataclasses** des *4D Developer Guide*.

- dataClass.{attributeName} Neu 17.0
- dataClass.all Neu 17.0
- dataClass.fromCollection Neu 17.0
- dataClass.get Neu 17.0
- dataClass.new Neu 17.0
- dataClass.newSelection Neu 17.0
- dataClass.query Neu 17.0

dataClass.{attributeName}

Parameter	Typ	Beschreibung
dataClass.{attributeName}	DataClassAttribute	Dataclass attribute description

Beschreibung

Attribute von Dataclasses sind Objekte, die direkt als Eigenschaften dieser Klassen verfügbar sind.

Die zurückgegebenen Objekte sind vom Typ *DataClassAttribute*. Sie haben Eigenschaften, die Sie verwenden und lesen können, um Information über Ihre Dataclass Attribute zu erhalten. Die Liste dieser Eigenschaften finden Sie im Abschnitt **ORDA - DataClassAttribute**.

Hinweis: Dataclass Attribute sind auch über die alternative Syntax mit eckigen Klammern [ ] abrufbar (siehe Beispiel).

Beispiel

```
$salary:=ds.Employee.salary  //gibt das Attribut salary in der Dataclass Employee zurück
$compCity:=ds.Company["city"]  //gibt das Attribut city n der Dataclass Company zurück
```

dataClass.all( )

dataClass.all () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	EntitySelection		Referenzen auf alle Entities in der Dataclass

Beschreibung

Die Methode **dataClass.all( )** sucht im Datastore nach allen Entities in der Dataclass und gibt sie als Entity-Selection zurück.

Die Entities werden in der standardmäßigen Reihenfolge zurückgegeben, d.h. so wie sie anfangs nacheinander angelegt wurden. Wurden dagegen Entities gelöscht oder neu hinzugefügt, gibt die standardmäßigen Reihenfolge nicht mehr die Reihenfolge der Erstellung wieder.

Wird keine entsprechende Entity gefunden, wird eine leere Entity-Selection zurückgegeben.

Lazy loading wird angewendet.

Beispiel

```
C_OBJECT($allEmp)
$allEmp:=ds.Employee.all()
```

dataClass.fromCollection( )

dataClass.fromCollection (objectCol) -> Funktionsergebnis

Parameter	Typ		Beschreibung
objectCol	Collection	⇒	Collection von Objekten zum Abbilden mit Entities
Funktionsergebnis	EntitySelection	⇐	Entity-Selection gefüllt aus der Collection

Beschreibung

Die Methode **dataClass.fromCollection( )** aktualisiert oder erstellt Entities in der Dataclass nach der im Parameter *objectCol* angegebenen Collection von Objekten und gibt die entsprechende Entity-Selection zurück.

Im Parameter *objectCol* übergeben Sie eine Collection von Objekten, um vorhandene Entities der Dataclass zu aktualisieren oder neue zu erstellen. Die Eigenschaftsnamen müssen dieselben sein wie die Attributsnamen in der Dataclass. Ist ein Eigenschaftsname in der Dataclass nicht vorhanden, wird er ignoriert. Ist ein Attributswert in der Collection nicht definiert, ist er null.

Die Abbildung zwischen den Objekten der Collection und den Entities erfolgt über die **Attributsnamen** und **passenden Typen**. Hat eine Eigenschaft des Objekts den gleichen Namen wie ein Attribut der Entity, aber ihre Typen passen nicht, wird das Attribut der Entity nicht gefüllt.

Modus Erstellen oder Aktualisieren
Für jedes Objekt von *objectCol* gilt folgendes:

- Enthält das Objekt **keine** boolean Eigenschaft "__NEW" oder ist sie auf **falsch** gesetzt, wird die Entity mit den entsprechenden Werten der Eigenschaften aus dem Objekt aktualisiert oder erstellt. Es wird keine Überprüfung des Primärschlüssels durchgeführt:
  - Ist der Primärschlüssel im Objekt angegeben und existiert, wird die Entity aktualisiert. In diesem Fall kann der Primärschlüssel wie vorgegeben sein oder als Eigenschaft "__KEY" (gefüllt mit dem Wert des Primärschlüssels).
  - Ist der Primärschlüssel im Objekt angegeben und existiert nicht, wird die Entity erstellt
  - Ist der Primärschlüssel im Objekt nicht angegeben, wird die Entity erstellt und der Wert des Primärschlüssels gemäß den standardmäßigen Datenbankregeln zugewiesen.
- Enthält das Objekt eine boolean Eigenschaft "__NEW", gesetzt auf **wahr**, wird die Entity mit den entsprechenden Werten der Attribute aus dem Objekt erstellt. Es wird eine Überprüfung des Primärschlüssels durchgeführt:
  - Ist der Primärschlüssel wie vorgegeben angegeben und existiert, wird ein Fehler gesendet
  - Ist der Primärschlüssel wie vorgegeben angegeben und existiert nicht, wird die Entity erstellt
  - Ist der Primärschlüssel nicht angegeben, wird die Entity erstellt und der Wert des Primärschlüssels gemäß den standardmäßigen Datenbankregeln zugewiesen.

  Hinweis: Die Eigenschaft "__KEY" mit einen Wert wird nur berücksichtigt, wenn die Eigenschaft "__NEW" auf **falsch** gesetzt ist oder weggelassen wird und eine entsprechende Entity existiert. In allen anderen Fällen wird der Wert der Eigenschaft "__KEY" ignoriert, der Wert des Primärschlüssels muss wie vorgegeben übergeben werden.

Verknüpfte Entities
Die Objekte von *objectCol* können eine oder mehrere eingebettete Objekte enthalten mit einer oder mehreren verknüpften Entities. Diese sind hilfreich, um Links zwischen Entities zu erstellen oder aktualisieren.
Eingebettete Objekte mit verknüpften Entities müssen das Attribut ___KEY enthalten (gefüllt mit dem Primärschlüssel der verknüpften Entity) oder den Primärschlüssel der verknüpften Entity selbst. Das Attribut __KEY ermöglicht Unabhängigkeit vom Namen des Primärschlüssels.

Hinweis: Der Inhalt von verknüpften Entities lässt sich nicht durch diesen Mechanismus erstellen bzw. aktualisieren.

Stempel
Ist ein Attribut __STAMP angegeben, wird eine Überprüfung mit dem Stempel im Datastore durchgeführt und im entsprechenden Fall ein Fehler zurückgegeben ("Angegebener Stempel passt nicht zum aktuellen für Datensatz# XX der Tabelle XXXX"). Weitere Informationen dazu finden Sie im Abschnitt **Entity sperren**.

Beispiel 1

Eine vorhandene Entity aktualisieren. Die Eigenschaft __NEW is nicht angegeben, der Primärschlüssel employee ist angegeben und existiert:

```
C_COLLECTION($empsCollection)
C_OBJECT($emp;$employees)

$empsCollection:=New collection
$emp:=New object
$emp.ID:=668  //Vorhandener Pirmärschlüssel in der Tabelle Employee
$emp.firstName:="Arthur"
$emp.lastName:="Martin"
$emp.employer:=New object("ID";121)  //Vorhandener Pirmärschlüssel in der verknüpften Dataclass Company
 // Für diesen Angestellten können wir die Firma über einen anderen Primärschlüssel in der verknüpften Dataclass Company ändern
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

Beispiel 2

Eine vorhandene Entity aktualisieren. Die Eigenschaft__NEW ist nicht angegeben, der Primärschlüssel *employee* hat das Attribut __KEY und existiert:

```
C_COLLECTION($empsCollection)
C_OBJECT($emp;$employees)

$empsCollection:=New collection
$emp:=New object
$emp.__KEY:=1720  //Vorhandender PK in der Tabelle Employee
$emp.firstName:="John"
$emp.lastName:="Boorman"
$emp.employer:=New object("ID";121)  //Vorhandender PK in der verknüpften dataClass Company
 // Für diesen Angestellten können wir über einen anderen vorhandenen PK in der verknüpften dataClass Company die Firma wechseln
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

Beispiel 3

Einfach eine neue Entity aus der Collection erstellen:

```
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)
```

Beispiel 4

Eine Entity erstellen. Die Eigenschaft __NEW ist **Wahr**, es gibt keinen Primärschlüssel employee:

```
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)  //Vorhandener Primärschlüssel in der verknüpften Dataclass Company
$emp.__NEW:=True
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

Beispiel 5

Eine Entity erstellen. Die Eigenschaft __NEW ist nicht angegeben, der Primärschlüssel employee ist angegeben und existiert nicht:

```
C_COLLECTION($empsCollection)
C_OBJECT($emp;$employees)

$empsCollection:=New collection
$emp:=New object
$emp.ID:=10000  //Nicht vorhandener Primärschlüssel
$emp.firstName:="Françoise"
$emp.lastName:="Sagan"
$empsCollection.push($emp)
$employees:=ds.Employee.fromCollection($empsCollection)
```

Beispiel 6

In diesem Beispiel wird die erste Entity erstellt und gesichert, die zweite schlägt dagegen fehl, da beide denselben Primärschlüssel nutzen:

```
C_COLLECTION($empsCollection)
C_OBJECT($emp;$emp2;$employees)

$empsCollection:=New collection
$emp:=New object
$emp.ID:=10001  // Primärschlüssel existiert nicht
$emp.firstName:="Simone"
$emp.lastName:="Martin"
$emp.__NEW:=True
$empsCollection.push($emp)

$emp2:=New object
$emp2.ID:=10001  // Gleicher Primärschlüssel, existiert bereits
$emp2.firstName:="Marc"
$emp2.lastName:="Smith"
$emp2.__NEW:=True
$empsCollection.push($emp2)
$employees:=ds.Employee.fromCollection($empsCollection)
 //Erste Entity wird erstellt
 //Fehler wegen doppeltem Schlüssel für die zweite Entity
```

dataClass.get( )

dataClass.get (primaryKey) -> Funktionsergebnis

Parameter	Typ		Beschreibung
primaryKey	Lange Ganzzahl, Text	→	Wert des Primärschlüssels der Entity
Funktionsergebnis	Entity	↩	Entity, die zum angegebenen Primärschlüssel passt

Beschreibung

Die Methode **dataClass.get( )** sucht in der Dataclass nach der Entity mit dem passenden Parameter *primaryKey*.

In *primaryKey* übergeben Sie den Wert des Primärschlüssels der zu findenden Entity. Der Typ des Werts muss zum Typ des Primärschlüssels passen, der im Datastore gesetzt wurde (Lange Ganzzahl oder Text). Mit der Methode **entity.getKey( )** und ihrem Parameter dk key as string können Sie sicherstellen, dass der Wert des Primärschlüssels immer als Text zurückgegeben wird.

Wird mit *primaryKey* keine Entity gefunden, wird eine **Null** Entity zurückgegeben.

Lazy loading wird angewendet, d.h. verknüpfte Daten werden nur bei Bedarf geladen.

Beispiel

```
C_OBJECT($entity)
$entity:=ds.Employee.get(167)  // gibt die Entity mit dem Primärschlüsselwert 167 zurück
$entity:=ds.Invoice.get("DGGX20030")  // gibt die Entity mit dem Primärschlüsselwert "DGGX20030" zurück
```

dataClass.new( )

dataClass.new () -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Entity	Neue Entity, die zur DataClass passt

Beschreibung

Die Methode **dataClass.new( )** erstellt im Speicher eine neue leere Entity, die mit der DataClass verknüpft ist.
Das Entity Objekt wird im Speicher angelegt und nicht in der Datenbank gesichert, bis die Methode **entity.save( )** aufgerufen wird. Wird die Entity vor dem Sichern gelöscht, kann sie nicht wiederhergestellt werden.

Beispiel

Dieses Beispiel erstellt eine neue Entity in der DataClass "Log" und speichert die Information im Attribut *info*:

```
C_OBJECT($entity)
$entity:=ds.Log.new()  //Eine Referenz erstellen
$entity.info:="New entry"  //ein paar Informationen speichern
$entity.save()  //die Entity sichern
```

dataClass.newSelection( )

dataClass.newSelection ({keepOrder}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
keepOrder	Lange Ganzzahl	→	dk keep ordered: Erstellt eine sortierte Entity-Selection dk non ordered: Erstellt eine unsortierte Entity-Selection (Standard ohne Angabe)
Funktionsergebnis	EntitySelection	←	Neue leere Entity Selection, verknüpft mit der DataClass

Beschreibung

Die Methode **dataClass.newSelection( )** erstellt im Speicher eine neue leere Entity-Selection, verknüpft mit der DataClass.

Zum Erstellen einer sortierten Entity-Selection übergeben Sie im Parameter *keepOrder* den Selector dk keep ordered. Ohne diesen Parameter oder über den Selector dk non ordered erstellt die Methode eine unsortierte Entity-Selection. Diese sind zwar schneller, Sie können sich jedoch nicht auf die Position von Entities innerhalb der Selection verlassen. Weitere Informationen dazu finden Sie im Abschnitt **Sortierte vs unsortierte Entity-Selections** des *4D Developer Guide*.

Wird die Entity Selection angelegt, enthält sie keine Entities (mySelection.length gibt 0 zurück). Durch aufeinanderfolgende Aufrufe der Methode **add( )** können Sie Entity-Selections schrittweise aufbauen.

Beispiel

```
C_OBJECT($USelection;$OSelection)
$USelection:=ds.Employee.newSelection()  //Eine unsortierte leere Entity-Selection erstellen
$OSelection:=ds.Employee.newSelection(dk keep ordered)  //Eine sortierte leere Entity-Selection erstellen
```

dataClass.query( )

dataClass.query (queryString {; value}{; value2 ; ... ; valueN}{; querySettings}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
queryString	Text	➡	Suchkriterium
value		➡	Wert(e) zum Vergleichen bei Platzhalter(n)
querySettings	Objekt	➡	Suchoptionen: parameters, queryPath, queryPlan
Funktionsergebnis	EntitySelection	↩	Neue Entity-Selection mit den Entities der dataClass, die zum Suchkriterium passen

Beschreibung

Die Methode **dataClass.query( )** sucht nach Entities, welche die in *queryString* und (optional) *value* angegebenen Suchkriterien erfüllen, für alle Entities in der DataClass oder Entity-Selection und gibt ein neues Objekt vom Typ *EntitySelection* zurück, das alle gefundenen Entities der DataClass enthält. *Lazy loading* wird angewendet.

Werden keine passenden Entities gefunden, wird eine leere *EntitySelection* zurückgegeben.

Parameter queryString

Der Parameter *queryString* verwendet folgende Syntax:

```
attributePath comparator value {logicalOperator attributePath comparator value}
```

Definitionen:

- **attributePath**: Name des Dataclass Attributs, in dem Sie die Suche ausführen wollen, z.B. "country". Das kann jeder gültige Attributspfad sein, z.B. "country.name".
  Bei einem Attributspfad vom Typ *Collection* verwenden Sie eckige Klammern [] zum Verwalten aller Vorkommen.
- **comparator**: Symbol zum Vergleichen von *attributePath* und *value*. Es gibt folgende Symbole:

Vergleich	Symbol(e)	Kommentar
Ist gleich	=, ==	Erhält passende Daten, unterstützt den Joker @, berücksichtigt weder Groß- und Kleinschreibung noch diakritische Zeichen.
	===, IS	Erhält passende Daten, bewertet @ als Standardzeichen und nicht als Joker, berücksichtigt weder Groß- und Kleinschreibung noch diakritische Zeichen.
Not equal to	#, !=	unterstützt den Joker (@)
	!==, IS NOT	bewertet @ als Standardzeichen und nicht als Joker
Kleiner als	<	
Größer als	>	
Kleiner als oder gleich	<=	
Größer als oder gleich	>=	
Enthalten in	IN	Erhält Daten, die mit mindestens einem Wert in einer Collection bzw. einem Satz Werte übereinstimmt
Nicht enthalten in einer Anweisung	NOT	Klammern sind zwingend, wenn NOT vor einer Anweisung mit mehreren Operatoren verwendet wird
Enthält Schlüsselwort	%	Schlüsselwörter lassen sich in Attributen vom Typ String oder Bild verwenden

- **value**: Wert zum Vergleichen mit dem aktuellen Wert der Eigenschaft jeder Entity in der Entity-Selection oder Element in der Collection. Das kann jeder Ausdruck vom gleichen Datentyp wie die Eigenschaft sein oder ein Platzhalter *:paramIndex* (siehe unten).
  Werte von Konstanten stehen zwischen einfachen Anführungszeichen. Die Schlüsselwörter *true*, *false* sind für Konstanten verboten. Mit dem Schlüsselwort "null" können Sie den Wert **Null** in einer Suche vergleichen. Diese Suche findet die Eigenschaften *null* und *undefined*.
  Für eine Suche nach einem String innerhalb eines anderen String (eine Suche "Enthalten in") verwenden Sie den Joker (@) in *value*, um den zu suchenden String zu isolieren, zum Beispiel: "@Smith@".
  Für numerische Werte dient Punkt als Dezimaltrenner. Datumsangeben müssen im Format "YYYY-MM-DD" sein.
  Bei einer Suche mit einem IN Vergleichsoperator muss *value* eine Collection sein bzw. Werte, die zum Typ des Attributspfads zwischen eckigen Klammern [] passen, getrennt durch Kommas (bei Strings müssen Anführungszeichen " mit "\" abschließen).
- **logicalOperator**: verbindet mehrere Bedingungen in der Suche (optional). Es gibt folgende logische Operatoren (Sie können Name oder Symbol übergeben):

Konjunktion	Symbol(e)
AND	&, &&, and
OR	|, ||, or

Anführungszeichen verwenden
Innerhalb der Suche müssen Sie einfache Anführungszeichen ' ' setzen, doppelte Anführungszeichen " " verwenden Sie zum Umrahmen der gesamten Suche. Andernfalls wird ein Fehler generiert. Zum Beispiel:

```
"employee.name = 'smith' AND employee.firstname = 'john'"
```

Klammern verwenden
Mit Klammern in einer Suche können Sie Prioritäten beim Berechnen setzen. Zum Beispiel:

```
"(employee.age >= 30 OR employee.age <= 65) AND (employee.salary <= 10000 OR employee.status = 'Manager')"
```

Parameter value und Platzhalter

Bei Suchen mit *Platzhaltern* sind Parameter *value* erforderlich. Platzhalter sind Tags, die Sie in Suchstrings einfügen und die beim Bewerten durch einen anderen Wert ersetzt werden. Sie können bis zu 128 Parameter *value* verwenden.

Hinweis: Werte für Platzhalter lassen sich auch als Collection in der Eigenschaft *parameters* des optionalen Parameters *querySettings* übergeben (nur Suchen für entitySelection und dataClass). Weitere Informationen dazu finden Sie unten im Abschnitt **Parameter querySettings**.

In *queryString* fügen Sie für jeden Platzhalter :*paramIndex* ein (d.h. "verwende den Parameter *paramIndex* der Suche als Wert zum Vergleichen") und dann übergeben Sie die angeforderten Werte als Parameter *value*. Beispiel: Für eine Suche nach Angestellten, die in Chicago wohnen und unter 10 000 verdienen, können Sie schreiben:

```
"employee.city = :1 & employee.salary < :2"; "Chicago";10000
```

Der Wert wird einmal am Anfang bewertet; er wird nicht für jedes Element bewertet.

Platzhalter in Suchen werden aus zwei Gründen empfohlen:

1. Verhindert Einfügen von böswilligem Code: Verwenden Sie direkt von Benutzern gefüllte Variablen im Suchstring, könnte ein Benutzer die Suchbedingungen durch Einfügen zusätzlicher Suchargumente verändern. Nehmen wir beispielsweise folgenden Suchstring:

   ```
   $vquery:="status = 'public'  &  name = "+myname  //Benutzer gibt Namen ein
   $result:=$col.query($vquery)
   ```

   Diese Suche scheint abgesichert, da nicht-öffentliche Daten gefiltert werden. Gibt der Benutzer jedoch im Bereich *myname* etwas ein wie *OR status='private'*, wäre der Suchstring beim Interpretieren verändert und könnte private Daten zurückgeben.
   Mit Platzhaltern ist ein Überschreiben der Sicherheitsbedingungen nicht möglich:

   ```
   $result:=$col.query("status='public'  &  name=:1";$myname)
   ```

   Gibt der Benutzer hier *OR status='private'* im Bereich *myname* ein, wird das im Suchstring nicht interpretiert, sondern nur als Wert übergeben. Die Suche nach einer Person mit Namen "smith OR status='private'" schlägt einfach fehl.

2. Sie müssen sich nicht um Formatierungsprobleme kümmern. Zusätzlich können Sie Variablen oder Ausdrücke in Suchargumenten verwenden, z.B.:

   ```
   $result:=$col.query("address.city = :1  &  name =:2";$city;$myVar+"@")
   ```

Nach Nullwerten suchen
Beim Suchen nach Nullwerten können Sie keine Syntax mit Platzhaltern verwenden, da die Such-Engine Null als einen unerwarteten Vergleichswert betrachtet. Führen Sie beispielsweise folgende Suche aus:

```
$vSingles:=ds.Person.query("spouse = :1";Null)  // funktioniert NICHT
```

erhalten Sie nicht das erwartete Ergebnis, da der Nullwert von 4D als ein Fehler gewertet wird, der sich aus der Bewertung des Parameters ergibt (z.B. ein Attribut aus einer anderen Suche). Für derartige Suchen müssen Sie die direkte Suchsyntax verwenden:

```
$vSingles:=ds.Person.query("spouse = null")  //korrekte Syntax
```

Parameter querySettings

Hinweis: Dieser Parameter wird nur von den Methoden **entitySelection.query( )** und **dataClass.query( )** unterstützt.

In den Parametern *querySettings* können Sie ein Objekt mit zusätzlichen Optionen übergeben. Es gibt folgende Eigenschaften:

Eigenschaft	Typ	Beschreibung
parameters	Collection	Werte zum Vergleichen beim Verwenden von Platzhaltern in *queryString* (alternativer Weg, um Platzhaltern Werte zu übergeben). Werte, die in Parametern *value* direkt übergeben wurden, werden an die Platzhalter-Sequenz angehängt.
queryPlan	Boolean	Gibt in der resultierenden Entity Collection direkt vor der Ausführung die ausführliche Beschreibung der Suche zurück oder nicht, z.B. die geplante Suche. Die zurückgegebene Eigenschaft ist ein Objekt mit jeder geplanten Suche und untergeordneten Suchen bei komplexen Suchläufen. Diese Option ist während der Entwicklungsphase einer Anwendung hilfreich und wird in der Regel zusammen mit queryPath verwendet. Standard wenn weggelassen: false
queryPath	Boolean	Gibt in der resultierenden Entity Collection die ausführliche Beschreibung der aktuell durchgeführten Suche zurück oder nicht. Die zurückgegebene Eigenschaft ist ein Objekt mit dem aktuellen Pfad für die Suche (in der Regel identisch mit queryPlan, kann unterschiedlich sein, wenn die Engine die Suche optimiert), sowie Bearbeitungszeit und die Anzahl der gefundenen Datensätze. Diese Option ist während der Entwicklungsphase einer Anwendung hilfreich. Standard wenn weggelassen: false

Über queryPlan und queryPath
Die in queryPlan/queryPath gespeicherten Angaben enthalten den Suchtyp (indiziert und sequentiell) und jede notwendige Untersuche zusammen mit den Verbindungsoperatoren. Suchpfade enthalten auch die Anzahl der gefundenen Entities und die erforderliche Zeit zum Ausführen jedes Suchkriteriums. Das sind nützliche Informationen, die Sie beim Entwickeln Ihrer Anwendung analysieren können. Generally, Die Beschreibung des Suchplans und seines Pfads sind im allgemeinen identisch. Sie können u.U. unterschiedlich sein, da 4D beim Ausführen der Suche dynamische Optimierungen zum Verbessern der Performance einfügen kann. Die 4D Engine kann z.B. dynamisch eine indizierte Suche in eine sequentielle umwandeln, wenn sie diese für schneller hält. Das kann bei geringer Anzahl der gesuchten Entities der Fall sein.
Führen Sie beispielsweise folgende Suche aus:

```
$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}]}]}]}]}
```

Beispiele

Verschiedene Beispiele für gültige Suchen.
Standardsuche mit Platzhaltern:

```
$entitySelection:=dataClass.query("(firstName = :1 or firstName = :2) and (lastName = :3 or lastName = :4)";"D@";"R@";"S@";"K@")
```

Standardsuche ohne Platzhalter:

```
$entitySelection :=dataClass.query("firstName = 'S@'")
```

Suche mit einer verknüpften dataClass:

```
$entitySelection:=dataClass.query("lastName = :1 and manager.lastName = :2";"M@";"S@")
```

Suche mit queryPlan und queryPath Objekten:

```
$entitySelection:=dataClass.query("(firstName = :1 or firstName = :2) and (lastName = :3 or lastName = :4)";"D@";"R@";"S@";"K@";New object("queryPlan";True;"queryPath";True))

 //Sie können dann diese Eigenschaften in der resultierenden Entity-Selection erhalten
C_OBJECT($queryPlan;$queryPath)
$queryPlan:=$entitySelection.queryPlan
$queryPath:=$entitySelection.queryPath
```

Suche mit Platzhaltern und Werten in Form einer Collection:

```
$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)
```

Suche in einer Anweisung NOT:

```
$entitySelection:=dataClass.query("not(firstName=Kim)")
```

Suche mit einem Attributspfad vom Typ *Collection*:

```
$entitySelection:=dataClass.query("additionalInfo.hobbies[].name = horsebackriding")
```

Suche mit einem Attributspfad vom Typ *Object*:

```
$entitySelection:=ds.Employee.query("extra.eyeColor = :1";"blue")
```

Suche mit einer Anweisung IN:

```
$entitySelection:=dataClass.query("firstName in :1";New collection("Kim";"Dixie"))
```

Suche mit einer Anweisung NOT (IN):

```
$entitySelection:=ds.Employee.query("not (firstName in :1)";New collection("John";"Jane"))
```

Suche mit einem Datum:

```
$entitySelection:=dataClass.query("birthDate > :1";"1970-01-01")
```

ORDA - DataClassAttribute

Weitere Informationen zu Dataclass Attributen und Beispiele finden Sie im Abschnitt **Attribute der Dataclass** des *4D Developer's Guide*.

- dataClassAttribute.kind Neu 17.0
- dataClassAttribute.name Neu 17.0
- dataClassAttribute.relatedDataClass Neu 17.0

dataClassAttribute.kind

Parameter	Typ	Beschreibung
dataClassAttribute.kind	String	Art des Attributs

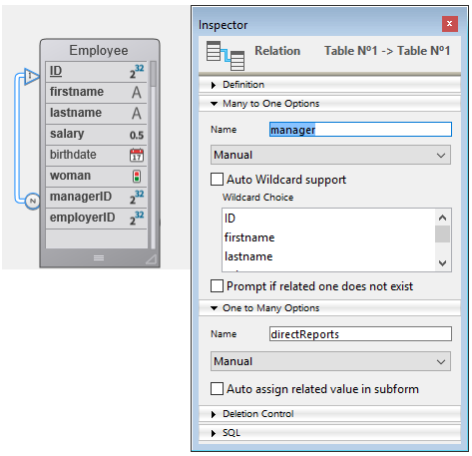
Beschreibung

Die Eigenschaft **dataClassAttribute.kind** gibt die Kategorie des Attributs zurück. Der zurückgegebene Wert kann folgendes sein:

- "storage": Skalares oder Speicherattribut , z.B. Attribut, das einen Wert speichert, nicht eine Referenz auf ein anderes Attribut
- "relatedEntity": N -> 1 Verknüpfungsattribut (Referenz auf eine Entity)
- "relatedEntities": 1 -> N Verknüpfungsattribut (Referenz auf eine Entity-Selection)

Beispiel

Gegeben sind folgende Tabelle und Verknüpfung:




```
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

Parameter	Typ	Beschreibung
dataClassAttribute.name	String	Name des Attributs, wie in der Datenbankstruktur definiert

Beschreibung

Die Eigenschaft **dataClassAttribute.name** gibt den Namen des Objekts *dataClassAttribute* als String zurück.

Beispiel

```
C_TEXT($attName)
$attName:=ds.Employee.lastname.name  //$attName="lastname"
```

dataClassAttribute.relatedDataClass

Parameter	Typ	Beschreibung
dataClassAttribute.relatedDataClass	String	Name der verknüpften Dataclass

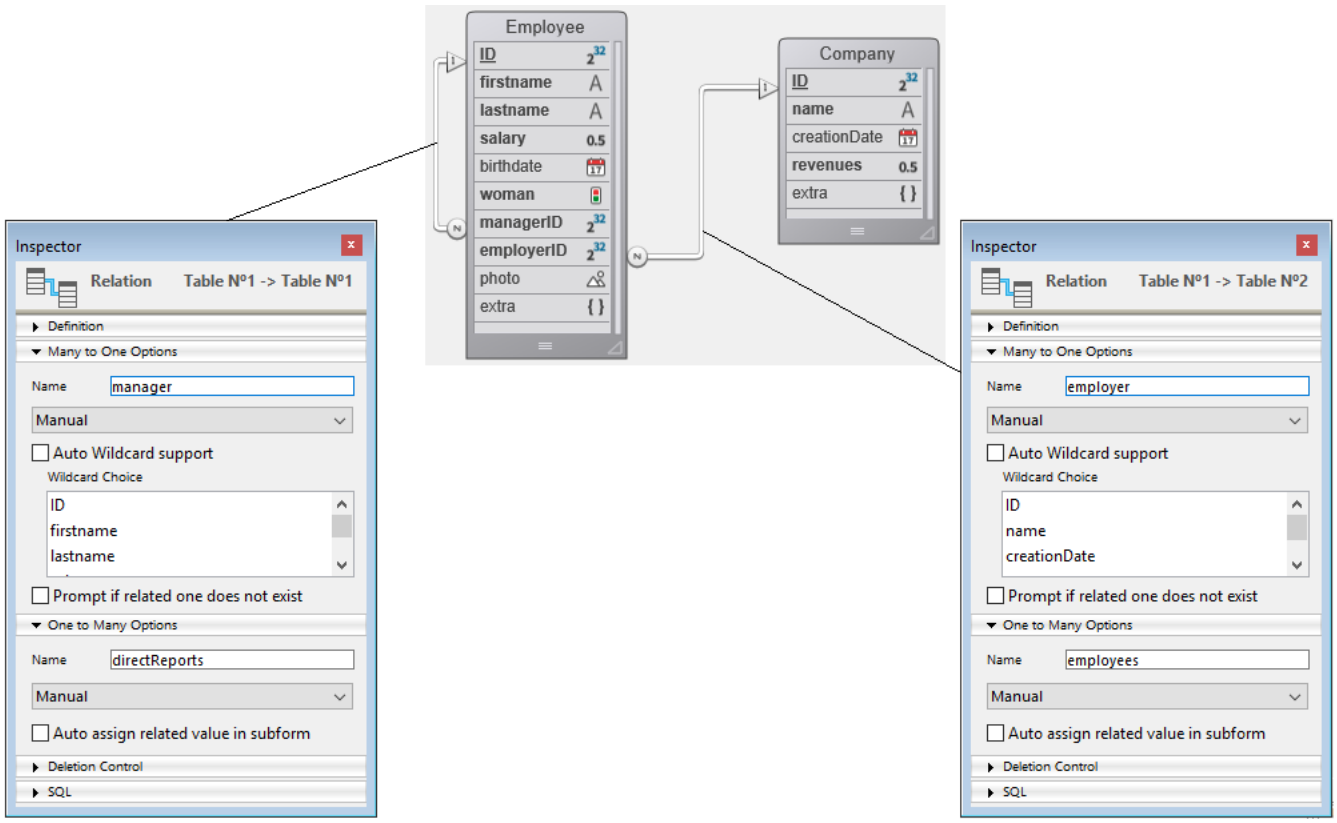
Beschreibung

Hinweis: Diese Eigenschaft ist nur mit Attributen der Eigenschaft "relatedEntity" oder "relatedEntities" **dataClassAttribute.kind** verfügbar.

Die Eigenschaft **dataClassAttribute.relatedDataClass** gibt den Namen der Dataclass zurück, die mit dem Attribut verknüpft ist.

Beispiel

Nehmen wir folgende Struktur:




```
C_TEXT($relClass1;$relClassN)
$relClass1:=ds.Employee.employer.relatedDataClass  //$relClass1="Company"
$relClassN:=ds.Employee.directReports.relatedDataClass  //$relClassN="Employee"
```

ORDA - DataStore

Ausführliche Informationen zum Datastore und Beispiele finden Sie auf der Seite **Datastore** des *4D Developer's Guide*.

ds.{dataclassName} Neu 17.0
ds Neu 17.0

ds.{dataclassName}

Parameter	Typ	Beschreibung
ds.{dataclassName}	DataClass	Objekt DataClass

Beschreibung

Jede Dataclass im Datastore ist als eine Eigenschaft des Objekts **ds** verfügbar. Das zurückgegebene Objekt enthält eine Beschreibung der Dataclass.

Dataclass Objekte können spezifische Methoden unter dem Thema **ORDA - DataClass** nutzen.

Beispiel

```
C_OBJECT($emp;$sel)
$emp:=ds.Employee  //$emp enthält die Dataclass Employee
$sel:=$emp.all()  //erhält eine Entity-Selection aller Angestellten

 //Sie können auch direkt schreiben:
$sel:=ds.Employee.all()
```

ds

ds -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Objekt	➲	Neue Referenz auf Datastore

Beschreibung

Der Befehl **ds** gibt eine neue Referenz auf den Datastore zurück, der zur aktuellen 4D Datenbank passt.

Hinweis: Mit **ds** muss Ihre Datenbank die für ORDA geltenden Anforderungen erfüllen. Weitere Informationen dazu finden Sie im Abschnitt **Voraussetzungen**.

Dieser Datastore wird automatisch geöffnet und ist durch **ds** direkt verfügbar. Es gelten folgende Prinzipien:

- Ein Datastore verweist nur auf Tabellen mit einem einzigen Primärschlüssel. Tabellen ohne Primärschlüssel oder mit zusammengesetzten Primärschlüsseln werden nicht berücksichtigt.
- Attribute vom Typ BLOB werden im Datastore nicht verwaltet.

Weitere Informationen dazu finden Sie im Abschnitt **Datastore**.

Beispiel

Den Datastore der 4D Datenbank verwenden:

```
$result:=ds.Employee.query("firstName = :1";"S@")
```

ORDA - Entity

Weitere Informationen zu Entities mit Code-Beispielen finden Sie im Abschnitt **Entities** des *4D Developer Guide*.

- entity.{attributeName} Neu 17.0
- entity.clone Neu 17.0
- entity.diff Neu 17.0
- entity.drop Neu 17.0
- entity.first Neu 17.0
- entity.fromObject Neu 17.0
- entity.getKey Neu 17.0
- entity.getSelection Neu 17.0
- entity.getStamp Neu 17.0
- entity.indexOf Neu 17.0
- entity.isNew Neu 17.0
- entity.last Neu 17.0
- entity.lock Neu 17.0
- entity.next Neu 17.0
- entity.previous Neu 17.0
- entity.reload Neu 17.0
- entity.save Neu 17.0
- entity.toObject Neu 17.0
- entity.touched Neu 17.0
- entity.touchedAttributes Neu 17.0
- entity.unlock Neu 17.0

entity.{attributeName}

Parameter	Typ	Beschreibung
entity.{attributeName}	Mixed	Aktueller Wert des Attributs in der Entity

Beschreibung

Jedes Attribut der Dataclass ist als Eigenschaft einer Entity verfügbar, die den Attributwert der Entity speichert.
Hinweis: Dataclass Attribute sind auch über die alternative Syntax mit [ ] zugänglich.
Der Typ des Attributwerts richtet sich nach der Art des Attributs (Verknüpfung oder Attribut):

- Ist *attributeName* vom Typ **storage**:
  gibt **entity.attributeName** einen Wert vom gleichen Typ wie *attributeName* zurück.
- Ist *attributeName* vom Typ **relatedEntity**:
  gibt **entity.attributeName** eine verknüpfte Entity zurück. Werte der verknüpften Entity sind direkt über verschachtelte Eigenschaften verfügbar, z.B. "myEntity.employer.employees[0].lastname".
- Ist *attributeName* vom Typ **relatedEntities**:
  gibt **entity.attributeName** eine neue Entity-Selection der verknüpften Entities zurück. Duplikate werden entfernt, die Entity-Selection wird unsortiert zurückgegeben.

Hinweis: Weitere Informationen dazu finden Sie im Abschnitt **dataClassAttribute.kind**.

Beispiel

```
C_OBJECT($myEntity)
$myEntity:=ds.Employee.new() //Ein neues Objekt vom Typ Entity erstellen
$myEntity.name:="Dupont" // 'Dupont' dem Attribut 'name' zuweisen
$myEntity.firstname:="John" //'John' dem Attribut 'firstname' zuweisen
$myEntity.save() //die Entity sichern
```

entity.clone( )

entity.clone () -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Entity	Neue Entity, die auf den Datensatz verweist

Beschreibung

Die Methode **entity.clone( )** erstellt im Speicher eine neue Entity, die auf denselben Datensatz wie die ursprüngliche Entity verweist.

Mit dieser Methode können Sie Entities getrennt aktualisieren. Beachten Sie, dass jede Änderung in Entities im referenzierten Datensatz nur gesichert wird, wenn die Methode **entity.save( )** ausgeführt wird.

Beispiel

```
C_OBJECT($emp;$empCloned)
$emp:=ds.Employee.get(672)
$empCloned:=$emp.clone()

$emp.lastName:="Smith"  //in $emp ausgeführte Updates sind in $empCloned nicht ausgeführt
```

entity.diff( )

entity.diff (entityToCompare {; attributesToCompare}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
entityToCompare	Entity	➡	Entity zum Vergleichen mit der ursprünglichen Entity
attributesToCompare	Collection	➡	Attributsnamen zum Vergleichen
Funktionsergebnis	Collection	⮌	Unterschiede zwischen den Entities

Beschreibung

Die Methode **entity.diff( )** vergleicht den Inhalt von zwei Entities und gibt die Unterschiede zurück.

In *entityToCompare* übergeben Sie die Entity zum Vergleichen mit der urspünglichen Entity.

In *attributesToInspect* können Sie spezifische Attribute zum Vergleichen angeben. Sind Attribute angegeben, wird der Vergleich nur mit diesen Attributen ausgeführt. Ohne Angabe werden alle Unterschiede zwischen den Entities zurückgegeben.

Die Unterschiede werden als Collection von Objekten mit folgenden Eigenschaften zurückgegeben:

Eigenschaftsname	Typ	Beschreibung
attributeName	String	Name des Attributs
value	Abhängig vom Attributstyp	Wert des Attributs in der Entity
otherValue	Abhängig vom Attributstyp	Wert des Attributs in *entityToCompare*

Nur Attribute mit unterschiedlichen Werten sind in der Collection enthalten. Werden keine Unterschiede gefunden, gibt **entity.diff( )** eine leere Collection zurück.

Die Methode gilt für Eigenschaften der Art *storage* oder *relatedEntity* (siehe **dataClassAttribute.kind**). Wurde eine verknüpfte Entity aktualisiert (d.h. der Fremdschlüssel), werden die Namen der verknüpften Entity und ihres Primärschlüssels als Eigenschaften *attributeName* zurückgegeben (*value* und *otherValue* sind leer für den Namen der verknüpften Entity).

Ist eine der verglichenen Entities Null, wird ein Fehler generiert.

Beispiel 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)  // Alle Unterschiede werden zurückgegeben
$diff2:=$clone.diff(employee;New collection"firstName";"lastName"))
 // Nur Unterschiede in firstName und lastName werden zurückgegeben
```

$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" } ]
```


Beispiel 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 (alle Unterschiede werden zurückgegeben):

[{ "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 von Company "otherValue": "[object Entity]"// Entity 118 von Company }]

vCompareResult2 (nur Unterschiede in $attributesToInspect werden zurückgegeben)

[{ "attributeName": "firstName", "value": "Karla update", "otherValue": "Karla" }, { "attributeName": "lastName", "value": "Marrero update", "otherValue": "Marrero" }]

vCompareResult3 (nur Unterschiede in betroffenen Attributen in $e1 werden zurückgegeben)

[{ "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]",// Entity 117 von Company "otherValue": "[object Entity]"// Entity 118 von Company }]

entity.drop( )

entity.drop ({mode}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
mode	Lange Ganzzahl	→	dk force drop if stamp changed: Löschen erzwingen, selbst wenn sich der Stempel inzwischen geändert hat.
Funktionsergebnis	Objekt	↩	Ergebnis des Löschvorgangs

Beschreibung

Die Methode **entity.drop( )** löscht aus dem Datastore die in der Entity enthaltenen Daten aus der Tabelle, die der DataClass zugewiesen ist. Beachten Sie, dass die Entity im Speicher erhalten bleibt.

In einer Applikation mit mehreren Benutzern oder mehreren Prozessen wird die Methode **entity.drop( )** mit dem optimistischen Sperrverfahren ausgeführt. Dabei wird der interne Sperrstempel bei jedem Sichern des Datensatzes automatisch erhöht. Weitere Informationen dazu finden Sie auf der Seite **Entity sperren**.

Standardmäßig, also ohne den Parameter *mode*, gibt die Methode einen Fehler zurück (siehe unten), wenn die gleiche Entity zwischenzeitlich durch einen anderen Prozess oder Benutzer geändert wurde (der Stempel hat sich geändert).
Mit dem Parameter *mode* können Sie die Option dk force drop if stamp changed übergeben, so dass die Entity gelöscht wird, auch wenn sich der Stempel geändert hat (und der Primärschlüssel gleich bleibt).

Ergebnis

Das von **entity.drop( )** zurückgegebene Opjekt enthält folgende Eigenschaften:

Eigenschaft		Typ	Beschreibung
success		Boolean	wahr bei erfolgreichem Löschvorgang, sonst falsch
			Nur bei Fehler verfügbar:
status(*)		Zahl	Fehler-Code, siehe unten
statusText(*)		Text	Beschreibung des Fehlers, siehe unten
			Nur bei Fehler beim pessimistischen Sperrverfahren verfügbar:
LockKindText		Text	"Gesperrt durch Datensatz"
lockInfo		Objekt	Information über Ursprung des Sperrens
	task_id	Zahl	Prozess Id
	user_name	Text	Benutzername der Sitzung auf dem Rechner
	user4d_id	Text	Benutzername im 4D Datenbankverzeichnis
	host_name	Text	Rechnername
	task_name	Text	Prozessname
	client_version	Text	
			Nur bei ernstem Fehler verfügbar (z.B. Primärschlüssel existiert bereits, Festplatte voll...):
errors		Collection von Objekten	
	message	Text	Fehlermeldung
	component signature	Text	interne Signatur der Komponente (z.B. "dmbg" steht für die Datenbankkomponente)
	errCode	Zahl	Fehlercode

(*) Bei einem Fehler in den Eigenschaften *status* und *statusText* des Objekts *Result* können folgende Werte zurückgegeben werden:

Konstante	Wert	Kommentar
dk status entity does not exist anymore	5	Die Entity existiert nicht mehr in den Daten. Dieser Fehler kann in folgenden Fällen auftreten: • Die Entity wurde gelöscht (der Stempel hat sich geändert und der Speicherplatz ist jetzt frei) • Die Entity wurde gelöscht und durch eine andere mit einem anderen Primärschlüssel ersetzt (der Stempel hat sich geändert und eine neue Entity verwendet jetzt den Speicherplatz). Mit **entity.drop( )** wird dieser Fehler beim Verwenden der Option dk force drop if stamp changed zurückgegeben. Mit **entity.lock( )** wird dieser Fehler beim Verwenden der Option dk reload if stamp changed zurückgegeben. **Zugewiesener Statustext**: "Entity does not exist anymore"
dk status locked	3	Die Entity wird durch pessimistisches Sperrverfahren gesperrt. **Zugewiesener Statustext**: "Already locked"
dk status serious error	4	Ein ernsthafter Fehler ist ein low-level Fehler in der Anwendung, wie z.B. duplizierter Schlüssel, Hardware Fehler, etc. **Zugewiesener Statustext**: "Other error"
dk status stamp has changed	2	Der interne Stempelwert der Entity passt nicht zum Wert der in den Daten gespeicherten Entity (optimistisches Sperrverfahren). • mit **entity.save( )**: nur Fehler, wenn die Option dk auto merge nicht verwendet wird • mit **entity.drop( )**: nur Fehler, wenn die Option dk force drop if stamp changed nicht verwendet wird • mit **entity.lock( )**: nur Fehler, wenn die Option dk reload if stamp changed nicht verwendet wird **Zugewiesener Statustext**: "Stamp has changed"

Beispiel 1

Beispiel ohne die Option 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
   :($status.success)
      ALERT("Sie haben "+$employee.firstName+" "+$employee.lastName gelöscht)  //Die gelöschte Entity bleibt im Speicher
   :($status.status=dk status stamp has changed)
      ALERT($status.statusText)
End case
```

Beispiel 2

Beispiel mit der Option 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
   :($status.success)
      ALERT("Sie haben "+$employee.firstName+" "+$employee.lastName gelöscht)  // Die gelöschte Entity bleibt im Speicher
   :($status.status=dk status entity does not exist anymore)
      ALERT($status.statusText)
End case
```

entity.first( )

entity.first () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Entity, Null		Erste Entity der Entity-Selection mit der ursprünglichen Entity

Beschreibung

Die Methode **entity.first( )** gibt eine Referenz zur Entity an erster Stelle in der Entity-Selection zurück, zu der die ursprüngliche Entity gehört.

Gehört die Entity zu keiner vorhandenen Entity-Selection (z.B. **entity.getSelection( )** gibt Null zurück), gibt die Methode **Null** zurück.

Beispiel

```
C_OBJECT($employees;$employee;$firstEmployee)
$employees:=ds.Employee.query("lastName = :1";"H@")  //Diese Entity-Selection enthält 3 Entities
$employee:=$employees[2]
$firstEmployee:=$employee.first()  //$firstEmployee ist die erste Entity in der Entity-Selection $employees
```

entity.fromObject( )

entity.fromObject (object)

Parameter	Typ		Beschreibung
object	Objekt	➡	Objekt zum Füllen der Entity

Beschreibung

Die Methode **entity.fromObject( )** füllt die Entity mit dem Inhalt von *object*.

Hinweis: Diese Methode ändert die ursprüngliche Entity.

Das Abbilden zwischen Objekt und Entity erfolgt über die Attributsnamen:

- Existiert eine Eigenschaft des Objekts nicht in der Dataclass, wird sie ignoriert.
- Die Datentypen müssen übereinstimmen. Bei unterschiedlichen Typen in Objekt und Dataclass wird das Attribut in der Entity nicht gefüllt. 4D versucht, den Datentyp soweit wie möglich anzupassen (siehe **Datentypen**), sonst bleibt das Attribut unangetastet.

object kann eine **verknüpfte Entity** unter folgenden Bedingungen verwalten:

- *object* enthält den Fremdschlüssel selbst oder
- *object* enthält eine Objekteigenschaft mit demselben Namen wie die verknüpfte Entity mit einer einzelnen Eigenschaft mit Namen "__KEY".
- existiert die verknüpfte Entity nicht, wird sie ignoriert.

Beispiel

Mit folgendem Objekt *$o*:

```
{ "firstName": "Mary", "lastName": "Smith", "salary": 36500, "birthDate": "1958-10-27T00:00:00.000Z", "woman": true, "managerID": 411,// relatedEntity gegeben mit PK "employerID": 20 // relatedEntity gegeben mit PK }
```

erstellt nachfolgender Code eine Entity mit den verknüpften Entities **manager** und **employer**.

```
C_OBJECT($o)
$entity:=ds.Emp.new()
$entity.fromObject($o)
$entity.save()
```

Sie können auch eine verknüpfte Entity verwenden, die als Objekt gegeben ist:

```
{ "firstName": "Marie", "lastName": "Lechat", "salary": 68400, "birthDate": "1971-09-03T00:00:00.000Z", "woman": false, "employer": {// relatedEntity gegeben als Objekt "__KEY": "21" }, "manager": {// relatedEntity gegeben als Objekt "__KEY": "411" } }
```

entity.getKey( )

entity.getKey ({mode}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
mode	Lange Ganzzahl	➡	dk key as string: Primärschlüssel wird als String zurückgegeben, egal, welcher Typ er ist
Funktionsergebnis	Lange Ganzzahl, Text	➲	Wert des Primärschlüssels der Entity

Beschreibung

Die Methode **entity.getKey( )** gibt den Wert des Primärschlüssels von Entity zurück.

The **entity.getKey( )** method [#descv]returns the primary key value of the entity from the table related to the dataclass[#/descv].

Primärschlüssel können Zahlen vom Typ Lange Ganzzahl oder Strings sein. Mit der Option dk key as string im Parameter *mode* können Sie erzwingen, dass der Wert des zurückgegebenen Primärschlüssels ein String ist, unabhängig vom Typ des aktuellen Primärschlüssels.

Beispiel

```
C_OBJECT($employees;$employee)
$employees:=ds.Employee.query("lastName=:1";"Smith")
$employee:=$employees[0]
ALERT("Der Primärschlüssel ist "+$employee.getKey(dk key as string))
```

entity.getSelection( )

entity.getSelection () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	EntitySelection, Null		Entity-Selection, zu der die Entity gehört

Beschreibung

Die Methode **entity.getSelection( )** gibt die Entity-Selection zurück, zu der die Entity gehört.
Gehört die Entity nicht zu einer Entity-Selection, gibt die Methode **Null** zurück.

Beispiel

```
C_OBJECT($emp;$employees;$employees2)
$emp:=ds.Employee.get(672)  // Diese Entity gehört zu keiner Entity-Selection
$employees:=$emp.getSelection()  // $employees ist Null

$employees2:=ds.Employee.query("lastName=:1";"Smith")  //Diese Entity-Selection enthält 6 Entities
$emp:=$employees2[0]  // Diese Entity gehört zu einer Entity-Selection
ALERT("Die Entity-Selection enthält "+String($emp.getSelection().length)+" Entities")
```

entity.getStamp( )

entity.getStamp () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Zahl	➲	Stempel der Entity (0, wenn Entity gerade erstellt wurde)

Beschreibung

Die Methode **entity.getStamp( )** gibt den aktuellen Wert des Stempels von Entity zurück.

Der interne Stempel wird von 4D automatisch mit jedem Sichern der Entity erhöht. Er verwaltet gleichzeitige Benutzerzugriffe und Änderungen in denselben Entities. Weitere Informationen dazu finden Sie auf der Seite **Entity sperren**.

Hinweis: Für eine neue Entity (nie gesichert) gibt die Methode 0 zurück. Um abzufragen, ob eine Entity gerade erstellt wurde, ist die Methode **entity.isNew( )** besser geeignet.

Beispiel

```
C_OBJECT($entity)
C_LONGINT($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}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
entitySelection	EntitySelection	➡	Position der Entity wird in Bezug auf ihre Entity-Selection angegeben
Funktionsergebnis	Lange Ganzzahl	➲	Position der Entity in einer Entity-Selection

Beschreibung

Die Methode **entity.indexOf( )** gibt die Position der Entity in einer Entity-Selection zurück.

Standardmäßig, d.h. ohne den Parameter *entitySelection*, gibt die Methode die Position der Entity in ihrer eigenen Entity-Selection zurück. Sonst gibt sie die Position der Entity innerhalb der in *entitySelection* angegebenen Entity-Selection zurück.

Der resultierende Wert liegt zwischen 0 und der Länge der Entity-Selection -1.

- Hat die Entity keine Entity-Selection oder gehört sie nicht zu *entitySelection*, gibt die Methode -1 zurück
- Ist *entitySelection* Null oder gehört nicht zur gleichen Dataclass wie die Entity, wird ein Fehler erzeugt

Beispiel

```
C_OBJECT($employees;$employee)
$employees:=ds.Employee.query("lastName = :1";"H@")  //Diese Entity-Selection enthält 3 Entities
$employee:=$employees[1]  //Diese Entity gehört zu einer Entity-Selection
ALERT("Der Index der Entity in ihrer eigenen Entity-Selection ist "+String($employee.indexOf()))  //1

C_OBJECT($employee)
$employee:=ds.Employee.get(725)  //Diese Entity gehört nicht zu einer Entity-Selection
ALERT("Der Index der Entity ist "+String($employee.indexOf()))  // -1
```

entity.isNew( )

entity.isNew () -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Wahr, wenn Entity gerade erstellt und noch nicht gesichert wurde. Sonst Falsch

Beschreibung

Die Methode **entity.isNew( )** gibt wahr zurück, wenn die Entity, auf die sie angewendet wird, gerade erstellt und noch nicht im Datastore gesichert wurde. Sonst gibt sie Falsch zurück.

Beispiel

```
C_OBJECT($emp)

$emp:=ds.Employee.new()

If($emp.isNew())
    ALERT("Dies ist eine neue Entity")
End if
```

entity.last( )

entity.last () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Entity, Null		Letzte Entity in der Entity-Selection mit der ursprünglichen Entity

Beschreibung

Die Methode **entity.last( )** gibt eine Referenz auf die Entity an letzter Stelle in der Entity-Selection zurück, zu der die ursprüngliche Entity gehört.

Gehört die Entity nicht zu einer existierenden Entity-Selection (z.B. **entity.getSelection( )** gibt Null zurück), gibt die Methode **Null** zurück.

Beispiel

```
C_OBJECT($employees;$employee;$lastEmployee)
$employees:=ds.Employee.query("lastName = :1";"H@")  //Diese Entity-Selection enthält 3 Entities
$employee:=$employees[0]
$lastEmployee:=$employee.last()  //$lastEmployee ist die letzte Entity in der Entity-Selection $employees
```

entity.lock( )

entity.lock ({mode}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
mode	Lange Ganzzahl	→	dk reload if stamp changed: Erneutes Laden vor Sperren, wenn sich Stempel geändert hat.
Funktionsergebnis	Objekt	←	Ergebnis des Sperrvorgangs

Beschreibung

Die Methode **entity.lock( )** setzt ein pessimistisches Sperrverfahren für den durch Entity repräsentierten Datensatz. Das Sperren erfolgt für einen Datensatz und alle Referenzen der Entity im aktuellen Prozess.

Weitere Informationen dazu finden Sie auf der Seite **Entity sperren**.

Andere Prozesse sehen diesen Datensatz als gesperrt (beim Versuch, dieselbe Entity mit dieser Methode zu sperren, enthält die Eigenschaft **result.success** falsch). Nur Methoden, die in der "sperrenden" Sitzung ausgeführt werden, können Attribute der Entity bearbeiten und sichern. Andere Sitzungen können die Entity im Nur-Lesen Modus laden, jedoch weder Werte eingeben noch sichern.

Ein gesperrter Datensatz wird entsperrt:

- wenn die Methode **unlock()** für eine passende Entity im gleichen Prozess aufgerufen wird
- automatisch, sobald keine Variable mehr auf diese Entity verweist. Ist die Sperre z.B. nur auf eine lokale Variable mit dieser Entity gesetzt, wird die Entity bei Beenden der Methode automatisch entsperrt. Solange es Referenzen auf die Entity im Speicher gibt, bleibt der Datensatz gesperrt.

Standardmäßig, d.h. ohne den Parameter *mode*, gibt die Methode einen Fehler zurück (siehe unten), wenn dieselbe Entity inzwischen von einem anderen Prozess oder Benutzer geändert wurde (z.B. der Stempel hat sich geändert).
Übergeben Sie im Parameter *mode* die Option dk reload if stamp changed, wird kein Fehler zurückgegeben und die Entity wird erneut geladen, auch wenn sich der Stempel geändert hat (wenn die Entity noch existiert und der Fremdschlüssel gleichgeblieben ist).

Ergebnis

Das von der Methode **entity.lock( )** zurückgegebene Objekt enthält folgende Eigenschaften:

Eigenschaft		Typ	Beschreibung
success		Boolean	wahr bei erfolgreicher Sperraktion (oder wenn die Entity im aktuellen Prozess bereits gesperrt ist), sonst falsch.
			Nur verfügbar bei Verwenden der Option dk reload if stamp changed:
wasReloaded		Boolean	wahr bei erfolgreichem Laden der Entity, sonst falsch.
			Nur verfügbar bei Auftreten eines Fehlers:
status(*)		Zahl	Fehlercode, siehe unten
statusText(*)		Text	Fehlerbeschreibung, siehe unten
			Nur verfügbar bei Auftreten eines Fehlers beim pessimistischen Sperrverfahren:
LockKindText		text	"Gesperrt durch Datensatz"
lockInfo		Objekt	Angabe zum Ursprung der Sperre
	task_id	Zahl	Prozess-id
	user_name	Text	Benutzername der Sitzung auf dem Rechner
	user4d_id	Text	Benutzername im Verzeichnnis der 4D Datenbank
	host_name	Text	Rechnername
	task_name	Text	Prozessname
	client_version	Text	
			Nur verfügbar bei Auftreten eines ernsthaften Fehlers (Primärschlüssel existiert bereits, Festplatte voll, ...):
errors		Collection von Objekten	
	message	Text	Fehlermeldung
	component signature	text	interne Signatur der Komponente (z.B. "dmbg" steht für die Komponente der Datenbank)
	errCode	Zahl	Fehlercode

(*) In den Eigenschaften *status* und *statusText* des Objekts *Result* können bei einem Fehler folgende Werte zurückgegeben werden:

Konstante	Wert	Kommentar
dk status entity does not exist anymore	5	Die Entity existiert nicht mehr in den Daten. Dieser Fehler kann in folgenden Fällen auftreten: • Die Entity wurde gelöscht (der Stempel hat sich geändert und der Speicherplatz ist jetzt frei) • Die Entity wurde gelöscht und durch eine andere mit einem anderen Primärschlüssel ersetzt (der Stempel hat sich geändert und eine neue Entity verwendet jetzt den Speicherplatz). Mit **entity.drop( )** wird dieser Fehler beim Verwenden der Option dk force drop if stamp changed zurückgegeben. Mit **entity.lock( )** wird dieser Fehler beim Verwenden der Option dk reload if stamp changed zurückgegeben. **Zugewiesener Statustext**: "Entity does not exist anymore"
dk status locked	3	Die Entity wird durch pessimistisches Sperrverfahren gesperrt. **Zugewiesener Statustext**: "Already locked"
dk status serious error	4	Ein ernsthafter Fehler ist ein low-level Fehler in der Anwendung, wie z.B. duplizierter Schlüssel, Hardware Fehler, etc. **Zugewiesener Statustext**: "Other error"
dk status stamp has changed	2	Der interne Stempelwert der Entity passt nicht zum Wert der in den Daten gespeicherten Entity (optimistisches Sperrverfahren). • mit **entity.save( )**: nur Fehler, wenn die Option dk auto merge nicht verwendet wird • mit **entity.drop( )**: nur Fehler, wenn die Option dk force drop if stamp changed nicht verwendet wird • mit **entity.lock( )**: nur Fehler, wenn die Option dk reload if stamp changed nicht verwendet wird **Zugewiesener Statustext**: "Stamp has changed"

Beispiel 1

Beispiel mit Fehler:

```
C_OBJECT($employee;$status)
$employee:=ds.Employee.get(716)
$status:=$employee.lock()
Case of
   :($status.success)
      ALERT("Sie haben "+$employee.firstName+" "+$employee.lastName gesperrt)
   :($status.status=dk status stamp has changed)
      ALERT($status.statusText)
End case
```

Beispiel 2

Beispiel mit der Option dk reload if stamp changed:

```
C_OBJECT($employee;$status)

$employee:=ds.Employee.get(717)
$status:=$employee.lock(dk reload if stamp changed)
Case of
   :($status.success)
      ALERT("Sie haben "+$employee.firstName+" "+$employee.lastName gesperrt)
   :($status.status=dk status entity does not exist anymore)
      ALERT($status.statusText)
End case
```

entity.next( )

entity.next () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Entity, Null	⮌	Referenz auf die nächste Entity in der Entity-Selection

Beschreibung

Die Methode **entity.next( )** gibt eine Referenz auf die nächste Entity in der Entity-Selection zurück, zu der die ursprüngliche Entity gehört.

Gehört die Entity nicht zu einer existierenden Entity-Selection (z.B. **entity.getSelection( )** gibt Null zurück), gibt die Methode **Null** zurück.

Gibt es keine gültige nächste Entity in der Entity-Selection (Sie sind z.B. bei der letzten Entity der Selection), gibt die Methode Null zurück. Wurde die nächste Entity verschoben, gibt die Methode die nächste gültige Entity zurück (und evtl. Null).

Beispiel

```
C_OBJECT($employees;$employee;$nextEmployee)
$employees:=ds.Employee.query("lastName = :1";"H@")  //Diese Entity-Selection enthält 3 Entities
$employee:=$employees[0]
$nextEmployee:=$employee.next()  //$nextEmployee ist die zweite Entity der Entity-Selection   $employees entity selection
```

entity.previous( )

entity.previous () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Entity, Null		Gibt die vorige Entity in einer Entity-Selection zurück

Beschreibung

Die Methode **entity.previous( )** gibt eine Referenz auf die vorige Entity in der Entity-Selection zurück, zu der die ursprüngliche Entity gehört.

Gehört die Entity nicht zu einer existierenden Entity-Selection (z.B. **entity.getSelection( )** gibt Null zurück), gibt die Methode **Null** zurück.

Gibt es keine gültige vorige Entity in der Entity-Selection (Sie sind z.B. bei der ersten Entity der Selection), gibt die Methode Null zurück. Wurde die vorige Entity verschoben, gibt die Methode die davor liegende gültige Entity zurück (und evtl. Null).

Beispiel

```
C_OBJECT($employees;$employee;$previousEmployee)
$employees:=ds.Employee.query("lastName = :1";"H@") //Diese Entity-Selection enthält 3 Entities
$employee:=$employees[1]
$previousEmployee:=$employee.previous() //$previousEmployee ist die vorige Entity in der Entity-Selection $employees
```

entity.reload( )

entity.reload () -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Objekt	Status object

Beschreibung

Die Methode **entity.reload( )** lädt erneut den Inhalt der Entity im Speicher. Das passiert nur, wenn die Entity mit demselben Primärschlüssel noch existiert.

Ergebnis

Das von **entity.reload( )** zurückgegebene Objekt enthält folgende Eigenschaften:

Eigenschaft	Typ	Beschreibung
success	Boolean	wahr, wenn erneutes Laden erfolgreich ist, sonst falsch
		Nur verfügbar bei Auftreten eines Fehlers:
status(*)	Zahl	Fehlercode, siehe unten
statusText(*)	Text	Fehlerbeschreibung, siehe unten

(*) Bei einem Fehler sind in den Eigenschaften *status* und *statusText* des Objekts *Result* folgende Werte möglich:

Konstante	Wert	Kommentar
dk status entity does not exist anymore	5	Die Entity existiert nicht mehr in den Daten. Dieser Fehler kann in folgenden Fällen auftreten: • Die Entity wurde gelöscht (der Stempel hat sich geändert und der Speicherplatz ist jetzt frei) • Die Entity wurde gelöscht und durch eine andere mit einem anderen Primärschlüssel ersetzt (der Stempel hat sich geändert und eine neue Entity verwendet jetzt den Speicherplatz). Mit **entity.drop( )** wird dieser Fehler beim Verwenden der Option dk force drop if stamp changed zurückgegeben. Mit **entity.lock( )** wird dieser Fehler beim Verwenden der Option dk reload if stamp changed zurückgegeben. **Zugewiesener Statustext**: "Entity does not exist anymore"
dk status serious error	4	Ein ernsthafter Fehler ist ein low-level Fehler in der Anwendung, wie z.B. duplizierter Schlüssel, Hardware Fehler, etc. **Zugewiesener Statustext**: "Other error"

Beispiel

```
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("Die Entity wurde gelöscht")
End case
```

entity.save( )

entity.save ({mode}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
mode	Lange Ganzzahl	→	dk auto merge: Aktiviert automatisches Mischen
Funktionsergebnis	Objekt	←	Ergebnis des Sichernvorgangs

Beschreibung

Die Methode **entity.save( )** sichert die Änderungen in der Entity in der Tabelle, die mit ihrer Dataclass verbunden ist. Sie müssen diese Methode nach Erstellen oder Ändern einer Entity aufrufen, wenn sie gesichert werden soll.

Das Sichern wird nur ausgeführt, wenn mindestens ein Attribut der Entity "angetastet" wurde (siehe Methoden **entity.touched( )** und **entity.touchedAttributes( )**). Sonst führt die Methode nichts aus (der Trigger wird nicht aufgerufen).

In einer Applikation mit mehreren Benutzern oder mehreren Prozessen wird die Methode **entity.save( )** mit dem optimistischen Sperrverfahren ausgeführt. Dabei wird der interne Sperrstempel bei jedem Sichern des Datensatzes automatisch erhöht. Weitere Informationen dazu finden Sie auf der Seite **Entity sperren**.

Standardmäßig, also ohne den Parameter *mode*, gibt die Methode einen Fehler zurück (siehe unten), wenn die gleiche Entity zwischenzeitlich durch einen anderen Prozess oder Benutzer geändert wurde, egal welches Attribut geändert wurde.
Mit dem Parameter *mode* können Sie die Option dk auto merge übergeben: Ist automatisches Mischen aktiviert, und macht ein anderer Prozess bzw. Benutzer zwischenzeitlich eine Änderung in derselben Entity, aber in einem anderen Attribut, gibt es keinen Fehler. Die in der Entity gesicherten Daten sind eine Kombination ("merge") aller nicht-konkurrierenden Änderungen. Nur bei Änderungen im gleichen Attribut schlägt das Sichern fehl und ein Fehler wird zurückgegeben, und das auch mit dem Modus "auto merge".

Hinweis: "auto merge" ist nicht verfügbar für Attribute vom Typ Bild, Objekt und Text, wenn sie außerhalb des Datensatzes gesichert werden. Gleichzeitige Änderungen in diesen Attributen führen zum Fehler dk status stamp has changed.

Ergebnis

Das von **entity.save( )** zurückgegebene Objekt enthält folgende Eigenschaften:

Eigenschaft		Typ	Beschreibung
success		Boolean	wahr bei erfolgreichem Sichern, sonst falsch
			Nur bei Verwenden von dk auto merge verfügbar:
autoMerged		Boolean	wahr bei Ausführen von "auto merge", sonst falsch
			Nur bei Fehler verfügbar:
status(*)		Zahl	Fehlercode, siehe unten
statusText(*)		Text	Fehlerbeschreibung, siehe unten
			Nur bei Fehler beim pessimistischen Sperrverfahren verfügbar:
LockKindText		Text	"Gesperrt durch Datensatz"
lockInfo		Objekt	Information über Ursprung des Sperrens
	task_id	Zahl	Prozess Id
	user_name	Text	Benutzername der Sitzung auf dem Rechner
	user4d_id	Text	Benutzername im 4D Datenbankverzeichnis
	host_name	Text	Rechnername
	task_name	Text	Prozessname
	client_version	Text	
			Nur bei ernstem Fehler verfügbar (z.B. Primärschlüssel existiert bereits, Festplatte voll...):
errors		Collection von Objekten	
	message	Text	Fehlermeldung
	component signature	Text	interne Signatur der Komponente (z.B. "dmbg" steht für die Datenbankkomponente)
	errCode	Zahl	Fehlercode

(*) Bei einem Fehler in den Eigenschaften *status* und *statusText* des Objekts *Result* können folgende Werte zurückgegeben werden:

Konstante	Wert
dk status automerge failed	6
dk status entity does not exist anymore	5
dk status locked	3
dk status serious error	4
dk status stamp has changed	2

Beispiel 1

Eine neue Entity erstellen:

```
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
```

Beispiel 2

Eine Entity ohne die Option dk auto merge aktualisieren:

```
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
```

Beispiel 3

Eine Entity mit der Option dk auto merge aktualisieren:

```
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 (filter {; options}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
filter	String, Collection	➡	Gibt die Eigenschaften zum Extrahieren an
options	Lange Ganzzahl	➡	dk with primary key: fügt die Eigenschaft _KEY hinzu, dk with stamp: fügt die Eigenschaft _STAMP hinzu
Funktionsergebnis	Objekt	⮌	Aus der Entity erstelltes Objekt

Beschreibung

Die Methode **entity.toObject( )** gibt ein Objekt zurück, das aus der Entity erstellt wurde. Eigenschaftsnamen im Objekt entsprechen den Attributsnamen der Entity.

Im Parameter *filter* übergeben Sie die zu extrahierenden Entity Attribute. Zwei Syntaxarten sind erlaubt:

- Ein String mit Eigenschaftspfaden, getrennt durch Kommas: "propertyPath1, propertyPath2, ...".
- Eine Collection von Strings: ["propertyPath1","propertyPath2";...]

Wird *filter* für Attribute vom Typ **relatedEntity** angegeben:

- propertyPath = "relatedEntity" -> wird als einfache Form extrahiert: ein Objekt mit der Eigenschaft __KEY (Primärschlüssel).
- propertyPath = "relatedEntity.*" -> alle Eigenschaften werden extrahiert
- propertyPath = "relatedEntity.propertyName1; relatedEntity.propertyName2; ..." -> nur diese Eigenschaften werden extrahiert

Wird *filter* für Attribute vom Typ **relatedEntities** angegeben:

- propertyPath = "relatedEntities.*" -> alle Eigenschaften werden extrahiert
- propertyPath = "relatedEntities.propertyName1; relatedEntities.propertyName2; ..." -> nur diese Eigenschaften werden extrahiert

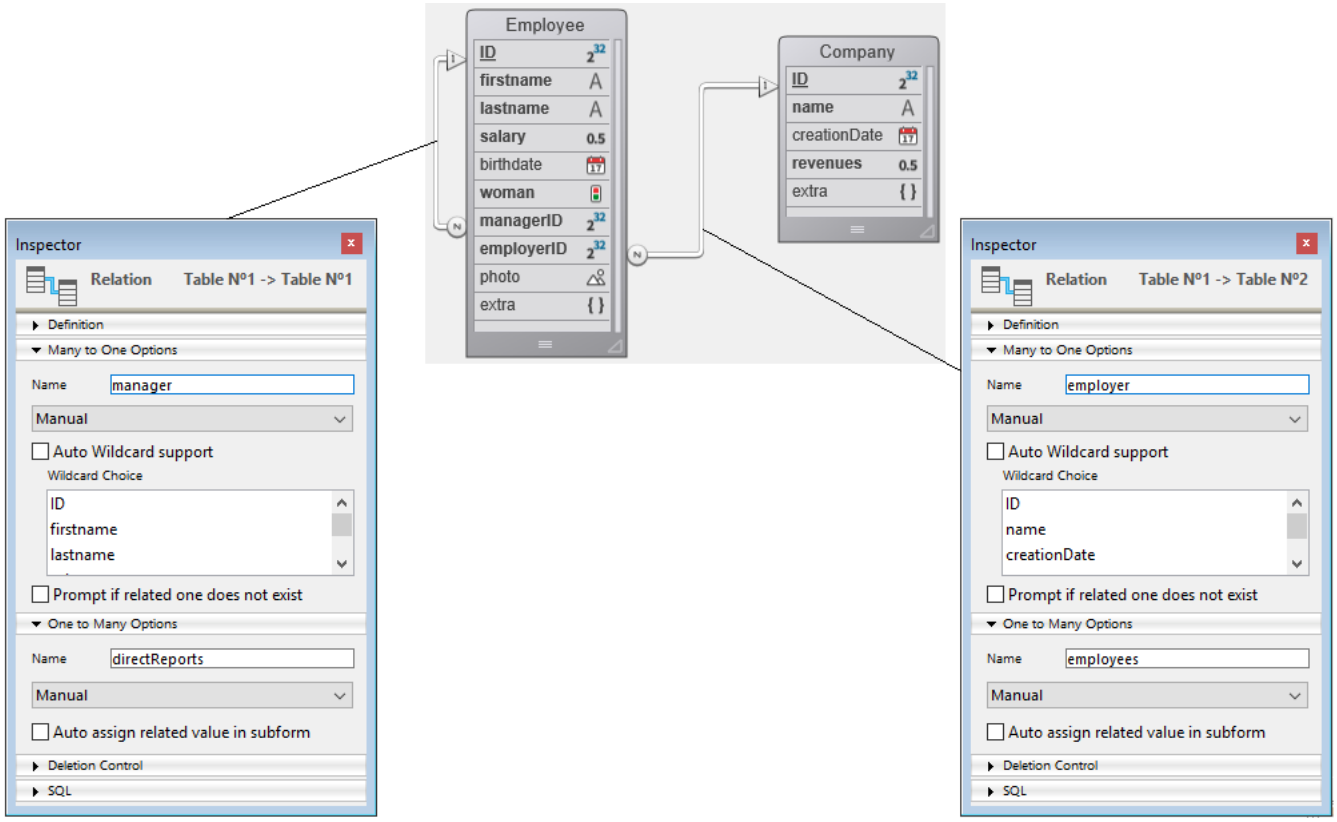
Enthält der Parameter *filter* einen leeren String oder "*", enthält das zurückgegebene Objekt:

- alle Speicherattribute der Entity
- Attribute vom Typ **relatedEntity**: Sie erhalten eine Eigenschaft mit demselben Namen wie die verknüpfte Entity (Name der Verknüpfung Viele-zu-Eine). Attribut wird als einfache Form extrahiert
- Attribute vom Typ **relatedEntities** werden nicht zurückgegeben.

Im Parameter *options* können Sie die Selektoren dk with primary key bzw. dk with stamp übergeben, um die Primärschlüssel bzw. Stempel in extrahierten Objekten hinzuzufügen.

Beispiel 1

Folgende Struktur wird für alle Beispiele dieses Abschnitts verwendet:



Ohne Parameter *filter*:

```
employeeObject:=employeeSelected.toObject()
```

Ergibt:

{ "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 als einfache Form "__KEY": 20 }, "manager": { "__KEY": 412 } }

Beispiel 2

Primärschlüssel und Stempel extrahieren:

```
employeeObject:=employeeSelected.toObject("";dk with primary key+dk with stamp)
```

Ergibt:

{ "__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 } }

Beispiel 3

Alle Attribute der relatedEntities extrahieren:

```
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 } }] }

Beispiel 4

Einige Attribute von relatedEntities extrahieren:

```
employeeObject:=employeeSelected.toObject("firstName, directReports.lastName")
```

Ergibt:

{ "firstName": "Greg", "directReports": [{ "lastName": "Boothe" }, { "lastName": "Caudill" }, { "lastName": "Gomes" }] }

Beispiel 5

Eine relatedEntity in einfacher Form extrahieren:

```
$coll:=New collection("firstName";"employer")
employeeObject:=employeeSelected.toObject($coll)
```

Ergibt:

{ "firstName": "Greg", "employer": { "__KEY": 20 } }

Beispiel 6

Alle Attribute einer relatedEntity extrahieren:

```
employeeObject:=employeeSelected.toObject("employer.*")
```

Ergibt:

{ "employer": { "ID": 20, "name": "India Astral Secretary", "creationDate": "1984-08-25T00:00:00.000Z", "revenues": 12000000, "extra": null } }

Beispiel 7

Einige Attribute einer relatedEntity extrahieren:

```
$col:=Create collection
$col.push("employer.name")
$col.push("employer.revenues")
employeeObject:=employeeSelected.toObject($col)
```

Ergibt:

{ "employer": { "name": "India Astral Secretary", "revenues": 12000000 } }

entity.touched( )

entity.touched () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean	➲	Wahr, wenn mindestens ein Entity Attribut geändert und noch nicht gesichert wurde, sonst falsch

Beschreibung

Die **entity.touched( )** Methode testet, ob seit dem Sichern der Entity oder Laden in den Speicher ein Entity Attribut geändert wurde.

Wurde ein Attribut geändert oder berechnet, gibt die Methode wahr zurück, sonst falsch. Mit dieser Methode können Sie herausfinden, ob Sie die Entity sichern müssen.

Diese Methode gibt für eine gerade erstellte neue Entity falsch zurück (mit **dataClass.new( )**). Verwenden Sie jedoch eine Methode zum Berechnen eines Attributs der Entity, gibt **entity.touched( )** wahr zurück. Rufen Sie zum Beispiel **entity.getKey( )** zum Berechnen des Primärschlüssels auf, gibt die Methode **entity.touched( )** wahr zurück.

Beispiel

Wir prüfen, ob es notwendig ist, die Entity zu sichern:

```
C_OBJECT($emp)
$emp:=ds.Employee.get(672)
$emp.firstName:=$emp.firstName  // Auch wenn das Attribut mit dem gleichen Wert aktualisiert wird, gilt es als angefasst

If($emp.touched())  //hat sich mindestens eins der Attribute geändert
    $emp.save()
 // sonst muss die Entity nicht gesichert werden
End if
```

entity.touchedAttributes( )

entity.touchedAttributes -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Collection		Namen der angefassten Attribute oder leere Collection

Beschreibung

Die Methode **entity.touchedAttributes( )** gibt die Namen der Attribute zurück, die seit dem Laden der Entity in den Speicher geändert wurden.

Dies gilt für Attribute vom Typ *storage* oder *relatedEntity* (siehe **dataClassAttribute.kind**).

Wurde eine verknüpfte Entity angefasst (*z.B.* der Fremdschlüssel), werden die Namen der verknüpften Entity und des Fremdschlüssels zurückgegeben.

Wurde kein Attribut der Entity angefasst, gibt die Methode eine leere Collection zurück.

Beispiel 1

```
C_COLLECTION($touchedAttributes)
C_OBJECT($emp)

$touchedAttributes:=New collection
$emp:=ds.Employee.get(725)
$emp.firstName:=$emp.firstName  //Auch wenn das Attribut mit dem gleichen Wert aktualisiert wurde, gilt das Attribut als angefasst
$emp.lastName:="Martin"
$touchedAttributes:=$emp.touchedAttributes()
 //$touchedAttributes: ["firstName","lastName"]
```

Beispiel 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"]
```

Es gilt folgendes:

- firstName und lastName sind vom Typ storage
- employer hat einen Typ relatedEntity
- employerID ist der Fremdschlüssel der verknüpften Entity employer

entity.unlock( )

entity.unlock () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Objekt		Status Objekt

Beschreibung

Die Methode **entity.unlock( )** hebt das pessimistische Sperrverfahren für den durch Entity repräsentierten Datensatz auf.

Weitere Informationen dazu finden Sie auf der Seite **Entity sperren**.

Ein Datensatz wird automatisch entsperrt, sobald keine Variable mehr auf diese Entity verweist (Beispiel: Ist die Sperre nur auf eine lokale Variable mit dieser Entity gesetzt, wird diese und somit auch der Datensatz entsperrt, wenn der Prozess endet).

Hinweis: Ist ein Datensatz gesperrt, muss er vom sperrenden Prozess und in der Entity Referenz, die die Sperre setzt, entsperrt werden. Zum Beispiel:

```
$e1:=dataClass.all()[0]
$e2:=dataClass.all()[0]
$e1.lock() // Entsperren muss auf $e1 ausgeführt werden. Entsperren auf $e2 schlägt fehl.
```

Ergebnis

Das von **entity.unlock( )** zurückgegebene Objekt enthält folgende Eigenschaft:

Eigenschaft	Typ	Beschreibung
success	Boolean	wahr bei erfolgreichem Entsperren, sonst falsch; *success* ist ebenfalls falsch, wenn Entsperren auf eine gelöschte Entity, einen nicht gesperrten Datensatz oder einen Datensatz ausgeführt wird, der von einem anderen Prozess oder einer anderen Entity gesperrt ist.

Beispiel

```
C_OBJECT($employee;$status)

$employee:=ds.Employee.get(725)
$status:=$employee.lock()
... //processing
$status:=$employee.unlock()
If($status.success)
    ALERT("Die Entity ist jetzt entsperrt")
End if
```

ORDA - EntitySelection

- entitySelection.{attributeName} Neu 17.0
- entitySelection.length Neu 17.0
- entitySelection.queryPath Neu 17.0
- entitySelection.queryPlan Neu 17.0
- entitySelection[index] Neu 17.0
- Create entity selection Neu 17.0
- entitySelection.add Neu 17.0
- entitySelection.and Neu 17.0
- entitySelection.average Neu 17.0
- entitySelection.contains Neu 17.0
- entitySelection.count Neu 17.0
- entitySelection.distinct Neu 17.0
- entitySelection.drop Neu 17.0
- entitySelection.first Neu 17.0
- entitySelection.isOrdered Neu 17.0
- entitySelection.last Neu 17.0
- entitySelection.max Neu 17.0
- entitySelection.min Neu 17.0
- entitySelection.minus Neu 17.0
- entitySelection.or Neu 17.0
- entitySelection.orderBy Neu 17.0
- entitySelection.query Neu 17.0
- entitySelection.slice Neu 17.0
- entitySelection.sum Neu 17.0
- entitySelection.toCollection Neu 17.0
- USE ENTITY SELECTION Neu 17.0

entitySelection.{attributeName}

Parameter	Typ	Beschreibung
entitySelection.{attributeName}	Collection, EntitySelection	Projektion der Attributwerte für die Entity-Selection

Beschreibung

Jedes Attribut der Dataclass lässt sich als Eigenschaft einer Entity-Selection verwenden, um eine "Projektion" von Werten für die Attribute in der Entity-Selection zurückzugeben. Die abgebildeten Werte können je nach Art des Attributs (Speicher oder Verknüpfung) eine Collection oder eine neue Entity-Selection sein.

- Ist *attributeName* vom Typ **storage**:
  gibt **entitySelection.attributeName** eine Collection mit Werten vom gleichen Typ wie *attributeName* zurück.
- Ist *attributeName* vom Typ **relatedEntity**:
  gibt **entitySelection.attributeName** eine neue Entity-Selection der verknüpften Werte vom gleichen Typ wie *attributeName* zurück. Duplikationen werden entfernt. Es wird eine unsortierte Entity-Selection zurückgegeben.
- Ist *attributeName* vom Typ **relatedEntities**:
  gibt **entitySelection.attributeName** eine neue Entity-Selection der verknüpften Werte vom gleichen Typ wie *attributeName* zurück. Duplikationen werden entfernt. Es wird eine unsortierte Entity-Selection zurückgegeben.

Wird ein Verknüpfungsattribut als Eigenschaft einer Entity-Selection verwendet, ist das Ergebnis immer eine andere Entity-Selection, selbst wenn nur eine Entity zurückgegeben wird. Werden keine Entities zurückgegeben, ist das Ergebnis eine leere Entity-Selection.
Weitere Informationen dazu finden Sie im Abschnitt **dataClassAttribute.kind**.

Beispiel 1

Projektion der Speicherwerte:

```
C_COLLECTION(firstNames)
$entitySelection:=ds.Employee.all()
firstNames:=$entitySelection.firstName  // firstName ist vom Typ String
```

Das ergibt eine Collection von Strings, zum Beispiel:

```
[    "Joanna",    "Alexandra",    "Rick" ]
```

Beispiel 2

Projektion der verknüpften Entity:

```
C_OBJECT($es;$entitySelection)
$entitySelection:=ds.Employee.all()
$es:=$entitySelection.employer  // employer ist mit der Dataclass Company verknüpft
```

Das ergibt eine Entity-Selection von Company; evtl. vorhandene Duplikationen werden entfernt.

Beispiel 3

Projection der verknüpften Entities:

```
C_OBJECT($es)
$es:=ds.Employee.all().directReports  // directReports ist mit der Dataclass Employee verknüpft
```

Das ergibt eine Entity-Selection von Employee; evtl. vorhandene Duplikationen werden entfernt.

entitySelection.length

Parameter	Typ		Beschreibung
entitySelection.length	Lange Ganzzahl	⮌	Anzahl der Entities in der Entity-Selection

Beschreibung

Die Eigenschaft **entitySelection.length** gibt die Anzahl Entities in der Entity-Selection zurück. Ist die Entity-Selection leer, gibt sie 0 zurück.
Entity-Selections haben immer eine Eigenschaft **length**.

Beispiel

```
C_LONGINT(vSize)
vSize:=ds.Employee.query("gender = :1";"male").length
ALERT(String(vSize)+" männliche Angestellte gefunden.")
```

entitySelection.queryPath

Parameter	Typ		Beschreibung
entitySelection.queryPath	Text		Beschreibung der gerade durchgeführten Suche

Beschreibung

Die Eigenschaft **entitySelection.queryPath** enthält die ausführliche Beschreibung der aktuell von 4D durchgeführte Suche. Sie ist für für durch Suchläufe generierte *entitySelection* Objekte verfügbar, wenn im Parameter *querySettings* der Methode **query()** die Eigenschaft *"queryPath":true* übergeben wurde.

Weitere Informationen dazu finden Sie im Abschnitt **Parameter querySettings**.

entitySelection.queryPlan

Parameter	Typ	Beschreibung
entitySelection.queryPlan	Text	Beschreibung der Suche direkt vor der Ausführung

Beschreibung

Die Eigenschaft **entitySelection.queryPlan** enthält eine ausführliche Beschreibung der Suche direkt vor der Ausführung (z.B. vor der geplanten Ausführung). Diese Eigenschaft ist für durch Suchläufe generierte *entitySelection* Objekte verfügbar, wenn im Parameter *querySettings* der Methode **query()** die Eigenschaft *"queryPlan":true* übergeben wurde.

Weitere Informationen dazu finden Sie im Abschnitt **Parameter querySettings**.

entitySelection[index]

Parameter	Typ		Beschreibung
entitySelection[index]	Entity		Entity zum angegebenen Index

Beschreibung

Über die Notation **entitySelection[index]** können Sie in der Entity-Selection über die Standardsyntax für Collection auf Entities zugreifen. Dazu übergeben Sie im Parameter *index* die Position der gewünschten Entity.

Beachten Sie, dass die entsprechende Entity vom Datastore geladen wird.

index kann jede Nummer zwischen 0 und **entitySelection.length**-1 sein.

- Ist *index* außerhalb des Bereichs, wird ein Fehler zurückgegeben.
- Entspricht *index* einer gelöschten Entity, wird Null zurückgegeben.

entitySelection[index] ist ein **nicht-zuweisbarer Ausdruck**, d.h. er lässt sich nicht als editierbare Entity Referenz mit Methoden wie **entity.lock( )** oder **entity.save( )** verwenden. Um mit der entsprechenden Entity zu arbeiten, müssen Sie den zurückgegebenen Ausdruck einem zuweisbaren Ausdruck zuordnen, z.B. eine Variable. Beispiele:

```
$sel:=ds.Employee.all()  //Die Entity-Selection erstellen
 //ungültige Anweisungen:
$result:=$sel[0].lock()  //funktioniert NICHT
$sel[0].lastName:="Smith"  //funktioniert NICHT
$result:=$sel[0].save()  //funktioniert NICHT
 //gültiger Code:
$entity:=$sel[0]  //OK
$entity.lastName:="Smith"  //OK
$entity.save()  //OK
```

Beispiel

```
C_OBJECT($employees;$employee)
$employees:=ds.Employee.query("lastName = :1";"H@")
$employee:=$employees[2]  // Die 3. Entity der Entity-Selection $employees wird von der Datenbank geladen
```

Create entity selection

Create entity selection (dsTable) -> Funktionsergebnis

Parameter	Typ		Beschreibung
dsTable	Tabelle	➡	Aktuelle Auswahl der Tabelle in der 4D Datenbank zum Anlegen der Entity-Selection
Funktionsergebnis	EntitySelection	↩	Entity-Selection in der Dataclass, die mit der gegebenen Tabelle verknüpft ist

Beschreibung

Der Befehl **Create entity selection** erstellt eine neue Entity-Selection, anhand der in *dsTable* angegebenen aktuellen Auswahl der Tabelle, die mit der Dataclass verbunden ist.

Es wird eine sortierte Entity-Selection angelegt, die die Reihenfolge der aktuellen Auswahl beibehält. Weitere Informationen dazu finden Sie im Abschnitt **Sortierte vs unsortierte Entity-Selections** des *4D Developer Guide*.

Ist *dsTable* nicht in **ds** abgebildet, wird ein Fehler zurückgegeben.

Beispiel

```
C_OBJECT($employees)
ALL RECORDS([Employee])
$employees:=Create entity selection([Employee])
 // Die Entity-Selection $employees enthält jetzt einen Satz Referenzen auf alle Entities in der Dataclass Employee
```

entitySelection.add( )

entitySelection.add (entity)

Parameter	Typ		Beschreibung
entity	Entity	➡	Entity zum Hinzufügen in der Entity-Selection

Beschreibung

Die Methode **entitySelection.add( )** fügt die angegebene *Entity* in der Entity-Selection hinzu.
Hinweis: Diese Methode ändert die ursprüngliche Entity-Selection.

- Bei sortierter Entity-Selection wird *entity* am Ende der Selection hinzugefügt. Gehört eine Referenz derselben Entity bereits zur Entity-Selection, wird sie dupliziert und eine neue Referenz hinzugefügt.
- Bei unsortierter Entity-Selection wird *entity* irgendwo in der Selection hinzugefügt.

Weitere Informationen dazu finden Sie im Abschnitt **Sortierte vs unsortierte Entity-Selections** des *4D Developer Guide*.
Sind *entity* und die Entity-Selection nicht mit derselben Dataclass verknüpft, tritt ein Fehler auf. Ist die Entity zum Hinzufügen Null, gibt es keinen Fehler.

Beispiel

```
C_OBJECT($employees;$employee)
$employees:=ds.Employee.query("lastName = :1";"S@")
$employee:=ds.Employee.new()
$employee.lastName:="Smith"
$employee.save()
$employees.add($employee)  //Die entity $employee wird in der Entity-Selection $employees hinzugefügt
```

entitySelection.and( )

entitySelection.and (entity | entitySelection) -> Funktionsergebnis

Parameter	Typ		Beschreibung
entity | entitySelection	Entity, EntitySelection	➡	Schnittmenge aus Entity und Entity-Selection
Funktionsergebnis	EntitySelection	⮌	Neue Entity-Selection mit durch logisches AND gebildeter Schnittmenge

Beschreibung

Die Methode **entitySelection.and( )** kombiniert die Entity-Selection mit dem Parameter *entity* oder *entitySelection* über den Operator logisches AND; sie gibt eine neue, unsortierte Entity-Selection zurück, die nur die Entities enthält, auf die in der Entity-Selection und im jeweiligen Parameter verwiesen wird..

- Übergeben Sie als Parameter *entity*, kombinieren Sie diese Entity mit der Entity-Selection. Gehört die Entity zur Entity-Selection, wird eine neue Entity-Selection nur mit dieser Entity zurückgegeben. Sonst wird eine leere Entity-Selection zurückgegeben.
- Übergeben Sie als Parameter *entitySelection*, kombinieren Sie beide Entity-Selections. Zurückgegeben wird eine neue Entity-Selection mit nur den Entities, auf die in beiden Selections verwiesen wird. Gibt es keine gemeinsame Entity, wird eine leere Selection zurückgegeben.

Hinweis: Sie können sortierte bzw. nicht sortierte Entity-Selections vergleichen. Die sich daraus ergebende Selection ist immer unsortiert. Weitere Informationen dazu finden Sie im Abschnitt **Sortierte vs unsortierte Entity-Selections** des *4D Developer Guide*.

Ist die ursprüngliche Entity-Selection oder der Parameter *entitySelection* leer, oder ist *entity* Null, wird eine leere Entity-Selection zurückgegeben.

Sind die ursprüngliche Entity-Selection und der jeweilige Parameter nicht mit derselben Dataclass verknüpft, wird ein Fehler generiert.

Beispiel 1

```
C_OBJECT($employees1;$employee;$result)
$employees1:=ds.Employee.query("lastName = :1";"H@")  //Die Entity-Selection $employees1 enthält die Entity mit dem Primärschlüssel 710 und andere Entities
 //z.B. "Colin Hetrick" / "Grady Harness" / "Sherlock Holmes" (Primärschlüssel 710)
$employee:=ds.Employee.get(710)  // Gibt "Sherlock Holmes" zurück

$result:=$employees1.and($employee)  //$result ist eine Entity-Selection, die nur die Entity mit dem Primärschlüssel 710 enthält ("Sherlock Holmes")
```

Beispiel 2

Eine Selection der Angestellten mit Namen "Jones" und Wohnsitz New York erhalten:

```
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 (attributePath) -> Funktionsergebnis

Parameter	Typ		Beschreibung
attributePath	Text	→	Pfad des Attributs für die Berechnung
Funktionsergebnis	Null, Zahl	←	Arithmetisches Mittel (Durchschnitt) der Attributswerte der Entity

Beschreibung

Die Methode **entitySelection.average( )** gibt das arithmetische Mittel (Durchschnitt) aller Nicht-Null-Werte von *attributePath* in der Entity-Selection zurück.

Im Parameter *attributePath* übergeben Sie den Attributpfad für die Berechnung. Nur numerische Werte werden berücksichtigt, außer *attributePath* der Entity-Selection enthält Werte vom Typ *mixed*. In diesem Fall berücksichtigt **entitySelection.average( )** alle skalaren Elemente zum Berechnen des Durchschnitts.

Hinweis: Werte vom Typ Datum werden in numerische Werte (Sekunden) umgewandelt und dann zum Berechnen des Durchschnitts verwendet.

entitySelection.average( ) gibt Null zurück, wenn die Entity-Selection leer ist.

Ein Fehler wird zurückgegeben, wenn

- *attributePath* ein verknüpftes Attribut ist oder keine numerischen Werte enthält,
- *attributePath* in der Dataclass der Entity-Selection nicht gefunden wird.

Beispiel

Liste der Angestellten mit einem Gehalt über dem Durchschnittswert erhalten:

```
C_REAL($averageSalary)
C_OBJECT($moreThanAv)
$averageSalary:=ds.Employee.all().average("salary")
$moreThanAv:=ds.Employee.query("salary > :1"$averageSalary)
```

entitySelection.contains( )

entitySelection.contains (entity) -> Funktionsergebnis

Parameter	Typ		Beschreibung
entity	Entity	➡	Entity zum Bewerten
Funktionsergebnis	Boolean	⮌	Wahr, wenn die Entity zur Entity-Selection gehört, sonst falsch

Beschreibung

Die Methode **entitySelection.contains( )** gibt **wahr** zurück, wenn die *entity* Referenz zur Entity-Selection gehört, sonst falsch.
In *entity* übergeben Sie die zu suchende Entity in der Entity-Selection. Ist Entity Null, gibt die Methode falsch zurück.
Gehören *entity* und die Entity-Selection nicht zur gleichen Dataclass, wird ein Fehler generiert.

Beispiel

```
C_OBJECT($employees;$employee)

$employees:=ds.Employee.query("lastName=:1";"H@")
$employee:=ds.Employee.get(610)

If($employees.contains($employee))
    ALERT("The entity with primary key 610 has a last name beginning with H")
Else
    ALERT("The entity with primary key 610 does not have a last name beginning with H")
End if
```

entitySelection.count( )

entitySelection.count (attributePath) -> Funktionsergebnis

Parameter	Typ		Beschreibung
attributePath	Text	→	Pfad des Attributs für die Berechnung
Funktionsergebnis	Zahl	↩	Anzahl der nicht-Null Attributspfade in der Entity-Selection

Beschreibung

Die Methode **entitySelection.count( )** gibt die Anzahl der Entities in der Entity-Selection mit einem nicht-Null Wert in *attributePath* zurück.

Hinweis: Nur skalare Werte werden berücksichtigt. Werte vom Typ Objekt oder Collection werden als Nullwerte betrachtet.

Ein Fehler wird zurückgegeben, wenn:

- *attributePath* ein verknüpftes Attribut ist
- *attributePath* in der Dataclass der Entity-Selection nicht gefunden wird.

Beispiel

Die Gesamtzahl der Angestellten eines Unternehmens abfragen, ohne diejenigen, deren Jobname nicht angegeben ist:

```
C_OBJECT($sel)
C_REAL($count)

$sel:=ds.Employee.query("employer = :1";"Acme, Inc")
$count:=$sel.count("jobname")
```

entitySelection.distinct( )

entitySelection.distinct (attributePath {; option}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
attributePath	Text	➡	Pfad des Attributs zum Erhalten der nicht-wiederholten Werte
option	Lange Ganzzahl	➡	dk diacritical: diakritische Bewertung, dann ist z.B. "A" # "a"
Funktionsergebnis	Collection	⬅	Collection nur mit nicht-wiederholten Werten

Beschreibung

Die Methode **entitySelection.distinct( )** gibt eine Collection nur mit nicht-wiederholten (unterschiedlichen) Werten von *attributePath* in der Entity-Selection zurück.

Die zurückgegebene Collection ist automatisch sortiert. Es werden keine **Null** Werte zurückgegeben.

Im Parameter *attributePath* übergeben Sie das Entity Attribut, dessen nicht-wiederholte Werte Sie erhalten wollen. Das ist nur für skalare Werte möglich, also Text, Zahl, Boolean oder Datum. Ist *attributePath* ein Objektattribut mit Werten unterschiedlichen Typs, werden sie erst nach Typ gruppiert und anschließend sortiert. Typen werden in folgender Reihenfolge zurückgegeben:

1. Boolean
2. String
3. Zahl
4. Datum

Standardmäßig wird die Bewertung ohne diakritische Zeichen durchgeführt. Sollen Klein- und Großschreibung und Akzente berücksichtigt werden, übergeben Sie im Parameter *option* die Konstante dk diacritical.

Ein Fehler wird zurückgegeben, wenn:

- *attributePath* ein verknüpftes Attribut ist
- *attributePath* nicht in der Dataclass der Entity-Selection gefunden wird.

Beispiel

Eine Collection mit einem einzigen Element pro Ländername erhalten:

```
C_COLLECTION($countries)
$countries:=ds.Employee.all().distinct("address.country")
```

entitySelection.drop( )

entitySelection.drop ({mode}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
mode	Lange Ganzzahl	→	dk stop dropping on first error: Stoppt die Ausführung der Methode bei der ersten nicht löschbaren Entity
Funktionsergebnis	EntitySelection	↩	Leere Entity-Selection, wenn erfolgreich; sonst Entity-Selection mit den nicht löschbaren Entities

Beschreibung

Die Methode **entitySelection.drop( )** löscht die Entities, die zur Entity-Selection gehören. Die Entity-Selection bleibt im Speicher.

Hinweis: Löschen von Entities ist dauerhaft und lässt sich nicht rückgängig machen. Wir empfehlen daher, diese Aktion in einer Transaktion aufzurufen, um eine Rollback Option zu haben.

Wird beim Ausführen von **entitySelection.drop( )** eine gesperrteEntity gefunden, wird sie nicht gelöscht. Standardmäßig bearbeitet die Methode alle Entities der Entity-Selection und gibt nicht-löschbare Entities in der Entity-Selection zurück. Soll die Methode bei der ersten gefundenen nicht-löschbaren Entity stoppen, übergeben Sie im Parameter *mode* die Konstante dk stop dropping on first error.

Beispiel

Beispiel ohne die Option dk stop dropping on first error:

```
C_OBJECT($employees;$notDropped)
$employees:=ds.Employee.query("firstName=:1";"S@")
$notDropped:=$employees.drop() // $notDropped ist eine Entity-Selection mit allen nicht gelöschten Entities
If($notDropped.length=0) //Löschen ist erfolgreich, alle Entities wurden gelöscht
    ALERT("You have dropped "+String($employees.length)+" employees") //Die gelöschte Entity-Selection bleibt im Speicher
Else
    ALERT("Problem during drop, try later")
End if
```

Beispiel mit der Option dk stop dropping on first error:

```
C_OBJECT($employees;$notDropped)
$employees:=ds.Employee.query("firstName=:1";"S@")
$notDropped:=$employees.drop(dk stop dropping on first error) //$notDropped ist eine Entity-Selection mit der ersten nicht gelöschten Entity
If($notDropped.length=0) //Löschen ist erfolgreich, alle Entities wurden gelöscht
    ALERT("You have dropped "+String($employees.length)+" employees") //Die gelöschte Entity-Selection bleibt im Speicher
Else
    ALERT("Problem during drop, try later")
End if
```

entitySelection.first( )

entitySelection.first () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Entity, Null		Referenz zur ersten Entity der Entity-Selection

Beschreibung

Die Methode **entitySelection.first( )** gibt eine Referenz auf die Entity an der ersten Stelle der Entity-Selection zurück.
Das Ergebnis dieser Methode ist ähnlich wie:

```
$entity:=$entitySel[0]
```

Beide Anweisungen unterscheiden sich jedoch, wenn die Selection leer ist:

```
C_OBJECT($entitySel;$entity)
$entitySel:=ds.Emp.query("lastName = :1";"Nonexistentname")  //keine passende Entity
 //Entity-Selection ist dann leer
$entity:=$entitySel.first()  //gibt Null zurück
$entity:=$entitySel[0]  //generiert einen Fehler
```

Beispiel

```
C_OBJECT($entitySelection;$entity)
$entitySelection:=ds.Emp.query("salary > :1";100000)
If($entitySelection.length#0)
    $entity:=$entitySelection.first()
End if
```

entitySelection.isOrdered( )

entitySelection.isOrdered () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean	➲	Wahr, wenn die Entity-Selection sortiert ist, sonst falsch

Beschreibung

Die Methode **entitySelection.isOrdered( )** gibt wahr zurück, wenn die Entity-Selection sortiert ist; falsch, wenn sie unsortiert ist.
Weitere Informationen dazu finden Sie im Abschnitt **Sortierte vs unsortierte Entity-Selections**.

Beispiel

```
C_OBJECT($employees;$employee)
C_BOOLEAN($isOrdered)
$employees:=ds.Employee.newSelection(dk keep ordered)
$employee:=ds.Employee.get(714)  // Erhält die Entity mit Primärschlüssel 714

 //In einer sortierten Entity-Selection können wir dieselbe Entity mehrmals hinzufügen (Duplizierungen werden beibehalten)
$employees.add($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 () -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Entity, Null	➲	Referenz auf die letzte Entity in der Entity-Selection

Beschreibung

Die Methode **entitySelection.last( )** gibt eine Referenz auf die Entity an der letzten Stelle in der Entity-Selection zurück.
Das Ergebnis dieser Methode ist ähnlich zu:

```
$entity:=$entitySel[length-1]
```

Ist die Entity-Selection leer, gibt die Methode Null zurück.

Beispiel

```
C_OBJECT($entitySelection;$entity)
$entitySelection:=ds.Emp.query("salary < :1";50000)
If($entitySelection.length#0)
    $entity:=$entitySelection.last()
End if
```

entitySelection.max( )

entitySelection.max (attributePath) -> Funktionsergebnis

Parameter	Typ		Beschreibung
attributePath	Text	→	Pfad des Attributs für die Berechnung
Funktionsergebnis	Mixed	←	Höchster Wert des Attributs

Beschreibung

Die Methode **entitySelection.max( )** gibt den höchsten (oder maximum) Wert unter allen Werten von *attributePath* in der Entity-Selection zurück. Sie gibt den Wert der ersten Entity in der Entity-Selection zurück, als ob sie mit der Methode **entitySelection.orderBy( )** in aufsteigender Reihenfolge sortiert wäre.

Haben Sie im Parameter *attributePath* einen Pfad zu einem Objekt Attribut mit Werten unterschiedlichen Typs übergeben, gibt die Methode **entitySelection.max( )** den Maximumwert im ersten skalaren Elementtyp in der 4D Typenliste zurück (siehe **collection.sort( )**). Existiert *attributePath* nicht im Objekt, gibt **entitySelection.max( ) null** zurück.

entitySelection.max( ) gibt **null** zurück, wenn die Entity-Selection leer ist.

Ein Fehler wird zurückgegeben, wenn

- *attributePath* ein verknüpftes Attribut ist,
- *attributePath* in der Dataclass der Entity-Selection nicht gefunden wird.

Beispiel

Unter allen weiblichen Angestellten das höchste Gehalt herausfinden:

```
C_OBJECT($sel)
C_REAL($maxSalary)
$sel:=ds.Employee.query("gender = :1";"female")
$maxSalary:=$sel.max("salary")
```

entitySelection.min( )

entitySelection.min (attributePath) -> Funktionsergebnis

Parameter	Typ		Beschreibung
attributePath	Text	→	Pfad des Attributs für die Berechnung
Funktionsergebnis	Mixed	←	Niedrigster Wert des Attributs

Beschreibung

Die Methode **entitySelection.min( )** gibt den niedrigsten (oder minimum) Wert unter allen Werten von *attributePath* in der Entity-Selection zurück. Sie gibt den Wert der ersten Entity in der Entity-Selection zurück, als ob sie mit der Methode **entitySelection.orderBy( )** in aufsteigender Reihenfolge sortiert wäre.

Haben Sie im Parameter *attributePath* einen Pfad zu einem Objekt Attribut mit Werten unterschiedlichen Typs übergeben, gibt die Methode **entitySelection.min( )** den Mindestwert im ersten skalaren Elementtyp in der 4D Typenliste zurück, (siehe unter **collection.sort( )**). Existiert *attributePath* nicht im Objekt, gibt **entitySelection.min( ) null** zurück.

entitySelection.min( ) gibt auch **null** zurück, wenn die Entity-Selection leer ist.

Ein Fehler wird zurückgegeben, wenn

- *attributePath* ein verknüpftes Attribut ist
- *attributePath* in der Dataclass der Entity-Selection nicht gefunden wird.

Beispiel

Unter allen weiblichen Angestellten das niedrigste Gehalt herausfinden:

```
C_OBJECT($sel)
C_REAL($minSalary)
$sel:=ds.Employee.query("gender = :1";"female")
$minSalary:=$sel.min("salary")
```

entitySelection.minus( )

entitySelection.minus (entity | entitySelection) -> Funktionsergebnis

Parameter	Typ		Beschreibung
entity | entitySelection	Entity, EntitySelection	➡	Entity oder Entity-Selection zum Entnehmen
Funktionsergebnis	EntitySelection	↩	Neue Entity-Selection oder neue Referenz auf die vorhandene Entity-Selection

Beschreibung

Die Methode **entitySelection.minus( )** schließt die *entity* oder Entities von *entitySelection* aus der Entity-Selection aus, in der sie ausgeführt wird und gibt die resultierende Entity-Selection zurück.

- Übergeben Sie *entity* als Parameter, erstellt die Methode eine neue Entity-Selection *ohne entity* (wenn *entity* zu dieser Entity-Selection gehört). Ist *entity* nicht in der ursprünglichen Entity-Selection enthalten, wird eine neue Referenz auf die Entity-Selection zurückgegeben.
- Übergeben Sie *entitySelection* als Parameter, gibt die Methode eine Entity-Selection mit den Entities der ursprünglichen Entity-Selection ohne die in *entitySelection* enthaltenen Entities zurück.

Hinweis: Sie können sortierte bzw. unsortierte Entity-Selections vergleichen. Die resultierende Selection ist immer unsortiert. Weitere Informationen dazu finden Sie im Abschnitt **Sortierte vs unsortierte Entity-Selections** des *4D Developer Guide*.

Ist die ursprüngliche Entity-Selection oder sind die ursprüngliche Entity-Selection und der Parameter *entitySelection* leer, wird eine leere Entity-Selection zurückgegeben.
Ist *entitySelection* leer oder ist *entity* Null, wird eine neue Referenz der ursprünglichen Entity-Selection zurückgegeben.

Sind die ursprüngliche Entity-Selection und die Parameter nicht mit der gleichen Dataclass verbunden, wird ein Fehler generiert.

Beispiel 1

```
C_OBJECT($employees;$employee;$result)

$employees:=ds.Employee.query("lastName = :1";"H@")  // Die Entity-Selection $employees enthält die Entity mit Primärschlüssel 710 und anderen Entities
  // z.B. "Colin Hetrick", "Grady Harness", "Sherlock Holmes" (Primärschlüssel 710)

$employee:=ds.Employee.get(710)  // Gibt "Sherlock Holmes" zurück

$result:=$employees.minus($employee)  //$result enthält "Colin Hetrick", "Grady Harness"
```

Beispiel 2

Eine Selection der weiblichen Angestellen mit Namen "Jones" und Wohnsitz New York erhalten:

```
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 (entity | entitySelection) -> Funktionsergebnis

Parameter	Typ		Beschreibung
entity | entitySelection	Entity, EntitySelection	➡	Vereinigungsmenge aus Entity und Entity-Selection
Funktionsergebnis	EntitySelection	⮌	Neue Entity-Selection oder neue Referenz zur ursprünglichen Entity-Selection

Beschreibung

Die Methode **entitySelection.or( )** kombiniert die Entity-Selection mit dem Parameter *entity* oder *entitySelection* über den logischen (nicht ausschließlichen) Operator OR; Sie gibt eine neue unsortierte Entity-Selection zurück mit allen Entities aus der Entity-Selection und dem Parameter.

- Mit dem Parameter *entity* vergleichen Sie diese Entity mit der Entity-Selection. Gehört die Entity zur Entity-Selection, wird eine neue Referenz zur Entity-Selection zurückgegeben. Sonst wird eine neue Entity-Selection mit der ursprünglichen Entity-Selection und der Entity zurückgegeben.
- Mit dem Parameter *entitySelection* vergleichen Sie Entity-Selections. Eine neue Entity-Selection mit den Entities aus der ursprünglichen Entity-Selection oder *entitySelection* wird zurückgegeben. OR ist nicht ausschließlich, d.h. heißt Entities, auf die in beiden Selections verwiesen wird, werden nicht doppelt angezeigt.

Hinweis: Sie können sortierte bzw. unsortierte Entity-Selections vergleichen. Die resultierende Selection ist immer unsortiert. Weitere Informationen dazu finden Sie im Abschnitt **Sortierte vs unsortierte Entity-Selections** des *4D Developer Guide*.

Sind die ursprüngliche Entity-Selection und der Parameter *entitySelection* leer, wird eine leere Entity-Selection zurückgegeben.
Ist die ursprüngliche Entity-Selection leer, wird eine Referenz auf *entitySelection* oder eine Entity-Selection nur mit *entity* zurückgegeben.
Ist *entitySelection* leer oder ist *entity* Null, wird eine neue Referenz auf die ursprüngliche Entity-Selection zurückgegeben.

Sind die ursprüngliche Entity-Selection und der Parameter nicht mit derselben Dataclass verknüpft, wird ein Fehler generiert.

Beispiel 1

```
C_OBJECT($employees1;$employees2;$result)
$employees1:=ds.Employee.query("lastName = :1";"H@")  //Gibt "Colin Hetrick","Grady Harness" zurück
$employees2:=ds.Employee.query("firstName = :1";"C@")  //Gibt "Colin Hetrick", "Cath Kidston" zurück
$result:=$employees1.or($employees2)  //$result enthält "Colin Hetrick", "Grady Harness","Cath Kidston"
```

Beispiel 2

```
C_OBJECT($employees;$employee;$result)
$employees:=ds.Employee.query("lastName = :1";"H@")  // Gibt "Colin Hetrick","Grady Harness", "Sherlock Holmes" zurück
$employee:=ds.Employee.get(686)  //Die Entity mit Primärschlüssel 686 gehört nicht zur Entity-Selection $employees
 //Sie passt zur Angestellten "Mary Smith"

$result:=$employees.or($employee)  //$result enthält "Colin Hetrick", "Grady Harness", "Sherlock Holmes", "Mary Smith"
```

entitySelection.orderBy( )

entitySelection.orderBy (criteria) -> Funktionsergebnis

Parameter	Typ		Beschreibung
criteria	Text, Collection	➡	Text: Attributpfad(e) und Reihenfolge(n), nach denen die Entity-Selection sortiert werden soll Collection: Collection der Objekte für Kriterien
Funktionsergebnis	EntitySelection	⮌	Neue Entity-Selection in der angegebenen Reihenfolge

Beschreibung

Die Methode **entitySelection.orderBy( )** gibt eine neue sortierte Entity-Selection mit allen Entities der Entity-Selection in der in *criteria* angegebenen Reihenfolge zurück .

Hinweise:

- Diese Methode ändert nicht die ursprüngliche Entity-Selection.
- Weitere Informationen dazu finden Sie im Abschnitt **Sortierte vs unsortierte Entity-Selections**.

Im Parameter *criteria* definieren Sie, wie die Entities sortiert werden sollen. Es gibt 2 Möglichkeiten:

- *criteria* ist vom **Typ Text** (Formel): In diesem Fall enthält *criteria* eine Formel aus 1 bis x Attributspfaden und (optional) Sortierreihenfolgen, getrennt durch Kommas. Die Syntax für die Formel lautet:

  ```
  "attributePath1 {desc oder asc}, attributePath2 {desc oder asc},..."
  ```

  Die Reihenfolge, in der die Attribute übergeben sind, bestimmt die Sortierpriorität der Entities. Standardmäßig werden Attribute in aufsteigender Reihenfolge sortiert. Sie können die Sortierreihenfolge einer Eigenschaft im *criteria* String setzen, vom Eigenschaftspfad getrennt durch ein Leerzeichen: Übergeben Sie "asc" zum Sortieren in aufsteigender Reihenfolge, "desc" für absteigende Reihenfolge.

- *criteria* ist vom **Typ Collection**: In diesem Fall enthält jedes Element der Collection ein Objekt mit folgender Struktur:
  ```
  {
     "propertyPath": string,
     "descending": boolean
  }
  ```
  Attribute werden standardmäßig in aufsteigender Reihenfolge sortiert ("absteigend" ist falsch).
  In der Collection *criteria* können Sie soviel Objekte, wie benötigt hinzufügen.

Hinweis: Nullwerte werden geringer als andere Werte gewertet.

Beispiel

```
// Sortierung nach Formel
$sortedEntitySelection:=$entitySelection.orderBy("firstName asc, salary desc")
$sortedEntitySelection:=$entitySelection.orderBy("firstName")

// Sortierung nach Collection mit oder ohne Reihenfolge
$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 (queryString {; value}{; value2 ; ... ; valueN}{; querySettings}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
queryString	Text	➡	Suchkriterium
value		➡	Werte zum Vergleichen bei Verwenden von Platzhalter(n)
querySettings	Objekt	➡	Suchoptionen
Funktionsergebnis	EntitySelection	↩	Neue Entity-Selection mit den Entities, für die das in queryString angegebene Suchkriterium passt

Beschreibung

Die Methode **entitySelection.query( )** sucht in allen Entities der Dataclass oder EntitySelection nach Entities, für die das in *queryString* angegebene Suchkriterium und (optional) *value* passen, und gibt ein neues Objekt vom Typ *EntitySelection* mit den gefundenen Entities zurück. *Lazy loading* wird angewendet.

Hinweis: Diese Methode ändert nicht die ursprüngliche Entity-Selection.

Werden keine passenden Entities gefunden, wird eine leere *EntitySelection* zurückgegeben.

Weitere Informationen zum Erstellen eine Suche mit den Parametern *queryString*, *value* und *querySettings* finden Sie unter der Methode **dataClass.query( )**.

Hinweis: Die zurückgegebene Entity-Selection ist nicht sortiert (weitere Informationen dazu finden Sie unter **Sortierte vs unsortierte Entity-Selections**). Im Client/Server-Betrieb verhält sich sich jedoch wie eine sortierte Entity-Selection, d.h. Entities werden am Ende der Selection hinzugefügt.

Beispiel 1

```
C_OBJECT($entitySelectionTemp)
$entitySelectionTemp:=dataClass.query("lastName = :1";"M@")
Form.emps:=$entitySelectionTemp.query("manager.lastName = :1";"S@")
```

Beispiel 2

Weitere Beispiele für Suchläufe finden Sie auf der Seite **dataClass.query( )**.

entitySelection.slice( )

entitySelection.slice (startFrom {; end}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
startFrom	Lange Ganzzahl	➡	Index, wo die Suche startet (einschließlich)
end	Lange Ganzzahl	➡	Indexende (ausschließlich)
Funktionsergebnis	EntitySelection	⮌	Neue Entity-Selection mit den geteilten Entities (flache Kopie)

Beschreibung

Die Methode **entitySelection.slice( )** gibt einen Teil einer Entity-Selection in einer neuen Entity-Selection zurück, ausgewählt vom Index *startFrom* bis *end* (*end* ist nicht enthalten). Diese Methode gibt eine flache Kopie der Entity-Selection zurück, d.h sie verwendet die gleichen Entity Referenzen.

Hinweis: Diese Methode ändert nicht die ursprüngliche Entity-Selection.

Die zurückgegebene Entity-Selection enthält die Entities ab *startFrom* und alle nachfolgenden Entities bis zur Entity in *end*. Diese ist jedoch nicht mehr enthalten. Ist nur der Parameter *startFrom* angegeben, enthält die zurückgegebene Entity-Selection alle Entities ab *startFrom* bis zur letzten Entity der ursprünglichen Entity-Selection.

- Ist *startFrom* < 0, wird es neu berechnet als *startFrom:=startFrom+length* (wird als Versatz vom Ende der Entity-Selection gewertet). Ist der berechnete Wert < 0, wird *startFrom* auf 0 gesetzt.
- Ist *startFrom* >= *length*, gibt die Methode eine leere Entity-Selection zurück.
- Ist *end* < 0, wird es neu berechnet als *end:=end+length*.
- Ist *end* < *startFrom* (übergebene oder berechnete Werte), führt die Methode nichts aus.

Enthält die Entity-Selection mittlerweile gelöschte Entities, sind diese auch im Ergebnis enthalten.

Beispiel 1

Eine Selection der ersten 10 Entities einer Entity-Selection erhalten:

```
C_OBJECT($sel;$sliced)
$sel:=ds.Employee.query("salary > :1";50000)
$sliced:=$sel.slice(0;9)
```

Beispiel 2

Vorgegeben ist ds.Employee.all().length = 10

```
C_OBJECT($slice)
$slice:=ds.Employee.all().slice(-1;-2)  //versucht, Entites von Index 9 zu 8 zurückzugeben, da 9 > 8 ist, gibt sie eine leere Entity-Selection zurück
```

entitySelection.sum( )

entitySelection.sum (attributePath) -> Funktionsergebnis

Parameter	Typ		Beschreibung
attributePath	Text	→	Pfad des Attributs zur Berechnung
Funktionsergebnis	Zahl	⮌	Summe der Werte in der Entity-Selection

Beschreibung

Die Methode **entitySelection.sum( )** gibt die Summe aller Werte *attributePath* in der Entity-Selection zurück.

Ist die Entity-Selection leer, gibt **entitySelection.sum( )** 0 zurück.

Die Summe lässt sich nur für Werte vom Typ Zahl ausführen. Ist *attributePath* vom Typ Objekt, werden zur Berechnung nur numerische Werte berücksichtigt (andere Typen werden ignoriert). Führt *attributePath* zu einer Eigenschaft, die im Objekt nicht existiert oder keine numerischen Werte enthält, gibt **entitySelection.sum( )** 0 zurück.

Es wird ein Fehler zurückgegeben, wenn:

- *attributePath* kein Attribut vom Typ numerisch oder Objekt ist,
- *attributePath* ein verknüpftes Attribut ist
- *attributePath* in der Dataclass der Entity-Selection nicht gefunden wird.

Beispiel

```
C_OBJECT($sel)
C_REAL($sum)

$sel:=ds.Employee.query("salary < :1";20000)
$sum:=$sel.sum("salary")
```

entitySelection.toCollection( )

entitySelection.toCollection ({filter ;}{ options {; begin {; howMany}}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
filter	String, Collection	➡	Gibt die Entity Eigenschaften zum Extrahieren an
options	Lange Ganzzahl	➡	dk with primary key: Fügt den Primärschlüssel hinzu dk with stamp: Fügt den Stempel hinzu
begin	Lange Ganzzahl	➡	Definiert den startenden Index
howMany	Lange Ganzzahl	➡	Anzahl der Entities zum Extrahieren
Funktionsergebnis	Collection	↩	Collection der Objekte mit Attributen und Werten der Entity Collection

Beschreibung

Die Methode **entitySelection.toCollection( )** erstellt und gibt eine Collection zurück, wo jedes Element ein Objekt mit einem Satz Eigenschaften und Werten ist, die den Attributsnamen und Werten für die Entity Collection entsprechen.

Wird der Parameter *filter* weggelassen, enthält einen leeren String oder "*", werden alle Attribute extrahiert. Attribute mit der Eigenschaft "kind" "relatedEntity" werden mit einfacher Form extrahiert: ein Objekt mit Eigenschaft __KEY (Primärschlüssel). Attribute mit Eigenschaft "kind" "relatedEntities" werden nicht extrahiert.

Im Parameter *filter* können Sie die Entity Attribute zum Extrahieren übergeben. Es gibt zwei Syntaxarten:

- Ein String mit Eigenschaftspfaden, getrennt durch Kommas: "propertyPath1, propertyPath2, ...".
- Eine Collection mit Strings: ["propertyPath1","propertyPath2";...]

Ist *filter* für Attribute vom Typ **relatedEntity** angegeben:

- propertyPath = "relatedEntity" -> wird mit einfacher Form extrahiert
- propertyPath = "relatedEntity.*" -> alle Eigenschaften werden extrahiert
- propertyPath = "relatedEntity.propertyName1; relatedEntity.propertyName2; ..." -> nur diese Eigenschaften werden extrahiert

Ist *filter* für Attribute vom Typ **relatedEntities** angegeben:

- propertyPath = "relatedEntities.*" -> alle Eigenschaften werden extrahiert
- propertyPath = "relatedEntities.propertyName1; relatedEntities.propertyName2; ..." -> nur diese Eigenschaften werden extrahiert

Im Parameter *options* können Sie den bzw. die Selektoren dk with primary key und/oder dk with stamp übergeben, um Primärschlüssel der Entity bzw. Stempel in extrahierten Objekten hinzuzufügen.

Mit dem Parameter *begin* können Sie den startenden Index der Entities zum Extrahieren angeben. Sie können jeden Wert zwischen 0 und Entity-Selection-1 übergeben.

Im Parameter *howMany* können Sie die Anzahl der zu extrahierenden Entities ab der in *begin* angegebenen Entity definieren. Entfernte Entities werden nicht zurückgegeben, jedoch gemäß dem Parameter *howMany* mitgezählt. Ist *howMany*= 3 und gibt es 1 entfernte Entity, werden nur 2 Entities extrahiert.

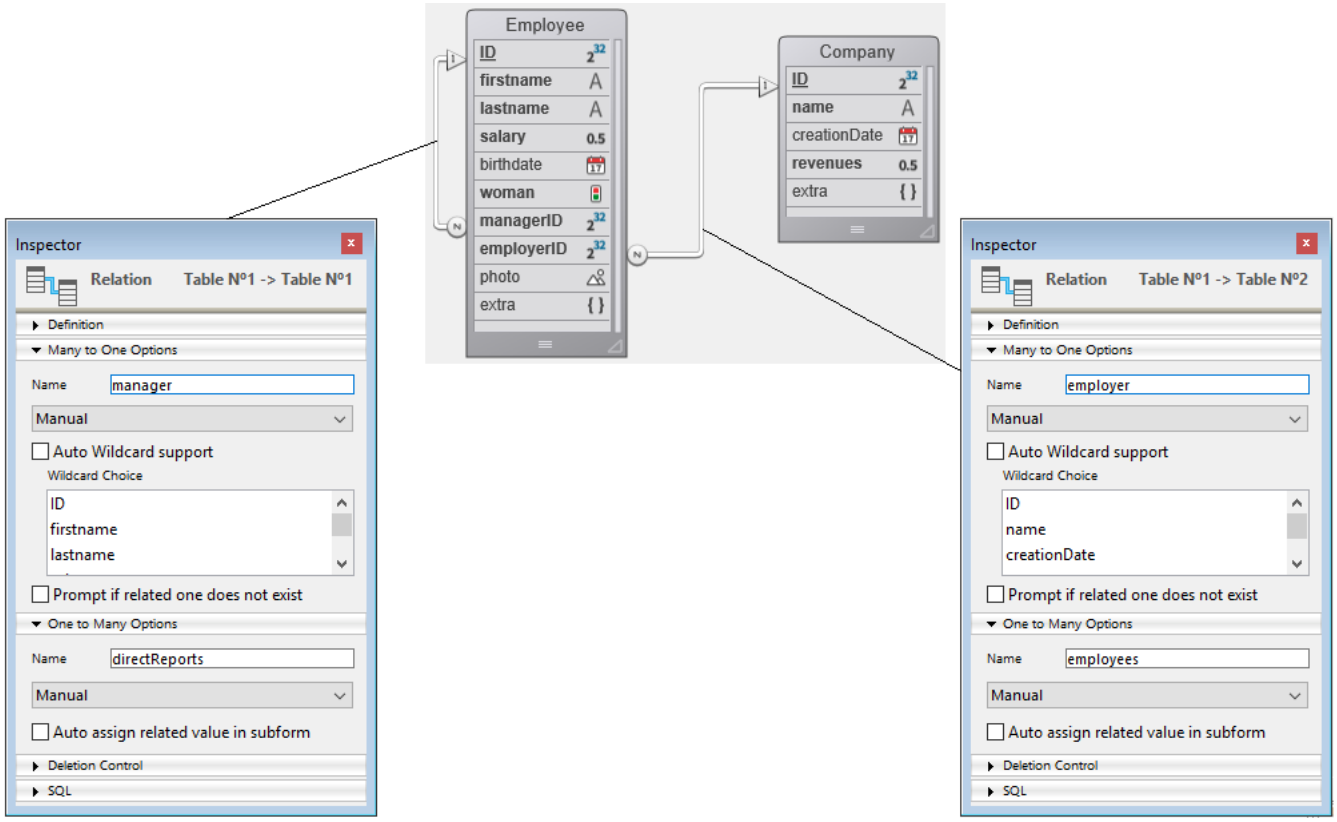
Ist *howMany* >Länge der Entity-Selection, gibt die Methode (Länge - *begin*) Objekte zurück.

Eine leere Collection wird zurückgegeben:

- wenn die Entity-Selection leer ist oder
- *begin* größer als die Länge der Entity-Selection ist

Beispiel 1

Folgende Struktur wird für alle Beispiele dieses Abschnitts verwendet:



Beispiel ohne die Parameter *filter* oder *options*:

```
C_COLLECTION($employeesCollection)
C_OBJECT($employees)

$employeesCollection:=New collection
$employees:=ds.Employee.all()
$employeesCollection:=$employees.toCollection()
```

Ergibt:


```
[ { "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 } } ]
```


Beispiel 2

Beispiel mit dem Parameter *options*:

```
$employeesCollection:=New collection
$employees:=ds.Employee.all()
$employeesCollection:=$employees.toCollection("";dk with primary key+dk with stamp)
```

Ergibt:


```
[ { "__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 } }]
```


Beispiel 3

Beispiel mit Aufteilen und Filtern auf Eigenschaften:

```
$employeesCollection:=New collection
$filter:=New collection
$filter.push("firstName")
$filter.push("lastName")
```

```
$employees:=ds.Employee.all()
$employeesCollection:=$employees.toCollection($filter;0;0;2)
```

Ergibt:

```
[ { "firstName": "Gregg", "lastName": "Wahl" }, { "firstName": "Irma", "lastName": "Durham" } ]
```

Beispiel 4

Beispiel mit dem Typ **relatedEntity** in einfacher Form:

```
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName,lastName,employer")
```

Ergibt:

```
[ { "firstName": "Gregg", "lastName": "Wahl", "employer": { "__KEY": 20 } }, { "firstName": "Irma", "lastName": "Durham", "employer": { "__KEY": 20 } }, { "firstName": "Lorena", "lastName": "Boothe", "employer": { "__KEY": 20 } } ]
```

Beispiel 5

Beispiel mit dem Parameter *filter* als Collection:

```
[#code4D]$employeesCollection:=New collection
$coll:=New collection("firstName";"lastName")
$employeesCollection:=$employees.toCollection($coll)#/code4D]
```

Ergibt:

```
[ { "firstName": "Joanna", "lastName": "Cabrera" }, { "firstName": "Alexandra", "lastName": "Coleman" } ]
```

Beispiel 6

Beispiel mit Extraktion aller Eigenschaften einer *relatedEntity*:

```
$employeesCollection:=New collection
$coll:=New collection
$coll.push("firstName")
$coll.push("lastName")
$coll.push("employer.*")
$employeesCollection:=$employees.toCollection($coll)
```

Ergibt:

```
[ { "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 } } ]
```

Beispiel 7

Beispiel mit Extraktion einiger Eigenschaften einer **relatedEntity**:

```
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName, lastName, employer.name")
```

Ergibt:

```
[ { "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" } }]
```

Beispiel 8

Beispiel mit Extraktion einiger Eigenschaften von **relatedEntities**:

```
$employeesCollection:=New collection()
$employeesCollection:=$employees.toCollection("firstName, lastName, directReports.firstName")
```

Ergibt:

```
[ { "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" }] }]

Beispiel 9

Beispiel mit Extraktion aller Eigenschaften von **relatedEntities**:

```
$employeesCollection:=New collection
$employeesCollection:=$employees.toCollection("firstName, lastName, directReports.*")
```

Ergibt

[{ "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)

Parameter	Typ		Beschreibung
entitySelection	EntitySelection	➡	Eine Entity-Selection

Beschreibung

Der Befehl **USE ENTITY SELECTION** aktualisiert die aktuelle Auswahl der Tabelle, die zur Dataclass des Parameters *entitySelection* passt, gemäß dem Inhalt der Entity-Selection.

Dieser Befehl funktioniert nur mit dem Datastore auf lokaler oder Client/Server Ebene, der vom Befehl **ds** zurückgegeben wird.

Hinweis: Nach Aufrufen von **USE ENTITY SELECTION** wird der erste Datensatz der aktualisierten aktuellen Auswahl (falls nicht leer) zum aktuellen Datensatz, er wird aber nicht in den Speicher geladen. Benötigen Sie die Werte der Felder im aktuellen Datensatz, verwenden Sie nach **USE ENTITY SELECTION** den Befehl **LOAD RECORD**.

Beispiel

```
C_OBJECT($entitySel)
$entitySel:=ds.Employee.query("lastName = :1";"M@")  //$entitySel ist mit der Dataclass Employee verbunden
REDUCE SELECTION([Employee];0)
USE ENTITY SELECTION($entitySel)  //Die aktuelle Auswahl der Tabelle Employee wird aktualisiert
```

Pasteboard

- Pasteboards verwalten
- 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

Pasteboards verwalten

Die Befehle in diesem Kapitel können sowohl die Aktionen Kopieren/Einsetzen (Zwischenablage) als auch Drag&Drop verwalten.
Dafür verwendet 4D zwei Übertragungsbereiche für die Daten: Einen für kopierte oder ausgeschnittene Daten, dies ist die Zwischenablage, die schon in früheren Versionen von 4D vorhanden war; den anderen für Daten, die per Drag&Drop-Technik bewegt werden, diese neue Ablage hat den Namen „Pasteboard".
Beide Übertragungsbereiche verwenden dieselben Befehle. Je nach Kontext greifen Sie auf die Zwischenablage oder Pasteboard zu:

- Der Übertragungsbereich für Drag&Drop (Pasteboard) ist nur innerhalb der Formularereignisse On Begin Drag Over, On Drag Over oder On Drop und in der **Datenbankmethode On Drop** verfügbar.
  In anderen Kontexten ist das Pasteboard für Drag&Drop nicht verfügbar.
- In allen anderen Fällen steht der Übertragungsbereich für Kopieren/Einsetzen zur Verfügung. Dieser behält die zu übertragenden Daten - im Gegensatz zum Pasteboard - während der gesamten Sitzung, solange sie nicht gelöscht oder erneut verwendet werden.

Datentypen

Während Drag&Drop Aktionen lassen sich verschiedene Datentypen in das Pasteboard setzen bzw. daraus entnehmen. Es gibt verschiedene Wege, um auf einen Datentyp zuzugreifen:

- Über seine **4D Signatur**: Die 4D Signatur ist eine Zeichenkette, die den Datentyp angibt, der 4D als Referenz dient. Die Verwendung von 4D Signaturen vereinfacht das Entwickeln von multi-plattform Anwendungen, da diese Signaturen auf Mac OS und Windows identisch sind.
  Die Liste der 4D Signaturen folgt im nächsten Abschnitt.
- Über ein **UTI** (Uniform Type Identifer, nur Mac OS): Der UTI Standard, von Apple definiert, weist jeder Art von nativen Objekten eine Zeichenkette zu. So haben z.B. GIF Bilder den UTI Typ "com.apple.gif." Die UTI Typen werden in der Dokumentation von Apple und in den davon betroffenen Editoren veröffentlicht
- Über seine *Nummer* oder seinen *Formatnamen* (nur Windows): Unter Windows hat jeder native Datentyp als Referenz eine Nummer ("3", "12", etc.) und einen Namen ("Rich Text Edit"). Microsoft spezifiziert standardmäßig verschiedene native Typen, das sind die Standarddateiformate. Darüberhinaus können Editoren von Drittherstellern Formatnamen im System sichern, die dann im Gegenzug eine Nummer erhalten. Weitere Informationen dazu finden Sie in der Dokumentation von Microsoft für Entwickler, insbesondere unter http://msdn2.microsoft.com/en-us/library/ms649013.aspx.

Hinweis: 4D Befehle behandeln die Nummern für Windows Formate als Text

Alle Befehle in diesem Kapitel können mit den oben beschriebenen Datentypen arbeiten. Über den Befehl **GET PASTEBOARD DATA TYPE** können Sie für jedes Format herausfinden, welcher Datentyp im Pasteboard vorhanden ist.

Hinweis: Die vierstellige Typbezeichnung (TEXT, PICT oder eigene Typen) wird zur Wahrung der Kompatibilität mit früheren Versionen von 4D beibehalten.

4D Signaturen

Hier ist die Liste der Standard 4D Signaturen und ihre Beschreibung:

Signatur	Beschreibung
"com.4d.private.text.native"	Text in native Zeichensatz
"com.4d.private.text.utf16"	Text in Unicode Zeichensatz
"com.4d.private.text.rtf"	Rich Text
"com.4d.private.picture.pict"	Bild in PICT Format
"com.4d.private.picture.png"	Bild in PNG Format
"com.4d.private.picture.gif"	Bild in GIF Format
"com.4d.private.picture.jfif"	Bild in JPEG Format
"com.4d.private.picture.emf"	Bild in EMF Format
"com.4d.private.picture.bitmap"	Bild in BITMAP Format
"com.4d.private.picture.tiff"	Bild in TIFF Format
"com.4d.private.picture.pdf"	PDF Dokument
"com.4d.private.file.url"	Pfadname der Datei

APPEND DATA TO PASTEBOARD

APPEND DATA TO PASTEBOARD (DatenTyp ; Daten)

Parameter	Typ		Beschreibung
DatenTyp	String	➡	Typ der hinzuzufügenden Daten
Daten	BLOB	➡	Daten, die im Pasteboard angefügt werden sollen

Beschreibung

Der Befehl **APPEND DATA TO PASTEBOARD** fügt im Pasteboard die im BLOB *Daten* enthaltenen Daten mit dem in *DatenTyp* angegebenem Typ an.

Hinweis: Bei Kopieren/Einsetzen Operationen entspricht Pasteboard der Zwischenablage.

In *DatenTyp* können Sie eine 4D Signatur, ein UTI (Mac OS), Formatnamen oder -nummer (Windows) oder einen Typ aus vier Zeichen (Kompatibilität) angeben. Weitere Informationen dazu finden Sie im Abschnitt **Pasteboards verwalten**.

Mit dem Befehl **APPEND DATA TO PASTEBOARD** hängen Sie normalerweise die verschiedenen Instanzen derselben Daten im Pasteboard an oder Daten, die weder vom Typ TEXT noch PICT sind. Wollen Sie neue Daten im Pasteboard anhängen, müssen Sie den Inhalt des Pasteboard zuerst mit dem Befehl **CLEAR PASTEBOARD** löschen.

Enthält ein BLOB jedoch derzeit Text oder ein Bild, können Sie mit dem Befehl **APPEND DATA TO PASTEBOARD** Text oder Bilder im Pasteboard anfügen.

Wollen Sie:

- Text im Pasteboard löschen und anfügen, verwenden Sie den Befehl **SET TEXT TO PASTEBOARD**
- Ein Bild im Pasteboard löschen und anfügen, verwenden Sie den Befehl **SET PICTURE TO PASTEBOARD**
- Den Pfadnamen (Drag and Drop) löschen oder anfügen, verwenden Sie den Befehl **SET FILE TO PASTEBOARD**

Beispiel

Mit den Befehlen für Pasteboards und BLOBs können Sie ausgeklügelte Schemata für Ausschneiden/Kopieren/Einsetzen erstellen, die auch mit strukturierten Daten und nicht nur mit einem einzigen Datenteil arbeiten können. Im nachfolgenden Beispiel können Sie mit Hilfe der Projektmethoden ***SET RECORD TO PASTEBOARD*** und ***GET RECORD FROM PASTEBOARD*** einen ganzen Datensatz als ein Teil behandeln, das in bzw. aus dem Pasteboard kopiert wird.

```
  ` Projektmethode SET RECORD TO PASTEBOARD
  ` SET RECORD TO PASTEBOARD ( Nummer )
  ` SET RECORD TO PASTEBOARD ( Tabellennummer )

C_LONGINT($1;$vlField;$vlFieldType)
C_POINTER($vpTable;$vpField)
C_STRING(255;$vsDocName)
C_TEXT($vtRecordData;$vtFieldData)
C_BLOB($vxRecordData)

  ` Inhalt des Pasteboard löschen (Gibt es keinen aktuellen Datensatz, bleibt sie leer)
CLEAR PASTEBOARD
  ` Hole Zeiger auf die Tabelle mit der als Parameter übergebenen Nummer
$vpTable:=Table($1)
  ` Gibt es einen aktuellen Datensatz für diese Tabelle
If((Record number($vpTable->)>=0)|(Is new record($vpTable->)))
  ` Initialisiere die Textvariable mit dem Textbild des Datensatzes
   $vtRecordData:=""
  ` Hole für jedes Feld des Datensatzes:
   For($vlField;1;Count fields($1))
  ` Feldtyp
      GET FIELD PROPERTIES($1;$vlField;$vlFieldType)
  ` Hole Zeiger auf das Feld
      $vpField:=Field($1;$vlField)
  ` Kopiere bzw. kopiere nicht je nach Feldtyp die Daten in geeigneter Weise
      Case of
         :(($vlFieldType=Is alpha field)|($vlFieldType=Is text))
            $vtFieldData:=$vpField->
         :(($vlFieldType=Is real)|($vlFieldType=Is integer)|
            ($vlFieldType=Is longint)|($vlFieldType=Is date)|
            ($vlFieldType=Is time))
            $vtFieldData:=String($vpField->)
         :($vlFieldType=Is Boolean)
            $vtFieldData:=String(Num($vpField->);"Yes;;No")
         Else
  ` Überspringe und ignoriere andere Felddatentypen
```

```
            $vtFieldData:=""
        End case
    ` Sammle die Felddaten in der Textvariablen mit dem Textbild des Datensatzes
        $vtRecordData:=$vtRecordData+Fieldname($1;$vlField)+":"+Char(9)+$vtFieldData+CR
    ` Beachte: Die Methode CR gibt unter Windows Char(13)+Char(10), auf Macintosh Char(13) zurück.
    End for
    ` Setze Textbild des Datensatzes in das Pasteboard
    SET TEXT TO PASTEBOARD($vtRecordData)
    ` Name für Übertragungsdatei in temporärem Ordner
    $vsDocName:=Temporary folder+"Scrap"+String(1+(Random%99))
    ` Lösche Übertragungsdatei, falls vorhanden (hier sollte Fehler getestet werden)
    DELETE DOCUMENT($vsDocName)
    ` Erstelle Teildatei
    SET CHANNEL(10;$vsDocName)
    ` Sende den ganzen Datensatz in die Übertragungsdatei
    SEND RECORD($vpTable->)
    ` Schließe die Übertragungsdatei
    SET CHANNEL(11)
    ` Lade Übertragungsdatei in ein BLOB
    DOCUMENT TO BLOB($vsDocName;$vxRecordData)
    ` Übertragungsdatei wird nicht länger benötigt
    DELETE DOCUMENT($vsDocName)
    ` Füge Gesamtbild des Datensatzes in das Pasteboard ein
    ` Hinweis: Wir verwenden wilkürlich "4Drc" als Datentyp
    APPEND DATA TO PASTEBOARD("4Drc";$vxRecordData)
    ` Das Pasteboard enthält nun:
    ` (1) Ein Textbild des Datensatzes (wie in u.a. Screenshots)
    ` (2) Ein Gesamtbild des Datensatzes (Bild, Unterdatei und darin enthaltene BLOB Felder)
End if
```

Geben Sie folgenden Datensatz ein:

und wenden die Methode ***SET RECORD TO PASTEBOARD*** auf die Tabelle [Angestellte] an, enthält die Zwischenablage das Textbild sowie das Gesamtbild des Datensatzes:

Über die Methode ***GET RECORD FROM PASTEBOARD*** können Sie dieses Bild des Datensatzes in einen anderen Datensatz kopieren:

```
<gen9>  ` Methode GET RECORD FROM PASTEBOARD
  ` GET RECORD FROM PASTEBOARD ( Nummer )
  ` GET RECORD FROM PASTEBOARD ( Tabellennummer )
C_LONGINT($1;$vlField;$vlFieldType;$vlPosCR;$vlPosColon)
C_POINTER($vpTable;$vpField)
C_STRING(255;$vsDocName)
C_BLOB($vxPasteboardData)
C_TEXT($vtPasteboardData;$vtFieldData)

  ` Hole Zeiger auf die Tabelle mit der in Parameter übergebenen Nummer
$vpTable:=Table($1)
  `Gibt es einen aktuellen Datensatz
If((Record number($vpTable->)>=0)|(Is new record($vpTable->)))
    Case of
  `Enthält das Pasteboard einen Datensatz als Gesamtbild?
       :(Pasteboard data size("4Drc")>0)
  ` Wenn ja, entnimm Inhalt dem Pasteboard
          GET PASTEBOARD DATA("4Drc";$vxPasteboardData)
  ` Name für Übertragungsdatei in temporärem Ordner
          $vsDocName:=Temporary folder+"Scrap"+String(1+(Random%99))
  ` Lösche Übertragungsdatei, falls vorhanden (hier sollte Fehler getestet werden)
          DELETE DOCUMENT($vsDocName)
  ` Sichere BLOB in der Übertragungsdatei
          BLOB TO DOCUMENT($vsDocName;$vxPasteboardData)
  ` Öffne Übertragungsdatei
          SET CHANNEL(10;$vsDocName)
  ` Erhalte den gesamten Datensatz aus der Übertragungsdatei
          RECEIVE RECORD($vpTable->)
  ` Schließe Übertragungsdatei
          SET CHANNEL(11)
  ` Übertragungsdatei wird nicht länger benötigt
          DELETE DOCUMENT($vsDocName)
  ` Enthält das Pasteboard TEXT?
       :(Pasteboard data size("TEXT")>0)
  ` Entnimm Text aus dem Pasteboard
          $vtPasteboardData:=Get text from pasteboard
  ` Initialisiere Feldnummer für die Erhöhung
          $vlField:=0
          Repeat
  ` Suche nach der nächsten Feldzeile im Text
             $vlPosCR:=Position(CR;$vtPasteboardData)
             If($vlPosCR>0)
  ` Entnimm die Feldzeile
                $vtFieldData:=Substring($vtPasteboardData;1;$vlPosCR-1)
  ` Bei einem Doppelpunkt ":"
                $vlPosColon:=Position(":";$vtFieldData)
                If($vlPosColon>0)
  ` Nimm nur die Felddaten (lösche den Feldnamen)
                   $vtFieldData:=Substring($vtFieldData;$vlPosColon+2)
                End if
  ` Erhöhe die Feldnummer
                $vlField:=$vlField+1
  ` Pasteboard enthält evtl. mehr Daten als benötigt
                If($vlField<=Count fields($vpTable))
  ` Hole Typ des Feldes
                   GET FIELD PROPERTIES($1;$vlField;$vlFieldType)
  ` Hole Zeiger auf das Feld
                   $vpField:=Field($1;$vlField)
  ` Je nach Feldtyp kopiere bzw. kopiere nicht Text in geeigneter Art
                   Case of
                      :(($vlFieldType=Is alpha field)|($vlFieldType=Is text))
                         $vpField->:=$vtFieldData
                      :(($vlFieldType=Is real)|($vlFieldType=Is integer)|
                         ($vlFieldType=Is longint))
                         $vpField->:=Num($vtFieldData)
                      :($vlFieldType=Is date)
                         $vpField->:=Date($vtFieldData)
                      :($vlFieldType=Is time)
                         $vpField->:=Time($vtFieldData)
                      :($vlFieldType=Is Boolean)
```

```
                    $vpField->:=($vtFieldData="Yes")
                End
` Überspringe und ignoriere andere Felddatentypen
                Else
` Alle Felder wurden zugewiesen, verlasse die Schleife
                    $vtPasteboardData:=""
                End if
` Entferne den soeben entnommenen Text
                $vtPasteboardData:=Substring($vtPasteboardData;$vlPosCR+Length(CR))
            Else
` Kein Begrenzer gefunden, verlasse die Schleife
                $vtPasteboardData:=""
            End if
` Wiederhole, solange Daten vorhanden
        Until(Length($vtPasteboardData)=0)
    Else
        ALERT("Pasteboard enthält keine Daten für die Übertragung als Datensatz.")
End case
End if</gen9>
```

Systemvariablen und Mengen

Wird BLOB Daten korrekt im Pasteboard angefügt, hat die Systemvariable OK den Wert 1; andernfalls hat OK den Wert 0 und es wird u.U. eine Fehlermeldung erzeugt.

CLEAR PASTEBOARD

CLEAR PASTEBOARD
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **CLEAR PASTEBOARD** löscht den Inhalt des Pasteboard. Enthält das Pasteboard mehrere Instanzen derselben Daten, werden alle Instanzen gelöscht. Das Pasteboard ist nach dem Aufrufen von **CLEAR PASTEBOARD** leer.

Sie müssen **CLEAR PASTEBOARD** einmal aufrufen, bevor Sie mit dem Befehl **APPEND DATA TO PASTEBOARD** neue Daten in das Pasteboard anfügen, da dieser vor dem Anfügen neuer Daten nicht den Inhalt der Zwischenablage löscht.

Rufen Sie einmal **CLEAR PASTEBOARD** auf und dann mehrere Male **APPEND DATA TO PASTEBOARD**, können Sie dieselben Daten unter verschiedenen Formaten ausschneiden bzw. kopieren.

Dagegen löschen die Befehle **SET TEXT TO PASTEBOARD** und **SET PICTURE TO PASTEBOARD** automatisch den Inhalt des Pasteboard, bevor Daten vom Typ TEXT oder PICT angefügt werden.

Beispiel 1

Folgender Code leert die Zwischenablage und hängt dann die Daten an:

```
CLEAR PASTEBOARD  ` Stellen Sie sicher, dass die Zwischenablage leer ist
APPEND DATA TO PASTEBOARD("com.4d.private.picture.gif;$vxSomeData)  ` Hänge einige gif-Bilder an
APPEND DATA TO PASTEBOARD("com.4d.private.text.rft";$vxSylkData)  ` Hänge RTF Text an
```

Beispiel 2

Siehe Beispiel zum Befehl **APPEND DATA TO PASTEBOARD**.

Get file from pasteboard

Get file from pasteboard (xIndex) -> Funktionsergebnis

Parameter	Typ		Beschreibung
xIndex	Lange Ganzzahl	➡	xte Datei innerhalb einer Drag-Aktion
Funktionsergebnis	String	⮌	Pfadname der aus dem Pasteboard entnommenen Datei

Beschreibung

Die Funktion **Get file from pasteboard** gibt den absoluten Pfadnamen der Datei innerhalb einer Drag&Drop Operation zurück. Der Parameter *xIndex* bezeichnet eine Datei aus dem Satz der ausgewählten Dateien.

Gibt es keine xte Datei im Pasteboard, gibt die Funktion einen leeren String zurück.

Beispiel

Das folgende Beispiel setzt in ein Array alle Pfadnamen der Dateien innerhalb einer Drag&Drop Operation:

```
ARRAY TEXT($filesArray;0)
C_TEXT($vfileArray)
C_INTEGER($n)
$n:=1
Repeat
    $vfileArray:=Get file from pasteboard($n)
    If($vfileArray#"")
        APPEND TO ARRAY($filesArray;$vfileArray)
        $n:=$n+1
    End if
Until($vfileArray="")
```

GET PASTEBOARD DATA

GET PASTEBOARD DATA (DatenTyp ; Daten)

Parameter	Typ		Beschreibung
DatenTyp	String	➡	Angeforderter Datentyp aus dem Pasteboard
Daten	BLOB	⬅	Angeforderte Daten aus dem Pasteboard

Beschreibung

Der Befehl **GET PASTEBOARD DATA** gibt im BLOB Feld oder in der Variablen *Daten* die im Pasteboard vorhandenen Daten zurück mit dem in *DatenTyp* übergebenen Typ. (Enthält das Pasteboard innerhalb 4D kopierten Text, wird im BLOB in der Regel der Zeichensatz UTF-16 verwendet.)

Hinweis: Im Rahmen von Copy/Paste Operationen entspricht Pasteboard der Zwischenablage.

In *DatenTyp* können Sie eine 4D Signatur, ein UTI (Mac OS), Formatnamen oder -nummer (Windows) oder einen Typ aus vier Zeichen (Kompatibilität) angeben. Weitere Informationen dazu finden Sie im Abschnitt **Pasteboards verwalten**.

Hinweis: Mit diesem Befehl können Sie keine Datendateien einlesen. Dafür benötigen Sie den Befehl **Get file from pasteboard**.

Beispiel

Folgende Objektmethoden für zwei Schaltflächen kopieren Daten und übertragen sie in das Array *asOptions* (PopUp-Menü, DropDown-Liste,...) in einem Formular:

```
  ` Objektmethode bCopyasOptions
If(Size of array(asOptions)>0)  ` Gibt es etwas zu kopieren?
   VARIABLE TO BLOB(asOptions;$vxClipData)  ` Sammle die Elemente des Array in einem BLOB
   CLEAR PASTEBOARD  ` Leere das Pasteboard
   APPEND DATA TO PASTEBOARD("artx";asOptions)  ` Die Datentypen wurden wilkürlich gewählt
End if

  ` Objektmethode bPasteasOptions
If(Pasteboard data size("artx")>0)  ` Gibt es Daten vom Typ "artx" im Pasteboard?
   GET PASTEBOARD DATA("artx";$vxClipData)  ` Entnimm Daten aus dem Pasteboard
   BLOB TO VARIABLE($vxClipData;asOptions)  ` Fülle Array mit BLOB Daten
   asOptions:=0  ` Setze ausgewähltes Element für das Array neu
End if
```

Systemvariablen und Mengen

Wurden die Daten korrekt entnommen, hat OK den Wert 1; sonst den Wert 0 und es wird ein Fehler erzeugt.

GET PASTEBOARD DATA TYPE

GET PASTEBOARD DATA TYPE (4DSignaturen ; NativeTypen {; FormatNamen})

Parameter	Typ		Beschreibung
4DSignaturen	Array Text	⇐	4D Signaturen der Datentypen
NativeTypen	Array Text	⇐	Native Datentypen
FormatNamen	Array Text	⇐	Formatnamen (nur Windows), auf Mac OS leere Strings

Beschreibung

Der Befehl **GET PASTEBOARD DATA TYPE** erhält die Liste der Datentypen, die im Pasteboard vorhanden sind. Dieser Befehl sollte generell im Rahmen der Formularereignisse On Drop oder On Drag Over des Zielobjekts verwendet werden. Er ermöglicht insbesondere, zu prüfen, ob im Pasteboard ein bestimmter Datentyp vorhanden ist.

Dieser Befehl gibt die Datentypen in verschiedenen Formularen über zwei oder drei Arrays zurück:

- Das Array *4DSignaturen* enthält die Datentypen, die über die interne 4D Signatur ausgedrückt werden, z.B. "com.4d.private.picture.gif".
  Erkennt 4D einen gefundenen Datentyp nicht, wird im Array ein leerer String ("") zurückgegeben.
- Das Array *NativeTypen* enthält die Datentypen, die über ihre native Typen ausgedrückt werden. Das Format ist zwischen Mac OS und Windows unterschiedlich:
  - Auf Mac OS werden native Typen als UTIs (Uniform Type Identifier) ausgedrückt.
  - Unter Windows werden native Typen als Zahl ausgedrückt, wobei jede Nummer einem Formatnamen zugeordnet ist. Das Array *NativeTypen* enthält diese Nummern in Form von Strings ("3", "12", usw.). Wollen Sie klarere Bezeichnungen verwenden, empfehlen wir, das optionale Array *FormatNamen* zu verwenden (siehe unten), da es die Formatnamen der native Typen unter Windows enthält.

  Das Array *NativeTypen* erlaubt jede Art von Datentyp, der im Pasteboard gefunden wird, inkl. Daten, deren Typ 4D nicht als Referenz enthält.

- Unter Windows können Sie auch das Array *FormatNamen* übergeben. Es empfängt die Namen der Datentypen, die im Pasteboard gefunden werden. Mit den hier zurückgegebenen Werten können Sie z.B. ein PopUp-Menü mit verschiedenen Formaten einrichten.
  Auf Mac OS gibt das Array *FormatNamen* leere Strings zurück.

Weitere Informationen dazu finden Sie im Abschnitt **Pasteboards verwalten**.

GET PICTURE FROM PASTEBOARD

GET PICTURE FROM PASTEBOARD (Bild)

Parameter	Typ		Beschreibung
Bild	Bild	⇐	Bild aus dem Pasteboard

Beschreibung

Der Befehl **GET PICTURE FROM PASTEBOARD** gibt das Bild aus dem Pasteboard im Bilddatenfeld oder der Variablen *Bild* zurück.
Hinweis: Bei Kopieren/Einsetzen Operationen entspricht Pasteboard der Zwischenablage.

Beispiel

Nachfolgende Objektmethode für die Schaltfläche weist das Bild im jpeg oder gif Format im Pasteboard (sofern vorhanden) dem Datenfeld [Employees]Photo zu:

```
If((Pasteboard data size("com.4d.private.picture.jpeg")>0)|(Pasteboard data size("com.4d.private.picture.gif")>0))
    GET PICTURE FROM PASTEBOARD([Employees]Photo)
Else
    ALERT("Das Pasteboard enthält keine Bilder.")
End if
```

Systemvariablen und Mengen

Wurde das Bild korrekt entnommen, hat OK den Wert 1; sonst den Wert 0.

Get text from pasteboard

Get text from pasteboard -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	String	Gibt Text aus dem Pasteboard zurück, sofern vorhanden

Beschreibung

Die Funktion **Get text from pasteboard** gibt den Text aus dem Pasteboard zurück.

Hinweis: Bei Operationen Kopieren/Einsetzen entspricht Pasteboard der Zwischenablage.

Enthält das Pasteboard Rich-Text, z.B. im RTF-Format, behält der Text seine Attribute beim Übertragen oder Kopieren, wenn der Zielbereich dazu kompatibel ist.

Textfelder und Variablen von 4D können bis zu 2 GB an Text enthalten.

Systemvariablen und Mengen

Wurde der Text korrekt entnommen, hat OK den Wert 1; sonst den Wert 0.

Pasteboard data size

Pasteboard data size (DatenTyp) -> Funktionsergebnis

Parameter	Typ		Beschreibung
DatenTyp	String	➡	String mit 4 Zeichen
Funktionsergebnis	Lange Ganzzahl	⮌	Größe (in Bytes) der im Pasteboard gespeicherten Daten oder Fehlercode

Beschreibung

Die Funktion **Pasteboard data size** prüft, ob es im Pasteboard Daten mit dem in *DatenTyp* übergebenen Typ gibt.

Warnung: Der in *DatenTyp* übergebene Wert berücksichtigt die Schreibweise. So ist z.B. "abcd" nicht gleich "ABCD."

Ist das Pasteboard leer oder enthält keine Daten vom angegebenen Typ, gibt die Funktion einen Fehler -102 zurück. Enthält das Pasteboard Daten vom angegebenen Typ, gibt die Funktion die Größe der Daten in Bytes zurück.

In *DatenTyp* können Sie eine 4D Signatur, ein UTI (Mac OS), Formatnamen oder -nummer (Windows) oder einen Typ aus vier Zeichen (Kompatibilität) angeben. Weitere Informationen dazu finden Sie im Abschnitt **Pasteboards verwalten**.

Haben Sie festgestellt, dass das Pasteboard Daten des entsprechenden Typs enthält, können Sie diese mit folgenden Befehlen aus dem Pasteboard entnehmen:

- Enthält das Pasteboard Daten vom Typ TEXT, erhalten Sie diese entweder mit der Funktion **Get text from pasteboard**, die einen Textwert zurückgibt, oder mit dem Befehl **GET PASTEBOARD DATA**, der den Text in einem BLOB zurückgibt.
- Enthält das Pasteboard Daten vom Typ Bild, erhalten Sie diese entweder mit dem Befehl **GET PICTURE FROM PASTEBOARD**, der das Bild in einem Bilddatenfeld oder einer Variablen zurückgibt, oder mit dem Befehl **GET PASTEBOARD DATA**, der das Bild in einem BLOB zurückgibt.
- Enthält das Pasteboard den Pfadnamen einer Datei, erhalten Sie diesen über die Funktion **Get file from pasteboard**, die den Pfadnamen der Datei zurückgibt.
- Verwenden Sie für alle anderen Datentypen den Befehl **GET PASTEBOARD DATA**, der die Daten in einem BLOB zurückgibt.

Beispiel 1

Nachfolgender Code prüft, ob die Zwischenablage ein Bild enthält. Wenn ja, wird es in eine 4D Variable kopiert:

```
If(Pasteboard data size(Picture data)=1)  //Gibt es ein Bild im Pasteboard?
   GET PICTURE FROM PASTEBOARD($vPicVariable)  //Wenn ja, entnimm das Bild aus dem Pasteboard
Else
   ALERT("Es gibt kein Bild im Pasteboard.")
End if
```

Beispiel 2

Anwendungen schneiden normalerweise Daten vom Typ TEXT bzw. Bild aus bzw. kopieren sie in das Pasteboard, da die meisten Anwendungen diese beiden Standarddatentypen erkennen. Eine Anwendung kann jedoch mehrere Instanzen derselben Daten in verschiedenen Formaten anfügen. Immer, wenn Sie z.B. Teile einer Tabellenkalkulation ausschneiden bzw. kopieren, könnte die Anwendung die Daten unter dem hypothetischen Format 'SPSH' anfügen, ebensogut aber auch in den Formaten SYLK und TEXT. Dann würde 'SPSH' die Daten gemäß der Datenstruktur enthalten, SYLK dieselben Daten im SYLK Format, das von den meisten anderen Tabellenkalkulationen erkannt wird und TEXT dieselben Daten ohne die Zusatzinformationen in SYLK bzw. 'SPSH'. Vorausgesetzt, Sie kennen die Beschreibung des 'SPSH' Formats und haben alles für die Übertragung von Daten im SYLK Format vorbereitet, können Sie nun Routinen für Ausschneiden/Kopieren/Einsetzen zwischen 4D und jener hypothetischen Tabellenkalkulation erstellen. Ihr Code könnte folgendermaßen aussehen:

```
Case of
   ` Prüfen Sie zuerst, ob das Pasteboard Daten aus der hypothetischen Tabellenkalkulation enthält
   :(Test clipboard('SPSH')>0)
   ` ...
   ` Prüfen Sie dann, ob das Pasteboard Daten vom Typ Sylk enthält
   :(Test clipboard('SYLK')>0)
   ` ...
   ` Prüfen Sie schließlich, ob das Pasteboard Daten vom Typ Text enthält
   :(Test clipboard('TEXT')>0)
   ` ...
End case
```

Anders gesagt, Sie versuchen aus dem Pasteboard die Instanz der Daten zu entnehmen, die am meisten Originalinformationen enthält.

Beispiel 3

Sie wollen eigene Daten aus verschiedenen Objekten in Ihr Formular ziehen. Sie können schreiben:

```
 //Quellobjekt
If(Form event=On Begin Drag Over)
   APPEND DATA TO PASTEBOARD("some.private.data";$data)
End if
```

```
 //Zielobjekt
If(Form event=On Drag Over)
   $0:=Choose(Pasteboard data size("some.private.data")>0;0;-1)
End if
```

Beispiel 4

Siehe Beispiel zum Befehl **APPEND DATA TO PASTEBOARD**.

SET FILE TO PASTEBOARD

SET FILE TO PASTEBOARD (Datei {; *})

Parameter	Typ		Beschreibung
Datei	String	➡	Dateiname oder vollständiger Pfadname der Datei
*	Operator	➡	Mit *: Hinzufügen; ohne *: Ersetzen

Beschreibung

Der Befehl **SET FILE TO PASTEBOARD** fügt im Pasteboard den vollständigen Pfadnamen hinzu, der im Parameter *Datei* übergeben wurde. Mit diesem Befehl können Sie Oberflächen einrichten, die z.B. Drag-and-Drop von 4D Objekten in Dateien auf dem Desktop erlauben.

In *Datei* können Sie entweder einen kompletten Pfadnamen oder nur einen Dateinamen übergeben. In diesem Fall muss die Datei neben der Strukturdatei der Datenbank liegen.

Der Befehl akzeptiert den Stern * als optionalen Parameter. Standardmäßig, d.h. ohne Stern, ersetzt er den Inhalt des Pasteboard durch den letzten in *Datei* angegebenen Pfadnamen. Mit Stern fügt der Befehl die Datei im Pasteboard an. Auf diese Weise kann er einen Stapel von Datei-Pfadnamen enthalten. In beiden Fällen werden im Pasteboard vorhandene Daten, die keine Pfadnamen sind, entfernt.

Hinweis: Das Pasteboard ist während dem Formularereignis On Drag Over im Nur-Lesen Modus. Deshalb lässt sich dieser Befehl nicht in diesem Kontext verwenden.

SET PICTURE TO PASTEBOARD

SET PICTURE TO PASTEBOARD (Bild)

Parameter	Typ		Beschreibung
Bild	Bild	➡	Bild, dessen Kopie in das Pasteboard gelegt werden soll

Beschreibung

Der Befehl **SET PICTURE TO PASTEBOARD** leert das Pasteboard und legt eine Kopie des in *Bild* übergebenen Bildes hinein.

Hinweis: Bei Kopieren/Einsetzen Operationen entspricht Pasteboard der Zwischenablage.

Haben Sie ein Bild in das Pasteboard gelegt, finden Sie es wieder mit dem Befehl **GET PICTURE FROM PASTEBOARD** oder z.B. mit der Anweisung **GET PASTEBOARD DATA** ("com.4d.private.picture.gif";...).

Beispiel

Über ein Palettenfenster wird ein Formular angezeigt mit dem Array *asEmployeeName*, das die Namen der Angestellten der Tabelle [Employees] anzeigt. Bei jedem Anklicken des Namens soll das Bild des Angestellten in die Zwischenablage kopiert werden. Die Objektmethode für das Array lautet:

```
If(asEmployeeName#0)
   QUERY([Employees];[Employees]Last name=asEmployeeName{asEmployeeName})
   If(Picture size([Employees]Foto)>0)
      SET PICTURE TO PASTEBOARD([Employees]Foto)  ` Kopiere Foto des Angestellten
   Else
      CLEAR PASTEBOARD  ` Es wurde weder Bild noch Datensatz gefunden
   End if
End if
```

Systemvariablen und Mengen

Wurde das Bild korrekt in die Zwischenablage kopiert, hat die OK Variable den Wert 1.
Reicht der Speicher zum Kopieren des Bildes nicht aus, setzt der Befehl die Variable OK auf 0. Es wird jedoch kein Fehler generiert.

SET TEXT TO PASTEBOARD

SET TEXT TO PASTEBOARD (Text)

Parameter	Typ		Beschreibung
Text	String	➡	Text, dessen Kopie in das Pasteboard kopiert werden soll.

Beschreibung

Der Befehl **SET TEXT TO PASTEBOARD** leert das Pasteboard und legt dann eine Kopie des in *Text* übergebenen Textes in das Pasteboard.

Hinweis: Bei Kopieren/Einsetzen Operationen ist Pasteboard identisch mit der Zwischenablage.

Einen in die Zwischenablage gelegten Text finden Sie wieder mit der Funktion **Get text from pasteboard** oder z.B. mit der Anweisung **GET PASTEBOARD DATA** ("com.4d.private.text.native";...).

Textausdrücke in 4D können bis zu 2 GB groß sein.

Hinweis: Das Pasteboard ist während dem Formularereignis On Drag Over im Nur-Lesen Modus. Von daher ist es nicht möglich, den Befehl in diesem Kontext zu verwenden.

Beispiel

Siehe Beispiel zum Befehl **APPEND DATA TO PASTEBOARD**.

Systemvariablen und Mengen

Wurde der Text korrekt in die Zwischenablage gelegt, setzt der Befehl die OK Variable auf 1. Reicht der Speicher zum Kopieren des Textes nicht aus, setzt der Befehl die OK Variable auf 0. Es wird jedoch kein Fehler generiert.

PHP

PHP Skripte in 4D ausführen
PHP Execute
PHP GET FULL RESPONSE
PHP GET OPTION
PHP SET OPTION
Unterstützung von PHP Modulen

PHP Skripte in 4D ausführen

In 4D können Sie direkt PHP Skripte ausführen. So können Sie die Vorteile von "utility libraries" nutzen, die über PHP verfügbar sind. Diese libraries bieten im Einzelnen folgende Funktionalitäten:

- Ciphering (MD5) und Hashing
- ZIP Dateien verwalten
- Bilder verwalten
- LDAP Zugriff
- COM Zugriff (Steuern von MS Office Dokumenten), etc.

Diese Liste ist nur ein Auszug. Eine komplette Übersicht über die PHP Module, die standardmäßig mit 4D verfügbar sind, finden Sie im Abschnitt **Unterstützung von PHP Modulen**. Beachten Sie, dass Sie auch zusätzliche eigene Module installieren können.

Um ein PHP Skript bzw. eine Funktion auszuführen, müssen Sie den Befehl **PHP Execute** verwenden. 4D enthält standardmäßig Version 5.4.11 von PHP. Ausgeführte Skripte müssen mit dieser Version und den installierten Modulen kompatibel sein.

Eine ausführliche Beschreibung der PHP Befehle und Syntax finden Sie in der umfangreichen PHP Dokumentation, die Sie über Internet abrufen können. Hier ein paar Beispiele:
http://us.php.net/manual/en/
http://phpdeveloper.org/
http://php.start4all.com/
http://php.resourceindex.com/Documentation/

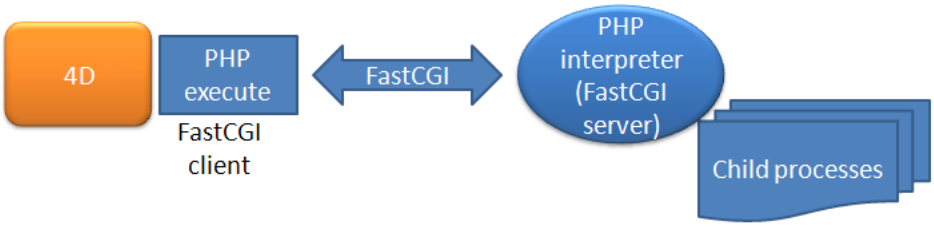
Architektur

4D bietet einen PHP Interpreter, kompiliert in FastCGI. Das ist ein Kommunikationsprotokoll vom Typ Client-Server zwischen einer Anwendung und einem PHP Interpreter.

Der PHP Interpreter verwaltet eine Reihe von Ausführungsprozessen im System, genannt "Kindprozesse". Diese Prozesse dienen zum Bearbeiten von Anfragen, die 4D sendet. Die Ausführung der Anfragen läuft synchron. Aus Optimierungsgründen können standardmäßig fünf Kindprozesse gleichzeitig laufen. Dieser Wert lässt sich über die Datenbank-Eigenschaften bzw. den Befehl **SET DATABASE PARAMETER** verändern. Auf Mac OS werden diese Prozesse bei der ersten Anfrage gestartet und vom PHP Interpreter dauerhaft beibehalten. Unter Windows erstellt 4D die Prozesse nach Bedarf und recycelt sie, falls notwendig. 4D unterstützt automatisch die Verwaltung von Prozessen, die der PHP Interpreter standardmäßig liefert, wie Starten und Schließen.

Hinweis: Endet das 4D Programm unerwartet, während noch PHP Kindprozesse aktiv sind, müssen Sie diese über das Systemfenster zur Prozessverwaltung von Hand löschen.

Nachfolgende Übersicht zeigt die 4D/PHP Struktur von 4D:



Diese Architektur arbeitet mit einem System interner Anfragen, die 4D an eine vordefinierte TCP Adresse sendet (IP Adresse und Port Nummer). Diese Adresse lässt sich bei Bedarf über die Datenbank-Eigenschaften oder über den Befehl **SET DATABASE PARAMETER** ändern, z.B. wenn auf demselben Rechner mehrere PHP Interpreter ausgeführt werden.

Warnung: Starten Sie zwei 4D Anwendungen auf dem gleichen Rechner und führen in beiden PHP Anweisungen aus (via **PHP Execute**), müssen Sie unbedingt die gelisteten Ports des Interpreters FastCGI PHP ändern, so dass Sie für jede Anwendung einen anderen benutzen. Andernfalls werden PHP Anweisungen u.U. mit Fehlern und nicht vorhersehbar ausgeführt und können sogar Ihre 4D Anwendung einfrieren.

Anderen PHP Interpreter und andere php.ini Datei verwenden

Sie können auch einen anderen PHP Interpreter verwenden als 4D zur Verfügung stellt. So können Sie bei einem Update von 4D denselben PHP Interpreter beibehalten. Außerdem können Sie alle gewünschten eigenen Module installieren - Mit dem in 4D enthaltenen Interpreter können Sie keine eigene PHP Datei verwenden. Sie benötigen einen externen Interpreter, wenn Sie andere PHP Konfigurationen als die standardmäßig mitgelieferten verwenden wollen. Ein eigener PHP Interpreter muss zwei Bedingungen erfüllen:

- Er muss in FastCGI kompiliert sein,
- Er muss auf demselben Rechner wie 4D liegen.

Wollen Sie einen eigenen PHP Interpreter verwenden, müssen Sie diesen lediglich so konfigurieren, dass er auf eine spezifische Adresse und TCP Port anspricht und den von 4D aufgerufenen internen Interpreter deaktivieren. Diese Parameter können Sie entweder über die Datenbankeinstellungen oder für die Sitzung über den Befehl **SET DATABASE PARAMETER** setzen. Dann müssen Sie natürlich das Starten und Arbeiten des Interpreters selbst verwalten.

Die Initialisationsdatei *php.ini* liegt im Ordner **Resources** der Datenbank. Über die php.ini Datei können Sie insbesondere den Speicherort der PHP Erweiterungen deklarieren.
Ist sie beim ersten Aufruf nicht vorhanden, erstellt 4D die Datei mit den passenden Konfigurationsoptionen.

Die *php.ini* Datei des externen Interpreters muss folgende Eingänge enthalten:

- **auto_prepend_file** liefert den vollständigen Pfadnamen zum Utility Skript *4D_Execute_PHP.php* . Dieses Skript liegt unter [4D application]Resources/php/Windows bzw. /Mac. Ohne diesen Eingang lassen sich nur vollständige Skripte ausführen: Aufrufe einer Routine innerhalb eines Skript funktionieren nicht.
- **display_errors** setzt ein "stderr" so dass 4D informiert werden kann, wenn während der Ausführung von PHP Code ein Fehler auftritt.
  Hier ein Beispiel:

```
; stderr - Melde die Fehler an STDERR (gilt nur für CGI/CLI)
; Um die Fehler für STDERR an CGI/CLI zu richten:
display_errors = "stderr"
```

Weitere Informationen zur Konfiguration eigener php.ini Dateien finden Sie in den 4D Kommentaren innerhalb der *php.ini* Datei.

PHP Execute

PHP Execute (SkriptPfad {; FunktionName {; * | phpErgebnis {; Param} {; Param2 ; ... ; ParamN}}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
SkriptPfad	Text	➡	Pfadname des PHP Skript oder "" zum Ausführen einer PHP Funktion
FunktionName	Text	➡	Auszuführende PHP Funktion
* | phpErgebnis	Operator, Variable, Feld	⬅	Ergebnis der Ausführung der PHP Funktion oder *, um kein Ergebnis zu erhalten.
Param	Text, Boolean, Zahl, Lange Ganzzahl, Datum, Zeit	➡	Parameter der PHP Funktion
Funktionsergebnis	Boolean	➲	Wahr = Ausführung korrekt, Falsch = Fehler bei der Ausführung

Beschreibung

Die Funktion **PHP Execute** ermöglicht, ein PHP Skript bzw. Funktion auszuführen.

Im Parameter *SkriptPfad* übergeben Sie den Pfadnamen des auszuführenden PHP Skript. Das kann ein relativer Pfad sein, wenn die Datei neben der Struktur der Datenbank liegt, oder ein absoluter Pfad. Der Pfadname lässt sich entweder in System Syntax oder in POSIX Syntax ausdrücken.

Um eine standardmäßige PHP Funktion direkt auszuführen, übergeben Sie einen leeren String ("") in *SkriptPfad*. Der Funktionsname muss im zweiten Parameter übergeben werden.

Übergeben Sie einen PHP Funktionsnamen in *FunktionName*, wenn Sie in *SkriptPfad* eine spezifische Funktion ausführen wollen. Übergeben Sie einen leeren String oder lassen den Parameter *FunktionName* weg, wird das Skript vollständig ausgeführt.

Hinweis: PHP berücksichtigt Klein- und Großschreibung für Funktionsnamen. Verwenden Sie keine Klammern, sondern geben nur den Funktionsnamen ein.

Der Parameter *phpErgebnis* empfängt das Ergebnis der Ausführung der PHP Funktion. Sie können folgendes übergeben:

- Variable, Array oder Feld zum Empfangen des Ergebnisses
- Das Zeichen *, wenn die Funktion kein Ergebnis zurückgibt oder Sie es nicht ausfindig machen wollen

Sie können eine Variable, ein Array oder ein Feld vom Typ Text, Lange Ganzzahl, Ganzzahl, Boolean oder Datum sowie (außer für Arrays) ein Feld vom Typ BLOB oder Zeit übergeben. 4D führt die Konvertierung der Daten und alle benötigten Anpassungen aus, wie im Absatz unten beschrieben.

- Haben Sie im Parameter *FunktionName* einen Funktionsnamen übergeben, empfängt *phpErgebnis* das, was der PHP Entwickler mit dem Befehl **return** im Hauptteil der Funktion zurückgegeben hat.
- Verwenden Sie den Befehl, ohne im Parameter *FunktionName* einen Funktionsnamen zu übergeben, empfängt *phpErgebnis* das, was der PHP Entwickler mit dem Befehl **echo** (oder einem ähnlichen Befehl) zurückgegeben hat.

Rufen Sie eine PHP Funktion auf, die Argumente erwartet, übergeben Sie einen oder mehrere Werte in *Param1...N*. Die Werte müssen durch Strichpunkte voneinander getrennt sein. Sie können vom Typ Alpha, Text, Boolean, Zahl, Ganzzahl, Lange Ganzzahl, Datum oder Zeit sein. Bilder, BLOBs und Objekte werden nicht akzeptiert. Sie können ein Array senden, dazu müssen Sie in der Funktion **PHP Execute** einen Zeiger auf das Array übergeben. Ansonsten wird der aktuelle Index des Array als Ganzzahl gesendet (siehe Beispiel). Der Befehl akzeptiert alle Arten von Arrays, außer für Zeiger, Bild und 2D Arrays.
Die Parameter *param1...N* werden in PHP im Format JSON in UTF-8 gesendet. Sie werden automatisch mit dem PHP Befehl **json_decode** entschlüsselt, bevor sie an die PHP Funktion *FunktionName* gesendet werden.

Hinweis: Aus technischen Gründen dürfen Parameter, die über das Protokoll Fast CGI übergeben werden, nicht größer als 64 KB sein. Sie müssen diese Beschränkung insbesondere bei Parametern vom Typ Text berücksichtigen.

Der Befehl gibt Wahr zurück, wenn die Ausführung auf 4D Seite korrekt ausgeführt wurde, d.h. wenn Starten der Ausführungsumgebung, Öffnen des Skripts und Herstellen der Kommunikation mit dem PHP Interpreter erfolgreich waren. Andernfalls wird ein Fehler erzeugt, den Sie mit **ON ERR CALL** abfangen und mit **GET LAST ERROR STACK** analysieren können. Darüberhinaus kann auch das Skript selbst PHP Fehler erzeugen. In diesem Fall müssen Sie **PHP GET FULL RESPONSE** zur Analyse der Fehlerquelle einsetzen (siehe Beispiel 4).

Hinweis: Über PHP lässt sich die Fehlerverwaltung konfigurieren. Weitere Informationen dazu finden Sie unter http://www.php.net/manual/en/errorfunc.configuration.php#ini.error-reporting.

Konvertierung der zurückgegebenen Daten

Nachfolgende Tabelle gibt an, wie 4D Daten gemäß dem Typ *phpErgebnis* interpretiert und konvertiert.

Typ des Parameters *phpErgenbis*	Bearbeitung durch 4D	Beispiel
BLOB	4D übernimmt die empfangenen Daten ohne Änderungen(*).	
Text	4D erwartet Daten codiert in UTF-8 (*). Der PHP Entwickler muss u.U. den PHP Befehl **utf8_encode** verwenden.	Beispiel für PHP Skript: `echo utf8_encode(myText)`
Datum	4D erwartet ein Datum, das als String im Format RFC 3339 übergeben wird (manchmal in PHP DATE_ATOM genannt). Dieses Format ist vom Typ "YYYY-MM-DDTHH:MM:SS", zum Beispiel: 2005-08-15T15:52:01+00:00. 4D ignoriert den Zeit-Teil und gibt das Datum in UTC zurück.	
Zeit	4D erwartet eine Zeit, gesendet als String im Format RFC 3339 (siehe Typ Datum). 4D ignoriert den Datumsteil und gibt die ab Mitternacht abgelaufenen Sekunden zurück, während das Datum in der lokalen Zeitzone bewertet wird.	Beispiel für PHP Skript zum Senden von 2h30'45": `echo date( DATE_ATOM, mktime( 2,30,45))`
Ganzzahl oder Zahl	4D interpretiert den numerischen Wert, ausgedrückt in Zahlen mit vorangestelltem + oder - Zeichen bzw. 'e' für Exponent. Ein '.' oder ',' Zeichen wird als Dezimaltrenner interpretiert.	Beispiel des PHP Skript: `echo -1.4e-16;`
Boolean	4D gibt Wahr zurück, wenn es den String "true" von PHP erhält oder wenn die numerische Bewertung einen Wert gibt, der nicht Null ist.	Beispiel für PHP Skript: `echo (a==b);`
Array	4D wertet, dass das PHP Array im JSON Format zurückgegeben wurde.	Beispiel für PHP Skript zum Zurückgeben eines Array mit zwei Texten: `echo json_encode( array( "hello", "world"));`

(*) Standardmäßig werden keine HTTP zurückgegeben:
- Verwenden Sie **PHP Execute** mit Übergeben einer Funktion im Parameter *FunktionName*, werden in *phpErgebnis* nie HTTP Header zurückgegeben. Sie sind nur über den Befehl **PHP GET FULL RESPONSE** verfügbar.
- Verwenden Sie **PHP Execute** ohne einen Funktionsnamen (der Parameter *FunktionName* wird weggelassen oder enthält einen leeren String), können Sie HTTP Header zurückgeben, wenn Sie die Option PHP Raw result auf Wahr setzen oder den Befehl **PHP SET OPTION** verwenden.

Hinweis: Müssen Sie über PHP große Datenmengen übernehmen, ist es in der Regel effizienter, über den Puffer *stdOut* zu gehen (Befehl **echo** oder ähnlich) anstatt über die Funktion **return**. Weitere Information dazu finden Sie in der Beschreibung zum Befehl **PHP GET FULL RESPONSE**.

Umgebungsvariablen verwenden

Mit dem Befehl **SET ENVIRONMENT VARIABLE** können Sie die vom Skript verwendeten Umgebungsvariablen definieren.
Warnung: Nach Aufrufen von **LAUNCH EXTERNAL PROCESS** oder **PHP Execute** wird der Satz Umgebungsvariablen wieder entfernt.

Spezialfunktionen

4D bietet folgende Spezialfunktionen:

- **quit_4d_php**: Um den PHP Interpreter und alle seine Kindprozesse zu verlassen. Führt mindestens ein Kindprozess ein Skript aus, beendet der Interpreter nicht und die Funktion **PHP Execute** gibt Falsch zurück.
- **relaunch_4d_php**: Um den PHP Interpreter wieder zu starten.

Beachten Sie, dass der Interpreter automatisch gestartet wird, wenn **PHP Execute** die erste Anfrage sendet.

Beispiel 1

Das Skript "myPhpFile.php" ohne Funktion aufrufen. Sein Inhalt lautet:

```
<?php
echo 'Current PHP version: ' . phpversion();
?>
```

Der folgende 4D Code:

```
C_TEXT($result)
C_BOOLEAN($isOK)
$isOK:=PHP Execute("C:\php\myPhpFile.php";"";$result)
ALERT($Result)
```

... zeigt an "Current PHP version: 5.3"

Beispiel 2

Die Funktion *myPhpFunction* im Skript "myNewScript.php" mit Parametern aufrufen. Der Inhalt des Skript lautet:

```
<?php
// . . . PHP code. . .
function myPhpFunction($p1, $p2) {
    return $p1 . ' '. $p2;
}
// . . . PHP code.. . .
?>
```

Aufruf mit Funktion:

```
C_TEXT($result)
C_TEXT($param1)
C_TEXT($param2)
C_BOOLEAN($isOk)
$param1 :="Hallo"
$param2 :="4D Welt!"
$isOk:=PHP Execute("C:\MyFolder\myNewScript.php";"myPhpFunction";$result;$param1;$param2)
ALERT($result)  // zeigt an "Hallo 4D Welt!"
```

Beispiel 3

PHP Interpreter beenden:

```
$ifOk:=PHP Execute("";"quit_4d_php")
```

Beispiel 4

Fehlerverwaltung:

```
 // Installation der Fehlerverwaltungsmethode
phpCommError:=""  // Modified by PHPErrorHandler
$T_saveErrorHandler :=Method called on error
ON ERR CALL("PHPErrorHandler")</p><p>  // Skriptausführung
C_TEXT($T_result)
If(PHP Execute("C:\MyScripts\MiscInfos.php";"";$T_result))
 // Kein Fehler, $T_Result enthält das Ergebnis
Else
 // Ein Fehler wird gefunden, verwaltet von PHPErrorHandler
   If(phpCommError="")
      ... // PHP Fehler, verwende PHP GET FULL RESPONSE
   Else
      ALERT(phpCommError)
   End if
End if

 // Methode deinstallieren
ON ERR CALL($T_saveErrorHandler)
```

Die Methode ***PHP_errHandler*** lautet:

```
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
```

Beispiel 5

Dynamische Skript-Erstellung von 4D vor seiner Ausführung:

```
DOCUMENT TO BLOB("C:\Scripts\MyScript.php";$blobDoc)
If(OK=1)
    $strDoc:=BLOB to text($blobDoc;UTF8 text without length)
    $strPosition:=Position("Funktion2Umbenennen";$strDoc)
    $strDoc:=Insert string($strDoc;"_v2";Length("Funktion2Umbenennen")+$strPosition)

    TEXT TO BLOB($strDoc;$blobDoc;UTF8 text without length)
    BLOB TO DOCUMENT("C:\Scripts\MyScript.php";$blobDoc)
    If(OK#1)
        ALERT("Fehler bei Skript-Erstellung")
    End if
End if
```

Dann wird das Skript ausgeführt:

```
$err:=PHP Execute("C:\Scripts\MyScript.php";"Funktion2Umbenennen_v2";*)
```

Beispiel 6

Wert vom Typ Datum und Zeit direkt ausfindig machen. Das Skript dafür lautet:

```
<?php
// . . . code php. . .
echo date(DATE_ATOM, mktime(1, 2, 3, 4, 5, 2009));
// . . . code php. . .
?>
```

Datum auf 4D Seite empfangen:

```
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?
```

Beispiel 7

Daten in Arrays verteilen:

```
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 enthält die alphanumerischen Werte "11", "22", "33", etc.
 // $arLong enthält die Zahlen 11, 22, 33, etc.
```

Beispiel 8

Initialisation eines Array:

```
ARRAY TEXT($arText ;0)
$phpok :=PHP Execute("";"array_pad";$arText;->$arText;50;"undefined")
 // Execute in PHP: $arrTest = array_pad($arrTest, 50, 'undefined');
 // Füllt das Array $arText mit 50 "undefinierten" Elementen
```

Beispiel 9

Parameter über ein Array übergeben:

```
ARRAY INTEGER($arInt;0)
$phpok :=PHP Execute("";"json_decode";$arInt;"[13,51,69,42,7]")
 // Execute in PHP: $arInt = json_decode('[13,51,69,42,7]');
 // Füllt das Array mit den Ausgangswerten
```

Beispiel 10

Einfaches Beispiel zur Trim Funktion, um zusätzliche Leerzeichen bzw. unsichtbare Zeichen von Anfang bis Ende eines String zu entfernen:

```
C_TEXT($T_String)
$T_String:="    Hello   "
C_BOOLEAN($B)
$B:=PHP Execute("";"trim";$T_String;$T_String)
```

Weitere Informationen zur Trim Funktion finden Sie in der PHP Dokumentation.

PHP GET FULL RESPONSE

PHP GET FULL RESPONSE (stdOut {; errBezeichnungen ; errWerte} {; httpHeaderFelder {; httpHeaderWerte}})

Parameter	Typ		Beschreibung
stdOut	Textvariable, BLOB Variable	⬅	Inhalte des Puffers stdOut
errBezeichnungen	Array Text	⬅	Bezeichnungen der Fehler
errWerte	Array Text	⬅	Werte der Fehler
httpHeaderFelder	Array Text	⬅	Namen der HTTP Header
httpHeaderWerte	Array Text	⬅	Werte der HTTP Header

Beschreibung

Der Befehl **PHP GET FULL RESPONSE** ermöglicht, zusätzliche Information über die vom PHP Interpreter zurückgegebene Antwort zu erhalten. Dieser Befehl ist besonders hilfreich, wenn während der Ausführung des Skript ein Fehler auftritt.

Das PHP Skript kann Daten in den *stdOut* Puffer (echo, print, etc.) schreiben. Der Befehl gibt die Daten direkt in der Variablen *stdOut* zurück und wendet dieselben Konvertierungsprinzipien an, wie im Befehl **PHP Execute** beschrieben.

Die aufeinander abgestimmten Text Arrays *errBezeichnungen* und *errWerte* werden gefüllt, wenn die Ausführung der PHP Skripte Fehler verursacht. Diese Arrays bieten insbesondere Information über den Ursprung des Fehlers, Skript und Zeile. Beide Arrays sind untrennbar, d.h. ist *errBezeichnungen* übergeben, muss auch *errWerte* übergeben werden. Da der Austausch zwischen 4D und dem PHP Interpreter über FastCGI erfolgt, funktioniert der PHP Interpreter, als ob er von einem HTTP Server aufgerufen wurde und sendet deshalb HTTP Header. Sie können diese Header und ihre Werte in den Arrays *httpHeaderFelder* und *httpHeaderWerte* wiederfinden.

PHP GET OPTION

PHP GET OPTION (Option ; Wert)

Parameter	Typ		Beschreibung
Option	Lange Ganzzahl	➡	Zu erhaltende Option
Wert	Text, Boolean	⬅	Aktueller Wert der Option

Beschreibung

Der Befehl **PHP GET OPTION** ermöglicht, den aktuellen Wert einer Option bei der Ausführung von PHP Skripten zu finden.

Im Parameter *Option* übergeben Sie eine Konstante unter dem Thema **PHP**, um die zu erhaltende Option zu bestimmen. Der Befehl gibt den aktuellen Wert der Option im Parameter *Wert* zurück. Sie können eine der folgenden Konstanten übergeben:

Konstante	Typ	Wert	Kommentar
PHP privileges	Lange Ganzzahl	1	Definition spezifischer Benutzerprivilegien zur Ausführung von Skripts. **Mögliche Werte**: String in Form von "User:Password". Zum Beispiel: "root:jd51254d"
PHP raw result	Lange Ganzzahl	2	Definition des Bearbeitungsmodus für HTTP Header, die von PHP im Ausführungsergebnis zurückgegeben werden, wenn das Ergebnis vom Typ Text ist (bei einem Ergebnis vom Typ BLOB werden Header immer beibehalten). **Mögliche Werte**: Boolean. Falsch (Standardwert = HTTP Header aus Ergebnis entfernen. Wahr = HTTP Header beibehalten.

Hinweis: Verwenden Sie die Konstante PHP Privileges mit diesem Befehl, wird nur das Benutzerkonto zurückgegeben.

Beispiel

Wir wollen das aktuelle Benutzerkonto finden:

```
C_TEXT($userAccount)
PHP GET OPTION(PHP privileges;$userAccount)
ALERT($userAccount)
```

PHP SET OPTION

PHP SET OPTION (Option ; Wert {; *})

Parameter	Typ		Beschreibung
Option	Lange Ganzzahl	➡	Zu setzende Option
Wert	Text, Boolean	➡	Neuer Wert der Option
*	Operator	➡	Mit *: Änderung gilt nur für den nächsten Aufruf

Beschreibung

Der Befehl **PHP SET OPTION** setzt spezifische Optionen vor Aufrufen der Funktion **PHP Execute**. Die Reichweite dieses Befehls ist der aktuelle Prozess.

Im Parameter *Option* übergeben Sie eine Konstante unter dem Thema **PHP**, um die zu ändernde Option zu bestimmen. Im Parameter *Wert* übergeben Sie den neuen Wert für die Option. Es gibt folgende Konstanten:

Konstante	Typ	Wert	Kommentar
PHP privileges	Lange Ganzzahl	1	Definition spezifischer Benutzerprivilegien zur Ausführung von Skripts. **Mögliche Werte**: String in Form von "User:Password". Zum Beispiel: "root:jd51254d"
PHP raw result	Lange Ganzzahl	2	Definition des Bearbeitungsmodus für HTTP Header, die von PHP im Ausführungsergebnis zurückgegeben werden, wenn das Ergebnis vom Typ Text ist (bei einem Ergebnis vom Typ BLOB werden Header immer beibehalten). **Mögliche Werte**: Boolean. Falsch (Standardwert = HTTP Header aus Ergebnis entfernen. Wahr = HTTP Header beibehalten.

PHP SET OPTION setzt standardmäßig die Option für alle nachfolgenden Aufrufe von **PHP Execute** des Prozesses. Wollen Sie das auf den nächsten Aufruf beschränken, übergeben Sie den Parameter Stern (*).

Beispiel

Führen Sie das Skript "myAdminScript.php" mit Admin Zugriffsrechten aus:

```
PHP SET OPTION(PHP privileges;"admin:mypwd";*)
  `Da wir * übergeben, werden Admin-Rechte nur einmal verwendet
C_TEXT($result)
C_BOOLEAN($isOK)
$isOK:=PHP Execute("myAdminScript.php";$result)
If($isOK)
    ALERT($result)
End if
```

Unterstützung von PHP Modulen

Dieser Anhang beschreibt, wie Sie PHP Module in 4D integrieren können. Folgende Themen werden behandelt:

- Liste der Standard PHP Module, die mit dem PHP Interpreter von 4D standardmäßig geliefert werden
- Liste der Standard PHP Module, die in 4D nicht enthalten sind
- Anweisungen zum Installieren zusätzlicher Module

Hinweis: Um zusätzliche Module zu installieren, müssen Sie einen externen FastCGI-php Interpreter verwenden (siehe **Anderen PHP Interpreter und andere php.ini Datei verwenden**).

Standardmäßig gelieferte Module

Nachfolgende Tabelle listet die PHP Module auf, die mit 4D standardmäßig geliefert werden.

Generische Module

Name	Web Site	Beschreibung

BCMath	http://php.net/bc	Binärer Kalkulator zum Verwalten von Nummern jeglicher Größe und Genauigkeit, dargestellt als Strings.
		Beispiel:
		`C_LONGINT($value;$result)`
		`$value:=4`
		`$ok:=PHP Execute("";"bcpow";$result;$value;3)`

Calendar	http://php.net/calendar	Satz Funktionen, um die Konvertierung verschiedener Kalenderformate zu vereinfachen. Basiert auf der Berechnung nach julianischem Datum.
		Beispiel:
		`C_LONGINT($NumberOfDays)`
		`$ok:=PHP Execute("";"cal_days_in_month";$NumberOfDays;1;2;2010)`

Ctype	http://php.net/ctype	Funktionen zum Prüfen, ob ein Zeichen oder eine Zeichenkette zu einer bestimmten Zeichenklasse gehört, abhängig von der aktuellen lokalen Konfiguration.
		Beispiel:
		`// Prüfen, ob alle Zeichen des gelieferten String Satzzeichen sind`
		`C_TEXT($myString)`
		`$myString:=",.;/"`
		`$ok:=PHP Execute("";"ctype_punct";$isPunct;$myString)`

Date and Time	http://php.net/datetime	Datum und Zeit von dem Server zurückbekommen, wo das PHP Skript ausgeführt wurde
		Beispiel:
		`//Berechnung der Zeit des Sonnenaufgangs in Lissabon, Portugal`
		`//Breitengrad: 38.4 Nord`
		`//Längengrad: 9 West`
		`//Zenit ~= 90`
		`//Zeitverschiebung: +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	XML Dokumente via DOM API in PHP 5 verwenden.
Exif	http://php.net/exif	Mit Bild Metadaten arbeiten.
Fileinfo(*)	http://php.net/fileinfo	Typ des Inhalts und Codierung einer Datei herausfinden.

Filter	http://php.net/filter	Daten von einer nicht-sicheren Quelle bestätigen und filtern, z.B. Benutzereingaben.
		Beispiel:
		`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	Detaillierter Zugriff auf einen FTP Server

Hash	http://php.net/hash	Message Digest engine. Ermöglicht direktes oder inkrementelles Bearbeiten von Meldungen mit wilkürlicher Länge über eine Reihe von Hash Algorithmen.
		Beispiel:
		`C_TEXT($md5Result)`
		`$ok:=PHP Execute("";"md5";$md5Result;"this is my string to hash")`

GD (Graphics Draw) Library	http://php.net/gd	Mit Bildern arbeiten
Iconv	http://php.net/iconv	Konvertierung von Dateien zwischen verschiedenen Zeichensätzen
JSON (JavaScript Object Notation)	http://php.net/json	Implementierung des JSON Datenaustauschformats
LDAP	http://php.net/ldap	LDAP ist ein Zugriffsprotokoll auf "folder servers", die Information in Form eines Baumdiagramm speichern.

LibXML	http://php.net/libxml	Library mit XML Funktionen und Konstanten
LibXSLT	http://php.net/xsl	Library mit Funktionen für XSL Transformation
Multibyte String	http://php.net/mbstring	Satz Funktionen zum Arbeiten mit Zeichenketten, zur Verwaltung der Codierung von multi-byte Zeichen oder zum Konvertieren von Zeichenketten.
OpenSSL	http://php.net/openssl	Verwendung der Funktionen OpenSSL zum Generieren und Verifizieren von Signaturen, zum Versiegeln (Verschlüsseln) und Öffnen (Entschlüsseln) von Daten.

PCRE (Perl Compatible Regular Expressions)	http://php.net/pcre	Satz Funktionen, die rationale Ausdrücke implementieren, welche dieselbe Syntax und semantische Perl 5 verwenden
		Beispiel:
		`// Dieses Beispiel entfernt unnötige Leerzeichen in einer Zeichenkette.`
		`C_TEXT($myString)`
		`$myString:="foo o bar"`
		`PHP Execute("";"preg_replace";$myString;"/\\s\\s+/";" ";$myString)`
		`ALERT($myString)`
		`// $myString ist nun "foo o bar" ohne wiederholte Leerzeichen`

PDO (PHP Data Objects) (*)	http://php.net/pdo	Schnittstelle für Zugriff auf eine Datenbank. Erfordert einen datenbankspezifischen PDO Treiber.
PDO_SQLITE (*)	http://php.net/pdo_sqlite	Treiber, der die PHP Data Objects (PDO) Schnittstelle implementiert, um PHP Zugriff auf SQLite 3 Datenbanken zu ermöglichen.
Reflection	http://php.net/reflection	Komplette Reflection API zur Untersuchung/Erkennung von Klassen, Interfaces, Funktionen und Methoden sowie Erweiterungen
Phar (PHP Archive)	http://php.net/phar	Einfügen einer kompletten PHP Anwendung in einer einmaligen Datei mit Namen "phar" (PHP Archiv), um ihre Installation und Konfiguration zu erleichtern

Session	http://php.net/session	Unterstützung der PHP Sitzungen
		Beispiel:
		In Web Anwendungen dienen Sitzungen dazu, den Kontext zwischen jeder Anfrage zu behalten. Rufen Sie **PHP Execute** in 4D auf, kann das PHP Skript eine Sitzung starten und alles, was als Kontext erhaltenswert ist, im Array $_SESSION speichern. Verwendet ein PHP Skript Sitzungen, müssen Sie die von PHP zurückgegebene Sitzungs ID über den Befehl **PHP GET FULL RESPONSE** erhalten und sie vor jedem Aufruf von **PHP Execute** über den Befehl **SET ENVIRONMENT VARIABLE** spezifizieren.
		`// "PHP Execute with context" method`
		`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	Sehr einfache und leicht-verwendbare Werkzeuge, um XML in ein Objekt zu konvertieren, das mit seinen Eigenschaften und Array Iteratoren bearbeitet werden kann.
Sockets	http://php.net/sockets	Implementierung einer low-level Schnittstelle zu Funktionen für Socket Kommunikation, basierend auf BSD Sockets und mit der Möglichkeit, sowohl als Socket-Server als auch als -Client zu operieren.
SPL (Standard PHP Library)	http://php.net/spl	Sammlung von Interfaces und Klassen, die zum Lösen von Standardproblemen dienen.
SQLite (*)	http://php.net/sqlite	Erweiterung für die SQLite Datenbank Engine. Diese Engine ist einbindbar.
SQLite3 (*)	http://php.net/sqlite3	Unterstützung für SQLite Version 3 Datenbanken.
Tokenizer	http://php.net/tokenizer	Funktionen, über die Sie Ihre eigenen PHP Analyse- oder Änderungswerkzeuge schreiben können, ohne sich mit Sprachspezifikationen auf lexikalischer Ebene befassen zu müssen.
XML (eXtensible Markup Language)	http://php.net/xml	Durchlaufen von XML Dokumenten.
XMLreader	http://php.net/xmlreader	XML Pull Parser
XMLwriter	http://php.net/xmlwriter	Generierung von Streams oder Dateien im XML Format
Zlib	http://php.net/zlib	Komprimierte gzip (.gz) Dateien lesen und schreiben **Beispiel:** `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	ZIP komprimierte Archive und darin enthaltene Dateien lesen und schreiben.

(*) Diese Module sind in der aktuellen 4D Version unter Windows nicht verfügbar.

Nur unter Windows verfügbare Module

Aus strukturellen Gründen sind folgende PHP Module nur auf der Windows Plattform verfügbar.

Name	Web Site	Beschreibung
COM & .NET	http://php.net/com	COM (Component Object Model) ist einer der Hauptwege für Anwendungen und Komponenten zur Kommunikation auf Windows Plattformen. Darüberhinaus unterstützt 4D die Instantiation und Erstellung von .Net Ansammlungen über den COM layer.
ODBC (Open DataBase Connectivity)	http://php.net/odbc	Zusätzlich zur standardmäßigen ODBC Unterstützung gewähren die Unified ODBC Funktionen in PHP Zugriff auf verschiedene Datenbanken, die die Semantik der ODBC API zum Implementieren ihrer eigenen API übernommen haben.
WDDX (Web Distributed Data eXchange)	http://php.net/wddx	Erleichtert den Datenaustausch zwischen Web Anwendungen über das Web, unabhängig von der Plattform.

Deaktivierte Module

Nachfolgende PHP Module wurden nicht integriert. Die dritte Spalte gibt den Grund dafür an:

Name	Web Site	Grund - Alternative
Mimetype	http://php.net/mime-magic	Veraltet (Deprecated) - Fileinfo verwenden
POSIX (Portable Operating System Interface)	http://php.net/posix	Veraltet (Deprecated)
Regular Expression (POSIX Extended)	http://php.net/regex	Veraltet (Deprecated) - PCRE verwenden
Crack	http://php.net/crack	Restriktive Lizenz
ffmpeg	http://ffmpeg-php.sourceforge.net/	Restriktive Lizenz - ffmpeg in Befehlszeile mit LAUNCH EXTERNAL PROCESS verwenden
Image Magick	http://php.net/manual/book.imagick.php	Restriktive Lizenz - GD 2 verwenden
IMAP (Internet Message Access Protocol)	http://php.net/imap	Restriktive Lizenz - Das integrierte Plug-In 4D Internet Commands verwenden.
PDF (Portable Document Format)	http://php.net/pdf	Restriktive Lizenz - Haru PDF verwenden
Mysqlnd (MySQL Native Driver)	http://dev.mysql.com/downloads/connector/php-mysqlnd/	Nicht relevant in der 4D Umgebung

Die Datei php.ini verändern

Die Datei "php.ini" zum Ändern (siehe unten) kann entweder in den Ordner **Resources\php** des 4D Programms (Standarddatei) oder in den Ordner **Resources** der Datenbank (eigene Datei) gelegt werden. Weitere Informationen dazu finden Sie im Abschnitt **PHP Skripte in 4D ausführen**.

Warnung: Das Ändern der Datei "php.ini" sollte mit Bedacht ausgeführt werden und erfordert gute Kenntnis von PHP. Weitere Informationen zur Konfiguration eigener php.ini Dateien finden Sie in den 4D Kommentaren in der php.ini Datei.

Hinweis: Dauert die PHP Bearbeitung relativ lange (über 30 Sekunden), wird In 4D standardmäßig ein 'timeout' Fehler zurückgegeben und die Bearbeitung abgebrochen. In solchen Fällen können Sie das standardmäßige *Timeout* verändern, um der PHP Ausführung mehr Zeit zu geben. Es gibt zwei Wege:

- Die Variable **max_execution_time** in der Datei "php.ini" setzen (übergeben Sie einen Wert in Sekunden). Achtung: Diese Einstellung wirkt sich auf alle Skripte aus.
- Den Befehl **set_time_limit(nbSec)** im PHP Ausführungsskript setzen, das längere Bearbeitung benötigt. In *nbSec* übergeben Sie die max. Dauer für die Ausführung des PHP Skripts.
  Wir empfehlen diese Einstellung, da sie nur für dieses Skript gilt. Aus Sicherheitsgründen ist es in der Regel besser, für PHP Skripte als Timeout einen kleinen Wert zu setzen.

Programmiersprache

- 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 Neu 17.0
- TRACE
- Type
- Undefined
- Value type

Command name

Command name (Befehl {; Info {; Kapitel}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Befehl	Lange Ganzzahl	➡	Befehlsnummer
Info	Lange Ganzzahl	⬅	Eigenschaft Thread-Sicherheit des Befehls
Kapitel	Text	⬅	Kapitel zum Befehl
Funktionsergebnis	String	➲	Lokalisierter Befehlsname

Beschreibung

Die Funktion **Command name** gibt den Namen des Befehls und optional seine Eigenschaften zu der in *Befehl* übergebenen Nummer zurück.

Hinweis: Die entsprechende Nummer zum Befehlsnamen erscheint im Explorer und in dieser Dokumentation jeweils in der rechten oberen Ecke unter **Eigenschaften**.

Hinweis zur Kompatibilität: Da der Befehlsname zwischen verschiedenen Versionen variieren kann (umbenannte Befehle), konnte über diese Funktion ein Befehl direkt mit seiner Nummer angegeben werden, insbesondere in nicht-tokenisierten Code-Teilen. Diese Vorgehensweise hat im Zuge der Evolution von 4D heutzutage an Bedeutung verloren, da 4D für nicht-tokenisierte Anweisungen (Formeln) eine Syntax mit Tokens anbietet. Damit lassen sich potentielle Probleme bei geänderten Befehlsnamen oder anderen Elementen, wie z.B. Tabellen, vermeiden und es ist trotzdem möglich, diese Namen in lesbarer Form einzugeben. Weitere Informationen dazu finden Sie im Abschnitt **Tokens in Formeln verwenden**. Außerdem wird ab 4D v15 standardmäßig in allen 4D Programmiersprachen die englische Version verwendet. Jedoch kann eine französische 4D Version über die Option "Verwende regionale Systemeinstellungen" auf der **Seite Methoden** der 4D Einstellungen weiterhin die französischen 4D Befehlsnamen verwenden.

Es gibt zwei optionale Parameter:

- *Info*: Eigenschaften des Befehls. Der zurückgegebene Wert ist ein *bit Feld*, wobei aktuell nur das erste Bit (bit 0) von Bedeutung ist und auf 1 gesetzt wird, wenn der Befehl thread-safe ist (0 ist thread-unsafe). Nur thread-safe Befehle lassen sich in preemptive Prozessen verwenden.
- *Kapitel*: Gibt den Kapitelnamen zum Befehl in der 4D Programmiersprache zurück.

Command name setzt die Variable *OK* auf 1, wenn *Befehl* zu einer vorhandenen Befehlsnummer passt, andernfalls auf 0. Bei deaktivierten Befehlen gibt **Command name** einen leeren String zurück (siehe letztes Beispiel).

Beispiel 1

Mit folgendem Code können Sie alle gültigen 4D Befehle in ein Array laden:

```
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)  //Befehlsnummer existiert
      If(Length($Txt_command)>0)  //Befehl ist nicht deaktiviert
         APPEND TO ARRAY($tTxt_commands;$Txt_command)
         APPEND TO ARRAY($tLon_Command_IDs;$Lon_id)
      End if
   End if
Until(OK=0)  //Ende der vorhandenen Befehle
```

Beispiel 2

Sie wollen in einem Formular eine DropDown-Liste mit den Grundbefehlen für Summen füllen. Die Objektmethode dafür lautet:

```
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
```

In der englischen/deutschen Version von 4D lautet die DropDown-Liste: Sum, Average, Min und Max. In der französischen Version (wenn die Option "Verwende regionale Systemeinstellungen" markiert ist) lautet sie: Somme, Moyenne, Min und Max.

Beispiel 3

Eine Methode erstellen, die **wahr** zurückgibt, wenn der Befehl mit der im Parameter übergebenen Nummer thread-safe ist, sonst falsch.

```
//Projektmethode Is_Thread_Safe
//Is_Thread_Safe(numCom) -> Boolean

C_LONGINT($1;$threadsafe)
C_TEXT($name)
C_BOOLEAN($0)
$name:=Command name($1;$threadsafe;$theme)
If($threadsafe ?? 0)  //wenn das erste Bit auf 1 gesetzt wird
   $0:=True
Else
   $0:=False
End if
```

Dann können Sie z.B. für den Befehl "SAVE RECORD" (53) folgende Anweisung schreiben:

```
$isSafe:=Is_Thread_Safe(53)
// gibt wahr zurück
```

Count parameters

Count parameters -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Lange Ganzzahl	Anzahl der aktuell übergebenen Parameter

Beschreibung

Die Funktion **Count parameters** gibt die Anzahl der Parameter zurück, die einer Projektmethode übergeben wurde.

Warnung: Count parameters ist nur in einer Projektmethode von Bedeutung, die von einer anderen Methode aufgerufen wird. Ist die Projektmethode, die diese Funktion aufruft, einem Menü zugeordnet, gibt sie den Wert 0 (Null) zurück.

Beispiel 1

Projektmethoden von 4D akzeptieren optionale Parameter.
Sie können zum Beispiel die Methode ***MyMethod***(a;b;c;d) auf folgende Arten aufrufen:

```
MyMethod(a;b;c;d)  ` Alle Parameter wurden übergeben
MyMethod(a;b;c)  ` Der letzte Parameter wurde nicht übergeben
MyMethod(a;b)  ` Die letzten beiden Parameter wurden nicht übergeben
MyMethod(a)  ` Nur der erste Parameter wurde übergeben
MyMethod  ` Kein Parameter wurde übergeben
```

Mit **Count parameters** innerhalb ***MyMethod*** stellen Sie die aktuelle Anzahl der Parameter fest und führen je nach Ergebnis verschiedene Operationen aus. Folgendes Beispiel zeigt eine Textmeldung und kann diese in einen 4D Write Bereich einfügen bzw. an ein Dokument auf der Festplatte senden:

```
  ` Projektmethode APPEND TEXT
  ` APPEND TEXT ( Text { ; Lang { ; Zeit } } )
  ` APPEND TEXT ( Text { ; 4D Write Bereich { ; DocRef } } )

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
```

Mit dieser Projektmethode in Ihrer Anwendung können Sie schreiben:

```
APPEND TEXT(vtSomeText)  ` Zeigt nur die Textmeldung an
APPEND TEXT(vtSomeText;$wrArea)  ` Zeigt die Textmeldung an und fügt sie dem Bereich $wr an
APPEND TEXT(vtSomeText;0;$vhDocRef)  ` Zeigt die Textmeldung an und schreibt sie in $vhDocRef
```

Beispiel 2

Projektmethoden von 4D akzeptieren verschiedene Anzahlen von Parametern desselben Typs. Dazu verwenden Sie einen Compilerbefehl, in dem Sie *${N}* als eine Variable übergeben. N gibt dabei den ersten Parameter an. Mit **Count parameters** haben Sie über eine *For* Schleife und die Parameter Indirektion Syntax Zugriff auf diese Parameter. Dieses Beispiel ist eine Funktion, die die größte Nummer zurückgibt, die als Parameter empfangen wird:

```
  ` Projektmethode Max of
  ` Max of ( Zahl { ; Zahl2... ; ZahlN } ) -> Zahl
  ` Max of ( Wert { ; Wert2... ; WertN } ) -> Größter Wert

C_REAL($0;${1})  ` Alle Parameter sowie Funktionsergebnis sind vom Typ ZAHL
$0:=${1}
For($vlParam;2;Count parameters)
   If(${$vlParam}>$0)
      $0:=${$vlParam}
```

```
    End if
End for
```

Mit dieser Projektmethode in Ihrer Anwendung können Sie schreiben:

```
vrResult:=Max of(Records in set("Operation A");Records in set("Operation B"))
```

oder:

```
vrResult:=Max of(r1;r2;r3;r4;r5;r6)
```

Current method name

Current method name -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	String		Name der aufrufenden Methode

Beschreibung

Die Funktion **Current method name** gibt den Namen der Methode zurück, in der sie aufgerufen wurde. Diese Funktion ist hilfreich zur Fehlerbehebung in generischen Methoden.

Je nach Art der aufrufenden Methode wird folgender String zurückgegeben:

Aufrufende Methode	Zurückgegebener String
Datenbankmethode	MethodenName
Trigger	Trigger auf [TabellenName]
Projektmethode	MethodenName
Tabellenformularmethode	[TabellenName].FormularName
Projektformularmethode	FormularName
TabellenformularObjektmethode	[TabellenName].FormularName.ObjektName
ProjektformularObjektmethode	FormularName.ObjektName
Komponente Projektmethode	MethodenName
Komponente Projektformularmethode	FormularName(KomponenteName)
Komponente ProjektformularObjektmethode	FormularName(KomponenteName).ObjektName(KomponenteName)

Diese Funktion lässt sich nicht innerhalb einer 4D Formel (Execute) aufrufen.

Hinweis: Damit diese Funktion im kompilierten Modus funktioniert, müssen Sie die Datenbank mit der Option "Bereichsüberprüfung" kompiliert haben. Dies wählen Sie in den Datenbank-Eigenschaften auf der Seite **Compiler** aus.
Um die Bereichsprüfung in einer Methode oder Teilen davon lokal zu deaktivieren, verwenden Sie folgende Kommentare:

```
`%R- Zum Deaktivieren der Bereichsprüfung
`%R+ Zum Aktivieren der Bereichsprüfung
`%R* Zum Wiederherstellen des ursprünglichen Status der Bereichsprüfung, d.h. so wie sie in den Einstellungen zum Kompilieren definiert wurden.
```

EXECUTE METHOD

EXECUTE METHOD (Methodenname {; Ergebnis {; Param}}{; Param2 ; ... ; ParamN})

Parameter	Typ		Beschreibung
Methodenname	String	➡	Name der auszuführenden Projektmethode
Ergebnis	Variable, Operator	⬅	Variable zum Empfangen des Methodenergebnisses oder * für Methode, die kein Ergebnis zurückgibt
Param	Ausdruck	➡	Parameter der Methode

Beschreibung

Der Befehl **EXECUTE METHOD** löst die Ausführung der Projektmethode *Methodenname* aus, in *Parameter* können Sie dieser Methode Parameter übergeben. Sie können den Namen jeder Methode übergeben, die sich aus der Datenbank oder der Komponente aufrufen lässt, welche den Befehl ausführen.

In *Ergebnis* können Sie eine Variable übergeben, die das Ergebnis aus der Ausführung von *Methodenname* empfängt (Wert in $0 innerhalb von *Methodenname*). Gibt die Methode kein Ergebnis zurück, übergeben Sie * als 2. Parameter. Gibt die Methode kein Ergebnis zurück und benötigt keine weiteren Parameter, genügt es, nur den Parameter *Methodenname* zu übergeben.
Der Kontext der Ausführung bleibt in der aufgerufenen Methode erhalten, d.h. die Definition des aktuellen Formulars und aller aktuellen Formularereignisse bleiben bestehen.
Rufen Sie diesen Befehl von einer Komponente aus auf und übergeben Sie in *Methodenname* einen Methodennamen aus der Host Datenbank oder umgekehrt, muss die Methode gemeinsam nutzbar sein (Option "Gemeinsam von Komponenten und Host benutzt" in Methoden-Eigenschaften).

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

Get action info

Get action info (Aktion {; Ziel}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Aktion	String	➡	Standardaktion Name oder Vorlage mit Parameter, wenn erforderlich
Ziel	Lange Ganzzahl	➡	Ziel der Aktion zum Erhalten der Information: Hauptformular oder aktuelles Formular
Funktionsergebnis	Objekt	⬅	Objekt mit Aktionsstatus als Boolean Eigenschaften: istAktiviert, istSichtbar, istMarkiert, istGemischt, istUnbekannterStatus

Beschreibung

Die Funktion **Get action info** gibt je nach dem aktuellen Kontext der Anwendung verschiedene Informationen über die definierte *Aktion* in *Ziel* zurück; dazu gehören auch Verfügbarkeit und Status.

In *Aktion* übergeben Sie den Namen der Standardaktion zum Prüfen zurück. Das kann ein String oder eine Konstante unter **Standardaktion** sein. Die komplette Übersicht finden Sie im Abschnitt **Standardaktionen** des Handbuchs *4D Designmodus*.

Hinweis: Einige Aktionen akzeptieren Parameter. In diesem Fall müssen Sie folgende Vorgabe verwenden: *actionName?ParameterName=ParameterValue.* Beispiel: *"gotoPage?value=2"*

In *Ziel* können Sie - wenn verfügbar - das Formular zum Ausführen von *Aktion* übergeben. Sie können eine der folgenden Konstanten unter dem Thema **Standardaktion** verwenden:

Konstante	Typ	Wert	Kommentar
ak current form	Lange Ganzzahl	1	Aktuelles Formular ist das Formular, wo die Aktion aufgerufen wurde. Das kann das Hauptformular oder ein Palettenfenster vor dem Hauptformular des aktuellen Prozesses sein.
ak main form	Lange Ganzzahl	2	Hauptformular ist das vorderste Dokument oder Dialogfenster des Prozesses, ohne Paletten- oder PopUp-Fenster.

Hinweis: Ist *Ziel* nicht angegeben, wird standardmäßig der Kontext ak current form verwendet.

Die Funktion **Get action info** gibt Information als Objekt mit folgenden Eigenschaften zurück:

Eigenschaft	Typ	Beschreibung	
Aktiviert	Boolean	wahr, wenn die Aktion auslösbar ist, sonst falsch	
Status	String	Wert kann ein String sein:	
		"markiert"	die Aktion ist markiert, d.h. die Eigenschaft ist gesetzt. Beispiel: Der ausgewählte Text ist fett, die Standardaktion ak font bold der Eigenschaft "Status" enthält "checked"
		"unchecked"	Die Aktion ist nicht markiert, d.h. die Eigenschaft ist nicht gesetzt. Beispiel: Der gewählte Text ist nicht fett, die Standardaktion ak font bold der Eigenschaft "status" enthält "unchecked".
		"mixed"	Die Aktion ist gemischt, d.h. die Eigenschaft ist teilweise gesetzt. Beispiel: Ein Teil des gewählten Textes ist "fett", die Standardaktion ak font bold der Eigenschaft "status" enthält "mixed".
Titel	Text	Aktueller lokalisierter Name der Aktion. Beispiel: "Undo", "Paste" für die englische Version	
Sichtbar	Boolean	wahr, wenn die Aktion im Formular sichtbar ist	
Wert	String	Aktueller Wert des Parameters Aktion (wenn vorhanden). Ist beispielsweise die Standardaktion "fontSize?value=10pt", enthält die Eigenschaft *Wert* "10pt"	

Lässt sich kein Status der Aktion bestimmen (z.B. weil sie für kein Objekt oder Menübefehl zutrifft), gibt die Funktion ein Nullobjekt zurück (undefiniert).

Beispiel

Abfragen, ob die Aktion Kopieren verfügbar ist (z.B. ob Daten ausgewählt sind):

```
C_OBJECT($actionInfo)
C_BOOLEAN($isEnabled)
$actionInfo:=Get action info(ak copy)
If(OB Is defined($actionInfo))  //Aktion ist im Prozess definiert
   If(OB Get($actionInfo;"enabled"))
 //die Aktion Kopieren ist verfügbar
   End if
End if
```

Get pointer

Get pointer (Name) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Name	String	→	Name einer Prozessvariablen
Funktionsergebnis	Zeiger	↩	Zeiger auf Prozessvariable

Beschreibung

Die Funktion **Get pointer** gibt einen Zeiger auf die in *varName* übergebene Variable zurück.
Für einen Zeiger auf ein Feld verwenden Sie **Field**. Für einen Zeiger auf eine Tabelle verwenden Sie **Table**.

Hinweis:
Sie können in **Get pointer** Ausdrücke wie *ArrName+"{3}"* übergeben, sowie 2D Array-Elemente wie *ArrName+"{3}{5}"* .
Dagegen können Sie keine Variablenelemente wie *ArrName+"{myVar}"* übergeben.

Beispiel 1

In einem Formular erstellen Sie eine Matrix 5 x 10 mit eingebbaren Variablen mit den Bezeichnungen v1, v2... v50. Um all diese Variablen zu initialisieren, schreiben Sie:

```
` ...
For($vlVar;1;50)
   $vpVar:=Get pointer("v"+String($vlVar))
   $vpVar->:=""
End for
```

Beispiel 2

Zeiger auf Elemente eines zweidimensionalen Arrays verwenden:

```
$pt:=Get pointer("a{1}{2}")
 //$pt=->a{1}{2}
$pt2:=Get pointer("atCities"+"{2}{6}")
 //$pt2=->atCities{2}{6}
```

INVOKE ACTION

INVOKE ACTION (Aktion {; Ziel})

Parameter	Typ		Beschreibung
Aktion	String	➡	Name der Standardaktion oder Vorgabe mit Parameter, wenn erforderlich
Ziel	Lange Ganzzahl	➡	Ziel zum Ausführen der Aktion: Aktuelles Formular (Standard) oder Hauptformular

Beschreibung

Der Befehl **INVOKE ACTION** löst die im Parameter *Aktion* definierte Standardaktion aus, optional im in *Ziel* angegebenen Kontext.

In *Aktion* übergeben Sie den Namen der auszuführenden Standardaktion. Das kann ein String oder eine Konstante unter dem Thema **Standardaktion** sein.

Die komplette Übersicht der Aktionen finden Sie im Abschnitt **Standardaktionen** des Handbuchs *4D Designmodus*.

Hinweise:

- Einige Aktionen akzeptieren einen Parameter. In diesen Fall müssen Sie folgende Vorgabe verwenden: *actionName?parameterName=parameterValue.* Beispiel: *"GotoPage?value=2"*
- Es gibt zusätzlich spezifische Aktionen für 4D Write Pro Dokumente. Weitere Informationen dazu finden Sie im Abschnitt **4D Write Pro Standardaktionen verwenden** des 4D Write Pro Handbuchs.

In *Ziel* können Sie das Formular übergeben, in dem *Aktion* ausgeführt werden soll. Sie können eine der folgenden Konstanten unter dem Thema **Standardaktion** verwenden:

Konstante	Typ	Wert	Kommentar
ak current form	Lange Ganzzahl	1	Aktuelles Formular ist das Formular, wo die Aktion aufgerufen wurde. Das kann das Hauptformular oder ein Palettenfenster vor dem Hauptformular des aktuellen Prozesses sein.
ak main form	Lange Ganzzahl	2	Hauptformular ist das vorderste Dokument oder Dialogfenster des Prozesses, ohne Paletten- oder PopUp-Fenster.

Hinweis: Ist *Ziel* nicht übergeben, wird standardmäßig der Kontext ak current form verwendet.

Der Befehl **INVOKE ACTION** wird je nach angegebenem *Ziel* synchron oder asynchron ausgeführt:

- Mit ak current form als *Ziel* läuft er synchron; die Aktion wird im aktuellen Zyklus beim Aufrufen des Befehls ausgeführt.
- Mit ak main form als *Ziel* läuft er asynchron; die Aktion wird im nächsten Zyklus nach Beenden der Methode des Formularobjekts ausgeführt.

Hinweis: Standardaktionen zum Bearbeiten (Ausschneiden, Kopieren, Einfügen, Alle auswählen, Löschen, Rückgängig/Wiederherstellen) ignorieren den Parameter *Ziel*. Solche Aktionen werden immer synchron mit dem editierbaren Objekt mit Fokus ausgeführt.

INVOKE ACTION generiert keinen Fehler, z.B. wenn die angefragte Aktion im aktuellen Kontext nicht verfügbar ist. Sie müssen die erwartete Aktion über die Funktion **Get action info** bestätigen.

Beispiel 1

Die Standardaktion **Kopieren** im aktuellen Formular ausführen:

```
INVOKE ACTION(ak copy;ak current form)
```

Beispiel 2

Eine Standardaktion **Gehe zu Seite** (Seite 3) im Hauptformular ausführen:

```
INVOKE ACTION(ak goto page+"?value=3";ak main form)
```

Is a variable

Is a variable (Zeiger) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Zeiger	Zeiger	➡	Zu testender Zeiger
Funktionsergebnis	Boolean	⮌	TRUE = Zeiger zeigt auf eine Variable FALSE = Zeiger zeigt nicht auf eine Variable

Beschreibung

Die Funktion **Is a variable** gibt TRUE zurück, wenn der in *Zeiger* übergebene Zeiger eine Variable ist. In allen anderen Fällen gibt sie FALSE zurück (Zeiger auf Datenfeld oder Tabelle, Zeiger Nil, usw.).

Wollen Sie den Namen der referenzierten Variablen oder die Feldnummer erfahren, können Sie den Befehl **RESOLVE POINTER** verwenden.

Is nil pointer

Is nil pointer (Zeiger) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Zeiger	Zeiger	⇒	Zu testender Zeiger
Funktionsergebnis	Boolean	⇐	TRUE = Zeiger Nil (->[]) FALSE = Gültiger Zeiger auf vorhandenes Objekt

Beschreibung

Die Funktion **Is nil pointer** gibt TRUE zurück, wenn der in *Zeiger* zurückgegebene Zeiger gleich Nil (->[]) ist. In allen anderen Fällen gibt sie FALSE zurück (Zeiger auf Feld, Tabelle oder Variable).

Wollen Sie den Namen der referenzierten Variablen bzw. die Nummer des referenzierten Feldes herausfinden, können Sie den Befehl **RESOLVE POINTER** verwenden.

Beispiel

```
C_POINTER($ptr)
...
If(Is nil pointer($ptr))
End if
 // ist gleichwertig mit
If($ptr=Null)
End if
```

NO TRACE

NO TRACE
Dieser Befehl benötigt keine Parameter

Beschreibung

Nach dem Aufruf von **NO TRACE** wird das Debugger-Fenster (Schrittfenster) nicht mehr angezeigt.

NO TRACE schaltet den Debugger ab, der durch **TRACE** aktiviert wurde, entweder durch einen Fehler oder vom Benutzer. Dieses Fenster erscheint, wenn der Befehl **TRACE** in einer Methode aufgerufen wurde, oder wenn Sie nach einer Fehlermeldung von 4D auf die Schaltfläche **Schritt** geklickt haben.

In kompilierten Datenbanken wird **NO TRACE** ignoriert.

Null

Null -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Null	Nullwert

Beschreibung

Die Funktion **Null** gibt den Wert Null vom Typ null zurück.
Mit dieser Funktion können Sie den Wert **Null** zuweisen oder mit folgenden Elementen der Programmiersprache vergleichen:

Elemente der Programmiersprache	Befehle
Werte der Objekteigenschaft	Vergleich von **Null** mit einer Objekteigenschaft gibt wahr zurück, wenn der Wert der Eigenschaft Null ist; sonst falsch. Zur Vereinfachung des Code gibt **Null** auch wahr zurück, wenn die Eigenschaft im Objekt nicht existiert (ist z.B. **Undefined**), siehe Beispiel 5.
Collection Elemente	Wird eine Collection mit nicht-zusammenhängenden Elementen erweitert, erhalten alle dazwischenliegenden Elemente automatisch den Wert **Null**.
Objekt Variablen (**C_OBJECT**)	Wird der Wert **Null** diesem Variablentyp zugewiesen, wird sein Inhalt gelöscht. In diesem Fall hat die Funktion denselben Effekt wie Aufrufen von **CLEAR VARIABLE**.
Collection Variablen (**C_COLLECTION**)	Wird der Wert **Null** diesem Variablentyp zugewiesen, wird sein Inhalt gelöscht. In diesem Fall hat die Funktion denselben Effekt wie Aufrufen von **CLEAR VARIABLE**.
Zeiger Variablen (**C_POINTER**)	Wird der Wert **Null** diesem Variablentyp zugewiesen, wird sein Inhalt gelöscht. In diesem Fall hat die Funktion denselben Effekt wie Aufrufen von **CLEAR VARIABLE**.
Bild Variablen (**C_PICTURE**)	(*) Wird der Wert **Null** diesem Variablentyp zugewiesen, wird sein Inhalt gelöscht. In diesem Fall hat die Funktion denselben Effekt wie Aufrufen von **CLEAR VARIABLE**.

Der Wert **Null** lässt sich nicht als Parameter an eine Methode übergeben oder als Funktionsergebnis bei eigenen Funktionen ($0) zurückgeben.

Hinweis: Diese Funktion lässt sich nicht mit skalaren Feldern der Datenbank verwenden. Nullwerte in Feldern der Datenbank werden von der SQL Engine verwaltet und über die Befehle **Is field value Null** und **SET FIELD VALUE NULL** verwaltet,

Beispiel 1

Den Wert **Null** für eine Objekteigenschaft zuweisen und testen:

```
C_OBJECT(vEmp)
vEmp:=New object
vEmp.name:="Smith"
vEmp.children:=Null

If(vEmp.children=Null)  //wahr
End if
If(vEmp.name=Null)  //falsch
End if
If(vEmp.parent=Null)  //wahr
End if
```

Hinweis: Für dieses Beispiel muss die Objekt Notation in den Struktureinstellungen aktiviert sein.

Beispiel 2

Den Wert **Null** einem Collection Element zuweisen und vergleichen:

```
C_COLLECTION(myCol)
myCol:=New collection(10;20;Null)
...
If(myCol[2]=Null)
  // wenn das 3. Element null ist
    ...
End if
```

Beispiel 3

Diese Beispiele zeigen verschiedene Wege, um Variablen den Wert **Null** zuzuweisen oder zu vergleichen:

```
 //Objektvariable
C_OBJECT($o)
$o:=New object
$o:=Null  //entspricht CLEAR VARIABLE($o)
If($o#Null)  //entspricht If (OB Is defined($o))
End if
```

```
 //Collection-Variable
C_COLLECTION($c)
$c:=New collection
$c:=Null  //entspricht CLEAR VARIABLE($c)
If($c#Null)
End if
```

```
 //Zeigervariable
C_POINTER($p)
$p:=->$v
$p:=Null  //entspricht CLEAR VARIABLE($p)
If($p=Null)  //entspricht If (Is Nil pointer($p))
End if
```

```
 //Bildvariable
C_PICTURE($i)
$i:=$vpicture
$i:=Null  //entspricht CLEAR VARIABLE($i)
If($i#Null)  //entspricht If (Picture size($i)#0)
End if
```

Beispiel 4

Hier sehen Sie je nach Kontext unterschiedliche Ergebnisse der Funktionen **Undefined** und **Null** mit Objekteigenschaften:

```
C_OBJECT(vEmp)
vEmp:=New object
vEmp.name:="Smith"
vEmp.children:=Null

$undefined:=Undefined(vEmp.name)  // Falsch
$null:=(vEmp.name=Null)  //Falsch

$undefined:=Undefined(vEmp.children)  // Falsch
$null:=(vEmp.children=Null)  //Wahr

$undefined:=Undefined(vEmp.parent)  // Wahr
$null:=(vEmp.parent=Null)  //True
```

RESOLVE POINTER

RESOLVE POINTER (Zeiger ; Name ; TabNum ; FeldNum)

Parameter	Typ		Beschreibung
Zeiger	Zeiger	⇒	Zeiger für den das referenzierte Objekt gefunden werden soll
Name	String	⇐	Name der referenzierten Variablen oder leerer String
TabNum	Lange Ganzzahl	⇐	Ziffer der referenzierten Tabelle bzw. Arrayelemente oder 0 oder -1
FeldNum	Lange Ganzzahl	⇐	Ziffer des referenzierten Feldes oder 0

Beschreibung

Der Befehl **RESOLVE POINTER** findet die Information des referenzierten Objekts über *Zeiger* und gibt ihn in den Parametern *VarName*, *TabNum* und *FeldNum* zurück.

RESOLVE POINTER gibt je nach Art des referenzierten Objekts folgende Werte zurück:

Referenziertes Objekt	Parameter		
	VarName	TabNum	FeldNum
None (Zeiger NIL)	"" (leerer String)	0	0
Variable	Name der Variablen	-1	-1
Array	Name des Array	-1	-1
Array Element	Name des Array	Elementnummer	-1
2D Array Element	Name des 2D Array	Element Zeilennummer	Element Spaltennummer
Tabelle	"" (leerer String)	Tabellennummer	0
Feld	"" (leerer String)	Tabellennummer	Feldnummer

Hinweise:

- Ist der in *Zeiger* übergebene Wert kein Zeigerausdruck, tritt ein Syntaxfehler auf.
- **RESOLVE POINTER** funktioniert nicht mit Zeigern auf lokale Variablen. Da per Definition mehrere lokale Variablen mit dem gleichen Namen an verschiedenen Stellen existieren können, kann der Befehl nicht die korrekte Variable finden.

Beispiel 1

Sie erstellen in einem Formular eine Gruppe von 100 eingebbaren Variablen, bezeichnet mit v1, v2... v100. Dazu führen Sie folgendes aus:

a. Erstellen Sie eine eingebbare Variable mit der Bezeichnung v.

b. Legen Sie die Eigenschaften des Objekts fest.

c. Weisen Sie diesem Objekt folgende Methode zu:

```
DoSomething(Self)  ` DoSomething ist eine Projektmethode in Ihrer Datenbank
```

d. An dieser Stelle können Sie nun entweder die Variable so oft wie benötigt duplizieren oder das Symbol für Matrix im Formulareditor verwenden.

e. Müssen Sie den Index der Variablen wissen, für den die Methode aufgerufen wurde, schreiben Sie in der Methode *DoSomething*:

```
RESOLVE POINTER($1;$vsVarName;$vlTableNum;$vlFieldNum)
$vlVarNum:=Num(Substring($vsVarName;2))
```

Bauen Sie Ihr Formular auf diese Weise auf, schreiben Sie die Methoden für die 100 Variablen nur einmal; Sie müssen nicht schreiben *DoSomething (1), DoSomething (2)...,DoSomething (100)*.

Beispiel 2

Zur Fehlerbehebung müssen Sie prüfen, ob der zweite Parameter ($2) einer Methode ein Zeiger auf eine Tabelle ist. Zu Beginn dieser Methode schreiben Sie:

```
  ` ...
If(◇DebugOn)
   RESOLVE POINTER($2;$vsVarName;$vlTableNum;$vlFieldNum)
   If(Not(($vlTableNum>0)&($vlFieldNum=0)&($vsVarName="")))
  ` WARNUNG: Der Zeiger ist keine Referenz auf eine Tabelle
      TRACE
      End
   End if
  ` ...
```

Beispiel 3

Siehe Beispiel zum Befehl **DRAG AND DROP PROPERTIES**.

Beispiel 4

Beispiel für einen Zeiger auf ein 2D Array:

```
ARRAY TEXT(atCities;100;50)
C_POINTER($city)
atCities{1}{2}:="Rom"
atCities{1}{5}:="Paris"
atCities{2}{6}:="New York"
 // ...andere Werte
$city:=->atCities{1}{5}
RESOLVE POINTER($city;$var;$rowNum;$colNum)
 //$var="atCities"
 //$rowNum="1"
 //$colNum="5"
```

Self

Self -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Zeiger		Zeiger auf Formularobjekt (sofern vorhanden) dessen Methode gerade ausgeführt wird. Sonst Nil (->[]), wenn außerhalb des Kontextes

Hinweis zur Kompatibilität

Dieser Befehl wird zur Wahrung der Kompatibilität beibehalten. Ab 4D Version 12 empfehlen wir, den Befehl **OBJECT Get pointer** zu verwenden.

Beschreibung

Die Funktion **Self** gibt einen Zeiger auf das Objekt zurück, dessen Objektmethode gerade ausgeführt wird.

Self wird verwendet, um auf eine Variable innerhalb ihrer eigenen Objektmethode zu verweisen. Sie gibt einen gültigen Zeiger zurück, wenn sie in einer Objektmethode oder Projektmethode (direkt oder indirekt über eine Objektmethode) aufgerufen wird. Wird **Self** außerhalb des Kontexts aufgerufen, gibt sie einen Zeiger **Is nil pointer** (->[]) zurück.

Tipp: Self ist hilfreich in einem Formular, wenn mehrere Objekte die gleiche Aktion ausführen sollen. Dann genügt der Aufruf einer globalen Methode mit einem Zeiger auf das aktuelle Objekt, also auf "sich selbst".

Bei Verwenden in einer Listbox gibt die Funktion je nach Kontext einen Zeiger auf die Listbox oder auf die Spalte der Listbox zurück. Weitere Informationen dazu finden Sie unter der Funktion **Einführung in Listboxen**.

Beispiel

Siehe Beispiel zum Befehl **RESOLVE POINTER**.

This

This -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Objekt	Das aktuelle Element

Beschreibung

Der Befehl **This** gibt eine Referenz auf das zu bearbeitende Objekt zurück.
Der Befehl dient zur Verwendung in folgendem Kontext:

- eine Listbox, die einer Collection oder Entity-Selection zugeordnet ist
- während dem Ereignis On Display Detail oder On Data Change

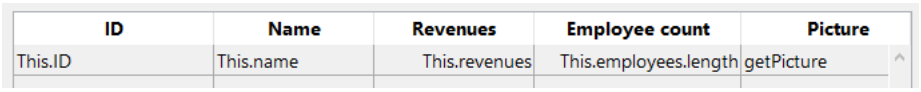
In diesem spezifischen Kontext gibt **This** eine Referenz auf das Collection-Element oder auf die Entity zurück, worauf die Listbox zugreift, um die aktuelle Zeile anzuzeigen. In anderen Fällen gibt er **Null** zurück.
Sie können auf alle Eigenschaften des Collection-Elements oder der Entity über **This.<*propertyPath*>** zugreifen. Zum Beispiel: *This.name* oder *This.employer.lastName* sind gültige Pfade zu Eigenschaften eines Elements bzw. einer Entity.
Hinweis: Verwenden Sie eine Collection mit skalaren Werten, erstellt 4D ein Objekt für jedes Element mit einer einzelnen Eigenschaft **value**. So ist der Wert des Elements durch den Ausdruck **This.value** verfügbar.

Beispiel 1

Wir nehmen eine Collection von Objekten, jeweils mit dieser Struktur:

{ "ID": 1234 "name": "Xavier", "revenues": 47300, "employees": ["Allan", "Bob", "Charlie"] },{ "ID": 2563 "name": "Carla", "revenues": 55000, "isFemale": true "employees": ["Igor", "Jane"] },...

In der Listbox bezieht sich jede Spalte auf eine der Eigenschaften des Objekts, entweder direkt (This.name), indirekt (This.employees.length) oder über einen Ausdruck (getPicture), in dem sie direkt verwendet werden kann. Die Listbox sieht folgendermaßen aus:

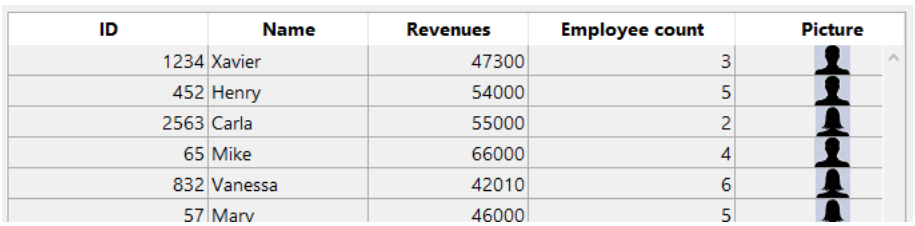


ID	Name	Revenues	Employee count	Picture
This.ID	This.name	This.revenues	This.employees.length	getPicture

Die Projektmethode *GetPicture* wird automatisch während dem Ereignis On Display Detail ausgeführt.

```
//GetPicture Methode
C_PICTURE($0)
If(This.isFemale)
    $0:=Form.genericFemaleImage
Else
    $0:=Form.genericMaleImage
End if
```

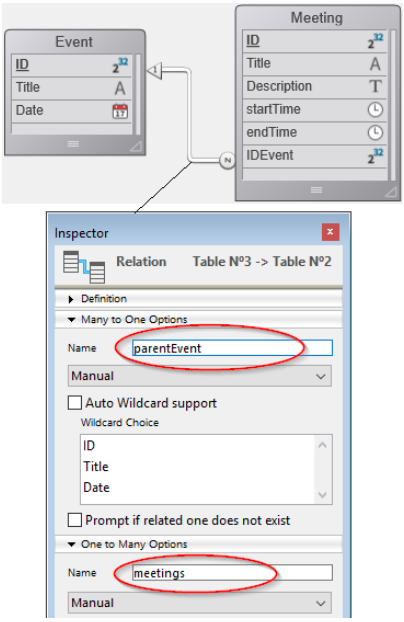
Nach Ausführen des Formulars erhalten Sie dieses Ergebnis:



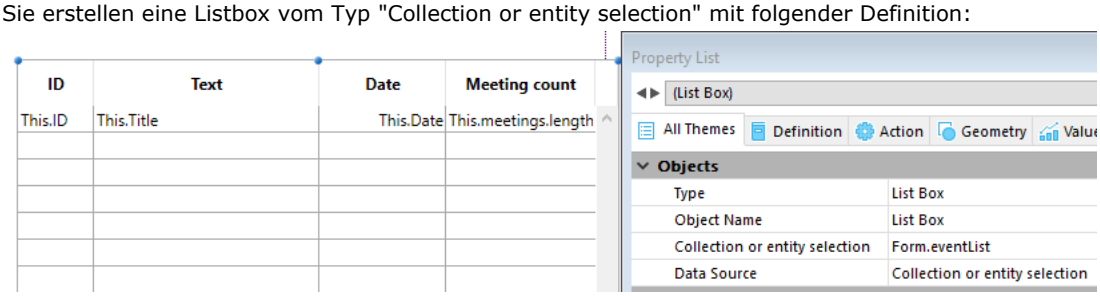
ID	Name	Revenues	Employee count	Picture
1234	Xavier	47300	3	
452	Henry	54000	5	
2563	Carla	55000	2	
65	Mike	66000	4	
832	Vanessa	42010	6	
57	Mary	46000	5	

Beispiel 2

Entities aus der folgenden Struktur in einer Listbox anzeigen:



Sie erstellen eine Listbox vom Typ "Collection or entity selection" mit folgender Definition:



Dabei gilt folgendes:

- *This.ID*, *This.Title* und *This.Date* beziehen sich direkt auf die entsprechenden Attribute in der Dataclass ds.Event.
- *This.meetings* ist ein verknüpftes Attribut (basierend auf dem Namen der *One-to-Many* Verknüpfung), das eine Entity-Selection der Dataclass *ds.Meeting* zurückgibt.
- **Form.eventList** ist eine Entity-Selection, die der Listbox zugewiesen ist. Der Code zur Iinitialisierung lässt sich in das Formularereignis On Load setzen:

```
Case of
   :(Form event=On Load)
      Form.eventList:=ds.Event.all()  //gibt eine Entity-Selection mit allen Entities zurück
End case
```

Nach Ausführen des Formulars wird die Listbox automatisch mit der Entity-Selection gefüllt:

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
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **TRACE** unterbricht die laufende Methode und zeigt den Debugger (Schrittfenster). Das Fenster erscheint, bevor die nächste Codezeile ausgeführt wird. Es wird für jede ausgeführte Codezeile erneut angezeigt. Sie können den Debugger während der Ausführung von Code auch unter Windows mit der Tastenkombination **alt + Umschalttaste + rechte Maustaste**, auf Mac OS mit **ctrl + Wahltaste + Maustaste** aufrufen.

In kompilierten Datenbanken wird **TRACE** ignoriert.

4D Server: Rufen Sie **TRACE** in einer Projektmethode auf, die in einer Serverprozedur ausgeführt wird, erscheint das Debugger-Fenster auf dem Server-Rechner.

Tipp: Setzen Sie keine Aufrufe von **TRACE** in Formularen, wenn die Ereignisse On Activate und On Deactivate aktiv sind. Denn diese Ereignisse werden immer ausgelöst, wenn das Debugger-Fenster erscheint; Sie geraten dann in eine Endlos-Schleife zwischen diesen Ereignissen und dem Debugger-Fenster. Um solch eine Situation zu beenden, klicken Sie im Debugger mit gedrückter **Umschalttaste** auf die Schaltfläche **Weiter ohne Schritt**. Alle folgenden Aufrufe von **TRACE** innerhalb des Prozesses werden dann ignoriert.

Beispiel

Folgender Code erwartet, dass die Prozessvariable *BUILD_LANG* gleich ist mit "US" oder "FR". Ist das nicht der Fall, wird die Projektmethode ***DEBUG*** aufgerufen:

```
` ...
Case of
   :(BUILD_LANG="US")
      vsBHCmdName:=[Commands]CM US Name
   :(BUILD_LANG="FR")
      vsBHCmdName:=[Commands]CM FR Name
   Else
      DEBUG("Unerwarteter Wert BUILD_LANG")
End case
```

Die Projektmethode ***DEBUG*** lautet:

```
` Projektmethode DEBUG
` DEBUG (Text)
` DEBUG (Optionale Debug Information)

C_TEXT($1)

If(◇vbDebugOn) ` In die Methode gesetzte Interprozeßvariable
   If(Compiled application)
      If(Count parameters>=1)
         ALERT($1+Char(13)+"Call Designer at x911")
      End if
   Else
      TRACE
   End if
End if
```

Type

Type (FeldVar) -> Funktionsergebnis

Parameter	Typ		Beschreibung
FeldVar	Feld, Variable	➡	Zu testender Ausdruck
Funktionsergebnis	Lange Ganzzahl	⮌	Datentypnummer

Beschreibung

Die Funktion **Type** gibt einen numerischen Wert zurück mit dem Typ des Feldes oder der Variablen, der in *FeldVar* übergeben wurde.

4D bietet unter dem Thema **Feld und Variablentypen** folgende vordefinierten Konstanten:

Konstante	Typ	Wert
Array 2D	Lange Ganzzahl	13
Blob array	Lange Ganzzahl	31
Boolean array	Lange Ganzzahl	22
Date array	Lange Ganzzahl	17
Integer array	Lange Ganzzahl	15
Is alpha field	Lange Ganzzahl	0
Is BLOB	Lange Ganzzahl	30
Is Boolean	Lange Ganzzahl	6
Is collection	Lange Ganzzahl	42
Is date	Lange Ganzzahl	4
Is float	Lange Ganzzahl	35
Is integer	Lange Ganzzahl	8
Is integer 64 bits	Lange Ganzzahl	25
Is longint	Lange Ganzzahl	9
Is null	Lange Ganzzahl	255
Is object	Lange Ganzzahl	38
Is picture	Lange Ganzzahl	3
Is pointer	Lange Ganzzahl	23
Is real	Lange Ganzzahl	1
Is string var	Lange Ganzzahl	24
Is subtable	Lange Ganzzahl	7
Is text	Lange Ganzzahl	2
Is time	Lange Ganzzahl	11
Is undefined	Lange Ganzzahl	5
LongInt array	Lange Ganzzahl	16
Object array	Lange Ganzzahl	39
Picture array	Lange Ganzzahl	19
Pointer array	Lange Ganzzahl	20
Real array	Lange Ganzzahl	14
String array	Lange Ganzzahl	21
Text array	Lange Ganzzahl	18
Time array	Lange Ganzzahl	32

Sie können **Type** auf Felder, Interprozessvariablen, Prozessvariablen, lokale Variablen und dereferenzierte Zeiger für diese Objekttypen anwenden. Sie können **Type** auf die Parameter *($1, $2 ... ${...})* einer Projektmethode oder Funktionsergebnisse *($0)* anwenden.

Hinweis: Sie können **Type** nicht auf skalare Ausdrücke wie Objekteigenschaften (*emp.name*) oder Elemente einer Collection (*myColl[5]*) anwenden. Dafür müssen Sie die Funktion **Value type** verwenden.

Beispiel 1

Folgende Projektmethode leert einige oder alle Felder des aktuellen Datensatzes der Tabelle mit einem als Zeiger übergebenen Parameter. Das geschieht ohne Löschen oder Ändern des aktuellen Datensatzes:

```
` Projektmethode EMPTY RECORD
` EMPTY RECORD ( Zeiger {; Lang } )
` EMPTY RECORD ( -> [Tabelle] { ; Typ Flags } )

C_POINTER($1)
C_LONGINT($2;$vlTypFlags)

If(Count parameters>=2)
   $vlTypFlags:=$2
```

```
Else
   $vlTypFlags:=0xFFFFFFFF
End if
For($vlField;1;Get last field number($1))
   $vpField:=Field(Table($1);$vlField)
   $vlFieldType:=Type($vpField->)
   If($vlTypeFlags ??$vlFieldType )
      Case of
         :(($vlFieldType=ls alpha field)|($vlFieldType=ls text))
            $vpField->:=""
         :(($vlFieldType=ls real)|($vlFieldType=ls integer)|($vlFieldType=ls longint))
            $vpField->:=0
         :($vlFieldType=ls date)
            $vpField->:=!00/00/00!
         :($vlFieldType=ls time)
            $vpField->:=?00:00:00?
         :($vlFieldType=ls Boolean)
            $vpField->:=False
         :($vlFieldType=ls picture)
            C_PICTURE($vgEmptyPicture)
            $vpField->:=$vgEmptyPicture
         :($vlFieldType=ls subtable)
            Repeat
               ALL SUBRECORDS($vpField->)
               DELETE SUBRECORD($vpField->)
            Until(Records in subselection($vpField->)=0)
         :($vlFieldType=ls BLOB)
            SET BLOB SIZE($vpField->;0)
      End case
   End if
End for
```

Mit dieser Projektmethode in Ihrer Datenbank können Sie schreiben:

```
  ` Leere den gesamten aktuellen Datensatz der Tabelle [Dinge tun]
EMPTY RECORD(->[Dinge tun])
  ` Leere Text, BLOB und Bildfelder für den aktuellen Datensatz der Tabelle [Dinge tun]
EMPTY RECORD(->[Dinge tun];0?+ls text?+ls BLOB?+ls picture)
  ` Leere den gesamten aktuellen Datensatz der Tabelle [Dinge tun] außer alphanumerische Felder
EMPTY RECORD(->[Dinge tun];-1?-ls alpha field)
```

Beispiel 2

In manchen Fällen, z.B. beim Schreiben von generischem Code, müssen Sie wissen, ob ein Array ein unabhängiges Standard Array oder die Zeile eines 2D Arrays ist. Dafür können Sie folgenden Code verwenden:

```
ptrmyArr:=->myArr{6}  ` Ist myArr{6} die Zeile eines 2D Array?
RESOLVE POINTER(ptrmyArr;varName;tableNum;fieldNum)
If(varName#"")
   $ptr:=Get pointer(varName)
   $thetype:=Type($ptr->)
  ` Ist myArr{6} die Zeile eines 2D Array, ist $thetype gleich 13
End if
```

Beispiel 3

Siehe Beispiel zum Befehl **APPEND DATA TO PASTEBOARD**.

Beispiel 4

Siehe Beispiel zum Befehl **DRAG AND DROP PROPERTIES**.

Undefined

Undefined (Variablenname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Variablenname	Variable, Ausdruck, Feld	➡	Zu testende Variable
Funktionsergebnis	Boolean	⮌	True = Variable ist derzeit undefiniert, False = Variable ist derzeit definiert

Beschreibung

Die Funktion **Undefined** gibt den Wert **True** zurück, wenn der Parameter *Variablenname* nicht definiert ist und **False**, wenn *Variablenname* definiert ist.

- Eine Variable ist definiert, wenn sie über eine Kompilierungsdirektive erstellt oder ein Wert zugewiesen ist. In allen anderen Fällen ist sie undefiniert. Wurde die Datenbank kompiliert, gibt die Funktion für alle Variablen den Wert **False** zurück.
- Eine Objekteigenschaft ist undefiniert, wenn sie im Objekt nicht existiert.

Hinweis: Für Feldreferenzen gibt **Undefined** immer **False** zurück.

Beispiel 1

Folgender Code verwaltet die Prozesserstellung, wenn eine Menüzeile für ein spezifisches Plug-In Ihrer Anwendung ausgewählt wird. Existiert der Prozess bereits, bringen Sie ihn nach vorne; existiert er nicht, starten Sie ihn. Dafür richten Sie für jedes Plug-In der Anwendung eine Interprozessvariable ◇*PID_...* ein, die Sie in der **Datenbankmethode On Startup** initialisieren.

Beim Entwickeln der Datenbank fügen Sie neue Plug-Ins hinzu. Anstatt jedes Mal die **Datenbankmethode On Startup** zu ändern (Hinzufügen der Initialisierung der entsprechenden ◇*PID_...*) und die Datenbank zur Reinitialisierung erneut zu öffnen, können Sie das Hinzufügen eines neuen Plug-Ins mit der Funktion **Undefined** direkt verwalten:

```
//Globale Prozedur M_ADD_CUSTOMERS

If(Undefined(◇PID_ADD_CUSTOMERS))
  ` Kümmert sich um zukünftige Entwicklungsphasen
   C_LONGINT(◇PID_ADD_CUSTOMERS)
   ◇PID_ADD_CUSTOMERS:=0
End if

If(◇PID_ADD_CUSTOMERS=0)
   ◇PID_ADD_CUSTOMERS:=New process("P_ADD_CUSTOMERS";64*1024;"P_ADD_CUSTOMERS")
Else
   SHOW PROCESS(◇PID_ADD_CUSTOMERS)
   BRING TO FRONT(◇PID_ADD_CUSTOMERS)
End if
 // Hinweis: P_ADD_CUSTOMERS, die Hauptprozessmethode, setzt ◇PID_ADD_CUSTOMERS beim Beenden auf Null.
```

Beispiel 2

Hier sehen Sie je nach Kontext unterschiedliche Ergebnisse der Funktionen **Undefined** und **Null** mit Objekteigenschaften:

```
C_OBJECT(vEmp)
vEmp:=New object
vEmp.name:="Smith"
vEmp.children:=Null

$undefined:=Undefined(vEmp.name)  // Falsch
$null:=(vEmp.name=Null)  //Falsch

$undefined:=Undefined(vEmp.children)  // Falsch
$null:=(vEmp.children=Null)  //Wahr

$undefined:=Undefined(vEmp.parent)  // Wahr
$null:=(vEmp.parent=Null)  //True
```

Value type

Value type (Ausdruck) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Ausdruck	Ausdruck	➡	Ausdruck, dessen Ergebniswert getestet werden soll
Funktionsergebnis	Lange Ganzzahl	⮌	Nummer des Datentyps

Beschreibung

Die Funktion **Value type** gibt den Typ des Werts zurück, der sich aus der Bewertung des Parameters *Ausdruck* ergibt.
Sie gibt einen numerischen Wert zurück, der sich mit einer der folgenden Konstanten unter dem Thema **Feld und Variablentypen** vergleichen lässt:

Konstante	Typ	Wert
Is BLOB	Lange Ganzzahl	30
Is Boolean	Lange Ganzzahl	6
Is collection	Lange Ganzzahl	42
Is date	Lange Ganzzahl	4
Is longint	Lange Ganzzahl	9
Is null	Lange Ganzzahl	255
Is object	Lange Ganzzahl	38
Is picture	Lange Ganzzahl	3
Is pointer	Lange Ganzzahl	23
Is real	Lange Ganzzahl	1
Is text	Lange Ganzzahl	2
Is time	Lange Ganzzahl	11
Is undefined	Lange Ganzzahl	5
Object array	Lange Ganzzahl	39

Diese Funktion dient zum Zurückgeben des Typs skalarer Ausdruck, z.B. Werte, die in *Ausdruck* gespeichert oder von diesem zurückgegeben werden. Sie lässt sich auf folgende 4D Ausdrücke anwenden:

- Objekteigenschaften (*emp.name*),
- Collection Elemente (*myCol[5]*).

Hinweis: Numerische Objekteigenschaften werden immer als Zahl gewertet:

```
C_OBJECT($o)
$o:=New object("value";42)
$vType:=Value type($o.value)  //$vType=Is real
```

Value type lässt sich auf jeden gültigen 4D Ausdruck anwenden, inkl. Felder, Variablen und Parameter. Sie gibt im Unterschied zur Funktion **Type** den *internen* Typ des Werts zurück, der sich aus der Bewertung von *Ausdruck* ergibt, und nicht seinen *deklarierten* Typ. Da die 4D Programmiersprache einige Wertetypen intern konvertiert, kann sich das Ergebnis von **Value type** vom deklarierten Typ unterscheiden. Beispielsweise konvertiert 4D intern die Feldwerte vom Typ **Ganzzahl 64 bit**. Das liefert folgende Ergebnisse:

```
$vType1:=Type([myTable]Long64field)  //$vType=Is integer 64 bits
$vType2:=Value type([myTable]Long64field)  //$vType=Is real (im interpretierten Modus)
```

Weitere Unterschiede gibt es bei Arrays (ihre Bewertung gibt den aktuellen Element Index zurück) und kompiliertem Modus. Folgende Tabelle zeigt diese Unterschiede:

Deklarierter Typ	Ergebnis mit Type	Ergebnis (interpretiert) mit Value type	Ergebnis (kompiliert) mit Value type	Kommentar
ARRAY TEXT($t;1)	Text array	Is real	Is longint	$t enthält den aktuellen Elementindex, d.h. eine Nummer
Feld **Alpha**	Is alpha field	Is text	Is text	4D behandelt alle Strings intern als Texte
Feld **Ganzzahl**	Is integer	Is real	Is longint	Aus Optimierungsgründen werden im interpretierten Modus alle numerischen Werte als Zahl gewertet und...
Feld **Lange Ganzzahl**	Is longint	Is real	Is longint	... im kompilierten Modus alle Werte vom Typ Ganzzahl als Lange Ganzzahl (*)
Feld **Ganzzahl 64 bit**	Is integer 64 bits	Is real	Is longint	

(*) Zum Testen eines numerischen Wertes, der für kompilierten und interpretierten Modus gültig ist, können Sie folgenden Code schreiben:

```
If(Value type($myValue)=Is longint)|(Value type($myValue)=Is real)
```

Hinweis zur Kompatibilität: Ab 4D v16 R6 werden Datumsangaben in Objekteigenschaften entweder als Datumstyp oder als Text im ISO Datumsformat gespeichert. Weitere Informationen dazu finden Sie unter dem Selector Dates inside objects des Befehls **SET DATABASE PARAMETER**.

Beispiel 1

Die verschiedenen möglichen Typen eines Wertes Objekteigenschaft verwalten:

```
Case of
    :(Value type($o.value)=Is real)
  //Einen numerischen Wert verwalten
    :(Value type($o.value)=Is text)
  //Einen Text verwalten
    :(Value type($o.value)=Is object)
  //Ein Unterobjekt verwalten
        ...
End case
```

Beispiel 2

Alle numerischen Werte in einer Collection zusammenzählen:

```
C_COLLECTION($col)
C_REAL($sum)
$col:=New collection("Hello";20;"World2";15;50;Current date;True;10)
For($i;0;$col.length-1) //-1 da Collection bei 0 startet
    If(Value type($col[$i])=Is real)
        $sum:=$sum+$col[$i]
    End if
End for
ALERT(String($sum)) //95
```

Prozess (Kommunikation)

- Über Semaphoren
- Über Worker
- CALL WORKER
- CLEAR SEMAPHORE
- GET PROCESS VARIABLE
- KILL WORKER
- Semaphore
- SET PROCESS VARIABLE
- Test semaphore
- VARIABLE TO VARIABLE

Über Semaphoren

Was ist eine Semaphore?

In einem Computerprogramm ist eine Semaphore ein Tool zum Schutz von Aktionen, die nur von einem Prozess oder einem Benutzer zur selben Zeit ausgeführt werden dürfen.

In 4D sind Semaphoren beispielsweise beim Ändern eines Interprozess Arrays notwendig: Ändert ein Prozess die Werte des Array, darf kein anderer Prozess die Möglichkeit haben, zum gleichen Zeitpunkt dasselbe zu tun. Der Entwickler verwendet eine Semaphore, um einem Prozess anzuzeigen, dass er die Operationen nur ausführen kann, wenn kein anderer Prozess gerade dieselben Aktionen ausführt. Stößt ein Prozess auf eine Semaphore, gibt es drei Möglichkeiten:

- Er erhält sofort das Recht zur Ausführung
- Er wartet, bis er an der Reihe ist und das Recht zur Ausführung erhält
- Er setzt seinen Weg fort und verwirft die Idee, diese Aufgaben auszuführen

Die Semaphore schützt also Teile des Code. Sie erlaubt nur einen Prozess zur selben Zeit und blockiert den Zugriff, bis der aktuell berechtigte Prozess seinen Vorrang durch Freigabe der Semaphore beendet.

Befehle zum Arbeiten mit Semaphoren

In 4D setzen Sie eine Semaphore durch Aufrufen der Funktion **Semaphore**. Um sie wieder freizugeben, rufen Sie den Befehl **CLEAR SEMAPHORE** auf.

Die Funktion **Semaphore** hat ein sehr spezielles Verhalten, da sie zwei Aktionen gleichzeitig durchführt:

- Ist die Semaphore bereits zugewiesen, gibt die Funktion **Wahr** zurück
- Ist die Semaphore nicht zugewiesen, weist die Funktion sie dem Prozess zu und gibt zur gleichen Zeit **Falsch** zurück.

Diese Doppelaktion derselben Funktion stellt sicher, dass zwischen Testen und Zuweisen der Semaphore keine externe Operation eingefügt werden kann.

Über die Funktion **Test semaphore** können Sie herausfinden, ob eine Semaphore bereits zugewiesen ist oder nicht. Sie wird hauptsächlich als Teil von langen Operationen verwendet, wie z.B. beim Jahresabschluss in der Buchhaltung. Hier steuert **Test semaphore** die Oberfläche und verhindert den Zugriff zu bestimmten Operationen, wie z.B. Rechnungsdaten hinzufügen.

Semaphoren einsetzen

Für Semaphoren gelten folgende Prinzipien:

- Eine Semaphore soll in der gleichen Methode gesetzt und freigegeben werden
- Die Ausführung von Code, der durch eine Semaphore geschützt ist, soll so kurz wie möglich sein
- Der Code soll über den Parameter *tickCount* der Funktion **Semaphore** getaktet sein, um die Auflösung der Semaphore abzuwarten.

Hier ein typischer Code für eine Semaphore:

```
While(Semaphore("MySemaphore";300))
   IDLE
End while
 // hier durch Semaphore geschützten Code setzen
CLEAR SEMAPHORE("MySemaphore")
```

Eine nicht aufgelöste Semaphore kann Teile der Anwendung blockieren. Dieses Risiko lässt sich ausschließen, wenn Sie Semaphoren in derselben Methode setzen und auflösen.

Minimieren von Code, der durch eine Semaphore geschützt ist, erhöht die Durchlässigkeit der Anwendung und verhindert, dass die Semaphore wie ein Flaschenhals wirkt.

Zu guter Letzt optimiert der optionale Parameter *tickCount* der Funktion **Semaphore** das Abwarten bis zur Freigabe der Semaphore. Mit diesem Parameter funktioniert der Befehl wie folgt:

- Der Prozess wartet das Maximum der angegebenen Zahl Ticks (300 im Beispiel) für die gesetzte Semaphore ab, bis die Code Ausführung zur nächsten Zeile übergeht.
- Wird die Semaphore vor Ende dieses Limits aufgelöst, wird das sofort dem Prozess zugewiesen (**Semaphore** gibt Falsch zurück) und die Code Ausführung wird fortgesetzt.
- Wird die Semaphore erst mit Ende dieses Limits aufgelöst, wird die Code Ausführung dann fortgesetzt.

Die Funktion zieht auch Anfragen durch Einrichten einer Schleife vor. Auf diese Weise erhält der erste Prozess, der eine Semaphore abfragt, auch als erster eine.

Beachten Sie, dass die Wartezeit gemäß den Eigenheiten der Anwendung gesetzt wird.

Lokale oder globale Semaphoren

Es gibt zwei Arten von Semaphoren:

- **Lokale Semaphoren** werden nur von den Prozessen auf derselben Arbeitsstation und nur von dieser erkannt. Sie werden mit einem vorangestellten Dollarzeichen gekennzeichnet, z.B. $MeineSemaphore. Mit lokalen Semaphoren steuern Sie

Operationen zwischen Prozessen, die auf einer Arbeitsstation ausgeführt werden. Eine lokale Semaphore kann z.B. den Zugriff auf ein Interprozess-Array steuern, das sich alle Prozesse einer Arbeitsstation oder im 4D Einzelplatzbetrieb teilen.
- **Globale Semaphoren** sind für alle Benutzer und für alle Prozesse zugänglich. Mit globalen Semaphoren verwalten Sie Operationen zwischen Benutzern einer Datenbank im Mehrplatzbetrieb.

Globale und lokale Semaphoren funktionieren nach derselben Logik. Der Unterschied liegt in ihrer Reichweite. Im Client/Server Modus werden globale Semaphoren von allen Prozessen auf allen Clients und Servern gemeinsam genutzt. Eine lokale Semaphore wird nur von Prozessen auf dem Rechner genutzt, wo sie erzeugt wurde.
In 4D haben globale und lokale Semaphoren dieselbe Reichweite, da es nur einen Benutzer gibt. Wird die Datenbank jedoch in beiden Setups verwendet, stellen Sie sicher, dass - je nachdem, was ausgeführt werden soll - globale oder lokale Semaphoren verwendet werden.

Hinweis: Wir empfehlen, zum Verwalten lokaler Aspekte für den Client einer Anwendung, wie z.B. die Oberfläche oder ein Array mit Interprozessvariablen, lokale Semaphoren zu verwenden. Verwenden Sie dafür globale Semaphoren, verursacht das nicht nur unnötigen Austausch über das Netzwerk, sondern kann auch andere Client-Rechner beeinträchtigen. Über eine lokale Semaphore können Sie solche unerwünschten Nebenwirkungen vermeiden.

Über Worker

Überblick

Ein **Worker Prozess** ist ein einfacher und leistungsstarker Weg für den Informationsaustausch zwischen Prozessen. Dafür wurde ein asynchrones Nachrichtensystem eingerichtet, über das sich Prozesse und Formulare mit Parametern in ihrem eigenen Kontext aufrufen lassen.

Hinweis: Mit dem Befehl **CALL FORM** lässt sich eine Projektmethode auch mit Parametern im Kontext eines beliebigen Formulars ausführen.

Über den Befehl **CALL WORKER** kann jeder Prozess einen sog. "Worker" anheuern, um Projektmethoden mit Parametern in seinem eigenen Kontext auszuführen und so den Zugriff auf gemeinsam genutzte Information erlauben.

Diese Features bieten folgende Vorteile für die 4D Interprozess Kommunikation:

- Da sie sowohl von kooperativen als auch von preemptive Prozessen unterstützt werden, sind sie die perfekte Lösung für die Interprozess Kommunikation im preemptive Modus (Interprozessvariablen sind in preemptive Prozessen nicht erlaubt).
- Sie bieten eine einfache Alternative zu Semaphoren, die schwer einzurichten und komplex in der Verwendung sind (siehe **Über Semaphoren**).

Hinweis: Obwohl **CALL WORKER** und **CALL FORM** hauptsächlich für die Interprozess-Kommunikation im Kontext preemptive Prozesse dienen (nur 64-bit Versionen), sind sie auch in 32-bit verfügbar und mit kooperativen Prozessen verwendbar.

Worker verwenden

Ein **Worker** dient dazu, einen Prozess zum Ausführen von Projektmethoden aufzufordern. Er besteht aus:

- Einem einmaligen Namen **(*)**, der auch als Name des zugewiesenen Prozesses verwendet wird
- Einem zugewiesenen Prozess, der zu einem gegebenen Moment existiert oder nicht existiert
- Einer Nachrichtenbox
- Einer Start Methode (optional)

(*) Wichtig: Worker Namen berücksichtigen Klein- und Großschreibung, so kann es z.B. "myWorker" und "MyWorker" geben.

Über den Befehl **CALL WORKER** fordern Sie den Worker auf, eine Projektmethode auszuführen. Beim ersten Verwenden werden Worker und seine Nachrichtenbox erstellt; der zugewiesene Prozess wird ebenfalls beim ersten Mal automatisch gestartet. Endet der Worker Prozess später wieder, bleibt die Nachrichtenbox geöffnet und eine neue Nachricht in der Box startet einen neuen Worker Prozess.

Folgende Animation zeigt den Ablauf:

Im Gegensatz zur Funktion **New process** bleibt ein Worker Prozess am Leben, nachdem die Ausführung einer Prozessmethode beendet ist. Das bedeutet, dass alle Methodenausführungen für den gleichen Worker im gleichen Prozess laufen, der alle Informationen zum Prozess-Status behält (Prozessvariablen, aktueller Datensatz und aktuelle Auswahl...). Folglich greifen nacheinander ausgeführte Methoden auf die gleiche Information zu und nutzen sie gemeinsam. Das macht die Kommunikation zwischen Prozessen möglich. Die Nachrichtenbox des Worker verwaltet aufeinanderfolgende Aufrufe asynchron.

CALL WORKER bettet Methodenname und Befehlsargumente in eine Nachricht ein, die in die Nachrichtenbox des Workers gelegt wird. Der Worker Prozess wird dann gestartet, falls er noch nicht existiert und zum Ausführen der Nachricht aufgefordert, d.h. **CALL WORKER** gibt in der Regel zurück, bevor die Methode ausgeführt wird (die Bearbeitung ist asynchron). Aus diesem Grund gibt **CALL WORKER** keinen Wert zurück. Soll ein Worker Informationen an den Prozess zurücksenden, der in aufgerufen hat (callback), müssen Sie **CALL WORKER** erneut verwenden, um dem Aufrufer die nötige Information zu übergeben. Der Anrufer muss in diesem Fall natürlich selbst ein Worker sein.

Über **CALL WORKER** lässt sich keine Methode in einem Prozess ausführen, der über die Funktion **New process** erstellt wurde. Nur Worker Prozesse haben eine Nachrichtenbox und lassen sich über **CALL WORKER** aufrufen. Beachten Sie, dass ein mit **New process** erstellter Prozess zwar einen Worker aufrufen, aber nicht zurückgerufen werden kann.

Auf 4D Server lassen sich Worker Prozesse über Serverprozeduren erstellen: Sie können z.B. den Befehl **Execute on server** zum Ausführen einer Methode verwenden, die den Befehl **CALL WORKER** aufruft.

Ein Worker Prozess wird durch Aufrufen des Befehls **KILL WORKER** beendet. Er leert die Nachrichtenbox des Workers, fordert den zugewiesenen Prozess auf, die Bearbeitung von Nachrichten zu stoppen und ihre aktuelle Ausführung zu beenden.

CALL WORKER lässt sich nicht verwenden, um eine Methode in einem Prozess auszuführen, der über die Funktion **New process** erstellt wurde. **CALL WORKER** kann nur Worker Prozesse aufrufen.

Die Start Methode des Worker dient zum Erstellen des Worker (bei der ersten Verwendung). Wird **CALL WORKER** mit einem leeren Parameter *Methode* aufgerufen, wird automatisch wieder die Start Methode verwendet.

Der Hauptprozess, den 4D beim Öffnen einer Datenbank für Benutzer und Anwendungsmodi erstellt, ist ein Worker Prozess, der sich mit **CALL WORKER** aufrufen lässt. Beachten Sie, dass der Name des Hauptprozesses je nach der verwendeten Sprache des

4D Programms variieren kann, er hat aber immer die Prozessnummer 1; es ist also besser, ihn in **CALL WORKER** über die Prozessnummer anstatt dem Prozessnamen anzugeben.

Worker Prozesse identifizieren

Alle Worker Prozesse, mit Ausnahme des Hauptprozesses, haben als Prozesstyp Worker process (5), der über den Befehl **PROCESS PROPERTIES** zurückgegeben wird.

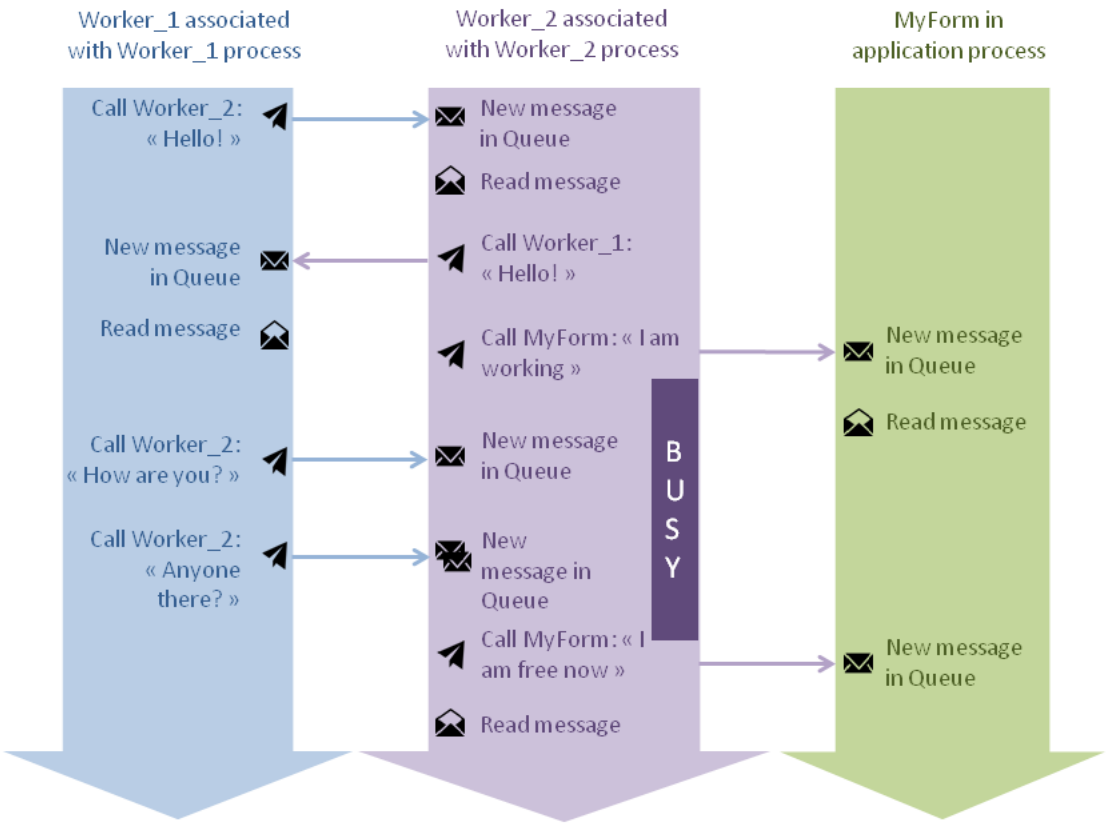
Im Runtime Explorer und im 4D Server Verwaltungsfenster gibt es neue Icons für Worker Prozesse:

Prozesstyp	Icon
Preemptive worker process	
Cooperative worker process	

Hinweis: Andere vorhandene Icons für Prozesse wurden in 4D v15 R5 aktualisiert.

Funktionsweise

Nachfolgende Darstellung zeigt den Ablauf der Kommunikation zwischen zwei Workern und einem Formular. Die ausgetauschte Information sind einfach Strings:



1. Worker_1 ruft Worker_2 auf und übergibt "Hello!" als Parameter.
2. Worker_2 erhält und liest die Nachricht. Er ruft Worker_1 zurück und übergibt "Hello!" als Parameter.
3. Worker_2 ruft dann das Formular MyForm auf und übergibt "I work" als Parameter. Er startet eine zeitaufwändige Operation, sein Status wird zu 'busy'.
4. Worker_1 ruft Worker_2 zweimal auf. Da das Nachrichtensystem asynchron läuft, werden die Nachrichten lediglich in Warteschleife gelegt und abgearbeitet, sobald Worker_2 verfügbar ist - nachdem Worker_2 MyForm aufgerufen hat.

CALL WORKER

CALL WORKER (Prozess ; Methode {; Param}{; Param2 ; ... ; ParamN})

Parameter	Typ		Beschreibung
Prozess	Text, Lange Ganzzahl	➡	Name oder Nummer des Worker Prozesses
Methode	Text	➡	Name der aufzurufenden Projektmethode
Param	Ausdruck	➡	Parameter für die Methode

Beschreibung

Der Befehl **CALL WORKER** verstellt oder ruft den Worker Prozess mit dem Namen oder der Nummer auf, übergeben in *Prozess*, und fordert auf, die *Methode* in ihrem Kontext mit dem optionalen Parameter *Param* auszuführen.

Der **CALL WORKER** bindet *Param* in eine Meldung ein und setzt sie in die Nachrichtenbox des Worker. Weitere Informationen dazu finden Sie im Abschnitt **Über Worker**.

Im Parameter *Prozess* können Sie den Worker über seinen Prozessnamen oder seine Prozessnummer angeben:

- Übergeben Sie die Nummer eines Prozesses, der nicht existiert oder wurde der Prozess weder mit **CALL WORKER** noch von 4D selbst erstellt (so wie der Hauptanwendungsprozess), führt **CALL WORKER** nichts aus.
- Übergeben Sie den Namen eines Prozesses, der nicht existiert, wird ein neuer Worker Prozess erstellt.

Hinweis: Der **Hauptprozess**, den 4D beim Öffnen einer Anwendung für die Benutzeroberfläche und den Anwendungsmodus erstellt, ist ein Worker Prozess. Er lässt sich über **CALL WORKER** aufrufen. Da sein Name jedoch je nach der 4D Programmiersprache variieren kann, empfehlen wir, diesen mit **CALL WORKER** über seine Nummer (immer 1) aufzurufen.

Der Worker Prozess erscheint in der Prozessliste des Runtime Explorer und wird vom Befehl **PROCESS PROPERTIES** zurückgegeben, wenn er auf diesen Prozess angewendet wird.

In *Methode* übergeben Sie den Namen der Projektmethode, die im Kontext des Worker Prozesses ausgeführt werden soll. Der Parameter *Methode* kann ein leerer String sein. In diesem Fall führt der Worker die Methode aus, die ursprünglich zum Starten ihres Prozesses, sofern vorhanden, verwendet wurde (z.B. die Start Methode des Worker).

Hinweis: *Methode* muss immer angegeben werden, wenn der Befehl den Hauptprozess (Prozessnummer 1) aufruft, da er nicht über eine Projektmethode gestartet wurde. Deshalb führt ***CALL WORKER*** *(1;"")* nichts aus.

In *Param* können Sie einen oder mehrere Parameter für *Methode* übergeben. Das funktioniert genauso wie Parameter für eine Unterroutine übergeben, siehe Abschnitt **Parameter in Methoden** . Beim Starten der Ausführung im Rahmen des Prozesses empfängt die Prozessmethode die Parameterwerte in *$1*, *$2*, etc. Beachten Sie auch, dass sich Arrays nicht als Parameter für eine Methode übergeben lassen. Außerdem müssen Sie für den Befehl **CALL WORKER** folgendes beachten:

- Zeiger auf Tabellen oder Felder sind erlaubt.
- Zeiger auf Variablen, insbesondere lokale und Prozessvariablen werden nicht empfohlen, da sie zum Zeitpunkt, wo die Prozessmethode auf sie zugreift, undefiniert sein können.
- Übergeben Sie einen Parameter vom Typ Objekt oder Collection, erstellt 4D im Zielprozess eine Kopie des Objekts oder der Collection, wenn der Worker nicht in dem Prozess liegt, den der Befehl **CALL WORKER** aufruft.

Ein Worker Prozess bleibt am Leben, bis die Anwendung geschlossen oder explizit der Befehl **KILL WORKER** aufgerufen wird. Rufen Sie diesen Befehl auf, wenn ein Worker Prozess nicht mehr benötigt wird, denn damit sparen Sie Speicherplatz.

Beispiel

Eine Schaltfläche im Formular löst eine Berechnung aus, z.B. Statistiken zum ausgewählten Jahr. Die Schaltfläche erstellt oder ruft einen Worker Prozess auf, der Daten berechnet, während der Benutzer im Formular weiterarbeiten kann.

Die Methode für die Schaltfläche lautet:

```
 //Den Worker vWorkerName mit dem Parameter
C_LONGINT(vYear)aufrufen
vYear:=2015  // Mögliche Benutzerauswahl im Formular
CALL WORKER("myWorker";"workerMethod";vYear;Current form window)
```

Der Code *von workerMethod* ist:

```
 // Das ist die Methode des Worker
 // Sie kann preemptive oder kooperativ sein
C_LONGINT($1) //erhält das Jahr
C_LONGINT($2) //erhält die Referenz des Fensters
C_OBJECT(vStatResults) //zum Speichern von Ergebnissen von Statistiken
... //Statistik berechnen
 //nach Beenden das Formular mit berechneten Werten zurückrufen
 //vStatResults kann Ergebnisse im Formular anzeigen
CALL FORM($2;"displayStats";vStatResults)
```

CLEAR SEMAPHORE

CLEAR SEMAPHORE (Semaphorenname)

Parameter	Typ		Beschreibung
Semaphorenname	String	➡	Zu löschende Semaphore

Beschreibung

CLEAR SEMAPHORE löscht die von der Funktion **Semaphore** gesetzte Semaphore *Semaphorenname*.

Vergessen Sie nicht, die Semaphore nach der Ausführung wieder zu löschen, damit andere Prozesse nicht behindert werden. Nicht gelöschte Semaphoren bleiben im Speicher erhalten, bis der sie erstellende Prozess endet. Ein Prozess kann nur Semaphoren löschen, die er erstellt hat. Versuchen Sie eine Semaphore in einem Prozess zu löschen, der diese nicht erstellt hat, wird nichts ausgeführt.

Wichtig: Beachten Sie, dass 4D zwischen Groß- und Kleinbuchstaben unterscheidet. So werden z.B. *MeineSemaphore* und *meinesemaphore* unterschiedlich gewertet.

Beispiel

Siehe Beispiel bei der Funktion **Semaphore**.

GET PROCESS VARIABLE

GET PROCESS VARIABLE (Prozess ; QuellVar ; ZielVar {; QuellVar2 ; ZielVar2 ; ... ; QuellVarN ; ZielVarN})

Parameter	Typ		Beschreibung
Prozess	Lange Ganzzahl	➡	Quellprozessnummer
QuellVar	Variable	➡	Quellvariable
ZielVar	Variable	⬅	Zielvariable

Beschreibung

Der Befehl **GET PROCESS VARIABLE** liest die Prozessvariablen *QuellVar* (srvVar2, etc.) aus dem Quellprozess mit der in *Prozess* übergebenen Nummer und gibt deren aktuelle Werte in den Variablen *ZielVar* (dstVar2, etc.) des aktuellen Prozesses zurück.

Jede Quellvariable kann eine Variable, ein Array oder ein Array-Element sein. Beachten Sie jedoch die unten aufgelisteten Einschränkungen.

In jedem Paar *QuellVar;ZielVar* müssen beide Variablen zueinander kompatibel sein, da sonst die erhaltenen Werte evtl. ohne Bedeutung sind.

Der aktuelle Prozess "überfliegt" die Variablen aus dem Quellprozess — dieser Prozess erhält keine Warnung, dass ein anderer Prozess die Instanz seiner Variablen liest.

4D Server: Mit 4D Client können Sie Variablen in einem Zielprozess lesen, der auf dem Server-Rechner ausgeführt wird (Serverprozedur). Versehen Sie dazu die im Parameter *Prozess* angegebene Nummer mit einem Minuszeichen.

Wichtig: Die Prozesskommunikation zwischen mehreren Rechnern über die Befehle **GET PROCESS VARIABLE**, **SET PROCESS VARIABLE** und **VARIABLE TO VARIABLE** ist nur vom Client zum Server möglich. Es ist immer ein Client-Prozess, der die Variablen einer Serverprozedur liest oder schreibt.

Tipp: Kennen Sie nicht die Kennummer der Serverprozedur, können Sie auch die Interprozessvariablen des Servers verwenden. Dazu können Sie jeden negativen Wert in *Prozess* eingeben. Sie müssen also nicht unbedingt die Kennummer des Prozesses wissen, um **GET PROCESS VARIABLE** mit den Interprozessvariablen des Servers einzusetzen. Das ist von Vorteil, wenn eine Serverprozedur mit der **Datenbankmethode On Server Startup** gestartet wird. Da Client-Rechner nicht automatisch die Kennummer dieses Prozesses wissen, können Sie im Parameter *Prozess* einen beliebigen negativen Wert übergeben.

Einschränkungen

GET PROCESS VARIABLE erlaubt keine lokalen Variablen als Quellvariablen.

Zielvariablen können dagegen Interprozess-, Prozess- oder lokale Variablen sein. Sie "empfangen" die Werte nur in Variablen, nicht in Feldern.

GET PROCESS VARIABLE erlaubt jede Art von Quellprozess oder Interprozessvariable. Davon ausgenommen sind:

- Zeiger
- Arrays von Zeigern
- Zweidimensionale Arrays

Der Quellprozess muss ein Benutzerprozess sein; er kann kein Kernelprozess sein. Gibt es keinen Quellprozess, hat der Befehl keine Auswirkung.

Hinweis: Im interpretierten Modus wird ein undefinierter Wert zurückgegeben, wenn es keine Quellvariable gibt. Das können Sie mit der Funktion **Type** herausfinden. Damit testen Sie die dazugehörige Zielvariable.

Beispiel 1

Diese Codezeile liest den Wert der Textvariablen *vtCurStatus* aus dem Prozess mit der Prozessnummer *$vlProcess*. Sie gibt den Wert in der Prozessvariablen *vtInfo* des aktuellen Prozesses zurück:

```
GET PROCESS VARIABLE($vlProcess;vtCurStatus;vtInfo)
```

Beispiel 2

Diese Codezeile führt dasselbe aus, gibt den Wert jedoch in der lokalen Variablen *$vtInfo* für die Methode zurück, die den aktuellen Prozess ausführt:

```
GET PROCESS VARIABLE($vlProcess;vtCurStatus;$vtInfo)
```

Beispiel 3

Diese Codezeile führt dasselbe aus, gibt den Wert jedoch in der Variablen *vtCurStatus* des aktuellen Prozesses zurück:

```
GET PROCESS VARIABLE($vlProcess;vtCurStatus;vtCurStatus)
```

Hinweis: Das erste *vtCurStatus* bezeichnet die Instanz der Variablen im Quellprozess. Das zweite *vtCurStatus* bezeichnet die Instanz der Variablen im aktuellen Prozess.

Beispiel 4

Dieses Beispiel liest sequentiell die Elemente eines Prozess-Arrays aus dem in *$vlProcess* angegebenen Prozess:

```
GET PROCESS VARIABLE($vlProcess;vl_IPCom_Array;$vlSize)
For($vlElem;1;$vlSize)
   GET PROCESS VARIABLE($vlProcess;at_IPCom_Array{$vlElem};$vtElem)
  ` Führe etwas aus mit $vtElem
End for
```

Hinweis: In diesem Beispiel enthält die Prozessvariable *vl_IPCom_Array* die Größe des Array *at_IPCom_Array* und muss vom Quellprozess aufrechterhalten werden.

Beispiel 5

Dieses Beispiel führt dasselbe wie das vorige aus, liest das Array jedoch als Ganzes anstatt sequentiell die Elemente:

```
GET PROCESS VARIABLE($vlProcess;at_IPCom_Array;$anArray)
For($vlElem;1;Size of array($anArray))
  ` Führe dasselbe aus mit $anArray{$vlElem}
End for
```

Beispiel 6

Dieses Beispiel liest die Instanzen der Variablen *v1*,*v2*,*v3* des Quellprozesses und gibt ihre Werte in den Instanzen derselben Variablen für den aktuellen Prozess zurück:

```
GET PROCESS VARIABLE($vlProcess;v1;v1;v2;v2;v3;v3)
```

Beispiel 7

Siehe Beispiel zum Befehl **DRAG AND DROP PROPERTIES**.

KILL WORKER

KILL WORKER {(Prozess)}

Parameter	Typ		Beschreibung
Prozess	Text, Lange Ganzzahl	➡	Nummer oder Name des zu stoppenden Prozesses (stoppt ohne Angabe den aktuellen Prozess)

Beschreibung

Der Befehl **KILL WORKER** sendet eine Nachricht an den Worker Prozess, mit dem in *Prozess* übergebenen Namen oder der Nummer, mit der Aufforderung, noch wartende Nachrichten zu ignorieren und die Ausführung nach Abschließen der aktuellen Aufgabe zu stoppen.

Dieser Befehl ist nur mit Worker Prozessen anwendbar. Weitere Informationen dazu finden Sie im Abschnitt **Über Worker**.

In *Prozess* übergeben Sie Name oder Nummer des Worker Prozesses, dessen Ausführung gestoppt werden soll. Gibt es keinen Worker mit dem angegebenen Prozessnamen oder der Prozessnummer, führt **KILL WORKER** nichts aus.
KILL WORKER ohne Parameter wird auf den aktuell laufenden Worker angewendet und entspricht deshalb **KILL WORKER** (aktueller Prozess).

Wird der Befehl in einem Worker aufgerufen, der nicht explizit mit dem Befehl **CALL WORKER** erstellt wurde (z.B. im Worker der Hauptanwendung), fordert er den Worker nur auf, seine Nachrichtenbox zu leeren.

Wird der Befehl **CALL WORKER** aufgerufen, um eine Meldung an einen Worker zu senden, der gerade von **KILL WORKER** gestoppt wurde, wird ein neuer Prozess gestartet. Um sicherzustellen, dass nur ein Prozess gleichzeitig für einen Worker läuft, startet der neue Prozess, nachdem der vorige beendet ist. Wird **CALL WORKER** dagegen von einem Worker aufgerufen, um sich selbst eine Meldung zu senden und wurde er gerade durch **KILL WORKER** gestoppt, führt der Befehl nichts aus.

Beispiel

Der folgende Code (z.B. über ein Formular ausgeführt) löst Beenden des Worker aus:

```
CALL WORKER(vWorkerName;"theWorker";"end")
```

Dazu wird in der Worker Methode (*theWorker*) folgender Code hinzugefügt:

```
 //Methode theWorker
C_TEXT($1) //param

Case of
   :($1="call") //der Worker wird aufgerufen
      ... //etwas ausführen
   :($1="end") //der Worker erhält die Aufforderung, zu stoppen
      KILL WORKER
End case
```

Semaphore

Semaphore (Semaphorenname {; ZähleTick}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Semaphorenname	String	➡	Zu testende und zu setzende Semaphore
ZähleTick	Lange Ganzzahl	➡	Max. Wartezeit
Funktionsergebnis	Boolean	⮌	Semaphore wurde erfolgreich gesetzt (FALSE) oder Semaphore ist bereits gesetzt (TRUE)

Beschreibung

Eine Semaphore ist ein Flag, das auf Arbeitsstationen oder in Prozessen auf einer Arbeitsstation bekannt ist. Eine Semaphore ist entweder vorhanden oder nicht vorhanden. Die Methoden, die ein Benutzer einsetzt, können testen, ob eine Semaphore vorhanden ist. Eine Semaphore lässt sich nur von der Arbeitsstation oder dem Prozess entfernen, welche(r) sie erstellt hat. Methoden können durch Erstellen und Testen von Semaphoren mit den Arbeitsstationen kommunizieren.
Sie verwenden Semaphoren nicht, um den Zugriff auf Datensätze zu verwalten. Das erledigen 4D und 4D Server automatisch. Sie setzen Semaphoren ein, damit nicht mehrere Benutzer gleichzeitig dieselbe Operation ausführen können.

Die Funktion **Semaphore** gibt TRUE zurück und führt nichts aus, wenn *Semaphorenname* vorhanden ist. Ist *Semaphorenname* nicht vorhanden, legt **Semaphore** die Semaphore an und gibt FALSE zurück. Eine Semaphore kann immer nur ein Benutzer gleichzeitig erstellen. Wird FALSE zurückgegeben, heißt das einerseits, dass die Semaphore nicht vorhanden war, andererseits aber auch, dass die Semaphore bereits für den Prozess gesetzt wurde, der sie aufgerufen hat.
Semaphore gibt FALSE zurück, wenn keine Semaphore gesetzt wurde bzw. wenn die Semaphore bereits im selben Prozess existiert, in dem der Aufruf erfolgt.
Der Name für eine Semaphore ist auf 255 Zeichen begrenzt, inkl. dem vorangestellten Zeichen $. Längere Namen werden abgeschnitten. Beachten Sie, dass Groß- und Kleinbuchschreibung berücksichtigt wird. Das Programm unterscheidet also zwischen *MeineSemaphore* und *meinesemaphore*.

Mit dem optionalen Parameter *ZähleTick* geben Sie eine Wartezeit in Ticks an, wenn *Semaphorenname* bereits gesetzt wurde. In diesem Fall wartet die Funktion entweder, bis die Semaphore freigegeben oder die Wartezeit abgelaufen ist und gibt erst dann *True* zurück.

Es gibt zwei Arten von Semaphoren:

- **Lokale Semaphoren** werden nur von den Prozessen auf derselben Arbeitsstation und nur von dieser erkannt. Sie werden mit einem vorangestellten Dollarzeichen gekennzeichnet, z.B. $MeineSemaphore. Mit lokalen Semaphoren steuern Sie Operationen zwischen Prozessen, die auf einer Arbeitsstation ausgeführt werden. Eine lokale Semaphore kann z.B. den Zugriff auf ein Interprozess-Array steuern, das sich alle Prozesse einer Arbeitsstation oder im 4D Einzelplatzbetrieb teilen.
- **Globale Semaphoren** sind für alle Benutzer und für alle Prozesse zugänglich. Mit globalen Semaphoren verwalten Sie Operationen zwischen Benutzern einer Datenbank im Mehrplatzbetrieb.

Globale und lokale Semaphoren funktionieren nach derselben Logik. Der Unterschied liegt in ihrer Reichweite. Im Client/Server Modus werden globale Semaphoren von allen Prozessen auf allen Clients und Servern gemeinsam genutzt. Eine lokale Semaphore wird nur von Prozessen auf dem Rechner genutzt, wo sie erzeugt wurde.
In 4D haben globale und lokale Semaphoren dieselbe Reichweite, da es nur einen Benutzer gibt. Wird die Datenbank jedoch in beiden Setups verwendet, stellen Sie sicher, dass - je nachdem, was ausgeführt werden soll - globale oder lokale Semaphoren verwendet werden.

Hinweis: Wir empfehlen, zum Verwalten lokaler Aspekte für den Client einer Anwendung, wie z.B. die Oberfläche oder ein Array mit Interprozessvariablen, lokale Semaphoren zu verwenden. Verwenden Sie dafür globale Semaphoren, verursacht das nicht nur unnötigen Austausch über das Netzwerk, sondern kann auch andere Client-Rechner beeinträchtigen. Über eine lokale Semaphore können Sie solche unerwünschten Nebenwirkungen vermeiden.

Beispiel 1

Hier eine typische Anweisung zum Verwenden einer Semaphore:

```
While(Semaphore("MySemaphore";300))
   IDLE
End while
 // hier durch Semaphore geschützten Code setzen
CLEAR SEMAPHORE("MySemaphore")
```

Beispiel 2

In diesem Beispiel wollen Sie verhindern, dass zwei Benutzer gleichzeitig die Preise in einer Produktetabelle global aktualisieren können. Schreiben Sie dazu folgende Methode:

```
If(Semaphore("UpdatePrices"))  ` Versuche die Semaphore zu erstellen
   ALERT("Ein anderer Benutzer aktualisiert bereits die Preise global. Versuchen Sie es später wieder.")
Else
   DoUpdatePrices  ` Aktualisiere alle Preise
   CLEAR SEMAPHORE("UpdatePrices"))  ` Lösche die Semaphore
End if
```

Beispiel 3

Folgendes Beispiel verwendet eine lokale Semaphore. Sie wollen in einer Datenbank mit mehreren Prozessen eine To-do Liste aufrechterhalten. Sie soll in einem Interprozess-Array und nicht in einer Tabelle verwaltet werden. Über eine Semaphore verhindern Sie den gleichzeitigen Zugriff. Für diesen Fall benötigen Sie nur eine lokale Semaphore, da nur ein Arbeitsplatz mit der To-Do Liste arbeitet.

Das Interprozess-Array wird in der Startup Methode initialisiert:

```
ARRAY TEXT(◇ToDoList;0)  ` Die To-Do Liste ist zuerst leer
```

Mit dieser Methode fügen Sie Einträge in die To-do Liste ein:

```
  ` Projektmethode ADD TO DO LIST
  ` ADD TO DO LIST ( Text )
  ` ADD TO DO LIST ( Eintrag in To-Do Liste )
C_TEXT($1)
If(Not(Semaphore("$AccessToDoList";300)))
  ` Warte 5 Sekunden, wenn Semaphore bereits existiert
   $vlElem:=Size of array(◇ToDoList)+1
   INSERT IN ARRAY(◇ToDoList;$vlElem)
   ◇ToDoList{$vlElem}:=$1
   CLEAR SEMAPHORE("$AccessToDoList")  ` Lösche die Semaphore
End if
```

Obige Methode können Sie von jedem Prozess aus aufrufen.

Beispiel 4

Diese Methode ermöglicht, eine Methode bei vorhandener Semaphore nicht auszuführen; die Methode informiert die aufrufende Methode mit Fehlernummer und Text.

Syntax:

```
$L_Error:=Semaphore_proof(->$T_Text_error)
```

```
  // Schützende Struktur über Semaphoren
C_LONGINT($0)
C_POINTER($1)  // Fehlermeldung

  // Methode starten
C_LONGINT($L_MyError)
$L_MyError:=1

C_TEXT($T_Sema_local)
$T_Sema_local:="$tictac"

If(Semaphore($T_Sema_local;300))
  // Wir haben 300 Ticks gewartet, ohne dass die Semaphore
  // von dem, der sie gesetzt hat, freigegeben wurde:
  // wir beenden hier
   $L_MyError:=-1

Else

  // Diese Methode startet nur von einem Prozess gleichzeitig

  // Wir haben die Semaphore, wie eingegeben, gesetzt
  // so sind wir die einzigen, die sie entfernen können

  // Etwas ausführen
   ...
  // Dann durch Entfernen der Semaphore beenden
   CLEAR SEMAPHORE($T_Sema_local)
End if

C_TEXT($T_Message)
If($L_MyError=-1)
   $T_Message:="Die Semaphore "+$T_Sema_local+" hat den Zugriff auf den restlichen Code blockiert"
Else
   $T_Message:="OK"
End if
```

```
$0:=$L_MyError
$1->:=$T_Message   // Die aufrufende Methode empfängt Fehlernummer und Erläuterung in einfachem Text.
```

SET PROCESS VARIABLE

SET PROCESS VARIABLE (Prozess ; ZielVar ; Ausdr {; ZielVar2 ; Ausdr2 ; ... ; ZielVarN ; AusdrN})

Parameter	Typ		Beschreibung
Prozess	Lange Ganzzahl	➡	Zielprozessnummer
ZielVar	Variable	➡	Zielvariable
Ausdr	Variable	➡	Quellausdruck (oder Quellvariable)

Beschreibung

Der Befehl **SET PROCESS VARIABLE** schreibt die Prozessvariablen *ZielVar* (dstVar2, etc.) des Zielprozesses mit der in Prozess übergebenen Nummer mit den in *Ausdr1* (expr2, etc.) übergebenen Werten.

Jede Zielvariable kann eine Variable oder ein Array-Element sein. Beachten Sie hierzu jedoch die unten aufgelisteten Einschränkungen.

In jedem Paar *QuellVar;ZielVar* müssen beide Variablen zueinander kompatibel sein, da sonst die erhaltenen Werte evtl. ohne Bedeutung sind. Gibt es keine Zielvariable im interpretierten Modus, wird sie erstellt und mit dem Ausdruck zugewiesen.

Der aktuelle Prozess "überfliegt" die Variablen aus dem Zielprozess — dieser Prozess erhält keine Warnung, dass ein anderer Prozess die Instanz seiner Variablen liest.

4D Server: Mit 4D Client können Sie Variablen in einen Zielprozess schreiben, der auf dem Server-Rechner ausgeführt wird (Serverprozedur). Versehen Sie dazu die im Parameter *Prozess* angegebene Nummer mit einem Minuszeichen.

Wichtig: Die Prozesskommunikation zwischen mehreren Rechnern über die Befehle **SET PROCESS VARIABLE**, **SET PROCESS VARIABLE** und **VARIABLE TO VARIABLE** ist nur vom Client zum Server möglich. Es ist immer ein Client-Prozess, der die Variablen einer Serverprozedur liest oder schreibt.

Tipp: Kennen Sie nicht die Kennummer der Serverprozedur, können Sie auch die Interprozessvariablen des Servers verwenden. Dazu können Sie jeden negativen Wert in *Prozess* eingeben. Sie müssen also nicht unbedingt die Kennummer des Prozesses wissen, um **SET PROCESS VARIABLE** mit den Interprozessvariablen des Servers einzusetzen. Das ist von Vorteil, wenn eine Serverprozedur mit der **Datenbankmethode On Server Startup** gestartet wird. Da Client-Rechner nicht automatisch die Kennummer dieses Prozesses wissen, können Sie im Parameter *Prozess* einen beliebigen negativen Wert übergeben.

Einschränkungen

SET PROCESS VARIABLE erlaubt keine lokalen Variablen als Zielvariablen.

SET PROCESS VARIABLE erlaubt jede Art von Zielprozess oder Interprozessvariablen. Davon ausgenommen sind:

- Zeiger
- Arrays jeglichen Typs. Wollen Sie ein Array als Ganzes von einem Prozess in einen anderen schreiben, verwenden Sie den Befehl **VARIABLE TO VARIABLE**. Beachten Sie jedoch, dass Sie mit **SET PROCESS VARIABLE** das Element eines Array schreiben können.
- Sie können nicht das Element eines Array von Zeigern oder eines Elements eines zweidimensionalen Array schreiben.

Der Zielprozess muss ein Benutzerprozess sein, er kann kein Kernel-Prozess sein. Gibt es keinen Zielprozess, wird ein Fehler erzeugt. Diesen Fehler können Sie mit einer Fehlerverwaltungsmethode abfangen, die mit **ON ERR CALL** installiert wurde.

Beispiel 1

Diese Codezeile setzt (zum leeren String) die Textvariable *vtCurStatus* des Prozesses mit der in *$vlProcess* übergebenen Nummer:

```
SET PROCESS VARIABLE($vlProcess;vtCurStatus;"")
```

Beispiel 2

Diese Codezeile setzt die Textvariable *vtCurStatus* des Prozesses mit der in *$vlProcess* übergebenen Nummer auf den Wert der Variablen *$vtInfo* aus der ausführenden Methode im aktuellen Prozess:

```
SET PROCESS VARIABLE($vlProcess;vtCurStatus;$vtInfo)
```

Beispiel 3

Diese Codezeile setzt die Textvariable *vtCurStatus* des Prozesses mit der in *$vlProcess* übergebenen Nummer auf den Wert derselben Variablen im aktuellen Prozess:

```
SET PROCESS VARIABLE($vlProcess;vtCurStatus;vtCurStatus)
```

Hinweis: Das erste *vtCurStatus* bezeichnet die Instanz der Variablen im Zielprozess. Das zweite *vtCurStatus* bezeichnet die Instanz der Variablen im aktuellen Prozess.

Beispiel 4

Dieses Beispiel setzt alle Elemente eines Prozessarray aus dem mit *$vlProcess* angegebenen Prozess sequentiell in Großbuchstaben:

```
GET PROCESS VARIABLE($vlProcess;vl_IPCom_Array;$vlSize)
For($vlElem;1;$vlSize)
   GET PROCESS VARIABLE($vlProcess;at_IPCom_Array{$vlElem};$vtElem)
   SET PROCESS VARIABLE($vlProcess;at_IPCom_Array{$vlElem};Uppercase($vtElem))
End for
```

Hinweis: In diesem Beispiel enthält die Prozessvariable *vl_IPCom_Array* die Größe des Array *at_IPCom_Array* und muss vom Quell-/Zielprozess aufrechterhalten werden.

Beispiel 5

Dieses Beispiel schreibt die Instanz der Variablen *v1*, *v2* und *v3* des Zielprozesses mit der Instanz derselben Variablen aus dem aktuellen Prozess:

```
SET PROCESS VARIABLE($vlProcess;v1;v1;v2;v2;v3;v3)
```

Test semaphore

Test semaphore (Semaphorenname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Semaphorenname	String	➡	Name der zu testenden Semaphore
Funktionsergebnis	Boolean	⮌	True = Semaphore vorhanden, False = Semaphore nicht vorhanden

Beschreibung

Die Funktion **Test semaphore** testet, ob eine Semaphore vorhanden ist.

Die Funktion **Semaphore** und **Test semaphore** unterscheiden sich darin, dass **Test semaphore** keine Semaphore erstellt, wenn sie nicht vorhanden ist. Ist die Semaphore vorhanden, gibt die Funktion *True* zurück, sonst *False*.

Wichtig: Beachten Sie, dass 4D zwischen Groß- und Kleinbuchstaben unterscheidet. So werden z.B. *MeineSemaphore* und *meinesemaphore* unterschiedlich gewertet.

Beispiel

Mit folgendem Beispiel erfahren Sie den Status eines Prozesses (hier beim Ändern des Code), ohne *Semaphorenname* zu ändern:

```
$Win:=Open window(x1;x2;y1;y2;-Palette window)
Repeat
   If(Test semaphore("Verschlüsselter Code"))
      POSITION MESSAGE($x3;$y3)
      MESSAGE("Verschlüsselten Code ändern.")
   Else
      POSITION MESSAGE($x3;$y3)
      MESSAGE("Änderung des verschlüsselten Code ist erlaubt.")
   End if
Until(StopInfo)
CLOSE WINDOW
```

VARIABLE TO VARIABLE

VARIABLE TO VARIABLE (Prozess ; ZielVar ; QuellVar {; ZielVar2 ; QuellVar2 ; ... ; ZielVarN ; QuellVarN})

Parameter	Typ		Beschreibung
Prozess	Lange Ganzzahl	➡	Zielprozessnummer
ZielVar	Variable	➡	Zielvariable
QuellVar	Variable	➡	Quellvariable

Beschreibung

Der Befehl **VARIABLE TO VARIABLE** schreibt die Prozessvariablen *ZielVar* (*ZielVar2*, etc.) des Zielprozesses mit der in *Prozess* übergebenen Nummer mit den Werten der Variablen *QuellVar1 QuellVar2*, etc.

VARIABLE TO VARIABLE führt dieselbe Aktion wie **SET PROCESS VARIABLE** aus bis auf folgende Unterschiede:

- In **SET PROCESS VARIABLE** übergeben Sie Quellausdrücke, und können deshalb ein Array nicht als Ganzes übergeben. In **VARIABLE TO VARIABLE** müssen Sie ausdrücklich Quellvariablen übergeben und können deshalb ein Array als Ganzes übergeben.
- Jede Zielvariable von **SET PROCESS VARIABLE** kann eine Variable oder ein Array-Element sein, jedoch nicht ein Array als Ganzes. Jede Zielvariable von **VARIABLE TO VARIABLE** kann eine Variable oder ein Array oder ein Array-Element sein.

In jedem Paar *ZielVar;QuellVar* müssen beide Variablen zueinander kompatibel sein, da sonst die erhaltenen Werte u.U. bedeutungslos sind. Gibt es im interpretierten Modus keine Zielvariable, wird sie erstellt und mit Typ und Wert der Quellvariablen zugewiesen

Der aktuelle Prozess "überfliegt" die Variablen aus dem Quellprozess – dieser Prozess erhält keine Warnung, dass ein anderer Prozess die Instanz seiner Variablen liest.

4D Server: Die Prozesskommunikation zwischen mehreren Rechnern über die Befehle **VARIABLE TO VARIABLE**, **SET PROCESS VARIABLE** und **VARIABLE TO VARIABLE** ist nur vom Client zum Server möglich. Es ist immer ein Client-Prozess, der die Variablen einer Serverprozedur liest oder schreibt.

Einschränkungen

VARIABLE TO VARIABLE erlaubt keine lokalen Variablen als Zielvariablen.

VARIABLE TO VARIABLE erlaubt für die Zielvariable jede Art von Prozess- oder Interprozessvariable. Davon ausgenommen sind:

- Zeiger
- Arrays von Zeigern
- Zweidimensionale Arrays

Der Zielprozess muss ein Benutzerprozess sein; er kann kein Kernelprozess sein. Gibt es keinen Zielprozess, wird ein Fehler generiert. Sie können ihn mit einer Fehlerverwaltungsmethode ausfindig machen, die mit **ON ERR CALL** installiert wurde.

Beispiel

Folgendes Beispiel liest ein Prozess-Array aus dem mit *$vlProcess* angegebenen Prozess, setzt die Elemente sequentiell in Großbuchstaben und schreibt das Array dann als Ganzes zurück:

```
GET PROCESS VARIABLE($vlProcess;at_IPCom_Array;$anArray)
For($vlElem;1;Size of array($anArray))
   $anArray{$vlElem}:=Uppercase($anArray{$vlElem})
End for
VARIABLE TO VARIABLE($vlProcess;at_IPCom_Array;$anArray)
```

Prozesse

- Einführung in Prozesse
- Preemptive 4D Prozesse
- 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

Einführung in Prozesse

Multitasking in 4D bedeutet, dass unterschiedliche Operationen der Datenbank gleichzeitig ausgeführt werden können. Diese Operationen werden Prozesse genannt.

Multiple Prozesse sind wie mehrere Benutzer am gleichen Rechner, wobei jeder seine eigene Aufgabe ausführt. Das bedeutet, dass sich jede Methode als ein anderes Task der Datenbank ausführen lässt.

Dieses Kapitel behandelt folgende Themen:

- Prozesse anlegen und löschen
- Elemente eines Prozesses
- Benutzerprozesse
- Von 4D erstellte Prozesse
- Lokale und globale Prozesse
- Record locking zwischen Prozessen

Hinweis: Dieses Kapitel beschreibt nicht die Serverprozeduren. Informationen dazu finden Sie im Handbuch *4D Server* im Abschnitt **Serverprozeduren**.

Prozesse anlegen und löschen

Es gibt verschiedene Wege, einen neuen Prozess zu erstellen:

- Sie führen in der Designumgebung eine Methode aus. Markieren Sie dazu im Dialogfenster **Methode ausführen** in der DropDown-Liste **Neuer Prozess**. Die gewählte Methode ist die Prozessmethode.
- Sie starten über eine Menüzeile einen Prozess. Wählen Sie dazu im **Menüleisteneditor** den Menübefehl und markieren das Kontrollkästchen **Starte Neuen Prozeß**. Die diesem Menübefehl zugeordnetete Methode ist die Prozessmethode.
- Verwenden Sie die Funktion **New process**. Die als Parameter übergebene Methode ist die Prozessmethode.
- Verwenden Sie die Funktion **Execute on server**, um einen Serverprozess auf dem Server zu erstellen. Die Methode, die als ein Parameter der Funktion übergeben wird, ist die Prozessmethode.
- Verwenden Sie den Befehl **CALL WORKER**. Existiert der Worker Prozess noch nicht, wird er angelegt.

Es gibt folgende Arten, einen Prozess zu löschen:

- Automatisch nach Beendigung der Methode
- Beim Schließen der Datenbank
- Durch einen Klick auf die Schaltflächen **Stop** oder **Bearbeiten** im Fenster für den Schrittmodus, wenn Sie sich in einer Hauptmethode befinden.
- Durch Markieren des Befehls **Stop** im Runtime-Explorer
- Durch Aufrufen des Befehls **KILL WORKER** (nur zum Löschen eines Worker Prozesses).

Ein Prozess kann einen anderen Prozess erstellen. Prozesse sind nicht hierarchisch organisiert — alle Prozesse sind gleich, unabhängig von welchem Prozess aus sie erstellt wurden. Hat ein Hauptprozess einen Unterprozess erstellt, kann der Unterprozess weiterlaufen, auch wenn der Hauptprozess nicht mehr ausgeführt wird.

Prozesselemente

Jeder Prozess enthält folgende Elementarten:

- **Elemente für die Bildschirmoberfläche**: Sie sind notwendig, um einen Prozess anzuzeigen
- **Elemente für die Daten**: Sie informieren über die Daten in der Datenbank.
- **Elemente für die Programmiersprache**: Sie werden in der Programmierung verwendet bzw. sind wichtig zum Entwickeln Ihrer eigenen Anwendung.

Elemente für die Bildschirmoberfläche

- **Menüleiste**: Jeder Prozess hat seine eigene Menüleiste. Sie wird angezeigt, wenn sich das Prozessfenster vorn befindet.
- **Fenster**: Jeder Prozess kann ein oder mehrere gleichzeitig geöffnete Fenster haben. Es gibt aber auch Prozesse ohne Fenster.
- **Aktives (vorderstes) Fenster**: Das Prozessfenster, das ganz vorn liegt, ist das aktive Fenster für diesen Prozess. Sie haben so viele aktive Fenster, wie geöffnete Prozesse mit eigenem Fenstern vorhanden sind.

Hinweise:

- Prozesse enthalten standardmäßig keine Menüleisten, d.h. die Tastenkürzel des Standardmenüs **Bearbeiten**, insbesondere Ausschneiden/Kopieren/Einfügen, sind im Prozessfenster nicht verfügbar. Damit Benutzer beim Aufrufen von Dialogfenstern oder 4D Editoren (Formulareditor, Sucheditor, **Request**, etc.) über einen Prozess solche Tastenkürzel nutzen können, müssen Sie sicherstellen, dass im Prozess eine Entsprechung des Menüs **Bearbeiten** installiert ist.
- Auf dem Server ausgeführte Prozesse (Serverprozeduren) dürfen keine Elemente der Oberfläche enthalten.

Elemente für die Daten

- **Eine aktuelle Auswahl pro Tabelle**: Jeder Prozess kann eine aktuelle Auswahl für jede Tabelle haben. Arbeiten Sie mit mehreren Prozessen gleichzeitig, können Sie auch mehrere aktuelle Auswahlen für eine Tabelle haben

- **Ein aktueller Datensatz pro Tabelle**: Ein Prozess kann einen aktuellen Datensatz für jede Tabelle haben. Arbeiten Sie mit mehreren Prozessen gleichzeitig, haben Sie auch mehrere aktuelle Datensätze für eine Tabelle.

Hinweis: Diese Beschreibung gilt für Datenelemente mit globalen Prozessen. Weitere Informationen dazu finden Sie im später aufgeführten Abschnitt **Globale und lokale Prozesse**.

Elemente für die Programmiersprache

- **Variablen**: Jeder Prozess hat seine eigenen Prozessvariablen. Weitere Informationen dazu finden Sie im Abschnitt **Variablen**. Prozessvariablen werden nur innerhalb des Prozesses erkannt.
- **Haupttabelle**: Jeder Prozess hat seine eigene Haupttabelle. Beachten Sie jedoch, dass der Befehl **DEFAULT TABLE** nur eine Sprachkonvention zum Programmieren ist.
- **Eingabe- und Ausgabeformulare**: Jeder Prozess hat für jede Tabelle ein aktuelles Ein- und Ausgabeformular.
- **Prozessmengen**: Jeder Prozess hat sowohl seine eigenen Prozessmengen. *LockedSet* ist eine Prozessmenge. Die Mengen werden nach Beendigung des Prozesses, der sie erzeugt hat, gelöscht. Weitere Informationen dazu finden Sie im Kapitel **Mengen**.
- **Fehlerbehandlung**: In jedem Prozess können Sie den Befehl **ON ERR CALL** aufrufen. Jeder Prozess hat seine eigene Fehlerbehandlung.
- **Debugger Fenster**: Jeder Prozess hat sein eigenes Fenster für den Schrittmodus.

Benutzerprozesse

Sie erstellen Benutzerprozesse, um bestimmte Tasks auszuführen. Diese teilen sich die Prozesszeit mit den Kernel-Prozessen. Web-Verbindungsprozesse sind zum Beispiel Benutzerprozesse.

Das 4D Programm erstellt auch Prozesse für seine eigenen Zwecke. 4D erstellt und verwaltet folgende Prozesse:

- **Hauptprozess**: Dieser Prozess steuert die Anzeigefenster der Benutzeroberfläche.
- **Design Prozess**: Er verwaltet die Fenster und Editoren der Designumgebung. In einer kompilierten Datenbank fehlt dieser Prozess.
- **Web Server Prozess**: Dieser Prozess läuft, wenn die Datenbank im Web publiziert wird. Weitere Informationen dazu finden Sie im Abschnitt **Web Server konfigurieren und Verbindung verwalten**.
- **Cache Manager**: Er verwaltet die Übertragung der Daten aus dem Cache-Speicher von 4D auf die Festplatte. Die Daten werden asynchron auf die Festplatte geschrieben und beeinträchtigen nicht die Arbeit des Anwenders. Er wird angelegt, sobald 4D Developer oder 4D Server laufen.
- **Index Prozess**: Dieser Prozess verwaltet das Anlegen von Indizes. Das Anlegen der Indizes geschieht in einem eigenen Prozess. Er wird angelegt, wenn ein Index für ein Feld erzeugt bzw. gelöscht wird.
- **On Event Manager**: Beim Aufruf des Befehls **ON EVENT CALL** wird dieser lokale Prozess erzeugt. Er wird kontinuierlich ausgeführt, auch wenn keine Methode abläuft. Die Ereignisverwaltung geschieht auch in der Designumgebung.

Kooperative und preemptive Prozesse

Ab 4D v15 R5 64 bits können Sie in 4D im kompililierten Modus preemptive Benutzerprozesse ausführen. In bisherigen Versionen waren nur kooperative Benutzerprozesse verfügbar.

Beim *preemptive* Modus (unterbrechend) ist der Prozess einer CPU zugeordnet. Die Prozessverwaltung wird dann an das System delegiert, das jede CPU einem multi-core Rechner einzeln zuweisen kann. Beim *kooperative*n Modus (nicht-unterbrechend) werden alle Prozesse vom übergeordneten Applikation Thread verwaltet und nutzen dieselbe CPU gemeinsam, und das auch auf einem multi-core Rechner.

Durch den preemptive Modus wird die globale Performance der Applikation erhöht, insbesondere auf multi-core Rechnern, da mehrfache Prozesse (*Threads*) real simultan laufen können. Der jeweilige Gewinn hängt jedoch von den ausgeführten Operationen ab. Außerdem gelten für Methoden, die preemptiv nutzbar sein sollen, spezifische Einschränkungen. Denn im preemptive Modus ist jeder Thread von den anderen unabhängig und wird nicht direkt von der Applikation verwaltet. Zusätzlich ist die preemptive Ausführung nur in ganz bestimmten Kontexten verfügbar.

Weitere Informationen zu preemptive Prozessen finden Sie im Abschnitt **Preemptive 4D Prozesse**.

Globale und lokale Prozesse

Prozesse können sowohl global als auch lokal sein. Standardmäßig sind alle Prozesse global.

Globale Prozesse können jede Operation ausführen, inkl. auf Daten zugreifen und Daten steuern. In den meisten Fällen verwenden Sie globale Prozesse.

Lokale Prozesse eignen sich nur für Operationen, die nicht auf Daten zugreifen. Sie verwenden diese beispielsweise für eine Methode zur Ereignisverwaltung oder zum Steuern von Oberflächenelementen, wie z.B. das Palettenfenster.

Der Name des lokalen Prozesses muss mit einem Dollarzeichen beginnen (*$*).

Warnung: Versuchen Sie, von einem lokalen Prozess aus auf Daten zuzugreifen, geschieht das über den Hauptprozess. Das kann Konflikte mit den Operationen geben, die innerhalb dieses Prozesses ausgeführt werden.

4D Server: Setzen Sie auf der Arbeitsstation lokale Prozesse ein für Operationen ohne Zugriff auf Daten. So bleibt mehr Prozesszeit für Server-intensive Tasks übrig.

Record Locking zwischen Prozessen

Ein Datensatz ist gesperrt, wenn ein anderer Prozess ihn erfolgreich zum Bearbeiten geladen hat. Ein gesperrter Datensatz kann von einem anderen Prozess geladen, jedoch nicht geändert werden. Der Datensatz wird nur in dem Prozess freigegeben, in welchem er bearbeitet wurde. Ein Datensatz ist nur ungesperrt ladbar, wenn die entsprechende Tabelle im Lese/Schreibmodus ist. Weitere Informationen dazu finden Sie im Abschnitt **Überblick zu Datensatz sperren**.

Preemptive 4D Prozesse

4D Developer Edition 64-bit für Windows und OS X bietet ein leistungsstarkes neues Feature zum Schreiben und Verwenden von **preemptive 4D Code**. Damit können Ihre kompilierten 4D Anwendungen die Vorteile von multi-core Computern nutzen, um die Ausführung zu beschleunigen und mehr angemeldete Benutzer zu unterstützen.

Was ist ein preemptive Prozess?

Beim *preemptive* Modus (unterbrechend) ist der Prozess einer CPU zugeordnet. Die Prozessverwaltung wird dann an das System delegiert, das jede CPU einem multi-core Rechner einzeln zuweisen kann.

Beim *kooperative*n Modus (nicht-unterbrechend - in 4D bis v15 R4 der einzige verfügbare Modus) werden alle Prozesse vom übergeordneten Applikation Thread verwaltet und nutzen dieselbe CPU gemeinsam, und das auch auf einem multi-core Rechner.

Durch den preemptive Modus wird die globale Performance der Applikation erhöht, insbesondere auf multi-core Rechnern, da mehrfache Prozesse (*Threads*) real simultan laufen können. Der jeweilige Gewinn hängt jedoch von den ausgeführten Operationen ab.

Außerdem gelten für Methoden, die preemptiv nutzbar sein sollen, spezifische Einschränkungen. Denn im preemptive Modus ist jeder Thread von den anderen unabhängig und wird nicht direkt von der Applikation verwaltet. Zusätzlich ist die preemptive Ausführung nur in ganz bestimmten Kontexten verfügbar.

Wo ist der preemptive Modus verfügbar?

Der preemptive Modus ist nur in **64-bit Versionen** verfügbar. Derzeit werden folgende Ausführungskontexte unterstützt:

	Preemptive Ausführung
4D Server	X
4D Remote	-
4D Einzelplatz	X
Kompilierter Modus	X
Interpretierter Modus	-

Unterstützt der Ausführungskontext den preemptive Modus und ist die Methode "thread-safe", wird ein neuer 4D Prozess, gestartet über die Befehle **New process** oder **CALL WORKER** oder über den Menübefehl "Methode ausführen", in einem preemptive Thread ausgeführt.

Rufen Sie dagegen **New process** oder **CALL WORKER** in einem Ausführungskontext auf, der nicht unterstützt wird (z.B. auf einem Rechner mit remote 4D), ist der Prozess immer kooperativ.

Hinweis: Von einem remote 4D aus kann ein Prozess im preemptive Modus laufen, wenn eine Serverprozedur auf dem Server per Programmierung aufgerufen wird, z.B. mit dem Befehl **Execute on server**.

Thread-safe versus thread-unsafe

4D Code kann nur unter ganz bestimmten Bedingungen in einem preemptive Thread laufen. Jeder Teil des ausgeführten Code (Befehle, Methoden, Variablen) muss mit einer preemptive Ausführung kompatibel sein. Elemente, die in preemptive Threads laufen können, werden **thread-safe** genannt. Elemente, die nicht in preemptive Threads laufen können, werden **thread-unsafe** genannt.

Hinweis: Da ein Thread unabhängig von der ihn startenden übergeordneten Prozessmethode verwaltet wird, darf die gesamte Aufrufkette keinen thread-unsafe Code enthalten, da dann keine preemptive Ausführung möglich ist. Weitere Informationen dazu finden Sie im Abschnitt **Wann startet ein Prozess preemptive?**

Die Eigenschaft "thread-safe" für jedes Element hängt vom Element selbst ab:

- 4D Befehle: Thread-Sicherheit ist eine interne Eigenschaft. Im *Handbuch 4D Programmiersprache* werden thread-safe Befehle mit dem Icon gekennzeichnet. Ein großer Teil der 4D Befehle kann im preemptive Modus laufen.
- Projektmethoden: Die Bedingungen für thread-safe werden im Abschnitt **Eine thread-safe Methode schreiben** beschrieben.

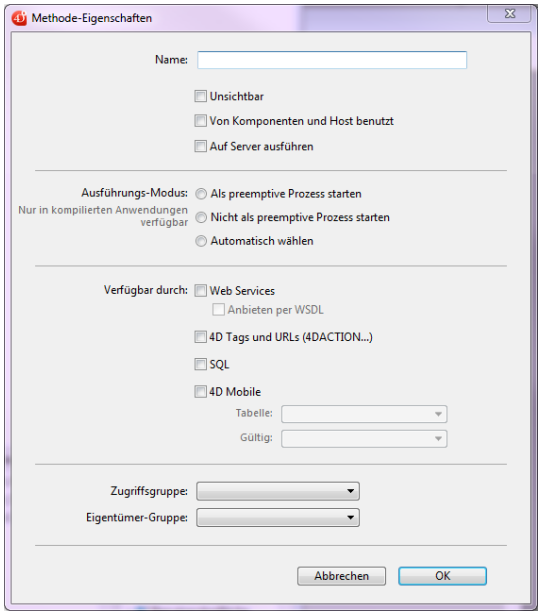
Im allgemeinen kann Code, der in preemptive Threads laufen soll, keine Teile mit externen Interaktionen aufrufen, wie Plug-In Code oder Interprozessvariablen. Zugriff auf Daten ist dagegen erlaubt, da der 4D Daten-Server die preemptive Ausführung unterstützt.

Preemptive Ausführungsmodus deklarieren

4D führt standardmäßig alle Projektmethoden im kooperativen Modus aus. Um die Vorteile des preemptive Modus zu nutzen, müssen Sie zuerst alle Methoden, die nach Möglichkeit im preemptive Modus starten sollen, explizit deklarieren. Das sind die als preemptiv nutzbar geltenden Methoden. Der Compiler prüft, ob solche Methoden tatsächlich thread-safe sind. Weitere Informationen dazu finden Sie im Abschnitt **Eine thread-safe Methode schreiben**. Bei Bedarf können Sie den preemptive Modus für einige Methoden auch nicht erlauben.

Beachten Sie, dass eine als preemptiv nutzbar deklarierte Methode lediglich für die preemptive Ausführung wählbar ist, aber nicht garantiert, dass sie in Echtzeit tatsächlich im preemptive Modus ausgeführt wird. Ob ein Prozess im preemptive Modus gestartet wird, richtet sich nach der Bewertung, die 4D für die Eigenschaften aller Methoden in der Aufrufkette des Prozesses durchführt. Weitere Informationen dazu finden Sie im Abschnitt **Wann startet ein Prozess preemptive?**.

Im Dialogfenster *Methode Eigenschaften* gibt es die neue Option **Ausführungsmodus**, um eine Methode für den preemptive Modus wählbar zu deklarieren:



Es gibt folgende Optionen:

- **Als preemptive Prozess starten**: Mit dieser Option deklarieren Sie, dass die Methode in einem preemptive Prozess laufen kann, d.h. sie soll wenn möglich, im preemptive Modus laufen.
  Der 4D Compiler überprüft, ob die Methode aktuell preemptiv nutzbar ist, andernfalls gibt er Fehler zurück -- das ist der Fall, wenn Befehle oder Methoden direkt oder indirekt aufgerufen werden, die nicht im preemptive Modus laufen können (die gesamte Aufrufkette wird durchlaufen, aber Fehler werden auf der ersten Unterebene protokolliert). Sie können dann die Methode bearbeiten, damit sie thread-safe wird, oder eine andere Option wählen.
  Ist die preemptive Nutzbarkeit der Methode bestätigt, wird sie intern als "thread-safe" markiert und im preemptive Modus ausgeführt, sobald alle erforderlichen Bedingungen zutreffen. Diese Eigenschaft definiert die Wahlmöglichkeit des preemptive Modus, garantiert aber nicht, dass die Methode tatsächlich in diesem Modus läuft, da dafür ein spezifischer Kontext erforderlich ist (siehe).
- **Nicht als preemptive Prozess starten**: Mit dieser Option deklarieren Sie, dass die Methode nie im preemptive Modus laufen soll, d.h. sie muss immer im kooperativen Modus laufen, wie in bisherigen 4D Versionen.
  Der 4D Compiler überprüft nicht, ob die Methode im preemptive Modus laufen kann; sie wird automatisch intern als "thread-unsafe" eingestuft (selbst wenn sie theoretisch preemptiv nutzbar wäre). Beim Aufrufen in Echtzeit kontaminiert diese Methode alle anderen Methoden im gleichen Thread und erzwingt so die Ausführung im kooperativen Modus (selbst wenn andere Methoden "thread-safe" sind).
- **Automatisch wählen**: Mit dieser Option deklarieren Sie, dass Sie die preemptive Nutzbarkeit für die Methode nicht verwalten wollen.
  Der 4D Compiler bewertet die preemptive Nutzbarkeit der Methode und stuft sie intern als "thread-safe" oder "thread-unsafe" ein. Es wird kein Fehler zur preemptive Ausführung gemeldet. Wird die Methode als "thread-safe" gewertet und in einem preemptive Kontext aufgerufen, verhindert das in Echtzeit nicht die preemptive Thread-Ausführung. Wird sie umgekehrt als "thread-unsafe" gewertet, verhindert das bei Aufrufen in Echtzeit jede preemptive Thread-Ausführung. Beachten Sie, dass mit dieser Option eine Methode, unabhängig von ihrer internen Bewertung der Thread-Sicherheit, immer im kooperativen Modus ausgeführt wird, wenn sie von 4D direkt als erste übergeordnete Methode aufgerufen wird (z.B. über die Funktion **New process**). Ist sie intern als thread-safe eingestuft, wird sie nur berücksichtigt, wenn sie von anderen Methoden innerhalb einer Aufrufkette aufgerufen wird.

Hinweis: Eine Komponentmethode, deklariert als "Gemeinsam von Komponenten und Host benutzt", muss auch als preemptiv nutzbar deklariert werden, damit sie über die Host Datenbank in einem preemptive Thread laufen kann.

Beim Export des Methoden-Code, z.B. mit **METHOD GET CODE**, wird die Eigenschaft "preemptiv" im Attribut des Parameters *Befehl* mit dem Wert "auto", "safe" oder "unsafe" exportiert. Auch die Befehle **METHOD GET ATTRIBUTES** und **METHOD SET ATTRIBUTES** erhalten oder setzen das Attribut "preemptiv" mit dem Wert "auto", "safe" oder "unsafe".

Nachfolgende Tabelle zeigt die Auswirkung der verschiedenen Optionen für preemptive Modus:

Option	Wert der Eigenschaft preemptiv (interpretiert)	Compiler Aktion	Internes Tag (kompiliert)	Ausführungsmodus bei thread-safe Kette
Kann in preemptive Prozessen laufen	Fähig	Prüft die Fähigkeit und gibt Fehler zurück, wenn unfähig	thread-safe	Preemptiv
Kann nicht in preemptive Prozessen laufen	Unfähig	Keine Bewertung	thread-unsafe	Kooperativ
Keine Differenzierung	auto	Bewertung, aber keine Fehlermeldung	thread-safe oder thread-unsafe	Wenn thread-safe: preemptiv; wenn thread-unsafe: kooperativ; bei direktem Aufruf: kooperativ

Wann startet ein Prozess preemptive?

Zur Erinnerung: Die preemptive Ausführung ist nur im kompilierten Modus möglich.

Beim Starten eines Prozesses, der entweder über **New process** oder **CALL WORKER** erstellt wurde, liest 4D im kompilierten Modus die preemptive Eigenschaft der Prozessmethode (auch *Eltern* Methode genannt) und führt den Prozess je nach definierter Eigenschaft im preemptive oder kooperativen Modus aus:

- Ist die Prozessmethode thread-safe (während der Kompilierung bestätigt), wird der Prozess in einem preemptive Thread ausgeführt
- Ist die Prozessmethode thread-unsafe, läuft der Prozess in einem kooperativen Thread.
- Lautet die preemptive Eigenschaft der Prozessmethode "automatisch wählen", läuft der Prozess zur Wahrung der Kompatibilität in einem kooperativen Thread (selbst wenn die Methode tatsächlich preemptiv nutzbar ist). Beachten Sie jedoch, dass diese Kompatibilität nur angewandt wird, wenn die Methode als Prozessmethode verwendet wird: Eine Methode im Ausführungsmodus "automatisch wählen", die aber vom Compiler intern als "thread-safe" eingestuft wird, lässt sich von einer anderen Methode als preemptiv aufrufen (siehe unten).

Die aktuelle Eigenschaft thread-safe richtet sich nach der Aufruffolge. Ruft eine als preemptiv nutzbar deklarierte Methode auf einer ihrer Unterebenen eine thread-unsafe Methode auf, wird ein Kompilierungsfehler zurückgegeben: Ist eine einzelne Methode innerhalb der gesamten Aufruffolge thread-unsafe, kontaminiert sie alle anderen Methoden und der Compiler verweigert die preemptive Ausführung. Ein preemptive Thread lässt sich nur erstellen, wenn die gesamte Aufruffolge thread-safe ist und die Prozessmethode als "Als preemptive Prozess starten" deklariert ist.
Andererseits lässt sich die gleiche thread-safe Methode in einer Aufruffolge in einem preemptive Thread ausführen, in einer anderen Aufruffolge dagegen im kooperativen Thread.

Hier sehen Sie ein paar Beispiele für Projektmethoden:

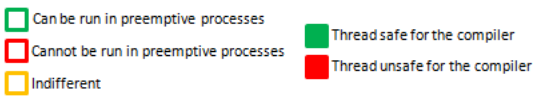
```
//Projektmethode MyDialog
//enthält Aufrufe der Oberfläche: ist intern thread unsafe
$win:=Open form window("tools";Palette form window)
DIALOG("tools")
```

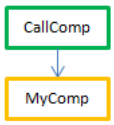
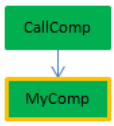
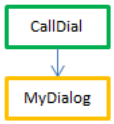
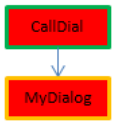
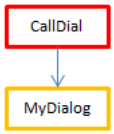
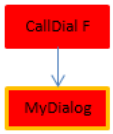
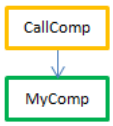
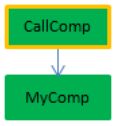
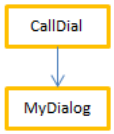
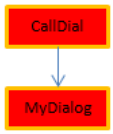
```
//Projektmethode MyComp
//enthält einfache Berechnung: ist intern thread safe
C_LONGINT($0;$1)
$0:=$1*2
```

```
//Projektmethode CallDial
C_TEXT($vName)
MyDialog
```

```
//Projektmethode CallComp
C_LONGINT($vAge)
MyCom($vAge)
```

Nachfolgende Übersicht zeigt die verschiedenen Situationen:



Deklaration und Aufruffolge	Kompilierung	Resultierende Thread Sicherheit	Kommentar	Ausführung
	OK		Methode A ist die übergeordnete Methode, deklariert als preemptive nutzbar; da Methode B intern thread-safe ist, ist Methode A thread-safe und der Prozess preemptiv	Preemptive
	Fehler		Methode C ist die übergeordnete Methode, deklariert als preemptive nutzbar; da aber Methode E intern thread-unsafe ist, kontaminiert sie die Aufruffolge und die Kompilierung schlägt fehl wegen Konflikt mit Methode C.	Die Lösung ist entweder, Methode E thread-safe zu machen (ausgehend, dass Methode D thread-safe ist), so dass die Ausführung preemptive läuft oder im kooperativen Modus zu arbeiten (die Deklaration der Methode C verändern).
	OK		Da Methode F als preemptive nicht nutzbar deklariert ist, ist die Kompilierung intern thread-unsafe, die Ausführung läuft immer kooperativ, egal welchen Status Methode G hat.	Kooperativ
	OK		Da Methode H die übergeordnete Methode ist (Eigenschaft "Automatisch wählen"), ist der Prozess kooperativ. Die Kompilierung ist erfolgreich, selbst wenn Methode I als preemptive nutzbar deklariert ist	Kooperativ
	OK		Methode J ist die übergeordnete Methode (Eigenschaft "Nicht-zugeordnet"), der Prozess ist dann kooperativ, selbst wenn die gesamte Aufruffolge thread-safe ist	Kooperativ

Den aktuellen Ausführungsmodus herausfinden

4D ermöglicht, den Ausführungsmodus für Prozesse im kompilierten Modus herauszufinden:

- Über den Befehl **PROCESS PROPERTIES** finden Sie heraus, ob ein Prozess im preemptive oder kooperativen Modus läuft.
- Das Runtime Explorer und das 4D Server Verwaltungsfenster zeigt für preemptive Prozesse und Worker Prozesse spezifische Icons an:

Prozesstyp	Icon
Preemptive Serverprozedur	
Preemptive Worker Prozess	
Cooperative Worker Prozess	

Eine thread-safe Methode schreiben

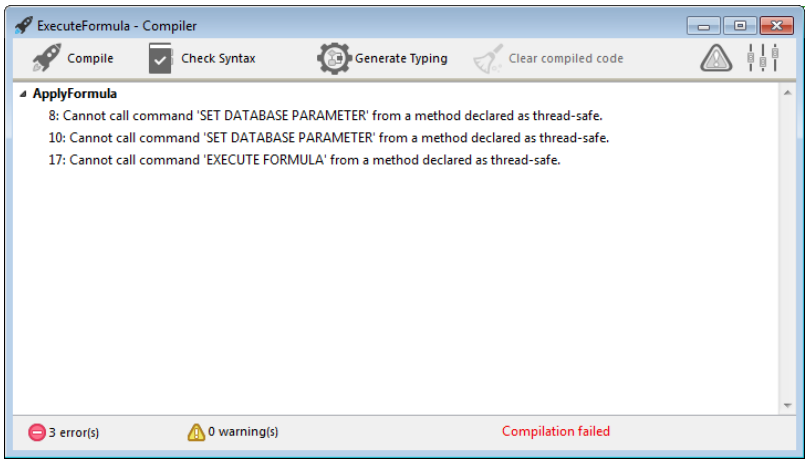
Für eine thread-safe Methode gelten folgende Regeln:

- Sie muss die Eigenschaft "Als preemptive Prozess starten" oder "Automatisch wählen" haben
- Sie ruft keinen 4D Befehl auf, der thread-unsafe ist
- Sie ruft keine andere Projektmethode auf, die thread-unsafe ist
- Sie ruft kein Plug-In auf
- Sie ruft keine Code-Sequenzen mit **begin sql/end sql** auf
- Sie verwendet keine Interprozessvariablen *
- Sie verwendet keine Objekte der Oberfläche ** (bis auf wenige Ausnahmen, siehe unten).

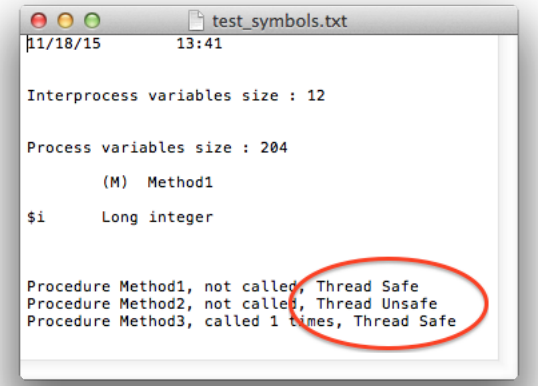
Für eine Methode mit der Eigenschaft "Gemeinsam von Komponenten und Host benutzt" muss auch die Eigenschaft "Als preemptive Prozess starten" markiert sein.

(*) Für den Datenaustausch zwischen preemptive Prozessen (und zwischen allen Prozessen) können Sie *shared collections* oder *shared objects* als Parameter an Prozesse übergeben bzw. den Katalog **Storage** verwenden. Weitere Informationen dazu finden Sie auf der Seite **Shared Objects und Shared Collections**.
Auch *Worker Prozesse* ermöglichen, Daten zwischen allen Prozessen auszutauschen, inkl. preemptive Prozesse. Weitere Informationen dazu finden Sie im Abschnitt **Über Worker**.
(**) Der Befehl **CALL FORM** bietet eine elegante Lösung, um Objekte der Oberfläche aus einem preemptive Prozess aufzurufen.

4D überprüft Methoden mit der Eigenschaft "Als preemptive Prozess starten" auch beim Kompilieren. Findet der Compiler etwas, das thread-safe verhindert, wird ein Kompilierungsfehler ausgegeben:



Ist die Symbol Datei aktiviert, zeigt sie für jede Methode den Status der Thread-Sicherheit:



Benutzeroberfläche

Aufrufen von Objekten der Benutzeroberfläche, z.B. Formulare oder Öffnen des Debuggers sind externe Zugriffe und deshalb in preemptive Threads nicht erlaubt.

Die Benutzeroberfläche ist von einem preemptive Thread aus nur in folgenden Fällen verwendbar:

- Standard Fehlerdialog: Er erscheint im Prozess Benutzermodus (4D Einzelplatz) oder Server Benutzeroberflächenprozess (4D Server). Die Schaltfläche **Schritt** ist deaktiviert.
- Standard Ablaufbalken
- Dialogfenster ALERT, REQUEST und CONFIRM. Er erscheint im Prozess Benutzermodus (4D Einzelplatz) oder Server Benutzeroberflächenprozess (4D Server).
  Beachten Sie, dass die Dialogfenster nicht erscheinen, wenn 4D Server als Service unter Windows gestartet wurde und keine Benutzeraktion erlaubt ist.

Thread-safe 4D Befehle

Eine beträchtliche Anzahl von 4D Befehlen sind thread-safe. Bei den Befehlen im *Handbuch 4D Programmiersprache* zeigt das Icon  im Bereich Eigenschaften rechts oben an, dass dieser Befehl thread-safe ist. Klicken Sie auf das Icon, erscheint die Liste der thread-safe Befehle.

Die Funktion **Command name** kann die Eigenschaft Thread-Sicherheit für jeden Befehl zurückgeben (siehe unten).

Trigger

Bei einer Methode mit einem Befehl, der Trigger aufrufen kann, bewertet der 4D Compiler zum Überprüfen der Thread-Sicherheit der Methode die Thread-Sicherheit des Trigger:

```
SAVE RECORD([Table_1]) //Trigger auf Table_1, wenn vorhanden, muss thread-safe sein
```

Hier die Liste der Befehle, die beim Kompilieren auf Thread-Sicherheit des Trigger geprüft werden:

- 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

Bei dynamischer Übergabe der Tabelle findet der Compiler manchmal nicht den Trigger zum Überprüfen heraus. Hierzu Beispiele:

```
DEFAULT TABLE([Table_1])
SAVE RECORD
SAVE RECORD($ptrOnTable->)
SAVE RECORD(Table(myMethodThatReturnsATableNumber())->)
```

In diesem Fall werden alle Trigger geprüft. Wird in mindestens einem Trigger ein thread-unsafe Befehl gefunden, wird die ganze Gruppe abgewiesen und die Methode als thread-unsafe deklariert.

Fehlerverwaltungsmethoden

Fehlerverwaltungsmethoden, die über den Befehl **ON ERR CALL** eingerichtet werden, müssen thread-safe sein, wenn sie von einem preemptive Prozess aufgerufen werden sollen. Für diesen Fall prüft der Compiler jetzt beim Kompilieren die Eigenschaft thread-safe der Fehlerverwaltungsmethoden im Befehl **ON ERR CALL** und gibt entsprechende Fehler zurück, wenn sie nicht zur preemptive Ausführung passen.

Beachten Sie, dass diese Prüfung nur möglich ist, wenn der Methodenname als Konstante übergeben ist und nicht berechnet wird, wie unten angezeigt:

```
ON ERR CALL("myErrMethod1") //wird vom Compiler geprüft
ON ERR CALL("myErrMethod"+String($vNum)) //wird vom Compiler nicht geprüft
```

Außerdem wird ab 4D v15 R5 der neue Fehler -10532 generiert, wenn sich eine Fehlerverwaltungsmethode nicht in Echtzeit aufrufen lässt (bei Unstimmigkeiten der Thread-Sicherheit oder aus einem anderen Grund, wie z.B. Methode wurde nicht gefunden). Die Fehlermeldung lautet: "Kann nicht die Projektmethode 'MethodeName' zur Fehlerverwaltung aufrufen".

Kompatibilität von Zeigern

Ein Prozess kann einen Zeiger nur dereferenzieren, um auf den Wert einer anderen Prozessvariable zuzugreifen, wenn beide Prozesse kooperativ sind. Sonst gibt 4D einen Fehler zurück. Versucht 4D Code in einem preemptive Prozess einen Zeiger auf eine Interprozessvariable zu dereferenzieren, gibt 4D einen Fehler aus.

Beispiel mit folgenden Methoden:

Methode1:

```
myVar:=42
$pid:=New process("Methode2";0;"Prozessname";->myVar)
```

Methode2:

```
$value:=$1->
```

Ist der Prozess mit Methode1 oder der Prozess mit Methode2 preemptive, gibt der Ausdruck "$value:=$1->" einen Ausführungsfehler aus.

Referenz DocRef

Der Parameter *DocRef* (geöffnete Dokumentreferenz, von **Open document**, **Create document**, **Append document**, **CLOSE DOCUMENT**, **RECEIVE PACKET**, **SEND PACKET** verwendet oder zurückgegeben) lässt sich folgendermaßen verwenden:

- Bei Aufruf von einem preemptive Prozess ist *DocRef* nur von diesem preemptive Prozess verwendbar.
- Bei Aufruf von einem kooperativen Prozess ist *DocRef* auch von anderen kooperativen Prozessen verwendbar.

Weitere Informationen dazu finden Sie im Abschnitt **DocRef: Referenznummer des Dokuments**.

Count tasks

Count tasks -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl	➲	Gesamtanzahl der Prozesse (inkl. Kernel Prozesse)

Beschreibung

Die Funktion **Count tasks** gibt die Höchstanzahl der Prozesse zurück, die in 4D Einzelplatz, 4D remote oder 4D Server (Serverprozeduren) vorhanden sind. Prozesse werden in der Reihenfolge, wie sie erstellt wurden, durchnummeriert. Wurde während der Sitzung noch kein Prozess abgebrochen, gibt diese Funktion die Gesamtanzahl der offenen Prozesse zurück.

Diese Anzahl berücksichtigt alle Prozesse, ,d.h. auch abgebrochene, wenn die Nummer nicht neu besetzt wurde und solche, die 4D automatisch verwaltet. Dazu gehören der Hauptprozess, Design-Prozess, Cache Manager-Prozess, Index-Prozess und der Web Server Prozess.

Beispiel

Siehe Beispiel zur Funktion **Process state** und zur **Semaphore**.

Count user processes

Count user processes -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Anzahl der "live" Prozesse (ohne interne Prozesse)

Beschreibung

Die Funktion **Count user processes** gibt die aktuelle Anzahl der "live" Prozesse in der 4D Anwendung zurück, die nicht vom Typ -25 (Internal Timer Process), -31 (Client Manager Process) und -15 (Server Interface Process) sind. Weitere Informationen zu Prozesstypen finden Sie unter dem Befehl **PROCESS PROPERTIES** und dem Konstantenthema **Prozesstypen**.
Count user processes gibt die Anzahl der Prozesse zurück, die vom Benutzer direkt oder indirekt geöffnet wurden. Das sind Prozesse, für die der Parameter *Ursprung*, zurückgegeben vom Befehl **PROCESS PROPERTIES**, größer oder gleich 0 (Null) ist.

Hinweis: "live" Prozesse sind Prozesse, dessen Status weder "abgebrochen", noch "existiert nicht" ist. Weitere Informationen dazu finden Sie unter der Funktion **Process state**.

Count users

Count users -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl	➲	Anzahl der Benutzer, die an den Server angemeldet sind

Beschreibung

Die Funktion **Count users** in einer Serverprozedur aufgerufen, gibt die Anzahl der Benutzer zurück, die an den Server-Rechner angemeldet sind.

Läuft auf dem Server mindestens eine Serverprozedur und wird **Count users** in einem anderen Kontext aufgerufen (Client-Rechner, Web Methode), gibt sie die Anzahl der Benutzer +1 zurück.

In 4D Developer gibt **Count users** den Wert 1 zurück.

Current process

Current process -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl	➲	Prozessnummer

Beschreibung

Die Funktion **Current process** gibt die Prozessnummer des gerade laufenden Prozesses zurück, der diese Funktion aufgerufen hat.

Beispiel

Siehe Beispiele zu den Befehlen **DELAY PROCESS** und **PROCESS PROPERTIES**.

Current process name

Current process name -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Text	Name des aktuellen Prozesses

Beschreibung

Der Befehl **Current process name** gibt den Namen des gerade laufenden Prozesses zurück, in dem diese Funktion aufgerufen wird.
Diese Funktion ist hilfreich im Kontext von Worker Prozessen (siehe Abschnitt **Über Worker**). Damit lässt sich der Worker Prozess identifizieren, der beim Schreiben von generischem Code aufgerufen werden soll.

Beispiel

Einen Worker aufrufen und den Namen des aufrufenden Prozesses als Parameter übergeben:

```
CALL WORKER(1;"myMessage";Current process name;"Start:"+String(vMax))
```

DELAY PROCESS

DELAY PROCESS (Prozess ; Dauer)

Parameter	Typ		Beschreibung
Prozess	Lange Ganzzahl	➡	Zu verzögernder Prozess
Dauer	Zahl	➡	Dauer in Ticks

Beschreibung

Der Befehl **DELAY PROCESS** verzögert den Prozess um die angegebene Anzahl der Ticks. 1 Tick ist der sechzigste Teil einer Sekunde. Während dieser Zeit ruht der Prozess und benutzt den Rechner nicht. Er ist aber immer noch im Arbeitsspeicher.

Sie können die Ausführung eines Prozesses auf eine Dauer kleiner als ein Tick verzögern. Übergeben Sie in *Dauer* z.B. 0,5, wird der Prozess um 1/2 Tick verzögert, also um 1/120stel Sekunde.

Ist der Prozess beim Aufruf von **DELAY PROCESS** verzögert, wird die neue Zeit nicht hinzugerechnet. Sie ersetzt vielmehr die alte Dauer. Übergeben Sie deshalb in *Dauer* Null (0), wenn Sie einen Prozess nicht länger verzögern wollen.

Ist der Prozess angehalten, hat der Befehl keine Wirkung. Mit dem Befehl **RESUME PROCESS** starten Sie einen verzögerten oder angehaltenen Prozess wieder neu.

Hinweis: Sie können mit diesem Befehl nicht von einem Client-Rechner aus eine Serverprozedur auf dem Server-Rechner zuweisen (Prozess<0).

Beispiel 1

Siehe Beispiel zu **Überblick zu Datensatz sperren**.

Beispiel 2

Siehe Beispiel zur Funktion **Process number**.

EXECUTE ON CLIENT

EXECUTE ON CLIENT (ClientName ; Methodenname {; Param1…N}{; Param1…N2 ; ... ; Param1…NN})

Parameter	Typ		Beschreibung
ClientName	String	➡	Registrierter Name von 4D Client
Methodenname	String	➡	Name der auszuführenden Methode
Param1…N		➡	Parameter der Methode

Beschreibung

Der Befehl **EXECUTE ON CLIENT** löst die Ausführung der Methode *Methodenname* mit den Parametern *Param1...N* aus; wenn erforderlich auf dem registrierten remote 4D mit Namen *ClientName*. Dieser Name wird über den Befehl **REGISTER CLIENT** definiert. Sie können ihn von einem remote 4D oder einer Serverprozedur auf 4D Server aufrufen.

Benötigt eine Methode einen oder mehrere Parameter, geben Sie diese nach dem Namen der Methode an.
Die Methode auf remote 4D wird in einem Prozess ausgeführt. Dieser Prozess wird automatisch auf der Arbeitsstation mit dem registrierten Namen des remote 4D angelegt.

Ruft derselbe Client diesen Befehl mehrmals auf, werden die Ausführungsanweisungen gestapelt. Die Methoden werden dann asynchron der Reihe nach abgearbeitet. Je mehr Methoden gestapelt sind, desto höher ist der Arbeitsaufwand für den Client. Mit dem Befehl **GET REGISTERED CLIENTS** können Sie den Status des Arbeitsaufwands für jeden Client abfragen.

Hinweis: Sie können das Stapeln der Ausführungsanweisungen nur ändern oder stoppen, wenn Sie mit dem Befehl **UNREGISTER CLIENT** die Registrierung des 4D Client rückgängig machen.

Sie können dieselbe Methode auf mehreren oder allen registrierten 4D Clients gleichzeitig ausführen. Verwenden Sie dazu im Parameter *ClientName* das Jokerzeichen (@).

Beispiel 1

Sie möchten die Methode “GenerateNums” auf der Arbeitsstation “Client1” ausführen:

```
EXECUTE ON CLIENT("Client1";"GenerateNums";12;$a;"Text")
```

Beispiel 2

Sollen alle Clients die Methode “EmptyTemp” ausführen, schreiben Sie:

```
EXECUTE ON CLIENT("@";"EmptyTemp")
```

Beispiel 3

Siehe Beispiel zum Befehl **REGISTER CLIENT**.

Systemvariablen und Mengen

Die Systemvariable *OK* hat den Wert 1, wenn 4D Server die Anfrage zum Ausführen einer Methode korrekt erhalten hat; das garantiert jedoch nicht, dass die Methode von 4D Client korrekt ausgeführt wird.

Execute on server

Execute on server (Methodenname ; Stapel {; Prozessname {; Param {; Param2 ; ... ; ParamN}}}{; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Methodenname	String	➡	Zu startende Methode im Prozess
Stapel	Lange Ganzzahl	➡	Größe des Stapelspeichers in Byte
Prozessname	String	➡	Name des erzeugten Prozesses
Param	Ausdruck	➡	Parameter für die Methode
*	Operator	➡	Einmaliger Prozess
Funktionsergebnis	Lange Ganzzahl	⮌	Prozessnummer für neu erstellten oder bereits laufenden Prozess

Beschreibung

Die Funktion **Execute on server** startet einen neuen Prozess auf dem Server-Rechner (aufgerufen im Client/Server-Betrieb) oder auf demselben Rechner (aufgerufen im Einzelbetrieb) und gibt die Prozessnummer für diesen Prozess zurück.

Mit dieser Funktion starten Sie eine Serverprozedur. Weitere Informationen dazu finden Sie im *4D Serverhandbuch* im Abschnitt **Serverprozeduren**.

Rufen Sie **Execute on server** auf einem Client-Rechner auf, gibt der Befehl eine negative Prozessnummer zurück. Rufen Sie die Funktion auf dem Server-Rechner auf, gibt sie eine positive Prozessnummer zurück. Beachten Sie, dass Aufrufen von **New process** auf dem Server-Rechner dasselbe ausführt wie Aufrufen von **Execute on server**.

Konnte der Prozess nicht erstellt werden, z.B. weil der Speicher dafür nicht ausreicht, gibt **Execute on server** den Wert Null (0) zurück und es wird ein Fehler generiert. Sie können diesen Fehler mit **ON ERR CALL** in einer Fehlerverwaltungsmethode ausfindig machen.

Prozessmethode

Methodenname ist der Name der Methode, die im Prozess gestartet werden soll.
Nachdem 4D den Kontext für den neuen Prozess bestimmt hat, beginnt es mit der Ausführung dieser Methode. Sie wird zur Prozessmethode.

Prozessstapel

Im Parameter *Stapel* können Sie die Größe des Stapelspeichers angeben. Im Stapelspeicher werden Methodenaufrufe, lokale Variablen, Parameter in Unterroutinen und gestapelte Datensatze angehäuft.

- Übergeben Sie in *Stapel* den Wert 0 für eine standardmäßige Stapelgröße, die für die meisten Anwendungen passt (empfohlene Einstellung)
- In bestimmten Fällen wollen Sie lieber einen eigenen Wert verwenden. Sie müssen ihn in Bytes angeben. Sie sollten mindestens 64.000 Bytes zuweisen. Bei sehr langen, verschachtelten Methoden weisen Sie 512.000 Bytes oder mehr zu. Wenn Sie zuwenig Speicher zuweisen, tritt die Fehlermeldung “Der Stapelspeicher ist voll” auf. Der angegebene Wert muss ein Vielfaches von 2 sein.

Hinweis: Der Stapelspeicher ist NICHT der Gesamtspeicher des Prozesses. Prozesse teilen den Speicher auf für Datensätze, Interprozessvariablen usw., Prozessvariablen werden in einem Extraspeicher abgelegt. Der Stapelspeicher hält nur lokale Variablen, Methodenaufrufe, Parameter in Unterroutinen und gestapelte Datensätze.

Hinweis für 64-bit 4D Server: Der Stapelspeicher für einen Prozess, der auf einem 64-bit 4D Server ausgeführt wird, benötigt mehr Speicher als ein 32-bit 4D Server (in etwa doppelt soviel). In Einklang mit den oben angegebenen Schätzungen emfehlen wir im Normalfall mindestens 128.000 Bytes, und 400.000 Bytes zum Verwalten einer anpassbaren Aufrufkette. Denken Sie daran, diesen Parameter zu prüfen, wenn Ihr Code zum Ausführen auf einem 64-bit Server dienen soll.

Prozessname

Sie übergeben den Namen des neuen Prozesses in *Prozessname*. Im Einzelplatz erscheint dieser Name in der Liste der Prozesse des Runtime Explorer. Er wird vom Befehl **PROCESS PROPERTIES** zurückgegeben, wenn er auf diesen neuen Prozess angewendet wird. Im Client/Server-Betrieb erscheint dieser Name im Hauptfenster von 4D Server in der Liste Serverprozeduren in Blau.
Geben Sie diesen Parameter nicht an, ist *Prozessname* ein leerer String.

Warnung: Machen Sie im Gegensatz zu **New process** mit der Funktion **Execute on server** den Prozess NICHT durch das vorangestellte Dollarzeichen lokal. Das funktioniert nur im Einzelplatzbetrieb, da **Execute on server** hier als **New process** agiert. Im Client/Server-Betrieb ruft das einen Fehler hervor.

Parameter für Prozessmethode

Sie können für die Prozessmethode Parameter übergeben, und zwar genauso wie für eine Unterroutine. Lediglich Ausdrücke vom Typ Zeiger und Arrays können Sie nicht übergeben. Sobald die Ausführung im Kontext des neuen Prozesses startet, erhält die Prozessmethode die Parameterwerte in $1, $2, etc.

Hinweis: Übergeben Sie Parameter in der Prozessmethode, müssen Sie auch *Prozessname* übergeben. In diesem Fall können Sie ihn nicht weglassen.

Übergeben Sie als Parameter ein 4D Objekt (**C_OBJECT**) oder eine Collection (**C_COLLECTION**), wird eine Kopie (keine Referenz) gesendet und das JSON Formular in UTF-8 für den Server verwendet. Enthält das Objekt oder die Collection Zeiger, werden die dereferenzierten Werte und nicht die Zeiger selbst gesendet.

Optionaler Parameter *

Geben Sie diesen Parameter an, erhält 4D die Anweisung, zuerst zu prüfen, ob bereits ein Prozess *Prozessname* vorhanden ist. Ist dies der Fall, startet 4D keinen neuen Prozess und gibt die Prozessnummer des Prozesses *Prozessname* zurück.

Beispiel

Folgendes Beispiel zeigt, wie Sie den Datenimport im Client/Server-Betrieb entscheidend beschleunigen können. Mit der nachfolgenden Methode **Regular Import** können Sie prüfen, wie lange der Import von Datensätzen mit dem Befehl **IMPORT TEXT** auf der Arbeitsstation dauert:

```
  ` Projektmethode Regular Import
$vhDocRef:=Open document("")
If(OK=1)
   CLOSE DOCUMENT($vhDocRef)
   FORM SET INPUT([Table1];"Import")
   $vhStartTime:=Current time
   IMPORT TEXT([Table1];Document)
   $vhEndTime:=Current time
   ALERT("Es dauerte "+String(0+($vhEndTime-$vhStartTime))+" Sekunden.")
End if
```

Bei regulären Importdaten zeigt 4D Client das Durchlaufen der Textdatei an, und erstellt dann für jeden Datensatz einen neuen Datensatz, füllt die Felder mit den importierten Daten und sendet den Datensatz an den Server-Rechner, damit er der Datenbank hinzugefügt wird. Auf diese Weise gehen viele Anfragen über das Netz. Diese Operation können Sie über eine Serverprozedur optimieren, die den Vorgang lokal auf dem Server-Rechner ausführt. Der Client-Rechner lädt das Dokument in ein BLOB, übergibt das BLOB als Parameter und startet so eine Serverprozedur. Die Serverprozedur speichert das BLOB in ein Dokument auf der Festplatte des Server-Rechners und importiert das Dokument dann lokal. Der Datenimport wird mit der Geschwindigkeit einer Einzelplatz-Version ausgeführt, da die meisten Anfragen über das Netz eliminiert wurden.
Hier ist die Projektmethode ***CLIENT IMPORT***. Sie wird auf dem Client-Rechner ausgeführt und startet die nachfolgende Serverprozedur *SERVER IMPORT*:

```
  ` Projektmethode CLIENT IMPORT
  ` CLIENT IMPORT ( Pointer ; String )
  ` CLIENT IMPORT ( -> [Table] ; Input form )

C_POINTER($1)
C_TEXT(31;$2)
C_TIME($vhDocRef)
C_BLOB($vxData)
C_LONGINT(spErrCode)

  `Wähle das zu importierende Dokument
$vhDocRef:=Open document("")
If(OK=1)
  ` Wurde ein Dokument ausgewählt, lasse es nicht offen
   CLOSE DOCUMENT($vhDocRef)
   $vhStartTime:=Current time
  ` Versuche, es in den Speicher zu laden
   DOCUMENT TO BLOB(Document;$vxData)
   If(OK=1)
  ` Das Dokument konnte ins BLOB geladen werden, Starte Serverprozedur, die Daten auf dem Server-Rechner importiert
      $spProcessID:=Execute on server("SERVER IMPORT";0;"Server Import Services";Table($1);$2;$vxData)
  ` Wir benötigen das BLOB nicht länger in diesem Prozess
      CLEAR VARIABLE($vxData)
  ` Warte bis von Serverprozedur ausgeführte Operation fertig ist.
      Repeat
         DELAY PROCESS(Current process;300)
         GET PROCESS VARIABLE($spProcessID;spErrCode;spErrCode)
         If(Undefined(spErrCode))
  ` <strong>Hinweis:</strong> Wurde die Serverprozedur nicht mit der eigenen Instanz der Variablen spErrCode initialisiert, wird evtl. eine undefinierte Variable zurückgegeben
            spErrCode:=1
         End if
      Until(spErrCode<=0)
  ` Teile der Serverprozedur mit, dass wir bestätigen
      spErrCode:=1
      SET PROCESS VARIABLE($spProcessID;spErrCode;spErrCode)
      $vhEndTime:=Current time
      ALERT("Es dauerte "+String(0+($vhEndTime-$vhStartTime))+" Sekunden.")
   Else
      ALERT("Der Speicher reicht nicht aus zum Laden des Dokuments.")
   End if
End if
```

Die Projektmethode ***SERVER IMPORT*** ausgeführt als Serverprozedur:

```
  ` SERVER IMPORT Project Method
  ` SERVER IMPORT ( Long ; String ; BLOB )
  ` SERVER IMPORT ( Tabellennummer ; Eingabeformular ; Importdaten )
```

```
C_LONGINT($1)
C_TEXT(31;$2)
C_BLOB($3)
C_LONGINT(spErrCode)

  ` Operation ist noch nicht beendet, setze spErrCode auf 1
spErrCode:=1
$vpTable:=Table($1)
INPUT FORM($vpTable->;$2)
$vsDocName:="Import File "+String(1+Random)
DELETE DOCUMENT($vsDocName)
BLOB TO DOCUMENT($vsDocName;$3)
IMPORT TEXT($vpTable->;$vsDocName)
DELETE DOCUMENT($vsDocName)
  ` Operation ist beendet, setze spErrCode auf 0
spErrCode:=0
  ` Warte bis der anfragende Client das Ergebnis erhalten hat
Repeat
   DELAY PROCESS(Current process;1)
Until(spErrCode>0)
```

Mit diesen beiden Projektmethoden in einer Datenbank können Sie einen Datenimport über eine Serverprozedur ausführen. Schreiben Sie z.B.:

```
CLIENT IMPORT(->[Table1];"Import")
```

Mit einigen Benchmarks werden Sie entdecken, dass Sie mit dieser Methode Datensätze bis zu 60 Mal schneller importieren können als über regulären Import.

Get process activity

Get process activity {(Optionen)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Optionen	Lange Ganzzahl	➡	Eine bestimmte Liste zurückgeben
Funktionsergebnis	Objekt	➲	Momentaufnahme der laufenden Prozesse bzw. Benutzersitzungen (nur 4D Server)

Beschreibung

Die Funktion **Get process activity** gibt eine Momentaufnahme (snapshot) der verbundenen Benutzersitzungen bzw. dazugehörigen laufenden Prozessen zu einem bestimmten Moment zurück. Sie gibt alle Prozesse zurück, d.h. auch interne Prozesse, die durch den Befehl **PROCESS PROPERTIES** nicht erreichbar sind.

- Bei Ausführung auf dem Server ohne den Parameter *Optionen* (Standardeinstellung) gibt **Get process activity** die Liste der Benutzersitzungen und der Prozesse zurück. Hierzu ein Beispiel:


```
{
"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"
                            }
             },...
    ]
}
```


Um eine der Listen festzulegen, übergeben Sie den Parameter *Optionen* und wählen eine der folgenden Konstanten unter dem Thema **4D Umgebung**:

Konstante	Typ	Wert	Kommentar
Processes only	Lange Ganzzahl	1	Gibt nur die Prozessliste zurück
Sessions only	Lange Ganzzahl	2	Gibt nur die Liste Benutzersitzung zurück

- Bei Ausführung von 4D im remote oder lokalen Modus gibt **Get process activity** nur die Prozessliste zurück. Der Parameter *Optionen* ist hier ohne Bedeutung.

Liste der Sitzungen

Die Eigenschaft "sessions" enthält eine Sammlung (collection) von Objekten mit allen laufenden Benutzersitzungen auf dem Server. Hierzu ein Beispiel:

Jedes Objekt Sitzung enthält folgende Eigenschaften (wenn nicht verfügbar, werden keine Eigenschaften zurückgegeben):

Name der Eigenschaft	Typ	Beschreibung
type	Text (num)	Sitzungstyp. Mögliche Werte: "remote", "storedProcedure", "web", "rest"
hostType	Text (num)	Hosttyp. Mögliche Werte: "windows", "mac", "browser"
userName	Text	Benutzername
machineName	Text	Name des remote Rechners
systemUserName	Text	Name der auf dem remote Rechner geöffneten Systemsitzung
IPAddress	Text	IP Adresse des remote Rechners
creationDateTime	Datum ISO 8601	Datum und Zeit der Verbindung auf dem remote Rechner
state	Text (num)	Sitzungsstatus. Mögliche Werte: "active", "postponed", "sleeping"
ID	Text	Sitzung UUID

Liste der Prozesse

Die Eigenschaft "process" enthält eine Sammlung (collection) von Objekten mit allen laufenden Prozessen auf dem Server. Hierzu ein Beispiel:

Jedes Objekt Prozess enthält folgende Eigenschaften (wenn nicht verfügbar, werden keine Eigenschaften zurückgegeben):

Name der Eigenschaft	Typ	Beschreibung
name	Text	Prozessname
sessionID	Text	Session UUID
ID	Lange Ganzzahl	Einmalige ID des Prozesses
visible	Boolean	Wahr wenn sichtbar, sonst falsch
systemID	Text	ID für den Benutzerprozess, 4D Prozess oder wartenden Prozess
type	Lange Ganzzahl	Typ des laufenden Prozesses. Sie können eine der folgenden Konstanten unter dem Thema **Prozesstypen** verwenden:
state	Lange Ganzzahl	Aktueller Status (siehe Konstantenliste **Prozesstatus**)
cpuUsage	Zahl	Diesem Prozess (zwischen 0 und 1) gewidmete Zeit (in Prozent)
cpuTime	Zahl	Laufende Zeit (Sekunden)
preemptive	Boolean	Wahr wenn er preemptive läuft, sonst falsch
session	Objekt	Spezifische Sitzung, in welcher der Prozess läuft. Undefiniert, wenn der Parameter

Konstante	Wert	Kommentar
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	Worker Prozess, vom Benutzer gestartet

Processes only übergeben ist.

Beispiel

Die Collection aller Benutzersitzungen erhalten:

```
 //Zur Ausführung auf dem Server
C_OBJECT($result)
C_COLLECTION($userCol)
$result:=Get process activity(Sessions only)
$userCol:=OB Get($result;"Benutzer")
```

GET REGISTERED CLIENTS

GET REGISTERED CLIENTS (ClientListe ; Methoden)

Parameter	Typ		Beschreibung
ClientListe	Array Text	⇐	Liste der gesicherten 4D Clients
Methoden	Array Lange Ganzzahl	⇐	Liste der auszuführenden Methoden

Beschreibung

Der Befehl **GET REGISTERED CLIENTS** füllt zwei Arrays:

- *ClientListe* enthält die Liste der Clients, die mit dem **REGISTER CLIENT** registriert wurden.
- *Methoden* liefert die Liste des Arbeitsaufkommens für jeden Client. Das ist die Anzahl der Methoden, die ein 4D Client mit dem Befehl **EXECUTE ON CLIENT** noch ausführen muss. Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **EXECUTE ON CLIENT**.

Beispiel 1

Sie möchten eine Liste aller registrierten Clients sowie der Methoden, die noch ausgeführt werden müssen:

```
ARRAY TEXT($clients;0)
ARRAY LONGINT($methods;0)
GET REGISTERED CLIENTS($clients;$methods)
```

Beispiel 2

Siehe Beispiel zum Befehl **REGISTER CLIENT**.

Systemvariablen und Mengen

War die Operation erfolgreich, hat die Systemvariable *OK* den Wert 1.

New process

New process (Methodenname ; Stapel {; Prozessname {; Param {; Param2 ; ... ; ParamN}}}{; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Methodenname	String	➡	Zu startende Methode
Stapel	Lange Ganzzahl	➡	Größe des Stapelspeichers in Byte
Prozessname	String	➡	Name des erzeugten Prozesses
Param	Ausdruck	➡	Parameter für die Methode
*	Operator	➡	Einmaliger Prozess
Funktionsergebnis	Lange Ganzzahl	⮌	Prozessnummer des neu erstellten bzw. bereits laufenden Prozesses

Beschreibung

Die Funktion **New process** startet auf demselben Rechner einen neuen Prozess und gibt seine Prozessnummer zurück. Diese Nummer ist zu einem bestimmten Zeitpunkt eindeutig. Ist der Prozess gelöscht, wird die Prozessnummer neu vergeben.

Konnte der Prozess nicht erstellt werden, z.B weil der Speicher dafür nicht ausreicht, gibt **New process** den Wert Null (0) zurück und es wird ein Fehler generiert. Sie können diesen Fehler über eine Fehlerverwaltungsmethode ausfindig machen, die **ON ERR CALL** einsetzt.

Prozessmethode

Methodenname ist der Name der Methode, die im Prozess gestartet werden soll.
Nachdem 4D den Kontext für den neuen Prozess bestimmt hat, beginnt es mit der Ausführung dieser Methode. Sie wird zur Prozessmethode.

Prozessstapel

Im Parameter *Stapel* können Sie die Größe des Stapelspeichers angeben. Im Stapelspeicher werden Methodenaufrufe, lokale Variablen, Parameter in Unterroutinen und gestapelte Datensatze angehäuft.

- Übergeben Sie in *Stapel* den Wert 0 für eine standardmäßige Stapelgröße, die für die meisten Anwendungen passt (empfohlene Einstellung)
- In bestimmten Fällen wollen Sie lieber einen eigenen Wert verwenden. Sie müssen ihn in Bytes angeben. Sie sollten mindestens 64.000 Bytes zuweisen. Bei sehr langen, verschachtelten Methoden weisen Sie 512.000 Bytes oder mehr zu. Wenn Sie zuwenig Speicher zuweisen, tritt die Fehlermeldung “Der Stapelspeicher ist voll” auf. Der angegebene Wert muss ein Vielfaches von 2 sein.

Hinweis: Der Stapelspeicher ist NICHT der Gesamtspeicher des Prozesses. Prozesse teilen den Speicher auf zwischen Datensätzen, Interprozessvariablen, usw.. Prozessvariablen werden in einem Extraspeicher gespeichert. Der Stapelspeicher enthält verschiedene 4D Informationen; die Menge an Informationen richtet sich nach der Anzahl der eingebundenen Methodenaufrufe, die der Prozess verwendet, der Anzahl der geöffneten und wieder geschlossenen Formulare, sowie Anzahl und Größe der lokalen Variablen, die in jedem eingebetteten Methodenaufruf verwendet werden.

Hinweis für 64-bit Versionen: Der Stapelspeicher für einen Prozess, der auf einer 64-bit Version von 4D ausgeführt wird, benötigt mehr Speicher als eine 32-bit Version von 4D (in etwa doppelt soviel). In Einklang mit den oben angegebenen Schätzungen empfehlen wir im Normalfall mindestens 128.000 Bytes, und 400.000 Bytes zum Verwalten einer anpassbaren Aufrufkette.
Denken Sie daran, diesen Parameter zu prüfen, wenn Ihr Code zum Ausführen auf einer 64-bit Version bestimmt ist.

Prozessname

Den Namen des neuen Prozesses übergeben Sie in *Prozessname*. Dieser Name erscheint in der Prozessliste des Runtime Explorers. Er wird von dem Befehl **PROCESS PROPERTIES** zurückgegeben, wenn er auf diesen neuen Prozess angewendet wird. Geben Sie diesen Parameter nicht an, ist *Prozessname* ein leerer String. Soll der Prozess nur lokal gelten, stellen Sie das Dollarzeichen voran (*$*).

Wichtig: Beachten Sie, dass lokale Prozesse keinen Zugriff auf Daten im Client/Server-Betrieb haben sollten.

Parameter für Prozessmethode

Sie übergeben für die Prozessmethode einen oder mehrere *Param* Parameter. Sie können die Parameter genauso wie für eine Unterroutine übergeben (siehe Abschnitt **Parameter in Methoden**). Sobald die Ausführung im Kontext des neuen Prozesses startet, erhält die Prozessmethode die Parameterwerte in *$1*, *$2*, etc.
Beachten Sie, dass Sie Arrays nicht als Parameter in einer Methode übergeben können. Auch beim Übergeben von Ausdrücken mit Zeigern müssen Sie auf zusätzliche Gegebenheiten achten:

- Zeiger auf Tabellen und Felder sind erlaubt,
- Zeiger auf Variablen, insbesondere lokale und Prozessvariablen werden nicht empfohlen, da diese Variablen im Moment, wo die Prozessmethode darauf zugreift, undefiniert sein können.
- Übergeben Sie einen Parameter vom Typ Objekt oder Collection, erstellt 4D in diesem Fall keine Referenz, sondern eine Kopie des Objekts oder der Collection im Zielprozess.

Hinweis: Übergeben Sie Parameter in der Prozessmethode, müssen Sie auch *Prozessname* übergeben. In diesem Fall können Sie ihn nicht weglassen.

Optionaler Parameter *

Geben Sie diesen Parameter an, erhält 4D die Anweisung, zuerst zu prüfen, ob bereits ein Prozess *Prozessname* vorhanden ist. Ist dies der Fall, startet 4D keinen neuen Prozess und gibt die Prozessnummer des Prozesses *Prozessname* zurück.

Beispiel

Wir gehen aus von der Projektmethode:

```
` ADD CUSTOMERS
SET MENU BAR(1)
Repeat
   ADD RECORD([Customers];*)
Until(OK=0)
```

Fügen Sie diese Projektmethode über den Menüleisteneditor der Designumgebung in einem eigenen Menüeintrag hinzu, für den die Eigenschaft **Starte Neuen Prozess** aktiviert ist, startet 4D automatisch einen neuen Prozess, der mit dieser Methode läuft. Der Aufruf **SET MENU BAR**(1) fügt dem neuen Prozess eine Menüleiste hinzu. Ist kein Fenster vorhanden (das sie mit der Funktion **Open window** öffnen können), öffnet der Aufruf von **ADD RECORD** automatisch ein Fenster.

Damit Sie den Prozess *Add Customers* über eine Schaltfläche in einem eigenen Kontrollfeld starten können, schreiben Sie:

```
` Objektmethode für Schaltfläche bAddCustomers
$vlProcessID:=New process("Add Customers";0;"Kunden hinzufügen")
```

Die Schaltfläche führt dasselbe aus wie der eigene Menüeintrag.

Wollen Sie beim Auswählen des Menüeintrags oder Anklicken der Schaltfläche den Prozess starten (wenn er nicht vorhanden ist) oder nach vorne bringen (wenn er schon läuft), können Sie die Methode ***START ADD CUSTOMERS*** einrichten:

```
` START ADD CUSTOMERS
$vlProcessID:=New process("Add Customers";0;"Kunden hinzufügen";*)
If($vlProcessID#0)
   BRING TO FRONT($vlProcessID)
End if
```

Die Objektmethode der Schaltfläche *bAddCustomers* lautet dann:

```
` Objektmethode für Schaltfläche bAddCustomers
START ADD CUSTOMERS
```

Im Menüleisteneditor ersetzen Sie die Methode ***ADD CUSTOMERS*** durch die Methode ***START ADD CUSTOMERS*** und deaktivieren die Eigenschaft **Starte Neuen Prozess** für den Menüeintrag.

PAUSE PROCESS

PAUSE PROCESS (Prozessnr)

Parameter	Typ		Beschreibung
Prozessnr	Lange Ganzzahl	➡	Prozessnummer

Beschreibung

Der Befehl **PAUSE PROCESS** hält einen laufenden Prozess an. Er muss wieder von **RESUME PROCESS** gestartet werden. **RESUME PROCESS** muss von einem anderen Prozess aufgerufen werden. Während dieser Zeit ruht der Prozess und benutzt den Rechner nicht. Er ist aber immer noch im Arbeitsspeicher.

Ist der Prozess bereits angehalten, hat **PAUSE PROCESS** keine Wirkung. Wurde der Prozess mit **DELAY PROCESS** verzögert, wird der Prozess angehalten. Mit dem Befehl **RESUME PROCESS** starten Sie einen verzögerten oder angehaltenen Prozess wieder neu.

Ist die Prozessausführung gestoppt, sind die zu diesem Prozess gehörenden Fenster nicht eingebbar. Das kann den Benutzer u.U. verwirren. Um das zu vermeiden, sollten Sie den Prozess ausblenden. Gibt es *Prozessnr* nicht, hat der Befehl keine Auswirkung.

Warnung: Verwenden Sie **PAUSE PROCESS** nur in gestarteten Prozessen. Er beeinträchtigt nicht den ursprünglichen Benutzer/Runtime-Prozess.

Hinweis: Sie können mit diesem Befehl nicht von einem Client-Rechner aus eine Serverprozedur auf dem Server-Rechner zuweisen (Prozess<0).

Process aborted

Process aborted -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		True = Der Prozess wird gleich abgebrochen False = Der Prozess wird nicht gleich abgebrochen

Beschreibung

Die Funktion **Process aborted** gibt **Wahr** zurück, wenn der aufrufende Prozess gleich unerwartet abgebrochen wird. Das bedeutet, dass die Ausführung des Befehls nicht normal zu Ende geführt werden konnte. Dieser Fall kann z.B. nach Aufrufen von **QUIT 4D** eintreten.

Process number

Process number (Auswahlname {; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Auswahlname	String	➡	Prozessname der zu findenden Prozessnummer
*		➡	Rückgabe der Prozessnummer vom Server
Funktionsergebnis	Lange Ganzzahl	⮌	Prozessnummer

Beschreibung

Die Funktion **Process number** gibt die Prozessnummer zum Prozess *Prozessname* zurück. Wird kein Prozess gefunden, gibt Prozessnummer den Wert Null (0) zurück.

Mit dem optionalen Parameter * können Sie von 4D Client aus die Prozessnummer eines Prozesses finden, der auf dem Server ausgeführt wird. Der zurückgegebene Wert ist dann negativ. Diese Option ist besonders hilfreich bei den Befehlen **GET PROCESS VARIABLE**, **SET PROCESS VARIABLE** und **VARIABLE TO VARIABLE**. Weitere Informationen dazu finden Sie in der Beschreibung zu diesen Befehlen.

Wird die Funktion mit dem Parameter * von einem Prozess auf dem Server ausgeführt, ist der zurückgegebene Wert positiv.

Beispiel

Sie erstellen ein eigenes Palettenfenster, das in einem eigenen Prozess läuft. Hier bauen Sie eigene Tools für die Interaktion mit der Designumgebung ein.
Sie wollen zum Beispiel einen Eintrag in einer hierarchischen Liste von Schlüsselwörtern auswählen und in das vorderste Fenster der Designumgebung einsetzen. Dazu können Sie die Zwischenablage verwenden, das Ereignis *Einsetzen* muss jedoch in einem Designprozess stattfinden. Folgende Funktion gibt die Prozessnummer des Design-Prozesses zurück, jedoch nur, sofern dieser aktiv ist:

```
` Projektmethode Design process number
` Design process number -> Lange Ganzzahl
` Design process number -> Nummer des Designprozesses

$0:=Process number(Get indexed string(170;3))
` Name des Designprozesses ist gespeichert in der 'STR#" Ressource ID=170, String #3 in 4D
```

Mit dieser Funktion setzt die folgende Projektmethode den als Parameter erhaltenen Text in das vorderste Fenster der Designumgebung:

```
` Projektmethode PASTE TEXT TO DESIGN
` PASTE TEXT TO DESIGN (Text)
` PASTE TEXT TO DESIGN (Text zum Einsetzen in das vorderste Designfenster)

C_TEXT($1)
C_LONGINT($vlDesignPID;$vlCount)

$vlDesignPID:=Design process number
If($vlDesignPID #0)
   ` Setze Text in die Zwischenablage
   SET TEXT TO CLIPBOARD($1)
   ` Sende Ereignis Ctrl-V / Befehl-V
   POST KEY(ASCII("v");Command key mask;$vlDesignPID)
   ` Rufe wiederholt DELAY PROCESS auf, so dass das Ereignis
   ` von der Zeit her in den Designprozess übertragen werden kann
   For($vlCount;1;5)
      DELAY PROCESS(Current process;1)
   End for
End if
```

PROCESS PROPERTIES

PROCESS PROPERTIES (Prozessnr ; Prozessname ; Prozesstatus ; Prozesszeit {; ProzModus {; EinmaligeID {; Ursprung}}})

Parameter	Typ		Beschreibung
Prozessnr	Lange Ganzzahl	➡	Prozessnummer
Prozessname	String	⬅	Prozessname
Prozesstatus	Lange Ganzzahl	⬅	Prozesstatus
Prozesszeit	Lange Ganzzahl	⬅	Vom Prozess angehäufte Zeit in Ticks
ProzModus	Boolean, Lange Ganzzahl	⬅	Wenn Boolean: Sichtbar (TRUE) oder ausgeblendet (FALSE) Wenn Lange Ganzzahl (bit Feld): bit 0 = Sichtbarkeit, bit 1 = Preemptive Ausführung
EinmaligeID	Lange Ganzzahl	⬅	Einmalige Prozessnummer
Ursprung	Lange Ganzzahl	⬅	Ursprung des Prozesses

Beschreibung

Der Befehl **PROCESS PROPERTIES** gibt verschiedene Informationen zum Prozess mit der in *Prozessnr* übergebenen Nummer zurück.

Nach dem Aufruf:

- gibt *Prozessname* den Namen des Prozesses zurück. Automatisch erzeugte Prozesse erhalten folgenden Namen:
  - "P_" gefolgt von einer Zahl, wenn der Prozess über das Dialogfenster **Methode ausführen** erzeugt wurde und die Option **Neuer Prozess** markiert ist.
  - "M_" oder "ML_" gefolgt von einer Zahl, wenn der Prozess in der Anwendungsumgebung erzeugt wird und die Eigenschaft **Starte Neuen Prozess** markiert ist.
  - "Web Process#" gefolgt von einer UUID, wenn der Prozess vom Web Server gestartet wurde.
  - Wurde der Prozess abgebrochen (und sein "Platz" noch nicht neu vergeben), wird der Prozessname weiter zurückgegeben. Um festzustellen, ob ein Prozess abgebrochen wurde, prüfen Sie, ob *Prozesstatus=-1* ist (siehe unten).
- *Prozesstatus* gibt den Zustand des Prozesses beim Aufruf zurück. Der Parameter kann den Wert einer vordefinierten Konstante unter dem Thema **Prozesstatus** zurückgeben:

Konstante	Typ	Wert
Aborted	Lange Ganzzahl	-1
Delayed	Lange Ganzzahl	1
Does not exist	Lange Ganzzahl	-100
Executing	Lange Ganzzahl	0
Hidden modal dialog	Lange Ganzzahl	6
Paused	Lange Ganzzahl	5
Waiting for input output	Lange Ganzzahl	3
Waiting for internal flag	Lange Ganzzahl	4
Waiting for user event	Lange Ganzzahl	2

- *Prozesszeit* gibt die ausgeführte Zeit seit dem Start des Prozesses in Ticks zurück. 1 Tick = 1/60 Sekunde.
- *ProzModus* ist optional und kann eine Variable vom Typ Boolean oder Lange Ganzzahl sein:
  - Beim Typ Boolean gibt er True zurück, wenn der Prozess sichtbar ist, False wenn er ausgeblendet ist.
  - Beim Typ Lange Ganzzahl enthält er nach Ausführen der Methode ein Bit Feld, wo die beiden ersten Bits gesetzt sind:
    - Bit 0 gibt die Eigenschaft Sichtbarkeit zurück: Es wird auf 1 gesetzt, wenn der Prozess sichtbar ist, und auf 0, wenn er ausgeblendet ist.
    - Bit 1 gibt die Eigenschaft preemptive Modus zurück: Es wird auf 1 gesetzt, wenn der Prozess im preemptive Modus läuft, und auf 0, wenn er im kooperativen Modus läuft.
      Hinweis: Diese Eigenschaft ist nur in 4D Anwendungen in 64-bit hilfreich, wo Prozesse preemptive oder kooperativ laufen können. Weitere Informationen dazu finden Sie im Abschnitt **Preemptive 4D Prozesse**.
- *EinmaligeID* ist optional. Geben Sie den Parameter an, wird die einmalige Prozessnummer zurückgegeben. Jedem Prozess wird eine Prozessnummer sowie pro Arbeitssitzung eine einmalige Prozessnummer zugewiesen. Mit dieser Nummer können Sie zwischen zwei Prozessen bzw. zwei Prozessitzungen unterscheiden. Sie entspricht der Prozessnummer, die während der Arbeitssitzung von 4D gestartet wurde.
- *Ursprung* ist optional. Geben Sie den Parameter an, wird ein Wert zurückgegeben, der den Ursprung des Prozesses beschreibt. 4D bietet unter dem Thema **Prozesstypen** folgende vordefinierten Konstanten:

Konstante	Typ	Wert	Kommentar
Apple event manager	Lange Ganzzahl	-7	
Backup process	Lange Ganzzahl	-19	
Cache manager	Lange Ganzzahl	-4	
Client manager process	Lange Ganzzahl	-31	
Compiler process	Lange Ganzzahl	-29	
Created from execution dialog	Lange Ganzzahl	3	
Created from menu command	Lange Ganzzahl	2	
Design process	Lange Ganzzahl	-2	
Event manager	Lange Ganzzahl	-8	
Execute on client process	Lange Ganzzahl	-14	
Execute on server process	Lange Ganzzahl	1	
External task	Lange Ganzzahl	-9	
HTTP Log flusher	Lange Ganzzahl	-58	
Indexing process	Lange Ganzzahl	-5	
Internal 4D server process	Lange Ganzzahl	-18	
Internal timer process	Lange Ganzzahl	-25	
Log file process	Lange Ganzzahl	-20	
Main 4D process	Lange Ganzzahl	-39	
Main process	Lange Ganzzahl	-1	
Method editor macro process	Lange Ganzzahl	-17	
Monitor process	Lange Ganzzahl	-26	
MSC process	Lange Ganzzahl	-22	
None	Lange Ganzzahl	0	
On exit process	Lange Ganzzahl	-16	
Other 4D process	Lange Ganzzahl	-10	
Other user process	Lange Ganzzahl	4	
Restore Process	Lange Ganzzahl	-21	
Serial Port Manager	Lange Ganzzahl	-6	
Server interface process	Lange Ganzzahl	-15	
SQL Method execution process	Lange Ganzzahl	-24	
Web process on 4D remote	Lange Ganzzahl	-12	
Web process with no context	Lange Ganzzahl	-3	
Web server process	Lange Ganzzahl	-13	
Worker process	Lange Ganzzahl	5	Worker Prozess, vom Benutzer gestartet

Hinweis: 4D's interne Prozesse geben einen negativen Wert, die vom Benutzer erzeugten Prozesse einen positiven Wert zurück. Ist der Prozess nicht vorhanden, bedeutet das, dass Sie im Parameter *Prozessnr* einen Wert übergeben haben, der nicht im Bereich zwischen 1 und der Anzahl vorhandener Prozesse (**Count tasks**) liegt. **PROCESS PROPERTIES** lässt die Parameter der Variablen dann unverändert.

Beispiel 1

Folgendes Beispiel gibt Name, Status und beanspruchte Zeit in den Variablen *vName*, *vState,* und *vTimeSpent* für den aktuellen Prozess zurück:

```
C_TEXT(vName)  ` Initialisiere die Variablen
C_LONGINT(vState)
C_LONGINT(vTime)
PROCESS PROPERTIES(Current process;vName;vState;vTimeSpent)
```

Beispiel 2

Siehe Beispiel zur **Semaphore**

Beispiel 3

Die Sichtbarkeit und den Ausführungsmodus für den aktuellen Prozess abfragen:

```
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  //wahr, wenn sichtbar
isPreemptive:=vFlags?? 1  //wahr, wenn preemptive
```

Process state

Process state (Prozessnr) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Prozessnr	Lange Ganzzahl	➡	Nummer des zu analysierenden Prozesses
Funktionsergebnis	Lange Ganzzahl	➲	Prozess-Status

Beschreibung

Die Funktion **Process state** gibt den Zustand des Prozesses *Prozessnr* zurück.
Das Funktionsergebnis kann ein Wert einer vordefinierten Konstante unter dem Thema **Prozesstatus** sein:

Konstante	Typ	Wert
Aborted	Lange Ganzzahl	-1
Delayed	Lange Ganzzahl	1
Does not exist	Lange Ganzzahl	-100
Executing	Lange Ganzzahl	0
Hidden modal dialog	Lange Ganzzahl	6
Paused	Lange Ganzzahl	5
Waiting for input output	Lange Ganzzahl	3
Waiting for internal flag	Lange Ganzzahl	4
Waiting for user event	Lange Ganzzahl	2

Ist der Prozess nicht vorhanden, d.h. die übergebene Nummer liegt nicht im Bereich von 1 bis **Count tasks**, gibt **Process state** den Wert *Does not exist* (-100) zurück.

Beispiel

Folgendes Beispiel setzt Namen und Referenznummer jedes Prozesses in die Arrays *asProcName* und *aiProcNum*. Die Methode prüft, ob der Prozess abgebrochen wurde. Name und Nummer dieses Prozesses werden den Arrays dann nicht hinzugefügt:

```
$vlNbTasks:=Count tasks
ARRAY TEXT(asProcName;$vlNbTasks)
ARRAY INTEGER(aiProcNum;$vlNbTasks)
$vlActualCount:=0
For($vlProcess;1;$vlNbTasks)
   If(Process state($vlProcess)>=Executing)
      $vlActualCount:=$vlActualCount+1
      PROCESS PROPERTIES($vlProcess;asProcName{$vlActualCount};$vlState;$vlTime)
      aiProcNum{$vlActualCount}:=$vlProcess
   End if
End for
  ` Entferne nicht verwendete Extra-Elemente
ARRAY TEXT(asProcName;$vlActualCount)
ARRAY INTEGER(aiProcNum;$vlActualCount)
```

REGISTER CLIENT

REGISTER CLIENT (ClientName {; Zeit}{; *})

Parameter	Typ		Beschreibung
ClientName	String	➡	Name der remote 4D Sitzung
Zeit	Lange Ganzzahl	➡	***Wird ab Version 11.3 ignoriert***
*	Operator	➡	Lokaler Prozess

Beschreibung

Der Befehl **REGISTER CLIENT** registriert ein remote 4D mit dem Namen *ClientName* auf 4D Server, so dass andere Clients oder evtl. 4D Server (über Servermethoden) über den Befehl **EXECUTE ON CLIENT** Methoden auf dieser Arbeitsstation ausführen können. Sobald ein remote 4D registriert ist, kann es eine oder mehrere Methoden für andere Clients ausführen.

Hinweise:

- Sie können auch jede Arbeitsstation, die sich an 4D Server anmeldet, automatisch registrieren. Wählen Sie dazu in den Einstellungen der Datenbank die Option "Client beim Starten registrieren".
- Verwenden Sie diesen Befehl im lokalen Modus von 4D, hat er keine Auswirkung.
- Mehrere remote 4D können denselben registrierten Namen haben.

Mit diesem Befehl wird auf der Arbeitsstation ein Prozess mit Namen *ClientName* erstellt. Dieser Prozess lässt sich nur mit dem Befehl **UNREGISTER CLIENT** abbrechen.
Übergeben Sie den optionalen Parameter *, ist der erstellte Prozess lokal. 4D stellt dem Prozessnamen automatisch das Dollarzeichen ($) voran. Andernfalls ist der Prozess global.

Hinweis zur Kompatibilität: Seit Version 11.3 von 4D sind die Abläufe zur Server/Client Kommunikation optimiert. Der Server sendet jetzt die Ausführung der Anfragen nach Bedarf direkt an die registrierten Clients (Technologie "push"). Damit entfällt das vorige Prinzip, bei dem Clients in periodischen Abständen am Server angefragt haben. Ist der Parameter *Zeit* übergeben, wird er ignoriert.

Nach Ausführung von **REGISTER CLIENT** können Sie den Namen der Arbeitsstation nicht mehr unmittelbar ändern. Dazu müssen Sie erst den Befehl **UNREGISTER CLIENT** und dann den Befehl **REGISTER CLIENT** aufrufen.

Beispiel

Im folgenden Beispiel erstellen wir ein kleines Meldungssystem, so dass die Arbeitsstationen miteinander kommunizieren können.
1) Mit der Methode ***Registration*** können Sie ein remote 4D registrieren und bereit machen, damit er Meldungen von einem anderen remote 4D empfangen kann:

```
  `Vor Registrieren unter einem anderen Namen müssen Sie zuerst die
  ` Registrierung rückgängig machen
UNREGISTER CLIENT
Repeat
   vPseudoName:=Request("Geben Sie Ihren Namen ein:";"Benutzer";"OK";"Abbrechen")
Until((OK=0)|(vPseudoName#""))
If(OK=0)
   ... ` Führe nichts aus
Else
   REGISTER CLIENT(vPseudoName)
End if
```

2) Mit folgender Anweisung erhalten Sie eine Liste des registrierten remote 4D. Sie können es in die **Datenbankmethode On Startup** setzen:

```
PrClientList:=New process("4D Client Liste";32000;"Liste der registrierten Clients")
```

3) Mit der Methode ***4D Client List*** finden Sie alle registrierten remote 4D wieder und solche, die Meldungen empfangen können:

```
If(Application type=4D Remote Mode)
  ` Nachfolgender Code ist nur im Client/Server-Betrieb gültig
   $Ref:=Open window(100;100;300;400;-(Palette window+Has window title);"Liste der registrierten Clients")
   Repeat
      GET REGISTERED CLIENTS($ClientList;$ListeCharge)
  `Finde die registrierten Clients wieder in $ClientList
      ERASE WINDOW($Ref)
      GOTO XY(0;0)
      For($p;1;Size of array($ClientList))
         MESSAGE($ClientList{$p}+Char(Carriage return))
      End for
  `Zeige jede Sekunde an
      DELAY PROCES(Current process;60)
```

```
    Until(False)  ` Endlos-Schleife
End if
```

4) Mit folgender Methode senden Sie eine Meldung an einen anderen registrierten remote 4D. Sie ruft die Methode ***Display_Message*** auf (siehe unten).

```
$Address:=Request("Empfänger der Meldung:";"")
  ` Gib Namen der Personen aus dem Fenster ein,
  ` das die Datenbankmethode On Startup erstellt
If(OK#0)
    $Message:=Request("Message:")  ` Meldung
    If(OK#0)
        EXECUTE ON CLIENT($Address;"Zeige_Meldung";$Message)  ` Sende Meldung
    End if
End if
```

5) Hier die Methode ***Display_Message***:

```
C_TEXT($1)
ALERT($1)
```

6) Mit dieser Methode ist eine Arbeitsstation für die anderen remote 4D nicht mehr sichtbar, sie erhält auch keine Meldungen mehr:

```
UNREGISTER CLIENT
```

Systemvariablen und Mengen

Wurde remote 4D korrekt registriert, hat die Systemvariable OK den Wert 1. Wurde remote 4D bereits registriert, führt der Befehl nichts aus, die Systemvariable OK hat den Wert 0 (Null).

RESUME PROCESS

RESUME PROCESS (Prozessnr)

Parameter	Typ		Beschreibung
Prozessnr	Lange Ganzzahl	➡	Wieder zu aktivierender Prozess

Beschreibung

Der Befehl **RESUME PROCESS** reaktiviert einen angehaltenen oder verzögerten Prozess. Ist der Prozess beim Aufruf nicht angehalten oder verzögert, hat der Befehl keine Wirkung.

Wurde *Prozessnr* zuvor verzögert, schlagen Sie bei den Befehlen **PAUSE PROCESS** oder **DELAY PROCESS** nach. Ist *Prozessnr* nicht vorhanden, hat der Befehl keine Wirkung.

Hinweis: Sie können mit diesem Befehl nicht von einem Client-Rechner aus eine Serverprozedur auf dem Server-Rechner zuweisen (*Prozessnr*<0).

UNREGISTER CLIENT

UNREGISTER CLIENT
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **UNREGISTER CLIENT** hebt die Registrierung eines remote 4D auf. Der Client muss zuvor mit dem Befehl **REGISTER CLIENT** registriert sein.

Hinweis: Die Registrierung eines remote 4D wird automatisch aufgehoben, wenn der Benutzer die Anwendung beendet.

Wurde die Arbeitsstation zuvor nicht registriert oder wird der Befehl im lokalen Modus von 4D ausgeführt, hat der Befehl keine Auswirkung.

Beispiel

Siehe Beispiel zum Befehl **REGISTER CLIENT**.

Systemvariablen und Mengen

Wurde die Registrierung des Client korrekt aufgehoben, hat die Systemvariable OK den Wert 1. War der Client nicht registriert, hat OK den Wert 0 (Null).

Prozessoberfläche

- BRING TO FRONT
- Frontmost process
- HIDE PROCESS
- SHOW PROCESS

BRING TO FRONT

BRING TO FRONT (Prozessnr)

Parameter	Typ		Beschreibung
Prozessnr	Lange Ganzzahl	➡	Nummer des Prozesses, dessen Fenster nach vorne geholt werden soll

Beschreibung

Der Befehl **BRING TO FRONT** holt die Fenster des Prozesses nach vorne.

Ist der Prozess bereits der vorderste Prozess, hat der Befehl keine Auswirkung. Ist der Prozess ausgeblendet, müssen Sie ihn mit **SHOW PROCESS** anzeigen, damit **BRING TO FRONT** aktiv wird. Ist *Prozessnr* ausgeblendet, hat der Befehl keine Wirkung.

Mit diesem Befehl lassen sich auch die Haupt- und Designprozesse nach vorne holen.

Hinweis: Hat der Prozess mehrere Fenster und wollen sie ein bestimmtes Fenster nach vorne holen, ist es besser, z.B. den Befehl **SET WINDOW RECT** zu verwenden.

Beispiel

Folgendes Beispiel ist eine Methode, die von einem Menü aus ausgeführt werden kann. Sie prüft, ob ◇*vlAddCust_PID* der vorderste Prozess ist. Wenn nicht, bringt diese Methode ihn nach vorne:

```
If(Frontmost process#◇vlAddCust_PID)
   BRING TO FRONT(◇vlAddCust_PID)
End if
```

Frontmost process

Frontmost process {(*)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	→	Prozessnummer des ersten Nicht-Palettenfensters
Funktionsergebnis	Ganzzahl	↩	Nummer des Prozesses, dessen Fenster vorne liegen

Beschreibung

Die Funktion **Frontmost process** gibt die Prozessnummer des Prozesses des vordersten Fensters zurück.

Ist das vorderste Fenster ein Palettenfenster, wird die Nummer des dazugehörigen Prozesses zurückgegeben.

Wird ein * als Parameter übergeben, wird die Nummer des Prozesses zurückgegeben, zu dem das vorderste Standardfenster gehört.

Beispiel

Siehe Beispiel zum Befehl **BRING TO FRONT**.

HIDE PROCESS

HIDE PROCESS (Prozessnr)

Parameter	Typ		Beschreibung
Prozessnr	Lange Ganzzahl	➡	Nummer des auszublendenden Prozesses

Beschreibung

HIDE PROCESS blendet alle Fenster aus, die zum Prozess *Prozessnr* gehören, bis zum Aufruf von **SHOW PROCESS**. Alle Oberflächenelemente werden unsichtbar.

Einzige Ausnahme ist das Fenster für den Schrittmodus. Wird dieses Fenster aufgerufen, während der Prozess ausgeblendet ist, werden alle Prozessfenster eingeblendet. Der Prozess ändert seinen Status und das oberste Fenster wird sein aktuelles Fenster.

Auf einen Designprozess oder den Hauptprozess ist **HIDE PROCESS** nicht anwendbar.

Der Prozess wird nur unsichtbar gemacht. Er bleibt weiter aktiv und wird weiterhin ausgeführt.

Beispiel

Folgendes Beispiel blendet alle Fenster des laufenden Prozesses aus:

```
HIDE PROCESS(Current process)
```

SHOW PROCESS

SHOW PROCESS (Prozessnr)

Parameter	Typ		Beschreibung
Prozessnr	Lange Ganzzahl	➡	Nummer des einzublendenden Prozesses

Beschreibung

SHOW PROCESS zeigt alle Fenster des Prozesses *Prozessnr* an. Er bringt die Fenster von *Prozessnr* aber nicht nach vorne. Dafür müssen Sie zusätzlich den Befehl **BRING TO FRONT** aufrufen.

Sind die Fenster des Prozesses nicht ausgeblendet, macht dieser Befehl nichts.

Beispiel

Folgendes Beispiel zeigt einen Prozess mit Namen Kunden, der zuvor ausgeblendet war. Die Prozessreferenz auf den Prozess Kunden ist in der Interprozessvariablen ◇*Customers* gespeichert:

```
SHOW PROCESS(◇Customers)
```

Rechtschreibprüfung

- 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 (Wort)

Parameter	Typ		Beschreibung
Wort	Text, Array Text	➡	Wörter oder Wortlisten zum Hinzufügen im Benutzerwörterbuch

Beschreibung

Der Befehl **SPELL ADD TO USER DICTIONARY** fügt ein oder mehrere Wörter im aktuellen Benutzerwörterbuch hinzu.

Ein Benutzerwörterbuch ist ein Wörterbuch, in das der Benutzer eigene Wörter hinzugefügt hat. Es ist eine Datei mit Namen *UserDictionaryxxx.dic*, wobei xxx die ID des aktuellen Wörterbuchs ist. Sie wird automatisch im aktuellen 4D Ordner angelegt. Für jedes aktuell verwendete Wörterbuch gibt es ein Benutzerwörterbuch.

In *Wort* übergeben Sie einen Text String oder ein Text Array mit dem bzw. den eigenen Wörtern, die Sie im Benutzerwörterbuch hinzufügen wollen. Ist ein Wort bereits vorhanden, wird es ignoriert.

Beispiel

Eigennamen im Benutzerwörterbuch hinzufügen:

```
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 (Text ; errPos ; errLength ; checkPos ; arrVorschlag)

Parameter	Typ		Beschreibung
Text	Text	➡	Text zum Prüfen
errPos	Lange Ganzzahl	⬅	Position des ersten Zeichens des unbekannten Worts
errLength	Lange Ganzzahl	⬅	Länge des unbekannten Worts
checkPos	Lange Ganzzahl	➡	Startposition für die Prüfung
arrVorschlag	Array Text	⬅	Liste der Vorschläge

Beschreibung

Der Befehl **SPELL CHECK TEXT** prüft den Inhalt des Parameters *Text* ab den Zeichen *checkPos* und gibt die Position des ersten gefundenen unbekannten Worts zurück - sofern das zutrifft.

Dieser Befehl gibt die Position des ersten Zeichens dieses unbekannten Worts in *errPos* und seine Länge in *errLength* zurück. Das Array *arrVorschlag* empfängt die Korrekturvorschläge, welche die Rechtschreibprüfung anbietet.

Startet die Prüfung ohne Fehler und wird ein unbekanntes Wort gefunden, wird die Systemvariable OK auf 0 gesetzt. Tritt während der Prüfung ein Initialisierungsfehler auf oder werden keine unbekannten Wörter gefunden, wird OK auf 1 gesetzt.

Hinweis: Ist in OS X die native Rechtschreibprüfung aktiviert, unterstützt dieser Befehl keine Grammatikkorrekturen.

Beispiel

Die Anzahl möglicher Fehler in einem Text zählen:

```
$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  // Zähle alle Fehler
      $errorWord:=Substring($myText;$errPos;$errLength)
      APPEND TO ARRAY($errors;$errorWord)  // Array der Fehler
      $pos:=$errPos+$errLength   //weiter prüfen
   End if
Until(OK=1)
 // Am Ende von $errCount=Size of array($errorWord)
```

SPELL CHECKING

SPELL CHECKING
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **SPELL CHECKING** löst die Rechtschreibprüfung im hervorgehobenen Feld bzw. in der Variablen im aktuell angezeigten Formular aus. Das markierte Objekt muss vom Typ String oder Text sein.

Hinweis: Wollen Sie die Rechtschreibprüfung durch Anklicken einer Schaltfläche im Formular auslösen, achten Sie darauf, dass diese Schaltfläche nicht die Eigenschaft "Fokusfähig" hat.

Die Rechtschreibprüfung startet mit dem ersten Wort in Bereich. Wird ein unbekanntes Wort gefunden, erscheint das Dialogfenster der Rechtschreibhilfe. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*. 4D verwendet das aktuelle Wörterbuch, welches zur Sprache der Anwendung gehört – außer Sie haben über den Befehl **SPELL SET CURRENT DICTIONARY** ein anderes definiert.

Warnung: SPELL CHECKING gilt für im Formular eingegebenen Text und nicht für die zugeordnete Datenquelle (Feld oder Variable), d.h. wenn Sie diesen Befehl über die Formularereignisse On Data Change oder On Losing Focus aufrufen (nicht empfohlen), gilt er nicht für den gespeicherten Text, da 4D an dieser Stelle den eingegebenen Text bereits der Datenquelle zugewiesen hat. In diesem Fall müssen Sie das bearbeitete Ergebnis über die Funktion **Get edited text** selbst der Datenquelle zuweisen. Zum Beispiel:

```
If(Form event=On Data Change)
   SPELL CHECKING
   theVariable:=Get edited text
End if
```

SPELL Get current dictionary

SPELL Get current dictionary -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		ID des Wörterbuchs zur Rechtschreibprüfung

Beschreibung

Die Funktion **SPELL Get current dictionary** gibt die ID Nummer des verwendeten Wörterbuchs zurück.

Beispiel

Die Sprache des aktuellen Wörterbuchs anzeigen:

```
 // Liste der geladenen Wörterbücher
SPELL GET DICTIONARY LIST($IDs_al;$Codes_at;$Names_at)
$curLangCode:=SPELL Get current dictionary
$countryName:=$Names_at{Find in array($IDs_al;$curLangCode)}
 // Meldung anzeigen
ALERT("Current dictionary: "+$countryName)  // Spanisch
```

SPELL GET DICTIONARY LIST

SPELL GET DICTIONARY LIST (SprachID ; SprachDateien ; SprachNamen)

Parameter	Typ		Beschreibung
SprachID	Array Lange Ganzzahl	⇐	Einmalige ID der Sprachen
SprachDateien	Array Text	⇐	Namen der installierten Sprachdateien
SprachNamen	Array Text	⇐	Lokale Namen der Sprachen

Beschreibung

Der Befehl **SPELL GET DICTIONARY LIST** gibt in den Arrays *SprachID*, *SprachDateien* und *SprachNamen* die IDs, Dateinamen und Sprachnamen der Hunspell Wörterbuchdateien zurück, die auf dem Rechner installiert sind.

Weitere Informationen zu Hunspell Wörterbüchern finden Sie im Abschnitt **Rechtschreibprüfung** des Handbuchs *4D Designmodus*.

- *SprachID* empfängt die ID Nummern, die automatisch generiert und mit dem Befehl **SPELL SET CURRENT DICTIONARY** verwendet werden. Die IDs sind einmalig und basieren auf den Dateinamen. Der Befehl ist besonders während der Entwicklung hilfreich; Sie müssen die IDs nicht jedes Mal, wenn die Datenbank ausgeführt wird, erneut generieren.
- *SprachDateien* empfängt die Namen der Wörterbuchdateien (ohne Endung), die auf dem Rechner installiert sind.
- *SprachNamen* empfängt die Sprachnamen, ausgedrückt in der aktuellen Anwendungssprache. Zum Beispiel wird für ein deutsches Wörterbuch auf einem Rechner mit deutschem System der Wert "deutsch" (Germany), auf einem englischen System der Wert "German (Germany) zurückgegeben.
  Für Hunspell Wörterbücher wird an den Sprachnamen "-Hunspell" angehängt. Dieses Feld ist nur für Dateien gültig, die 4D "kennt". Für unbekannte Dateien, z.B. eigene Dateien, wird der Name "N/A - Hunspell" zurückgegeben. Sie können das Wörterbuch trotzdem verwenden, wenn die Datei als gültig gewertet wird. Die zurückgegebene ID lässt sich auch im Befehl **SPELL SET CURRENT DICTIONARY** übergeben.

Beispiel

"fr-classic+reform1990.aff" und "fr-classic+reform1990.dic" sowie "fr-dentist.aff" und "fr-dentist.dic" in ein Hunspell Verzeichnis setzen:

```
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 (Wörterbuch)

Parameter	Typ		Beschreibung
Wörterbuch	Lange Ganzzahl, Text	➡	ID oder Name des Wörterbuchs, das verwendet werden soll

Beschreibung

Der Befehl **SPELL SET CURRENT DICTIONARY** ersetzt das aktuelle Wörterbuch durch das in *Wörterbuch* angegebene Wörterbuch. Die in 4D sowie in 4D Write Pro integrierte Rechtschreibhilfe verwendet zur Prüfung das aktuelle Wörterbuch. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*. Die Änderung des aktuellen Wörterbuchs gilt für alle Prozesse der Anwendung für die Arbeitssitzung, sowie für 4D Write Pro Bereiche.

4D verwendet standardmäßig

- Unter Windows das Hunspell Wörterbuch in der Sprache der Anwendung
- Auf Mac die native Rechtschreibhilfe

Hinweis: Für Mac können Sie über den Befehl **SET DATABASE PARAMETER** das Hunspell Wörterbuch verwenden. Weitere Informationen dazu finden Sie im Abschnitt **Rechtschreibprüfung konfigurieren** des Handbuchs *4D Designmodus*.

Mit dem Parameter *Wörterbuch* können Sie das Wörterbuch wechseln. Sie können folgendes übergeben:

- Hunspell Wörterbuch ID Nummer (zurückgegeben von der Funktion **SPELL GET DICTIONARY LIST**)
- Hunspell Wörterbuchname (entspricht dem Namen der Datei Hunspell Wörterbuch, mit oder ohne seine Endung)
- Sprachcode BCP 47, ISO 639-1 oder ISO 639-2. Zum Beispiel: mit dem Sprachcode BCP 47 gibt "en-US" amerikanisches Englisch und "en-GB" britisches Englisch an. Diese Codes werden intern zum entsprechenden aktuellen Wörterbuch umgeleitet (Hunspell oder native Mac).

Hinweis zur Kompatibilität: Bisherige Versionen haben Cordial Wörterbücher unterstützt. Zur Wahrung der Kompatibilität lässt sich im Parameter *Wörterbuch* weiterhin eine "Cordial" Wörterbuchnummer übergeben (Wert oder Konstante unter dem Thema **Wörterbücher**). Das Wörterbuch wird aber dann intern zum entsprechenden aktuellen Hunspell Wörterbuch umgeleitet (oder native Mac Wörterbuch).

Systemvariablen und Mengen

Wurde das Wörterbuch korrekt geladen, hat die Systemvariable OK den Wert 1, sonst den Wert 0 (Null). Dann wird ein Fehler zurückgegeben.

Beispiel

Das Wörterbuch "fr-classic" aus dem Ordner Hunspell laden:

```
SPELL SET CURRENT DICTIONARY("fr-classic")
 // SPELL SET CURRENT DICTIONARY ("FR-classic.dic") ist gültig
```

Ressourcen

- Einführung in Ressourcen
- 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*

Einführung in Ressourcen

Kompatibilitätshinweis zum Schreiben von Ressourcen (4D v13)

Wie bereits mit dem Release von 4D v11 angekündigt (siehe nächster Absatz), sind die Mechanismen zum Verwenden von Ressourcen Dateien überholt. **Befehle im Kapitel "Ressourcen"zum Schreiben von Ressourcendateien dürfen nicht mehr verwendet werden; sie werden in zukünftigen 4D Versionen entfernt.** Zur Wahrung der Kompatibilität werden Befehle zum Lesen von Ressourcen beibehalten.

Angepasste Arbeitsweise (4D v11)

Die Ressourcen-Verwaltung hat sich in 4D ab Version 11 geändert. Gemäß den Apple-Richtlinien, die auch in den neuesten Mac OS Versionen zum Tragen kommen, ist das Arbeiten mit Ressourcen überholt und wird nach und nach aufgegeben. Es wurden neue Funktionalitäten eingeführt, um die bisher von Ressourcen abgedeckten Anforderungen zu übernehmen: XLIFF Dateien zum Übersetzen von Text, Bilder im Format .png, etc. Ressource-Dateien werden bevorzugt durch standardmäßige Dateitypen ersetzt. 4D unterstützt diese Entwicklung und bietet neue Werkzeuge, um Übersetzungen der Datenbank zu verwalten, behält aber gleichzeitig die Kompatibilität zu vorhandenen Systemen bei.

Kompatibilität vorhandener Ressourcen

Zur Wahrung der Kompatibilität und um die schrittweise Anpassung vorhandener Anwendungen zu ermöglichen, bleibt die bisherige Arbeitsweise mit Ressourcen in 4D v11 bestehen. Es gibt jedoch ein paar wichtige Unterschiede:

- Zum gegenwärtigen Zeitpunkt unterstützt 4D weiterhin Ressource Dateien und das Öffnen mehrerer Ressource Dateien. Die .rsr oder .4dr Dateien konvertierter Datenbanken werden weiterhin automatisch geöffnet. Eigene Ressource Dateien können über Befehle aus diesem Kapitel geöffnet werden.
- Aufgrund der Weiterentwicklung der internen Architektur ist es jedoch nicht länger möglich, auf die Ressourcen der 4D Anwendung oder des Systems direkt zuzugreifen und zwar weder über Befehle aus diesem Kapitel noch über dynamische Referenzen. Manche Entwickler verwenden interne Ressourcen von 4D für ihre Oberfläche, z.B. Ressourcen mit den Monatsnamen oder Ressourcen der Programmiersprache. Diese Praxis wurde bisher schon nicht empfohlen und ist jetzt technisch nicht mehr möglich. In den meisten Fällen lassen sich anstelle interner Ressourcen andere Elemente verwenden, z.B. Konstanten, Befehle der Programmiersprache. Um die Auswirkung dieser Änderung auf vorhandene Datenbanken zu begrenzen, wurde ein Substitutionssystem eingerichtet, das die am häufigsten verwendeten Ressourcen quasi zu externen Ressourcen macht. Trotzdem raten wir dringend, konvertierte Datenbanken abzuändern und alle Aufrufe interner 4D Ressourcen daraus zu entfernen.
- Mit 4D v11 erstellte Datenbanken enthalten standardmäßig keine .RSR (Struktur Ressourcen) und .4DR (Daten Ressourcen) Dateien.

Neue Vorgehensweise bei Ressourcen

In 4D v11 ist der Begriff “Ressourcen” jetzt im weiteren Sinne zu verstehen als “Dateien, die zur Übersetzung von Oberflächen der Anwendung notwendig sind.”
Die aktuelle Architektur mit Ressourcen basiert auf einem Ordner mit Namen **Resources**, der neben der Strukturdatei der Datenbank liegen muss (.4db oder .4dc). In diesen Ordner müssen Sie alle Dateien ablegen, die zur Übersetzung oder Anpassung der Anwendungsoberfläche notwendig sind. Das sind Bilddateien, Textdateien, XLIFF-Dateien, usw.
Er kann auch “frühere Generationen” von Ressource Dateien der Datenbank enthalten (.rsr Dateien). Beachten Sie jedoch, dass diese Dateien nicht mehr automatisch geladen werden; Sie müssen diese mit Standardbefehlen zum Verwalten von 4D Ressourcen öffnen. 4D verwendet automatische Abläufe beim Arbeiten mit dem Inhalt dieses Ordners, insbesondere zum Verwalten von XLIFF Dateien. Weitere Informationen dazu finden Sie im **PICTURE TO BLOB** des Handbuchs *4D Designmodus*.
Zur Wahrung der Kompatibilität können Sie beiden Befehle **Get indexed string** und **STRING LIST TO ARRAY** für die Vorteile dieser Vorgehensweise nutzen. Wir empfehlen jedoch, jetzt die Funktion **Get localized string** aus dem Kapitel "String" zu verwenden.

CLOSE RESOURCE FILE

CLOSE RESOURCE FILE (ResDatei)

Parameter	Typ		Beschreibung
ResDatei	DokRef	➡	Referenznummer für Ressourcendatei

Beschreibung

Der Befehl **CLOSE RESOURCE FILE** schließt die Ressourcendatei mit der Referenznummer *ResID*.

Auch wenn Sie die gleiche Ressourcendatei mehrmals geöffnet haben, müssen Sie einmal **CLOSE RESOURCE FILE** aufrufen, um sie zu schließen.

Wenden Sie **CLOSE RESOURCE FILE** auf Ressourcendateien der 4D Anwendung oder der Datenbank an, bemerkt der Befehl das und führt nichts aus.

Übergeben Sie eine ungültige Referenznummer für die Ressourcendatei, hat der Befehl keine Auswirkung.

Vergessen Sie nicht, die mit **Open resource file** geöffnete Ressourcendatei über den Befehl **CLOSE RESOURCE FILE** wieder zu schließen. Verlassen Sie jedoch die Anwendung oder öffnen eine andere Datenbank, schließt 4D automatisch alle geöffneten Ressourcendateien.

GET ICON RESOURCE

GET ICON RESOURCE (ResID ; ResDaten {; ResDatei})

Parameter	Typ		Beschreibung
ResID	Lange Ganzzahl	➡	Kennumer der Icon-Ressource
ResDaten	Bild	➡	Bildfeld oder Variable zum Empfangen des Bildes
		⬅	Inhalt der Ressource cicn
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe alle geöffneten Ressourcendateien

Hinweis zur Kompatibilität

Dieser Befehl wird in 4D 64-bit Versionen nicht mehr unterstützt. Er gibt beim Ausführen in dieser Umgebung einen Fehler zurück.

Beschreibung

Der Befehl **GET ICON RESOURCE** gibt im Bilddatenfeld oder der Variablen *ResDaten* das Icon aus der Farbicon-Ressource ("cicn") mit der Kennummer *ResID* zurück.

Wird die Ressource nicht gefunden, bleibt der Parameter *ResDaten* unverändert, die OK Variable wird auf 0 (Null) gesetzt.

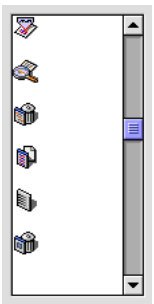
Übergeben Sie in *ResDatei* eine gültige Referenznummer für die Ressourcendatei, wird die Ressource nur in dieser Datei gesucht. Übergeben Sie *ResDatei* nicht, wird das erste Vorkommen der Ressource in der Ressourcendatei-Kette zurückgegeben.

Beispiel

Folgendes Beispiel lädt in ein Array vom Typ Bild die Farbicons aus der aktiven 4D Anwendung:

```
If(On Windows)
   $vh4DResFile:=Open resource file(Replace string(Application file;".EXE";".RSR"))
Else
   $vh4DResFile:=Open resource file(Application file)
End if
RESOURCE LIST("cicn";$alResID;$asResName;$vh4DResFile)
$vlNbIcons:=Size of array($alResID)
ARRAY PICTURE(ag4DIcon;$vlNbIcons)
For($vlElem;1;$vlNbIcons)
   GET ICON RESOURCE($alResID{$vlElem};ag4DIcon{$vlElem};$vh4DResFile)
End for
```

Nach Ausführung dieses Code erscheint das Array in einem Formular folgendermaßen:



Systemvariablen und Mengen

Wird die Ressource gefunden, wird OK auf 1 gesetzt, sonst auf 0 (Null).

Get indexed string

Get indexed string (ResID ; StrID {; ResDatei}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ResID	Lange Ganzzahl	➡	Kennummer der Ressource oder 'id' Attribut des 'group' Elements (XLIFF)
StrID	Lange Ganzzahl	➡	Stringnummer oder 'id' Attribut des 'trans-unit' Elements (XLIFF)
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe: alle XLIFF Dateien oder offene Ressourcendateien.
Funktionsergebnis	String	⮌	Wert des indizierten String

Beschreibung

Der Befehl **Get indexed string** gibt folgendes zurück:

- Einen String aus der Stringlisten-Ressource (“STR#”) mit der Kennummer *ResID*.
  ODER
- Einen String aus einer offenen XLIFF Datei mit dem 'id' Attribut des 'group' Elements, übergeben in *ResID* (siehe unten).

Sie übergeben die Stringnummer in *StrID*. Die Strings in der Stringlisten-Ressource sind numeriert von 1 bis N. Wollen Sie alle Strings (und ihre Nummern) aus der String-Listen Ressource erhalten, verwenden Sie den Befehl **STRING LIST TO ARRAY**. Wird die Ressource oder der String innerhalb einer Ressource nicht gefunden, wird ein leerer String zurückgegeben, die OK Variable wird auf 0 (Null) gesetzt.

Übergeben Sie in *ResDatei* eine gültige Referenznummer für die Ressourcendatei, wird die Ressource nur in dieser Datei gesucht. Übergeben Sie *ResDatei* nicht, wird das erste Vorkommen der Ressource in der Ressourcendatei-Kette zurückgegeben.

Hinweis: Jeder String einer Stringlisten-Ressource kann bis zu 255 Zeichen enthalten.

Kompatibilität mit XLIFF Architektur

Die Funktion **Get indexed string** ist ab 4D Version 11 mit der XLIFF Architektur kompatibel: Die Funktion sucht zuerst in allen offenen XLIFF Dateien nach Werten, die *ResID* und *StrID* entsprechen (wenn der Parameter *ResDatei* nicht übergeben ist). *ResID* bestimmt dann das id-Attribut des group-Elements und *StrID* das id-Attribut des trans-unit-Elements.
Wird der Wert nicht gefunden, setzt die Funktion die Suche in den offenen Ressource-Dateien fort. Weitere Informationen dazu finden Sie im **PICTURE TO BLOB** des Handbuchs *4D Designmodus*.

Systemvariablen und Mengen

Wird die Ressource gefunden, wird OK auf 1 gesetzt, sonst auf Null (0).

GET PICTURE RESOURCE

GET PICTURE RESOURCE (ResID ; ResDaten {; ResDatei})

Parameter	Typ		Beschreibung
ResID	Lange Ganzzahl	➡	Kennumer der Ressource
ResDaten	Feld, Variable	➡	Bildfeld oder Variable zum Empfangen des Bildes
		⬅	Inhalt der Ressource PICT
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe alle geöffneten Ressourcendateien

Beschreibung

Der Befehl **GET PICTURE RESOURCE** gibt im Bilddatenfeld oder der Variablen *ResDatei* das Bild aus der Bildressource "PICT" mit der Nummer *ResID* zurück.

Wird die Ressource nicht gefunden, bleibt der Parameter *ResDaten* unverändert, die OK Variable wird auf 0 (Null) gesetzt.

Übergeben Sie in *ResDatei* eine gültige Referenznummer für die Ressourcendatei, wird die Ressource nur in dieser Datei gesucht. Übergeben Sie *ResDatei* nicht, wird das erste Vorkommen der Ressource in der Ressourcendatei-Kette zurückgegeben.

Hinweis: Eine Bildressource kann mehrere Megabytes groß sein.

Beispiel

Siehe Beispiel zum Befehl **RESOURCE LIST**.

Systemvariablen und Mengen

Wird die Ressource gefunden, wird OK auf 1 gesetzt, sonst auf 0 (Null).

Fehlerverwaltung

Reicht der Speicher nicht aus, um das Bild zu laden, wird ein Fehler erzeugt. Sie können diesen Fehler mit **ON ERR CALL** in einer Fehlerverwaltungsmethode ausfindig machen.

GET RESOURCE

GET RESOURCE (ResTyp ; ResID ; ResDaten {; ResDatei})

Parameter	Typ		Beschreibung
ResTyp	String	➡	Ressourcentyp mit 4 Zeichen
ResID	Lange Ganzzahl	➡	Kennummer der Ressource
ResDaten	BLOB	➡	BLOB Feld oder Variable zum Empfangen der Daten
		⬅	Inhalt der Ressource
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe alle geöffneten Ressourcendateien

Beschreibung

Der Befehl **GET RESOURCE** gibt im BLOB Datenfeld bzw. der Variablen *ResDatei* den Inhalt der Ressource vom Typ *ResTyp* und der Nummer *ResID* zurück.

Wichtig: Sie müssen in *ResTyp* einen String mit vier Zeichen übergeben.

Wird die Ressource nicht gefunden, bleibt der Parameter *ResDaten* unverändert, die OK Variable wird auf 0 (Null) gesetzt.

Übergeben Sie in *ResDatei* eine gültige Referenznummer für die Ressourcendatei, wird die Ressource nur in dieser Datei gesucht. Übergeben Sie *ResDatei* nicht, wird das erste Vorkommen der Ressource in der Ressourcendatei-Kette zurückgegeben.

Hinweis: Eine Ressource kann mehrere Megabytes groß sein.

Plattformunabhängigkeit

Bedenken Sie, dass Sie mit Ressourcen arbeiten, die auf Mac OS basieren. Interne Ressourcendaten z.B. vom Typ Lange Ganzzahl werden, unabhängig von der Plattform, mit der Macintosh Byte Anordnung gespeichert. Unter Windows werden low-Bytes und high-Bytes bei Daten für die Standardressourcen (Stringlisten-, Bild-Ressourcen, etc.) bei Bedarf automatisch vertauscht. Verwenden Sie eigene interne Datenstrukturen, bleibt es Ihnen überlassen, ob Sie die Byte-Anordnung der Daten aus dem BLOB vertauschen. Sie können Macintosh byte ordering z.B. der Funktion **BLOB to longint** übergeben.

Beispiel

Siehe Beispiel zum Befehl **_o_SET RESOURCE**.

Systemvariablen und Mengen

Wird die Ressource gefunden, wird OK auf 1 gesetzt, sonst auf 0 (Null).

Fehlerverwaltung

Reicht der Speicher nicht aus, um die Ressource zu laden, wird ein Fehler erzeugt. Sie können diesen Fehler mit **ON ERR CALL** in einer Fehlerverwaltungsmethode ausfindig machen.

Get resource name

Get resource name (ResTyp ; ResID {; ResDatei}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ResTyp	String	➡	Ressourcentyp mit 4 Zeichen
ResID	Lange Ganzzahl	➡	Kennummer der Ressource
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe alle geöffneten Ressourcendateien
Funktionsergebnis	String	⮌	Name der Ressource

Beschreibung

Die Funktion **Get resource name** gibt den Namen der Ressource vom Typ *ResTyp* und mit der Kennummer *ResID* zurück.

Übergeben Sie in *ResDatei* eine gültige Referenznummer für die Ressourcendatei, wird die Ressource nur in dieser Datei gesucht. Übergeben Sie *ResDatei* nicht, wird die Ressource in den derzeit geöffneten Ressourcendateien gesucht.

Gibt es die Ressource nicht, gibt **Get resource name** einen leeren String zurück.

Get resource properties

Get resource properties (ResTyp ; ResID {; ResDatei}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ResTyp	String	→	Ressourcentyp mit 4 Zeichen
ResID	Lange Ganzzahl	→	Kennummer der Ressource
ResDatei	DokRef	→	Referenznummer der Ressourcendatei Ohne Angabe alle geöffneten Ressourcendateien
Funktionsergebnis	Lange Ganzzahl	↩	Attribute der Ressource

Beschreibung

Die Funktion **Get resource properties** gibt die Attribute der Ressource vom Typ *ResTyp* und mit der Kennummer *ResID* zurück.

Übergeben Sie in *ResDatei* eine gültige Referenznummer für die Ressourcendatei, wird die Ressource nur in dieser Datei gesucht. Übergeben Sie *ResDatei* nicht, wird die Ressource in der aktuell geöffneten Ressourcendatei gesucht.

Ist die Ressource nicht vorhanden, gibt die Funktion Null (0) zurück. Die OK Variable wird auf Null (0) gesetzt.

Der von **Get resource properties** zurückgegebene numerische Wert ist ein Bit Feldwert, dessen Bits spezielle Bedeutung haben.

Beispiel

Siehe Beispiel zur Funktion **Get resource name**.

Systemvariablen und Mengen

Ist die Ressource vorhanden, wird OK auf 1 gesetzt, sonst auf Null (0).

Get string resource

Get string resource (ResID {; ResDatei}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ResID	Lange Ganzzahl	→	Kennummer der Ressource
ResDatei	DokRef	→	Referenznummer der Ressource Ohne Angabe alle geöffneten Ressourcendateien
Funktionsergebnis	String	↩	Inhalt der Ressource STR

Beschreibung

Die Funktion **Get string resource** gibt den String aus der String-Ressource (“STR ”) mit der Nummer *ResID* zurück.

Wird die Ressource oder der String innerhalb einer Ressource nicht gefunden, wird ein leerer String zurückgegeben, die OK Variable wird auf Null (0) gesetzt.

Übergeben Sie in *ResDatei* eine gültige Referenznummer für die Ressourcendatei, wird die Ressource nur in dieser Datei gesucht. Übergeben Sie *ResDatei* nicht, wird das erste Vorkommen der Ressource in der Ressourcendatei-Kette zurückgegeben.

Hinweis: Eine String-Ressource kann bis zu 255 Zeichen enthalten.

Beispiel

Folgendes Beispiel zeigt den Inhalt der String-Ressource NR=20911 an. Er muss mindestens in einer der derzeit geöffneten Ressourcendateien enthalten sein:

```
ALERT(Get string resource(20911))
```

Systemvariablen und Mengen

Wird die Ressource gefunden, wird OK auf 1 gesetzt, sonst auf Null (0).

Get text resource

Get text resource (ResID {; ResDatei}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ResID	Lange Ganzzahl	→	Kennummer der Ressource
ResDatei	DokRef	→	Referenznummer der Ressourcendatei Ohne Angabe alle geöffneten Ressourcendateien
Funktionsergebnis	Text	↩	Inhalt der Ressource TEXT

Beschreibung

Die Funktion **Get text resource** gibt den Text aus der Ressource "TEXT" zurück mit der Kennummer *ResID*.

Wird die Ressource nicht gefunden, wird leerer Text zurückgegeben, die OK Variable wird auf 0 (Null) gesetzt.

Übergeben Sie in *ResDatei* eine gültige Referenznummer für die Ressourcendatei, wird die Ressource nur in dieser Datei gesucht. Übergeben Sie *ResDatei* nicht, wird das erste Vorkommen der Ressource in der Ressourcendatei-Kette zurückgegeben.

Hinweis: Eine Text-Ressource kann bis zu 32.000 Zeichen haben.

Beispiel

Folgendes Beispiel zeigt den Inhalt der Text-Ressource NR=20800. Er muss mindestens in einer der derzeit geöffneten Ressourcendateien enthalten sein:

```
ALERT(Get text resource(20800))
```

Systemvariablen und Mengen

Wird die Ressource gefunden, wird OK auf 1 gesetzt, sonst auf 0 (Null).

Open resource file

Open resource file (ResDateiName {; Dateityp}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ResDateiName	String	➡	Kurzer bzw. langer Name der Ressourcendatei oder leerer String für Standard-Öffnen Dialog für Datei
Dateityp	String	➡	Mac OS Dateityp (4-Zeichen String) oder Windows Dateierweiterung (1- bis 3-Zeichen String) ohne Angabe alle Dateien
Funktionsergebnis	DokRef	⬅	Referenznummer der Ressourcendatei

Beschreibung

Die Funktion **Open resource file** öffnet die Ressourcendatei mit dem Namen bzw. Pfadnamen *ResDateiName*.

Übergeben Sie einen Dateinamen, muss die Datei im gleichen Ordner liegen wie die Strukturdatei der Datenbank. Übergeben Sie einen Pfadnamen, können Sie eine Ressourcendatei aus einem anderen Ordner öffnen.

Übergeben Sie einen leeren String in *ResDateiName*, erscheint der Standard-Öffnen-Dialog. Sie können dann die entsprechende Ressourcendatei wählen. Brechen Sie diesen Dialog ab, wird keine Ressourcendatei geöffnet; **Open resource file** gibt eine Null DokRef zurück und setzt die OK Variable auf Null (0).

Bei korrekt geöffneter Ressourcendatei gibt die Funktion **Open resource file** deren Referenznummer zurück und setzt die OK Variable auf 1. Gibt es die Ressourcendatei nicht, oder ist die Datei keine Ressourcendatei, wird ein Fehler generiert.

- Verwenden Sie das Standard-Öffnen-Dialogfenster unter Windows, werden standardmäßig alle Dateien angezeigt. Soll ein bestimmter Dateityp angezeigt werden, übergeben Sie in *Dateityp* eine Windows-Erweiterung mit 1 bis 3 Zeichen oder über den Befehl **_o_MAP FILE TYPES** einen Macintosh Dateityp.
- Verwenden Sie das Standard-Öffnen-Dialogfenster auf Macintosh, werden standardmäßig alle Dateien angezeigt. Wollen Sie einen bestimmten Dateityp anzeigen, geben Sie diesen im optionalen Parameter *Dateityp* an.

Vergessen Sie nicht, die Ressourcendatei über den Befehl **CLOSE RESOURCE FILE** wieder zu schließen. Verlassen Sie jedoch die Datenbank oder öffnen eine andere Datenbank, schließt 4D mit den Funktionen **Open resource file** oder **_o_Create resource file** automatisch alle geöffneten Ressourcendateien.

Während die Funktion **Open document** ein Dokument (Datenfork auf Macintosh) ausschließlich im Lese-/Schreibzugriff öffnet, können Sie mit der Funktion **Open resource file** auch eine Ressourcendatei öffnen, die bereits in der 4D Sitzung geöffnet ist. Versuchen Sie zum Beispiel, dasselbe Dokument zweimal mit der Funktion **Open document** zu öffnen, erhalten Sie beim zweiten Versuch einen E/A Fehler. Versuchen Sie dagegen, eine Ressourcendatei, die bereits in der 4D Sitzung geöffnet ist, mit **Open resource file** erneut zu öffnen, erhalten Sie die Referenznummer der Ressourcendatei für die bereits geöffnete Datei. Auch wenn Sie eine Ressourcendatei mehrmals öffnen, müssen Sie dafür einmal den Befehl **CLOSE RESOURCE FILE** aufrufen, um sie zu schließen. Das gilt nur, wenn die Ressourcendatei innerhalb einer 4D Sitzung geöffnet wird. Versuchen Sie, eine bereits von einer anderen Anwendung geöffnete Ressourcendatei zu öffnen, erhalten Sie einen E/A Fehler.

Warnung

- Es ist verboten, auf eine Ressourcendatei einer 4D Anwendung und auf eine Ressourcendatei einer mit 4D Runtime erstellten Datenbank zuzugreifen.
- Auch wenn es technisch möglich ist, sollten Sie nicht die Ressourcendatei der Datenbankstruktur verwenden, da Ihr Code nicht funktioniert, wenn die Datenbank kompiliert ist und mit 4D Runtime verwendet wird. Greifen Sie dennoch auf eine Ressourcendatei der Datenbank zu und wollen per Programmierung Ressourcen hinzufügen, löschen oder ändern, prüfen Sie, in welcher Umgebung Sie sind. Mit 4D Server führt das höchstwarscheinlich zu schwerwiegenden Fehlern. Ändern Sie zum Beispiel eine Ressource auf dem Server-Rechner (über eine Datenbankmethode oder eine Serverprozedur), beeinträchtigt das definitiv die integrierten 4D Server Verwaltungsdienste, die die Ressourcen transparent auf die Arbeitsstationen verteilen. Bedenken Sie, dass Sie mit 4D Client keinen direkten Zugriff auf die Strukturdatei haben; diese liegt auf dem Server-Rechner.
- Wenn Sie Ressourcen verwenden, speichern Sie diese deshalb in Ihren eigenen Dateien.
- Arbeiten Sie mit eigenen Ressourcen, verwenden Sie KEINE negativen Ressourcennummern; sie sind für das Betriebssystem reserviert. Verwenden Sie KEINE Ressourcennummern im Bereich 0..14.999; dieser Bereich ist für 4D reserviert. Verwenden Sie für Ihre eigenen Ressourcen den Bereich 15.000..32.767. Bedenken Sie, dass sobald eine Ressourcendatei geöffnet ist, diese die erste Datei ist, nach der in einer Kette von Ressourcendateien gesucht wird. Speichern Sie in dieser Datei eine Ressource mit einer Nummer aus dem Bereich der System- bzw. 4D Ressourcen, finden Befehle wie **GET RESOURCE** und auch interne Routinen der 4D Anwendung diese Ressource. Das ist evtl. das Ergebnis, das Sie erreichen wollen. Wenn Sie jedoch nicht sicher sind, verwenden Sie diese Bereiche nicht für eigene Ressourcen, denn das kann zu Systemfehlern führen.
- Ressourcendateien sind stark strukturierte Dateien, sie akzeptieren max. 2.700 Ressourcen pro Datei. Arbeiten Sie mit Dateien, die bereits eine große Anzahl an Ressourcen enthalten, sollten Sie die Anzahl prüfen, bevor Sie neue Ressourcen hinzufügen. Sehen Sie sich dazu das Beispiel mit der Methode ***Count resources*** für den Befehl **RESOURCE TYPE LIST** an.

Haben Sie eine Ressourcendatei geöffnet, können Sie mit den Befehlen **RESOURCE TYPE LIST** und **RESOURCE LIST** den Inhalt der Datei analysieren.

Beispiel 1

Folgendes Beispiel versucht unter Windows die Ressourcendatei “MyPrefs.res” zu öffnen, die im Datenbankordner liegt:

```
$vhResFile:=Open resource file("MyPrefs";"res ")
```

Auf Macintosh versucht das Beispiel die Datei “MyPrefs” zu öffnen.

Beispiel 2

Folgendes Beispiel versucht unter Windows die Ressourcendatei “MyPrefs.rsr” zu öffnen, die im Datenbankordner liegt:

```
$vhResFile:=Open resource file("MyPrefs";"rsr")
```

Auf Macintosh versucht das Beispiel die Datei “MyPrefs” zu öffnen.

Beispiel 3

Folgendes Beispiel zeigt das Öffnen-Dialogfenster, das alle Dateitypen auflistet:

```
$vhResFile:=Open resource file("")
```

Beispiel 4

Folgendes Beispiel zeigt das Öffnen-Dialogfenster, das die Dateien mit dem Standardtyp auflistet, die von der Funktion **_o_Create resource file** erstellt wurden:

```
$vhResFile:=Open resource file("";"res ")
If(OK=1)
   ALERT("You just opened “"+Document+"”.")
   CLOSE RESOURCE FILE($vhResFile)
End if
```

Systemvariablen und Mengen

Wurde die Ressourcendatei erfolgreich geöffnet, wird die OK Variable auf 1 gesetzt. Konnte die Ressourcendatei nicht geöffnet werden oder hat der Benutzer im Öffnen-Dialogfenster auf Abbrechen geklickt, wird die OK Variable auf 0 (Null) gesetzt.

Wurde die Ressourcendatei über das Öffnen-Dialogfenster erfolgreich geöffnet, wird die Variable Document auf den Pfadnamen der Datei gesetzt.

Fehlerverwaltung

Konnte die Ressourcendatei wegen eines Ressourcen- bzw. E/A Problems nicht geöffnet werden, wird ein Fehler erzeugt. Sie können diesen Fehler mit **ON ERR CALL** in einer Fehlerverwaltungsmethode ausfindig machen.

RESOURCE LIST

RESOURCE LIST (ResTyp ; ResID ; ResNamen {; ResDatei})

Parameter	Typ		Beschreibung
ResTyp	String	➡	Ressourcentyp mit 4 Zeichen
ResID	Array Lange Ganzzahl	⬅	Ressourcennummern für Ressourcen dieses Typs
ResNamen	Array String	⬅	Ressourcennamen für Ressourcen dieses Typs
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe alle geöffneten Ressourcendateien

Beschreibung

Der Befehl **RESOURCE LIST** füllt die Arrays *ResDatei* und *ResNamen* mit den Ressourcennummern *ResID* und den Ressourcennamen vom Typ *ResTyp*.

Wichtig: Sie müssen in *ResTyp* einen String mit 4 Zeichen übergeben.

Übergeben Sie im optionalen Parameter *ResDatei* eine gültige Referenznummer für die Ressourcendatei, erscheinen nur die Ressourcen aus dieser Datei. Übergeben Sie diesen Parameter nicht, erscheinen alle Ressourcen aus den derzeit geöffneten Ressourcendateien.

Legen Sie vor dem Aufrufen von **RESOURCE LIST** Array-Typen fest, müssen Sie *ResID* als Array vom Typ Lange Ganzzahl und *ResNamen* als Array vom Typ String oder Text festlegen. Legen Sie nichts fest, erstellt der Befehl *ResID* als Array vom Typ Lange Ganzzahl und *ResNamen* als Array vom Typ Text.

Nach dem Aufruf können Sie über die Funktion **Size of array** die Anzahl der gefundenen Ressourcentypen im Array *ResID* oder *ResNamen* prüfen.

Beispiel 1

Folgendes Beispiel füllt die Arrays *$alResID* und *$atResName* mit den Nummern und Namen der String Ressourcen, die in der Datenbank vorhanden sind:

```
If(On Windows)
   $vhStructureResFile:=Open resource file(Replace string
   (Structure file;".4DB";".RSR"))
Else
   $vhStructureResFile:=Open resource file(Structure file)
End if
If(OK=1)
   RESOURCE LIST("STR#";$alResID;$atResName;$vhStructureResFile)
End if
```

Beispiel 2

Folgendes Beispiel kopiert alle Ressourcen vom Typ Bild aus den derzeit geöffneten Ressourcendateien in die Bildbibliothek der Datenbank:

```
RESOURCE LIST("PICT";$alResID;$atResName)
Open window(50;50;550;120;5;"Kopiere PICT Ressourcen...")
For($vlElem;1;Size of array($alResID))
   GET PICTURE RESOURCE($alResID{$vlElem};$vgPicture)
   If(OK=1)
      $vsName:=$atResName{$vlElem}
      If($vsName="")
         $vsName:="PICT resID="+String($alResID{$vlElem})
      End if
      ERASE WINDOW
      GOTO XY(2;1)
      MESSAGE("Bild “"+$vsName+"” in der DB Bildbibliothek hinzufügen.")
      SET PICTURE TO LIBRARY($vgPicture;$alResID{$vlElem};$vsName)
   End if
End for
CLOSE WINDOW
```

RESOURCE TYPE LIST

RESOURCE TYPE LIST (ResTypen {; ResDatei})

Parameter	Typ		Beschreibung
ResTypen	Array String	⇐	Liste der verfügbaren Ressourcentypen
ResDatei	DokRef	⇒	Referenznummer der Ressourcendatei Ohne Angabe alle geöffneten Ressourcendateien

Beschreibung

Der Befehl **RESOURCE TYPE LIST** füllt das Array *ResTypen* mit den Ressourcentypen, die in den Ressourcen der aktuell geöffneten Ressourcendateien vorhanden sind.

Übergeben Sie in dem optionalen Parameter *ResDatei* eine gültige Referenznummer für die Ressourcendatei, erscheinen nur die Ressourcen aus dieser Datei. Übergeben Sie diesen Parameter nicht, erscheinen alle Ressourcen aus den derzeit geöffneten Ressourcendateien.

Sie können für das Array *ResTypen* vor dem Aufrufen von **RESOURCE TYPE LIST** den Typ String oder Text festlegen. Legen Sie nichts fest, erstellt der Befehl *ResTypen* als Array vom Typ Text.

Nach dem Aufruf können Sie über die Funktion **Size of array** die Anzahl der gefundenen Ressourcentypen im Array *ResTypen* prüfen.

Beispiel 1

Folgendes Beispiel füllt das Array *atResType* mit den Ressourcentypen, die in allen derzeit geöffneten Ressourcendateien vorhanden sind:

```
RESOURCE TYPE LIST(atResType)
```

Beispiel 2

Folgendes Beispiel teilt mit, ob die 4D Strukturdatei auf Macintosh alte 4D Plug-Ins enthält, die für den Einsatz der Datenbank unter Windows aktualisiert werden müssen:

```
$vhResFile:=Open resource file(Structure file)
RESOURCE TYPE LIST(atResType;$vhResFile)
If(Find in array(atResType;"4DEX")>0)
   ALERT("Diese Datenbank enthält überholte Mac OS 4D Plug-Ins."+(Char(13)*2)+
   "Für den Einsatz unter Windows bitte aktualisieren.")
End if
```

Hinweis: Alte Plug-Ins sind nicht nur in der Strukturdatei enthalten. Die Datenbank kann auch eine Datei Proc.Ext enthalten.

Beispiel 3

Folgende Projektmethode gibt die Anzahl der Ressourcen zurück, die in einer Ressourcendatei enthalten sind:

```
` Projektmethode Count resources
` Count resources ( Zeit ) -> Lange Ganzzahl
` Count resources ( DokRef ) -> Anzahl der Ressourcen

C_LONGINT($0)
C_TIME($1)

$0:=0
RESOURCE TYPE LIST($atResType;$1)
For($vlElem;1;Size of array($atResType))
   RESOURCE LIST($atResType{$vlElem};$alResID;$atResName;$1)
   $0:=$0+Size of array($alResID)
End for
```

Ist diese Projektmethode in eine Datenbank integriert, können Sie schreiben:

```
$vhResFile:=Open resource file("")
If(OK=1)
   ALERT("Die Datei “"+Dokument+"” enthält "+String(Count resources($vhResFile))
   +" Ressource(n).")
```

```
    CLOSE RESOURCE FILE($vhResFile)
End if
```

STRING LIST TO ARRAY

STRING LIST TO ARRAY (ResID ; String {; ResDatei})

Parameter	Typ		Beschreibung
ResID	Lange Ganzzahl	➡	Ressourcennummer oder 'id' Attribut des 'group' Elements (XLIFF)
String	Array String	⬅	Strings aus der STR# Ressource oder Strings aus dem 'group' Element (XLIFF)
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe alle XLIFF Dateien oder offene Ressourcendateien

Beschreibung

Der Befehl **STRING LIST TO ARRAY**

- Den Strings aus der Stringlisten-Ressource ("STR#") mit der Nummer *ResID*
- Oder einem String aus einer offenen XLIFF Datei mit dem 'id' Attribut des 'group' Elements, übergeben in *ResID* (siehe unten)

Wird die Ressource nicht gefunden, bleibt das Array *String* unverändert, die OK Variable wird auf 0 (Null) gesetzt.

Übergeben Sie in *ResDatei* eine gültige Referenznummer für die Ressourcendatei, wird die Ressource nur in dieser Datei gesucht. Übergeben Sie diesen Parameter nicht, wird das erste Vorkommen der Ressource in der Kette der Ressourcendateien zurückgegeben.

Sie können für das Array *String* vor dem Aufrufen von **STRING LIST TO ARRAY** den Typ String oder Text festlegen. Legen Sie nichts fest, erstellt der Befehl *String* als Array vom Typ Text.

Hinweis: Jeder String einer Stringlisten-Ressource kann bis zu 255 Zeichen enthalten.

Tipp: Begrenzen Sie Ihre Stringlisten-Ressource auf die Größe 32K und auf ein paar hundert Strings pro Ressource.

Kompatibilität mit XLIFF Architektur

Der Befehl **STRING LIST TO ARRAY** ist ab 4D Version 11 kompatibel mit der XLIFF Architektur: Der Befehl sucht zuerst in allen offenen XLIFF Dateien nach Werten, die *ResID* und *String* entsprechen (wenn der Parameter *ResDatei* nicht übergeben ist) und füllt das Array *String* mit den entsprechenden Werten. *ResID* bestimmt dann das **id**-Attribut des **group**-Elements und das Array *String* enthält alle Strings des Eintrags. Wird der Wert nicht gefunden, setzt der Befehl die Suche in den offenen Ressource-Dateien fort. Weitere Informationen dazu finden Sie im **PICTURE TO BLOB** des Handbuchs *4D Designmodus*.

Systemvariablen und Mengen

Wird die Ressource gefunden, wird OK auf 1 gesetzt, sonst auf 0 (Null).

_o_ARRAY TO STRING LIST

_o_ARRAY TO STRING LIST (String ; ResID {; ResDatei})

Parameter	Typ		Beschreibung
String	Array String	➡	String oder Text Array (neuer Inhalt für die Ressource STR#)
ResID	Lange Ganzzahl	➡	Kennummer der Ressource
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe aktuelle Ressourcendatei

Beschreibung

Hinweis zur Kompatibilität: Seit 4D v13 sind Befehle, die Ressourcen schreiben, veraltet. Dieser Befehl sollte nicht mehr verwendet werden.

_o_Create resource file

_o_Create resource file (ResDateiName {; Dateityp {; *}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ResDateiName	String	➡	Kurzer oder langer Name der Ressourcendatei Oder leerer String für Standard Sichern-Dialog für Datei
Dateityp	String	➡	Mac OS Dateityp (4-Zeichen String) oder Windows Dateierweiterung (1- bis 3-Zeichen String), ohne Angabe Ressourcendokument ("res " /.RES)
*		➡	Wenn übergeben = Verwende Datafork
Funktionsergebnis	DokRef	⬅	Referenznummer der Ressourcendatei

Beschreibung

Hinweis zur Kompatibilität: Seit 4D v13 funktionieren Befehle, die Ressourcen schreiben, nicht mehr. Diese Funktion ist veraltet und sollte nicht mehr verwendet werden.

_o_DELETE RESOURCE

_o_DELETE RESOURCE (ResTyp ; ResID {; ResDatei})

Parameter	Typ		Beschreibung
ResTyp	String	➡	Ressourcentyp mit 4 Zeichen
ResID	Lange Ganzzahl	➡	Kennummer der Ressource
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe aktuelle Ressourcendatei

Beschreibung

Seit 4D v13 sind Befehle, die Ressourcen schreiben, veraltet und sollten nicht mehr verwendet werden.

_o_Get component resource ID

_o_Get component resource ID (KompName ; ResTyp ; OriginalResNum) -> Funktionsergebnis

Parameter	Typ		Beschreibung
KompName	String	➡	Name der 4D Komponente zur Ressource
ResTyp	String	➡	Ressourcentyp (4 Zeichen)
OriginalResNum	Lange Ganzzahl	➡	Originalnummer der Ressource vor Installation der 4D Komponente
Funktionsergebnis	Lange Ganzzahl	↩	Aktuelle Ressourcennummer

Beschreibung

Hinweis zur Kompatibilität: Diese Funktion arbeitete mit Komponenten früherer Generationen, die mit 4D Version 11 und höher nicht mehr kompatibel sind. Die Funktion ist überholt und sollte nicht mehr verwendet werden.

_o_SET PICTURE RESOURCE

_o_SET PICTURE RESOURCE (ResID ; ResDaten {; ResDatei})

Parameter	Typ		Beschreibung
ResID	Lange Ganzzahl	➡	Kennummer der Ressource
ResDaten	Bild	➡	Neuer Inhalt für die Ressource PICT
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe aktuelle Ressourcendatei

Beschreibung

Hinweis zur Kompatibilität: Seit 4D v13 funktionieren Befehle, die Ressourcen schreiben, nicht mehr. Dieser Befehl ist veraltet und sollte nicht mehr verwendet werden.

_o_SET RESOURCE

_o_SET RESOURCE (ResTyp ; ResID ; ResDaten {; ResDatei})

Parameter	Typ		Beschreibung
ResTyp	String	➡	Ressourcentyp mit 4 Zeichen
ResID	Lange Ganzzahl	➡	Kennummer der Ressource
ResDaten	BLOB	➡	Neuer Inhalt für die Ressource
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe aktuelle Ressourcendatei

Beschreibung

Hinweis zur Kompatibilität: Seit 4D v13 funktionieren Befehle, die Ressourcen schreiben, nicht mehr. Dieser Befehl ist veraltet und sollte nicht mehr verwendet werden.

_o_SET RESOURCE NAME

_o_SET RESOURCE NAME (ResTyp ; ResID ; ResName {; ResDatei})

Parameter	Typ		Beschreibung
ResTyp	String	➡	Ressourcentyp mit 4 Zeichen
ResID	Lange Ganzzahl	➡	Kennummer der Ressource
ResName	String	➡	Neuer Name für die Ressource
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe aktuelle Ressourcendatei

Beschreibung

Hinweis zur Kompatibilität: Seit 4D v13 funktionieren Befehle, die Ressourcen schreiben, nicht mehr. Dieser Befehl ist veraltet und sollte nicht mehr verwendet werden.

_o_SET RESOURCE PROPERTIES

_o_SET RESOURCE PROPERTIES (ResTyp ; ResID ; ResAttr {; ResDatei})

Parameter	Typ		Beschreibung
ResTyp	String	➡	Ressourcentyp mit vier Zeichen
ResID	Lange Ganzzahl	➡	Kennummer der Ressource
ResAttr	Lange Ganzzahl	➡	Neue Attribute für die Ressource
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe aktuelle Ressourcendatei

Beschreibung

Hinweis zur Kompatibilität: Seit 4D v13 funktionieren Befehle, die Ressourcen schreiben, nicht mehr. Dieser Befehl ist veraltet und sollte nicht mehr verwendet werden.

_o_SET STRING RESOURCE

_o_SET STRING RESOURCE (ResID ; ResDaten {; ResDatei})

Parameter	Typ		Beschreibung
ResID	Lange Ganzzahl	➡	Kennummer der Ressource
ResDaten	String	➡	Neuer Inhalt für die Ressource STR
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe aktuelle Ressourcendatei

Beschreibung

Hinweis zur Kompatibilität: Seit 4D v13 funktionieren Befehle, die Ressourcen schreiben, nicht mehr. Dieser Befehl ist veraltet und sollte nicht mehr verwendet werden.

_o_SET TEXT RESOURCE

_o_SET TEXT RESOURCE (ResID ; ResDaten {; ResDatei})

Parameter	Typ		Beschreibung
ResID	Lange Ganzzahl	➡	Kennumer der Ressource
ResDaten	String	➡	Neuer Inhalt für die Ressource TEXT
ResDatei	DokRef	➡	Referenznummer der Ressourcendatei Ohne Angabe aktuelle Ressourcendatei

Beschreibung

Hinweis zur Kompatibilität: Seit 4D v13 funktionieren Befehle, die Ressourcen schreiben, nicht mehr. Dieser Befehl ist veraltet und sollte nicht mehr verwendet werden.

Schnellbericht

- 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 (Bereich ; BLOB)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
BLOB	BLOB	➡	BLOB, das den Bericht enthält

Beschreibung

Der Befehl **QR BLOB TO REPORT** setzt den Bericht, enthalten in *BLOB,* in den *Bereich* für Schnellbericht.
Übergeben Sie eine ungültige Nummer für *Bereich*, wird der Fehler -9850 erzeugt.
Übergeben Sie einen ungültigen Namen in *BLOB*, wird der Fehler -9852 erzeugt.

Beispiel 1

Mit folgendem Code können Sie in *MeinBereich* die Berichtdatei mit Namen "report.4qr" anzeigen, die auf derselben Ebene wie die Datenbankstruktur liegt. Diese Datei kann auch aus einer früheren Version stammen:

```
C_BLOB($doc)
C_LONGINT(MeinBereich)
DOCUMENT TO BLOB("report.4qr";$doc)
QR BLOB TO REPORT(MeinBereich;$doc)
```

Beispiel 2

Folgende Anweisung findet den Schnellbericht, der in *Feld4* gespeichert ist und zeigt ihn in *MeinBereich* an:

```
QR BLOB TO REPORT(MeinBereich;[Tabelle 1]Feld4)
```

QR Count columns

QR Count columns (Bereich) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf Bereich
Funktionsergebnis	Lange Ganzzahl	⮌	Anzahl der Spalten in Bereich

Beschreibung

Die Funktion **QR Count columns** gibt die Anzahl Spalten zurück, die im *Bereich* für Schnellbericht enthalten sind.
Übergeben Sie eine ungültige Nummer für *Bereich*, wird Fehler -9850 generiert.

Beispiel

Nachfolgender Code findet die gezählten Spalten und fügt neben der letzten rechten Spalte eine Spalte an:

```
$ColNb:=QR Count columns(MyArea)
QR INSERT COLUMN(MyArea;$ColNb+1;->[Table 1]Field2)
```

QR DELETE COLUMN

QR DELETE COLUMN (Bereich ; SpaltenNr)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz des Bereichs
SpaltenNr	Lange Ganzzahl	➡	Spaltennummer

Beschreibung

Der Befehl **QR DELETE COLUMN** löscht die Spalten in *Bereich*, dessen Nummer in *SpaltenNr* übergeben wurde. Dieser Befehl gilt nicht für Kreuztabellen-Berichte.

Übergeben Sie eine ungültige Nummer für *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie eine ungültige Nummer für *SpaltenNr*, wird der Fehler -9852 generiert.

Beispiel

Folgendes Beispiel stellt sicher, dass der Bericht ein Listenbericht ist und löscht die dritte Spalte:

```
If(QR Get report kind(MyArea)=qr list report)
   QR DELETE COLUMN(MyArea;3)
End if
```

QR DELETE OFFSCREEN AREA

QR DELETE OFFSCREEN AREA (Bereich)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den zu löschenden Bereich

Beschreibung

Der Befehl **QR DELETE OFFSCREEN AREA** löscht im Speicher den Offscreen Schnellbericht, dessen Referenz in *Bereich* übergeben wurde.
Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

QR EXECUTE COMMAND

QR EXECUTE COMMAND (Bereich ; Befehl)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf Bereich
Befehl	Lange Ganzzahl	➡	Auszuführender Menübefehl

Hinweis zur Kompatibilität

Ab 4D v16 enthalten die 64-bit Versionen von 4D den neuen Editor **Schnellberichte (64-bit)**. Die Oberfläche dieses Editors mit Werkzeugleiste lässt sich nicht anpassen. Deshalb lassen sich die meisten Menübefehle für die u.a. Konstanten nicht mit dem Schnellberichteditor in 64-bit verwenden. Die kompatiblen Konstanten sind in der Tabelle angegeben.
Beachten Sie, dass Sie über den Bereich im 64-bit Editor eine komplett angepasste Oberfläche gestalten können. Weitere Informationen dazu finden Sie im Abschnitt **Schnellbericht erstellen**.

Beschreibung

Der Befehl **QR EXECUTE COMMAND** führt den Menübefehl oder Icon der Werkzeugleiste aus, dessen Referenz in *Befehl* übergeben wurde. Dieser Befehl wird hauptsächlich verwendet, um einen Befehl auszuführen, nachdem der Benutzer diesen Befehl gewählt hat und der Code ihn durch den Befehl **QR ON COMMAND** unterbrochen hat.

Sie können in *Befehl* einen Wert oder eine Konstante unter dem Thema **QR Befehle** übergeben:

Konstante	Typ	Wert	Kommentar
qr cmd 4D View destination	Lange Ganzzahl	2503	
qr cmd add column	Lange Ganzzahl	2608	
qr cmd alt back color palette	Lange Ganzzahl	1004	
qr cmd automatic width	Lange Ganzzahl	2605	
qr cmd average	Lange Ganzzahl	507	
qr cmd back color palette	Lange Ganzzahl	1003	
qr cmd back colors toolbar	Lange Ganzzahl	2052	
qr cmd bold	Lange Ganzzahl	500	
qr cmd borders	Lange Ganzzahl	2609	
qr cmd center justified	Lange Ganzzahl	504	
qr cmd columns toolbar	Lange Ganzzahl	2054	
qr cmd count	Lange Ganzzahl	510	
qr cmd default justified	Lange Ganzzahl	512	
qr cmd delete column	Lange Ganzzahl	2601	
qr cmd disk file destination	Lange Ganzzahl	2501	
qr cmd edit column	Lange Ganzzahl	2603	
qr cmd font color palette	Lange Ganzzahl	1002	
qr cmd font dropdown	Lange Ganzzahl	1000	
qr cmd format	Lange Ganzzahl	2606	
qr cmd generate	Lange Ganzzahl	2008	kompatibel mit 64-bit Editor (Befehl **QR RUN** wird empfohlen)
qr cmd graph destination	Lange Ganzzahl	2502	
qr cmd header and footer	Lange Ganzzahl	2005	
qr cmd hide column	Lange Ganzzahl	2602	
qr cmd hide line	Lange Ganzzahl	2607	
qr cmd HTML file destination	Lange Ganzzahl	2504	
qr cmd insert column	Lange Ganzzahl	2600	
qr cmd italic	Lange Ganzzahl	501	
qr cmd left justified	Lange Ganzzahl	503	
qr cmd max	Lange Ganzzahl	509	
qr cmd min	Lange Ganzzahl	508	
qr cmd move left	Lange Ganzzahl	3002	
qr cmd move right	Lange Ganzzahl	3003	
qr cmd new	Lange Ganzzahl	2000	
qr cmd open	Lange Ganzzahl	2001	
qr cmd operators toolbar	Lange Ganzzahl	2051	
qr cmd page setup	Lange Ganzzahl	2006	kompatibel mit 64-bit Editor
qr cmd plain	Lange Ganzzahl	511	
qr cmd presentation	Lange Ganzzahl	2611	
qr cmd print preview	Lange Ganzzahl	2007	kompatibel mit 64-bit Editor
qr cmd printer destination	Lange Ganzzahl	2500	
qr cmd repeated values	Lange Ganzzahl	2604	
qr cmd revert to save	Lange Ganzzahl	2004	
qr cmd right justified	Lange Ganzzahl	505	
qr cmd save	Lange Ganzzahl	2002	
qr cmd save as	Lange Ganzzahl	2003	
qr cmd standard deviation	Lange Ganzzahl	513	
qr cmd standard toolbar	Lange Ganzzahl	2053	
qr cmd style toolbar	Lange Ganzzahl	2050	
qr cmd sum	Lange Ganzzahl	506	
qr cmd totals spacing	Lange Ganzzahl	2610	
qr cmd underline	Lange Ganzzahl	502	

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie eine ungültige Nummer in *Befehl*, wird der Fehler -9852 generiert.

QR Find column

QR Find column (Bereich ; Ausdruck) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf Bericht
Ausdruck	String, Zeiger	➡	Objekt der Spalte
Funktionsergebnis	Lange Ganzzahl	⮌	Nummer der Spalte

Beschreibung

Die Funktion **QR Find column** gibt die Nummer der ersten Spalte zurück, deren Inhalt zum Parameter in *Ausdruck* passt.
Ausdruck kann entweder ein String oder ein Zeiger sein.
QR Find column gibt –1 zurück, wenn nichts gefunden wurde.
Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

Beispiel

Nachfolgender Code findet die Spaltennummer mit dem Feld [G.NQR Tests]Quarter und löscht diese Spalte:

```
$NumColumn:=QR Find column(MyArea;->[G.NQR Tests]Quarter)
```

oder:

```
$NumColumn:=QR Find column(MyArea;"[G.NQR Tests]Quarter")
```

gefolgt von:

```
[If ($NumColumn#-1)
QR DELETE COLUMN(MyArea;$NumColumn)
End if
```

QR Get area property

QR Get area property (Bereich ; Eigenschaft) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	→	Referenz auf den Bereich
Eigenschaft	Lange Ganzzahl	→	Zugewiesenes Oberflächenelement
Funktionsergebnis	Lange Ganzzahl	↩	0 = sichtbar, 1 = ausgeblendet

Beschreibung

Die Funktion **QR Get area property** gibt 1 zurück, wenn das in *Eigenschaft* übergebene Oberflächenelement (Werkzeug- oder Menüleiste) angezeigt wird; 0 (Null) wenn das Element ausgeblendet ist.

Menüleiste und Werkzeugleisten sind von oben nach unten mit 1 bis 6 nummeriert. Wert 7 ist für das Kontextmenü reserviert.

Sie können für die Oberflächenelemente die Konstanten unter dem Thema **QR Bereichseigenschaften** verwenden:

Konstante	Typ	Wert	Kommentar
qr view color toolbar	Lange Ganzzahl	5	Zeige Status der Werkzeugleiste für Farben (Angezeigt=1, Ausgeblendet=0)
qr view column toolbar	Lange Ganzzahl	6	Zeige Status der Werkzeugleiste für Spalten (Angezeigt=1, Ausgeblendet=0)
qr view contextual menus	Lange Ganzzahl	7	Zeige Status des Kontextmenüs (Angezeigt=1, Ausgeblendet=0)
qr view menubar	Lange Ganzzahl	1	Zeige Status der Menüleiste (Angezeigt=1, Ausgeblendet=0)
qr view operators toolbar	Lange Ganzzahl	4	Zeige Status der Werkzeugleiste für Operatoren (Angezeigt=1, Ausgeblendet=0)
qr view standard toolbar	Lange Ganzzahl	2	Zeige Status der Standard Werkzeugleiste (Angezeigt=1, Ausgeblendet=0)
qr view style toolbar	Lange Ganzzahl	3	Zeige Status der Werkzeugleiste für Stil (Angezeigt=1, Ausgeblendet=0)

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie eine ungültige Nummer in *Eigenschaft*, wird der Fehler -9852 generiert.

QR GET BORDERS

QR GET BORDERS (Bereich ; Spalte ; Zeile ; Rand ; Linie {; Farbe})

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Spalte	Lange Ganzzahl	➡	Nummer der Spalte
Zeile	Lange Ganzzahl	➡	Nummer der Zeile
Rand	Lange Ganzzahl	➡	Border value
Linie	Lange Ganzzahl	⬅	Linienstärke
Farbe	Lange Ganzzahl	⬅	Randfarbe

Beschreibung

Der Befehl **QR GET BORDERS** findet den Randstil für die Ränder einer bestimmten Zelle.
Bereich ist die Referenz auf *Bereich* des Schnellberichts. *Spalte* ist die Spaltennummer der Zelle.
Zeile gibt die Zeilennummer der Zelle zurück. Sie können entweder

- Eine positive Ganzzahl übergeben für die entsprechende Umbruchebene mit einer Untersumme
- Eine der vordefinierten Konstanten unter dem Thema **QR Zeilen für Eigenschaften** verwenden

Konstante	Typ	Wert	Kommentar
qr detail	Lange Ganzzahl	-2	Detailbereich des Berichts
qr grand total	Lange Ganzzahl	-3	Bereich Gesamtsumme
qr title	Lange Ganzzahl	-1	Titel des Berichts

Rand ist der Wert, der angibt, welcher Rand der Zelle betroffen ist. Sie können eine Konstante unter dem Thema **QR Rahmen** übergeben:

Konstante	Typ	Wert	Kommentar
qr bottom border	Lange Ganzzahl	8	Unterer Rand
qr inside horizontal border	Lange Ganzzahl	32	Innerer horizontaler Rand
qr inside vertical border	Lange Ganzzahl	16	Innerer vertikaler Rand
qr left border	Lange Ganzzahl	1	Linker Rand
qr right border	Lange Ganzzahl	4	Rechter Rand
qr top border	Lange Ganzzahl	2	Oberer Rand

Hinweis: QR GET BORDERS akzeptiert – im Gegensatz zu **QR SET BORDERS** keinen kumulierten Wert.
Sie müssen alle Parameter einzeln testen für einen Gesamtüberblick über die Zellenränder.
Linie gibt die Linienstärke an:

- 0 steht für keine Linie
- 1 gibt eine Stärke von 1/4 Punkt an
- 2 gibt eine Stärke von 1/2 Punkt an
- 3 gibt eine Stärke von 1 Punkt an
- 4 gibt eine Stärke von 2 Punkt an

Farbe bezeichnet die Linienfarbe, sie gibt den Wert der im Segment angewandten Farbe zurück.
Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie eine ungültige Nummer in *Spalte*, wird der Fehler -9852 generiert.
Übergeben Sie eine ungültige Nummer in *Zeile*, wird der Fehler -9853 generiert.
Übergeben Sie eine ungültige Nummer in *Rand*, wird der Fehler -9854 generiert.

QR Get command status

QR Get command status (Bereich ; Befehl {; Wert}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	→	Referenz auf den Bereich
Befehl	Lange Ganzzahl	→	Befehlsnummer
Wert	Lange Ganzzahl, Text	←	Wert für gewählten Untereintrag
Funktionsergebnis	Lange Ganzzahl	↩	Status des Befehls

Beschreibung

Die Funktion **QR Get command status** gibt 0 zurück, wenn *Befehl* deaktiviert ist, 1 wenn aktiviert.

Wert gibt den Wert des gewählten Untereintrags zurück – sofern vorhanden. Ist zum Beispiel der gewählte Befehl das Menü **Schrift** (1000) und die gewählte Schrift "Arial", gibt *Wert* "Arial" zurück; ist der gewählte Befehl ein Farbmenü (1002, 1003 or 1004), gibt *Wert* die Farbnummer zurück.

Sie können den Befehl auf zwei verschiedene Arten verwenden:

- Als einfache Anweisung, um zu bestimmen, ob ein Befehl aktiviert oder deaktiviert ist.
- Über die mit **QR ON COMMAND** eingerichtete Methode, die anzeigt, welcher Untereintrag gewählt wurde. In dieser Methode ist *$1* die Referenz auf den Bereich, *$2* die Nummer des Befehls.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie eine ungültige Nummer in *Befehl*, wird der Fehler -9852 generiert.

Sie können in *Befehl* einen Wert oder eine Konstante unter dem Thema **QR Befehle** übergeben.

QR GET DESTINATION

QR GET DESTINATION (Bereich ; Typ {; Merkmale})

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Typ	Lange Ganzzahl	⬅	Art des Berichts
Merkmale	String, Variable	⬅	Merkmale der Ausgabeart

Beschreibung

Der Befehl **QR GET DESTINATION** findet den *Typ* des Berichts für die in *Bereich* übergebene Referenz.
Sie können den Wert des Parameters *Typ* vergleichen mit den Konstanten unter dem Thema **QR Ausgabeart**.
Nachfolgende Tabelle beschreibt die möglichen Werte in den Parametern *Typ* und *Merkmale*:

Ziel	Typ (Wert)	Eigenheiten
Drucker	*qr printer* (1)	--
Text file	*qr text file* (2)	Pfadname der Datei
4D View	*qr 4D View area* (3)	--
4D Chart	*qr 4D Chart area* (4)	--
HTML file	*qr HTML file* (5)	Pfadname der HTML Datei

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 zurückgegeben.

QR Get document property

QR Get document property (Bereich ; Eigenschaft) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Eigenschaft	Lange Ganzzahl	➡	1 = Druckdialog, 2 = Dokumenteinheit
Funktionsergebnis	Lange Ganzzahl	⮌	Wert für die Eigenschaft

Beschreibung

Die Funktion **QR Get document property** liefert den Anzeigestatus des Druckdialogs oder die Einheit für das Lineal.
Sie können in *Eigenschaft* die Konstanten unter dem Thema **QR Dokumenteigenschaften** verwenden:

Konstante	Typ	Wert	Kommentar
qr printing dialog	Lange Ganzzahl	1	Druckdialog anzeigen: - Bei Wert = 0 wird der Druckdialog nicht vor dem Drucken angezeigt. - Bei Wert = 1 wird der Druckdialog vor dem Drucken angezeigt (Standardwert).
qr unit	Lange Ganzzahl	2	Maßeinheit für das Dokument: - Bei Wert = 0 ist die Einheit Punkte - Bei Wert = 1 ist die Einheit Zentimeter. - Bei Wert = 2 ist die Einheit Inches.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie eine ungültige Nummer in *Eigenschaft*, wird der Fehler -9852 generiert.

QR Get drop column

QR Get drop column (Bereich) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	→	Referenz auf den Bereich
Funktionsergebnis	Lange Ganzzahl	↩	Drop Wert

Beschreibung

Die Funktion **QR Get drop column** gibt einen Wert zurück, je nachdem, wo das Drop ausgeführt wurde:

- Ist der Wert negativ, gibt er eine Spaltennummer an (z.B. -3, wenn das Drop in der Spalte Nummer 3 durchgeführt wurde.)
- Ist der Wert positiv, gibt er an, dass das Drop auf einem Trenner vor der Spalte durchgeführt wurde. (z.B. 3, wenn das Drop nach Spalte 2 ausgeführt wurde). Beachten Sie, dass das Drop nicht vor einer vorhandenen Spalte möglich ist.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

QR GET HEADER AND FOOTER

QR GET HEADER AND FOOTER (Bereich ; Selector ; TitelLinks ; TitelMitte ; TitelRechts ; Höhe {; Bild {; BildAusrichtung}})

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Selector	Lange Ganzzahl	➡	1 = Kopfteil, 2 = Fußteil
TitelLinks	String	⬅	Text auf der linken Seite
TitelMitte	String	⬅	Text in der Mitte
TitelRechts	String	⬅	Text auf der rechten Seite
Höhe	Lange Ganzzahl	⬅	Höhe für Kopf- oder Fußteil
Bild	Bild	⬅	Anzuzeigendes Bild
BildAusrichtung	Lange Ganzzahl	⬅	Ausrichtungsattribut für Bild

Beschreibung

Der Befehl **QR GET HEADER AND FOOTER** liefert Inhalt und Größe des Kopf- bzw. Fußteils.
Über *Selector* wählen Sie Kopf- oder Fußteil aus:

- Ist *Selector* gleich 1, wird die Information des Kopfteils gefunden
- Ist *Selector* gleich 2, wird die Information des Fußteils gefunden

TitelLinks, *TitelMitte* und *TitelRechts* geben jeweils den Wert für den Kopfteil/Fußteil Links, Mitte und Rechts zurück.
Höhe gibt die Höhe des Kopfteils/Fußteils zurück, und zwar in der für den Bericht gewählten Einheit.
Bild gibt ein Bild zurück, das im Kopfteil bzw. Fußteil angezeigt wird.
BildAusrichtung ist das Attribut für die Ausrichtung des Bildes im Kopfteil/Fußteil.

- Gibt *BildAusrichtung* 1 zurück, ist das Bild linksbündig
- Gibt *BildAusrichtung* 2 zurück, ist das Bild zentriert
- Gibt *BildAusrichtung* 3 zurück, ist das Bild rechtsbündig

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie einen ungültigen Wert in *Selector*, wird der Fehler -9852 generiert.

Beispiel

Nachfolgender Code erhält die Werte der Titel sowie Größe des Kopfteils und zeigt sie als Meldungen an:

```
QR GET HEADER AND FOOTER(MyArea;1;$LeftText;$CenterText;$RightText;$height)
ALERT("Der linke Titel ist "+Char(34)+$LeftText+Char(34))
ALERT("Der zentrierte Titel ist "+Char(34)+$CenterText+Char(34))
ALERT("Der rechte Titel ist "+Char(34)+$RightText+Char(34))
ALERT("Die Höhe des Kopfteils ist "+String($height))
```

QR Get HTML template

QR Get HTML template (Bereich) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	→	Referenz des Bereichs
Funktionsergebnis	Text	↩	HTML Code, verwendet als Vorlage

Beschreibung

Die Funktion **QR Get HTML template** gibt die HTML Vorlage zurück, die aktuell für den Schnellbericht *Bereich* verwendet wird. Der zurückgegebene Wert ist ein Textwert, der den gesamten Inhalt der HTML Vorlage enthält.

Wurde keine spezifische Vorlage definiert, wird die Standardvorlage zurückgegeben. Beachten Sie, dass keine Vorlage zurückgegeben wird, wenn die Ausgabe weder manuell noch per Programmierung in die HTML Datei gesetzt wurde.

QR GET INFO COLUMN

QR GET INFO COLUMN (Bereich ; SpaltenNr ; Titel ; Objekt ; Ausblenden ; Größe ; Wiederholter Wert ; Format {; ErgebnisVar})

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
SpaltenNr	Lange Ganzzahl	➡	Spaltennummer
Titel	Text	⬅	Titel der Spalte
Objekt	Text	⬅	Dieser Spalte zugewiesenes Objekt
Ausblenden	Lange Ganzzahl	⬅	0 = Sichtbar, 1 = Ausgeblendet
Größe	Lange Ganzzahl	⬅	Spaltengröße
Wiederholter Wert	Lange Ganzzahl	⬅	0 = Nicht wiederholt, 1 = Wiederholt
Format	Text	⬅	Anzeigeformat der Daten
ErgebnisVar	Text	⬅	Name der Formelvariablen

Beschreibung

Modus Liste

Der Befehl **QR GET INFO COLUMN** findet die Parameter einer vorhandenen Spalte.

Bereich ist die Referenz auf den Schnellbericht *Bereich*.

SpaltenNr ist die Nummer der zu ändernden Spalte.

Titel gibt den Titel zurück, der im Kopfteil der Spalte erscheint.

Objekt gibt den Namen des aktuellen Objekts der Spalte zurück (Variable, Feldname oder Formel).

Hinweis: Der Befehl berücksichtigt keine virtuelle Struktur, die über die Befehle **SET TABLE TITLES** und **SET FIELD TITLES** definiert wurden. Der aktuelle Name des Feldes wird im Parameter *Objekt* zurückgegeben.

Ausblenden gibt an, ob die Spalte ein- oder ausgeblendet ist.

- Bei *Ausblenden* gleich 0 wird die Spalte eingeblendet,
- Bei *Ausblenden* gleich 1 wird die Spalte ausgeblendet.

Größe gibt die Größe der Spalte in Pixel zurück. Ist der zurückgegebene Wert negativ, ist die Spaltengröße automatisch.

WiederholterWert gibt den Status für Datenwiederholung zurück. Ändert sich zum Beispiel der Wert für ein Datenfeld oder eine Variable nicht von einem Datensatz zum nächsten, kann er wiederholt oder nicht wiederholt werden:

- Bei *WiederholterWert* gleich 0 werden die Werte nicht wiederholt,
- Bei *WiederholterWert* gleich 1 werden die Werte wiederholt.

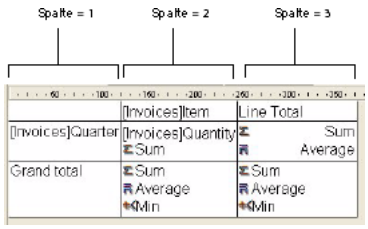
Format ist das Anzeigeformat. Anzeigeformate sind 4D Formate, die mit den angezeigten Daten kompatibel sind.

Ist der optionale Parameter *ErgebnisVar* übergeben, gibt er den Namen der Variablen zurück, den der Schnellberichteditor automatisch der Spalte mit Formel zugewiesen hat: "C1" für die erste Spalte mit Formel, "C2" für die zweite, etc. Hier speichert 4D das Ergebnis der letzten Ausführung der Formel der jeweiligen Spalte beim Generieren des Berichts.

Modus Kreuztabelle

Mit dem Befehl **QR GET INFO COLUMN** erhalten Sie dieselben Parameter, die Referenz auf die entsprechenden Bereiche ist jedoch anders und variiert je nach Parameter. In diesem Modus sind die Parameter *Titel*, *Ausblenden* und *WiederholterWert* ohne Bedeutung. Der Wert für *SpaltenNr* variiert, je nachdem, ob Sie Spaltengröße oder Datenquelle und Anzeigeformat erhalten wollen.

- Spaltengröße
  Dies ist ein visuelles Attribut, deshalb sind Spalten von links nach rechts, wie hier gezeigt, nummeriert:

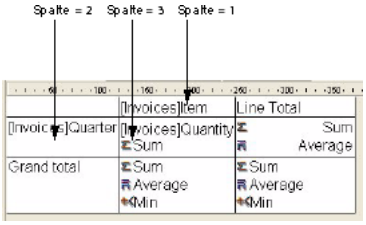


Die folgende Anweisung setzt die Größe für alle Spalten in einem Kreuztabellen-Bericht auf automatisch und lässt andere Elemente unverändert:

```
For($i;1;3)
   QR GET INFO COLUMN(qr_area;$i;$title;$obj;$hide;$size;$rep;$format)
   QR SET INFO COLUMN(qr_area;$i;$title;$obj;$hide;0;$rep;$format)
End for
```

Wollen Sie nur die Spaltengröße verändern, müssen Sie über den Befehl **QR GET INFO COLUMN** die Spalteneigenschaften aufrufen und bis auf die Spaltengröße unverändert in **QR SET INFO COLUMN** übernehmen.

- Datenquelle (Objekt) und Anzeigeformat
  In diesem Fall läuft die Spaltennummerierung wie folgt:



Übergeben Sie eine ungültige Bereichsnummer, wird der Fehler -9850 erzeugt.
Übergeben Sie einen ungültigen Wert für Spaltennummer, wird der Fehler -9852 erzeugt.

Beispiel

Sie haben folgenden Bericht gestaltet:

	[People]LastName	[People]FirstName	C1	[People]City	[People]Salary
Title	LastName	FirstName	Age	City	Salary
Detail					
Grand total					

Das Ergebnis der Spalte mit Formel speichern:

```
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 (Bereich ; Zeile) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den erstellten Bereich
Zeile	Lange Ganzzahl	➡	Anzeige Zeile
Funktionsergebnis	Lange Ganzzahl	⮌	0 = Sichtbar, 1 = Ausgeblendet

Beschreibung

Die Funktion **QR Get info row** erhält den Anzeigestatus der Zeile, deren Referenz in *Zeile* übergeben wurde.
Zeile zeigt an, für welche Zeile die Funktion gilt. Sie können entweder:

- Eine positive Ganzzahl für die entsprechende Umbruchebene der Untersumme angeben.
- Eine der vordefinierten Konstanten unter Thema **QR Zeilen für Eigenschaften** übergeben:

Konstante	Typ	Wert	Kommentar
qr detail	Lange Ganzzahl	-2	Detailbereich des Berichts
qr grand total	Lange Ganzzahl	-3	Bereich Gesamtsumme
qr title	Lange Ganzzahl	-1	Titel des Berichts

Das Funktionsergebnis gibt an, ob die Zeile sichtbar (Null) oder ausgeblendet (1) ist.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie einen ungültigen Wert in *Zeile*, wird der Fehler -9852 generiert.

QR Get report kind

QR Get report kind (Bereich) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	→	Referenz auf den Bereich
Funktionsergebnis	Lange Ganzzahl	↩	Art des Berichts

Beschreibung

Die Funktion **QR Get report kind** erhält die Berichtart für die in *Bereich* übergebene Referenz.

- Gibt die Funktion 1 zurück, ist der Bericht vom Typ Liste
- Gibt die Funktion 2 zurück, ist der Bericht vom Typ Kreuztabelle

Sie können das Funktionsergebnis auch mit den Konstanten unter dem Thema **QR Berichtstypen** vergleichen:

Konstante	Typ	Wert
qr cross report	Lange Ganzzahl	2
qr list report	Lange Ganzzahl	1

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

QR Get report table

QR Get report table (Bereich) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	→	Referenz auf den Bereich
Funktionsergebnis	Lange Ganzzahl	↩	Tabellennummer

Beschreibung

Die Funktion **QR Get report table** gibt die aktuelle Tabellennummer für den Bericht mit der in *Bereich* übergebenen Referenz zurück.
Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

QR GET SELECTION

QR GET SELECTION (Bereich ; Links ; Oben {; Rechts {; Unten}})

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Links	Lange Ganzzahl	⬅	Linker Rand
Oben	Lange Ganzzahl	⬅	Oberer Rand
Rechts	Lange Ganzzahl	⬅	Rechter Rand
Unten	Lange Ganzzahl	⬅	Unterer Rand

Beschreibung

Der Befehl **QR GET SELECTION** gibt die Koordinaten der ausgewählten Zelle zurück.

Links gibt die Nummer der Spalte am linken Rand der Auswahl zurück. Bei *Links* gleich 0 (Null), ist die ganze Zeile ausgewählt.

Oben gibt die Nummer der Zeile am oberen Rand der Auswahl zurück. Bei *Oben* gleich 0 (Null) wird die gesamte Spalte ausgewählt.

Hinweis: Sind *Links* und *Oben* gleich 0 (Null), wird der gesamte Bereich hervorgehoben.

Rechts ist die Nummer der Spalte am rechten Rand der Auswahl.

Unten ist die Nummer der Zeile am unteren Rand der Auswahl.

Hinweis: Gibt es keine Auswahl, werden *Links*, *Oben*, *Rechts* und *Unten* auf -1 gesetzt.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

QR GET SORTS

QR GET SORTS (Bereich ; Spalten ; Reihenfolge)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Spalten	Array Zahl	⬅	Sortierte Spalten
Reihenfolge	Array Zahl	⬅	Sortierreihenfolge

Beschreibung

Der Befehl **QR GET SORTS** füllt zwei Arrays:

- *Spalten*
  Dieses Array enthält alle Spalten mit einer Sortierreihenfolge.
- *Reihenfolge*
  Jedes Element dieses Array enthält die Sortierreihenfolgen der zutreffenden Spalte.
  - Bei *Reihenfolge{$i}* gleich 1 ist die Sortierreihenfolge aufsteigend.
  - Bei *Reihenfolge{$i}* gleich -1 ist die Sortierreihenfolge absteigend.

Modus Kreuztabelle

Im Modus Kreuztabelle kann das Array mit dem Ergebnis nicht mehr als zwei Elemente haben, da die Sortierung nur für Spalten (1) und Zeilen (2) durchführbar ist. (Werte für *Spalten*).

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

QR Get text property

QR Get text property (Bereich ; SpaltenNr ; ZeilenNr ; Eigenschaft) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
SpaltenNr	Lange Ganzzahl	➡	Spaltennummer
ZeilenNr	Lange Ganzzahl	➡	Nummer der Zeile
Eigenschaft	Lange Ganzzahl	➡	Nummer der Eigenschaft
Funktionsergebnis	Lange Ganzzahl, String	⮌	Wert der gewählten Eigenschaft

Beschreibung

Die Funktion **QR Get text property** gibt den Wert der Eigenschaft für die Textattribute der mit *SpaltenNr* und *ZeilenNr* festgelegten Zelle zurück.
Bereich ist die Referenz auf den Schnellbericht.
SpaltenNr ist die Nummer der Zellenspalte.
ZeilenNr ist die Referenz auf die Zellenzeile. Sie können folgendes übergeben:

- Einen positiven Wert, der die entsprechende Ebene für Zwischensumme (Umbruchebene) bezeichnet
- Eine Konstante unter dem Thema **QR Zeilen für Eigenschaften**:

Konstante	Typ	Wert	Kommentar
qr detail	Lange Ganzzahl	-2	Detailbereich des Berichts
qr footer	Lange Ganzzahl	-5	Seite Fußteil
qr grand total	Lange Ganzzahl	-3	Bereich Gesamtsumme
qr header	Lange Ganzzahl	-4	Seite Kopfteil
qr title	Lange Ganzzahl	-1	Titel des Berichts

Hinweise:
- Übergeben Sie als *ZeilenNr* -4 oder -5, müssen Sie auch in *SpaltenNr* eine Spaltennummer übergeben, auch wenn sie nicht benützt wird.
- Im Modus Kreuztabelle ist das Prinzip ähnlich, außer dass die Werte für Zeilen immer positiv sind.

Eigenschaft ist der Wert des zu erhaltenden Textattributs. Sie können die Konstanten unter dem Thema **QR Texteigenschaften** verwenden. Folgende Werte werden zurückgegeben:

Konstante	Typ	Wert	Kommentar
_o_qr font	Lange Ganzzahl	1	Ab 4D v14 R3 überholt (qr font name verwenden)
qr alternate background color	Lange Ganzzahl	9	Nummer für wechselnde Hintergrundfarbe
qr background color	Lange Ganzzahl	8	Nummer für Hintergrundfarbe
qr bold	Lange Ganzzahl	3	Stilattribut für Fett (0 oder 1)
qr font name	Lange Ganzzahl	10	Schriftname, wie er z.B. vom Befehl **FONT LIST** zurückgegeben wird
qr font size	Lange Ganzzahl	2	Schriftgröße ausgedrückt in Punkten (9 bis 255)
qr italic	Lange Ganzzahl	4	Stilattribut für Kursiv (0 oder 1)
qr justification	Lange Ganzzahl	7	Attribut für Ausrichtung (0 = Standard, 1 = linksbündig, 2 = zentriert, 3 = rechtsbündig)
qr text color	Lange Ganzzahl	6	Attribut für Farbnummer (Lange Ganzzahl)
qr underline	Lange Ganzzahl	5	Stilattribut für Unterstrichen

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie eine ungültige Nummer in *SpaltenNr*, wird der Fehler -9852 generiert.
Übergeben Sie eine ungültige Nummer in *ZeilenNr*, wird der Fehler -9853 generiert.
Übergeben Sie eine ungültige Nummer in *Eigenschaft*, wird der Fehler -9854 generiert.

QR GET TOTALS DATA

QR GET TOTALS DATA (Bereich ; SpaltenNr ; UmbruchNr ; Operator ; Text)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
SpaltenNr	Lange Ganzzahl	➡	Nummer der Spalte
UmbruchNr	Lange Ganzzahl	➡	Nummer des Umbruchs
Operator	Lange Ganzzahl	⬅	Operatorwert für die Zelle
Text	String	⬅	Inhalt der Zelle

Beschreibung

Modus Liste

Der Befehl **QR GET TOTALS DATA** liefert Einzelheiten zu einem bestimmten Umbruch.

Bereich ist die Referenz auf den Bereich Schnellbericht.

UmbruchNr ist die Nummer des Umbruchs, dessen Daten Sie erhalten

- Zwischensumme: Zwischen 1 und Anzahl der Zwischensumme/Sortierung.
- Gesamtsumme: -3 / Konstante: qr grand total

SpaltenNr ist die Nummer der Spalte, deren Daten Sie erhalten.

Operator ist die Summe aller in der Zelle vorhandenen Operatoren. Sie können zum Bearbeiten des zurückgegebenen Werts eine Konstante unter dem Thema **QR Operatoren** verwenden:

Konstante	Typ	Wert
qr average	Lange Ganzzahl	2
qr count	Lange Ganzzahl	16
qr max	Lange Ganzzahl	8
qr min	Lange Ganzzahl	4
qr standard deviation	Lange Ganzzahl	32
qr sum	Lange Ganzzahl	1

Text gibt den in der Zelle vorhandenen Text zurück.

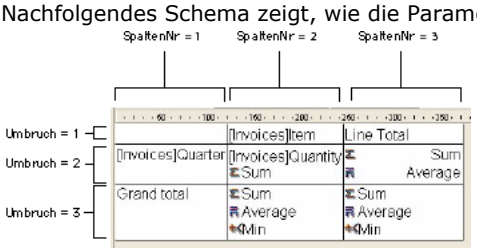
Hinweis: *Operator* und *Text* schließen sich gegenseitig aus. Sie erhalten entweder ein Ergebnis in *Operator* oder in *Text*.

Modus Kreuztabelle

Mit dem Befehl **QR GET TOTALS DATA** erhalten Sie Einzelheiten zu einer bestimmten Zelle.
Bereich ist die Referenz auf den Bereich des Schnellberichts.
SpaltenNr ist die Spaltennummer der Zelle, deren Daten gefunden werden.
UmbruchNr ist die Zeilennummer der Zelle, deren Daten gefunden werden.
Operator gibt die Summe aller in der Zelle vorhandenen Operatoren zurück. Sie können zum Bearbeiten des zurückgegebenen Werts eine Konstante unter dem Thema **QR Operatoren** verwenden (siehe oben).

Text gibt den in Zellen vorhandenen Text zurück.

Nachfolgendes Schema zeigt, wie die Parameter in einer Kreuztabelle kombiniert werden:



Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie eine ungültige Nummer in *SpaltenNr*, wird der Fehler -9852 generiert.
Übergeben Sie eine ungültige Nummer in *UmbruchNr*, wird der Fehler -9853 generiert.

QR GET TOTALS SPACING

QR GET TOTALS SPACING (Bereich ; ZwischensumNr ; Wert1...N)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
ZwischensumNr	Lange Ganzzahl	➡	Nummer der Zwischensumme
Wert1...N	Lange Ganzzahl	⬅	0=kein Abstand, 32000=fügt Seitenumbruch ein, >0=Abstand über Umbruchebene, <0=proportionale Erhöhung

Beschreibung

Der Befehl **QR GET TOTALS SPACING** liefert einen Abstand unter der Zeile mit der Zwischensumme. Der Befehl gilt nur für den Listenmodus.

Bereich ist die Referenz auf den Bereich Schnellbericht.

ZwischensumNr ist die betreffende Ebene für Umbruch bzw. Zwischensumme.

Wert gibt den Wert des Abstands an:

- Ist *Wert* gleich 0, wird kein Abstand hinzugefügt.
- Ist *Wert* gleich 32000, wird ein Seitenumbruch eingefügt.
- Ist *Wert* ein positiver Wert, gibt er den Abstand in Pixel an.
- Ist *Wert* ein negativer Wert, gibt er den Abstand als Prozentsatz der Zeile mit der Zwischensumme an. Zum Beispiel, -100 setzt einen Abstand von 100% unter die Zeile Zwischensumme.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie eine ungültige Nummer in *ZwischensumNr*, wird der Fehler -9852 generiert.

QR INSERT COLUMN

QR INSERT COLUMN (Bereich ; SpaltenNr ; Objekt)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
SpaltenNr	Lange Ganzzahl	➡	Nummer der Spalte
Objekt	Feld, Variable, Zeiger	➡	Einzufügendes Objekt für Spalte

Beschreibung

Der Befehl **QR INSERT COLUMN** fügt an der angegebenen Position eine Spalte ein oder erstellt sie neu. Rechts davon gelegene Spalten werden entsprechend verschoben.

SpaltenNr ist die Nummer der Spalte, beginnend von links nach rechts.

Der Standardtitel für die Spalte ist der in *Objekt* übergebene Wert.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

Hinweis: Dieser Befehl lässt sich nicht im Modus Kreuztabelle verwenden.

Beispiel

Nachfolgende Anweisung fügt die erste Spalte ein oder erstellt sie in einem Bereich Schnellbericht, fügt "Feld1" als Spaltentitel (Standardverhalten) ein und füllt den Inhalt des Hauptteils mit Werten aus Feld1.

```
QR INSERT COLUMN(MeinBereich;1;->[Tabelle 1]Feld1)
```

QR MOVE COLUMN

QR MOVE COLUMN (Bereich ; Spalte ; NeuePos)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Spalte	Lange Ganzzahl	➡	Spaltennummer
NeuePos	Lange Ganzzahl	➡	Neue Position für Spalte

Beschreibung

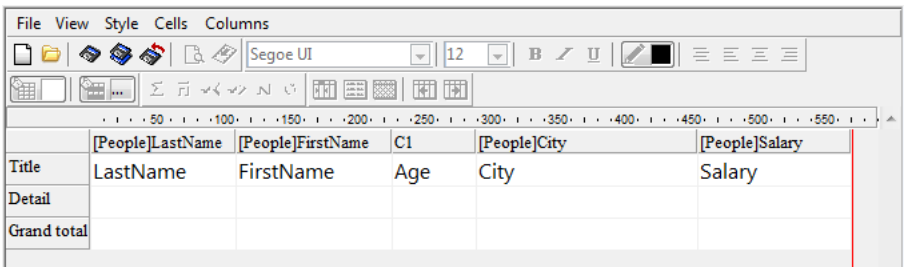
Der Befehl **QR MOVE COLUMN** bewegt die aktuelle Spalte an der Position *Spalte* auf die neue Position *NeuePos*.

Beide Parameter *Spalte* und *NeuePos* müssen gültige Spaltennummern sein (zwischen 1 und der Gesamtanzahl Spalten im Bericht); sonst wird der Fehler -9852 zurückgegeben.

Hinweis: Dieser Befehl lässt sich nur für Listenberichte verwenden.

Beispiel

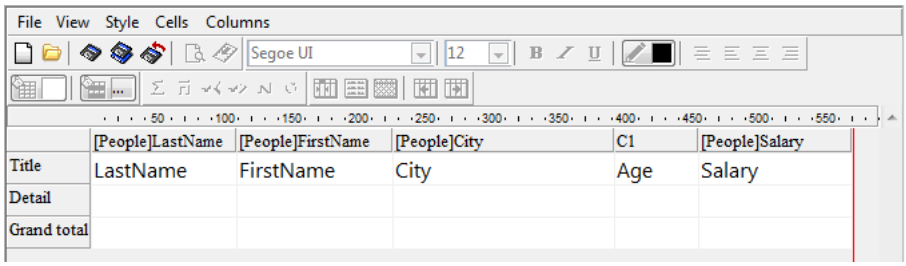
Sie haben folgenden Bericht erstellt:



Führen Sie folgenden Code aus:

```
QR MOVE COLUMN(Bereich;3;4)
```

erhalten Sie als Ergebnis:



QR NEW AREA

QR NEW AREA (ptr)

Parameter	Typ		Beschreibung
ptr	Zeiger	➡	Zeiger auf eine Variable

Beschreibung

Der Befehl **QR NEW AREA** erstellt einen neuen Bereich für Schnellbericht und speichert seine Referenznummer in der Variablen vom Typ Lange Ganzzahl, angegeben im Parameter *ptr*.

QR New offscreen area

QR New offscreen area -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Referenz auf den erstellten Bereich

Beschreibung

Die Funktion **QR New offscreen area** erstellt einen neuen Offscreen Bereich für den Schnellbericht und gibt die Referenz darauf zurück.

QR ON COMMAND

QR ON COMMAND (Bereich ; Methodenname)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Methodenname	String	➡	Name der ersetzenden Methode

Hinweis zur Kompatibilität

Ab 4D v16 enthalten 64-bit Versionen von 4D den neuen Editor **Schnellberichte (64-bit)**. Die Oberfläche dieses Editors mit Werkzeugleiste lässt sich nicht anpassen. Deshalb ist **QR ON COMMAND** in 64-bit Versionen mit den meisten Konstanten nicht verwendbar (siehe auch **QR EXECUTE COMMAND**).

Beschreibung

Der Befehl **QR ON COMMAND** führt die in *Methodenname* übergebene 4D Methode aus, wenn der Benutzer durch Wählen eines Menübefehls oder Anklicken einer Schaltfläche einen Befehl für Schnellbericht auslöst.
Ist *Bereich* gleich Null, gilt *Methodenname* für jeden Bereich des Schnellberichts, bis die Datenbank geschlossen oder der Befehl folgendermaßen aufgerufen wird: **QR ON COMMAND**(0;"")

Methodenname empfängt zwei Parameter:

- $1 ist die Referenz auf den Bereich (Lange Ganzzahl)
- $2 ist die Befehlsnummer des gewählten Befehls (Lange Ganzzahl). Sie können diesen Wert mit den Konstanten unter dem Thema **QR Befehle** vergleichen.

Hinweis: Wollen Sie die Datenbank kompilieren, müssen Sie $1 und $2 als Lange Ganzzahl deklarieren, auch wenn Sie diese nicht benutzen.

Soll der ursprüngliche Befehl ausgeführt werden, muss die aufgerufene Methode folgendes enthalten:
QR EXECUTE COMMAND($1;$2).

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

QR REPORT

QR REPORT ({Tabellenname ;} Dokumentname {; Hierarchisch {; Assistent {; Suchen {; Methodenname {; *}}}}})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Zu druckende Tabelle, ohne Angabe Standardtabelle
Dokumentname	String	➡	Zu ladendes Dokument Schnellbericht
Hierarchisch	Boolean	➡	Wahr = Zeigt N verknüpfte Tabellen Falsch/ohne Angabe = Nicht anzeigen (Standard)
Assistent	Boolean	➡	Wahr = Zeigt Schaltfläche Assistent an Falsch/ohne Angabe = Nicht anzeigen (Standard)
Suchen	Boolean	➡	Wahr = Zeigt Suchwerkzeuge und Haupttabelle Falsch/ohne Angabe = Nicht anzeigen (Standard)
Methodenname	String	➡	Name der aufzurufende Methode
*	Operator	➡	Druckdialoge löschen

Beschreibung

QR REPORT druckt einen Bericht für *Tabellenname*, der mit dem Schnellberichteditor erstellt wurde. Mit diesem Editor können Benutzer ihre eigenen Berichte erstellen. Weitere Informationen dazu finden Sie im Abschnitt **Schnellberichte** oder **Schnellberichte (64-bit)** des Handbuchs *4D Designmodus*.

Hinweise:

- Der Editor erscheint nicht, wenn *Tabellenname* als "Unsichtbar" definiert ist.
- Wird der Editor über den Befehl **QR REPORT** aufgerufen,behalten Verknüpfungen zwischen Tabellen ihren manuellen Status, sofern eingestellt. So kann der Entwickler den Status über die Befehle **SET AUTOMATIC RELATIONS** und **SET FIELD RELATION** selbst steuern. In 32-bit Versionen wird die Option **Alle Verknüpfungen auf automatisch** zum Ändern des Status automatisch/manuell der Verknüpfungen ausgeblendet.
- Der Editor wird in einem externen Fenster aufgerufen. In diesem Kontext kann der Befehl **QR ON COMMAND** nicht verwendet werden. Über den Parameter *Methodenname* können Sie jedoch eigenen Code ausführen, wenn ein Befehl der Oberfläche aktiviert ist (siehe unten).

Dokumentname (String) ist ein mit dem Schnellberichteditor erstellter und gesicherter Bericht. Dabei werden die Formatierungen des Berichts gesichert, nicht die darin enthaltenen Datensätze.
Ist *Dokumentname* leer (""), öffnet **QR REPORT** den Standard-Öffnen Dialog und der Benutzer kann den Bericht zum Drucken auswählen.
Gibt *Dokumentname* ein Dokument an, das nicht vorhanden ist, z.B.**Char(1)**, wird der Schnellberichteditor geöffnet.

Nur 32-bit Versionen:

- *Hierarchisch* (Boolean) definiert, ob in der Auswahlliste für Felder N-Tabellen angezeigt werden. Der Parameter hat standardmäßig den Wert 0 (Null), d.h. es werden keine verknüpften Tabellen angezeigt.
- *Assistent* (Boolean) gibt an, ob die Schaltfläche **Öffne Assistent** im Schnellberichteditor angezeigt wird. Der Parameter hat standardmäßig den Wert 0 (Null), d.h. der Assistent ist nicht verfügbar.
- *Suchen* (Boolean) gibt an, ob die Schaltfläche **Neue Suche** und die DropDown-Liste **Haupttabelle** im Schnellberichteditor angezeigt werden, um die aktuellen Tabellen bzw. die aktuelle Haupttabelle zu verändern. Der Parameter hat standardmäßig den Wert 0 (Null), d.h. es ist kein Zugriff auf die Suchwerkzeuge und die Haupttabelle möglich.
- Der Parameter *Methodenname* gibt eine 4D Projektmethode an, die ausgeführt wird, wenn ein Befehl des Schnellberichteditors durch Auswählen eines Menübefehls oder Anklicken einer Schaltfläche aufgerufen wird.
  Dieser Parameter ist die Entsprechung zum Befehl **QR ON COMMAND** im Rahmen des Dialogfensters Schnellberichteditor (**QR ON COMMAND** funktioniert nur für eingebundene Bereiche). Der neue Parameter eignet sich besonders, um den vom Schnellbericht verwendeten Zeichensatz zu ändern.
  Die Methode *Methodenname* empfängt zwei Parameter:
  - $1 enthält die Referenz des Bereichs (Lange Ganzzahl)
  - $2 enthält die Nummer des ausgewählten Befehls (Lange Ganzzahl). Sie können diesen Wert mit den Konstanten unter dem Thema **QR Befehle** vergleichen.

  Hinweis: Wenn Sie Ihre Anwendung kompilieren, müssen Sie die Parameter $1 und $2 explizit als Lange Ganzzahl deklarieren, auch wenn Sie diese nicht benutzen.

  Wollen Sie den anfangs vom Benutzer gewählten Befehl ausführen, verwenden Sie für die Methode *Methodenname* folgende Anweisung:

```
QR EXECUTE COMMAND($1;$2)
```

  Ist der Parameter *Methodenname* ein leerer String ("") oder wird er weggelassen, wird keine Methode aufgerufen und die Standardoperation von **QR REPORT** angewandt.

Ist ein Bericht ausgewählt, erscheinen die Dialogfenster zum Drucken. Mit dem Parameter * unterdrücken Sie die Druckdialoge. 4D benutzt dann die Druckeinstellungen, die bei der Erstellung des Berichts angegeben wurden.

Ist der Schnellberichteditor nicht beteiligt, hat die Variable OK den Wert 1, wenn ein Bericht gedruckt wird; den Wert 0 (Null), wenn nicht gedruckt wird (der Benutzer hat z.B. im Druckdialog auf die Schaltfläche **Abbrechen** geklickt).

4D Server: Dieser Befehl lässt sich auf 4D Server im Rahmen einer Serverprozedur ausführen. In diesem Kontext müssen Sie folgendes beachten:

- Stellen Sie sicher, dass auf dem Server Rechner kein Dialogfenster erscheint, außer für spezifische Anforderungen. Dazu müssen Sie den Befehl mit dem Parameter * aufrufen.

- Die Syntax zum Aufrufen des Schnellberichteditors funktioniert nicht mit 4D Server. In diesem Fall wird die Systemvariable OK auf 0 (Null) gesetzt.
- Bei einem Druckerproblem, z.B. kein Papier oder Drucker nicht verfügbar, erscheint keine Fehlermeldung.

Beispiel 1

Folgendes Beispiel lässt den Benutzer die Tabelle [People] suchen und druckt dann automatisch den Bericht "Detailliste":

```
QUERY([People])
If(OK=1)
   QR REPORT([People];"Detailliste";*)
End if
```

Beispiel 2

Folgendes Beispiel lässt den Benutzer die Tabelle [People] suchen und dann den Bericht für den Druck auswählen:

```
QUERY([People])
If(OK=1)
   QR REPORT([People];"")
End if
```

Beispiel 3

Folgendes Beispiel lässt den Benutzer die Tabelle [People] suchen und zeigt dann den Berichteditor an, so dass der Benutzer – mit oder ohne Assistenten – einen beliebigen Bericht erstellen, sichern, laden und drucken kann:

```
QUERY([People])
If(OK=1)
   QR REPORT([People];Char(1);True)
End if
```

Beispiel 4

Siehe Beispiel zum Befehl **SET FIELD RELATION**.

Beispiel 5

Sie wollen den Zeichensatz in einem Schnellbericht, der über **QR REPORT** aufgerufen wird, in Mac Roman ändern:

```
QR REPORT([MyTable];Char(1);False;False;False;"myCallbackMeth")
```

Die Methode ***myCallbackMeth*** konvertiert den Bericht, wenn er erzeugt wird:

```
C_LONGINT($1;$2)
If($2=qr cmd generate)  //wurde ein Bericht generiert
   C_BLOB($myblob)
   C_TEXT($path;$text)
   C_LONGINT($type)
   QR EXECUTE COMMAND($1;$2)  //den Befehl ausführen
   QR GET DESTINATION($1;$type;$path)  //Ziel finden
   If(($type=qr HTML file)|($type=qr text file))
      DOCUMENT TO BLOB($path;$myblob)
 //Konvertieren in Text in UTF-8
      $text:=Convert to text($myblob;"UTF-8")
 //Zeichensatz MacRoman verwenden
      CONVERT FROM TEXT($text;"MacRoman";$myblob)
 //Rückgabe des konvertierten Berichts
      BLOB TO DOCUMENT($path;$myblob)
   End if
Else  //andernfalls den Befehl ausführen
   QR EXECUTE COMMAND($1;$2)
End if
```

QR REPORT TO BLOB

QR REPORT TO BLOB (Bereich ; BLOB)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
BLOB	BLOB	⬅	Blob das den Schnellbericht enthält

Beschreibung

Der Befehl **QR REPORT TO BLOB** setzt den Bericht mit der in *Bereich* übergebenen Referenz in ein BLOB (Variable oder Feld). Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

Beispiel

Nachfolgender Code weist den in *MyArea* gespeicherten Schnellbericht einem Feld vom Typ BLOB zu.

```
QR REPORT TO BLOB(MyArea;[Table 1]Field4)
```

QR RUN

QR RUN (Bereich)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den auszuführenden Bereich

Beschreibung

Der Befehl **QR RUN** führt den Schnellbericht aus, dessen Referenz in *Bereich* mit den aktuellen Einstellungen des Schnellberichts inkl. Ausgabetyp übergeben wurde. Verwenden Sie den Befehl **QR SET DESTINATION**, um den Ausgabetyp zu verändern.

Der Bericht wird mit der Tabelle ausgeführt, der *Bereich* zugeordnet ist. Bezeichnet *Bereich* einen Offscreen-Bereich, müssen Sie die Tabelle über den Befehl **QR SET REPORT TABLE** spezifizieren.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

4D Server: Dieser Befehl lässt sich auf 4D Server in einer Serverprozedur ausführen. Achten Sie in diesem Kontext darauf, dass kein Druckdialog auf dem Server Rechner erscheint (außer für spezifische Anforderungen). Dazu müssen Sie den Befehl **QR SET DESTINATION** mit dem Parameter "*" aufrufen. Bei Druckerproblemen, wie Papierstau, Drucker abgeschaltet, etc., erscheint keine Fehlermeldung.

QR SET AREA PROPERTY

QR SET AREA PROPERTY (Bereich ; Eigenschaft ; Wert1...N)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Eigenschaft	Lange Ganzzahl	➡	Festgelegtes Oberflächenelement
Wert1...N	Lange Ganzzahl	➡	0 = Sichtbar, 1 = Ausgeblendet

Beschreibung

Der Befehl **QR SET AREA PROPERTY** zeigt das in *Eigenschaft* übergebene Oberflächenelement (Werkzeugleiste oder Menüleiste) an oder blendet es aus.

Menüleiste und Werkzeugleiste sind von oben nach unten von 1 bis 6 nummeriert, Wert 7 ist für das Kontextmenü reserviert. Sie können vordefinierte Konstanten unter dem Thema **QR Bereichseigenschaften** verwenden:

Konstante	Typ	Wert	Kommentar
qr view color toolbar	Lange Ganzzahl	5	Zeige Status der Werkzeugleiste für Farben (Angezeigt=1, Ausgeblendet=0)
qr view column toolbar	Lange Ganzzahl	6	Zeige Status der Werkzeugleiste für Spalten (Angezeigt=1, Ausgeblendet=0)
qr view contextual menus	Lange Ganzzahl	7	Zeige Status des Kontextmenüs (Angezeigt=1, Ausgeblendet=0)
qr view menubar	Lange Ganzzahl	1	Zeige Status der Menüleiste (Angezeigt=1, Ausgeblendet=0)
qr view operators toolbar	Lange Ganzzahl	4	Zeige Status der Werkzeugleiste für Operatoren (Angezeigt=1, Ausgeblendet=0)
qr view standard toolbar	Lange Ganzzahl	2	Zeige Status der Standard Werkzeugleiste (Angezeigt=1, Ausgeblendet=0)
qr view style toolbar	Lange Ganzzahl	3	Zeige Status der Werkzeugleiste für Stil (Angezeigt=1, Ausgeblendet=0)

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie einen ungültigen Parameter in *Eigenschaft*, wird der Fehler -9852 generiert.

QR SET BORDERS

QR SET BORDERS (Bereich ; Spalte ; Zeile ; Rand ; Linie {; Farbe})

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Spalte	Lange Ganzzahl	➡	Nummer der Spalte
Zeile	Lange Ganzzahl	➡	Nummer der Zeile
Rand	Lange Ganzzahl	➡	Zusammengesetzter Wert für Rand
Linie	Lange Ganzzahl	➡	Linienstärke
Farbe	Lange Ganzzahl	➡	Rahmenfarbe

Beschreibung

Der Befehl **QR SET BORDERS** setzt den Randstil für eine bestimmte Zelle.

Bereich ist die Referenz auf den Bereich Schnellbericht.
Spalte ist die Spaltennummer der Zelle.
Zeile ist Zeilennummer der Zelle. Sie können folgendes übergeben:
- Eine positive Ganzzahl zur Bezeichnung der entsprechenden Ebene für Zwischensumme (Umbruchebene)
- Eine Konstante unter dem Thema **QR Zeilen für Eigenschaften**:

Konstante	Typ	Wert	Kommentar
qr detail	Lange Ganzzahl	-2	Detailbereich des Berichts
qr grand total	Lange Ganzzahl	-3	Bereich Gesamtsumme
qr title	Lange Ganzzahl	-1	Titel des Berichts

Rand ist ein zusammengesetzter Wert, der die zutreffenden Ränder der Zelle angibt: Sie können die Ränder über Konstanten unter dem Thema **QR Rahmen** definieren:

Konstante	Typ	Wert	Kommentar
qr bottom border	Lange Ganzzahl	8	Unterer Rand
qr inside horizontal border	Lange Ganzzahl	32	Innerer horizontaler Rand
qr inside vertical border	Lange Ganzzahl	16	Innerer vertikaler Rand
qr left border	Lange Ganzzahl	1	Linker Rand
qr right border	Lange Ganzzahl	4	Rechter Rand
qr top border	Lange Ganzzahl	2	Oberer Rand

Beispiel: Der Wert 5 in *Rand* betrifft den rechten und linken Rand.

Linie ist die Stärke der Linie:

- 0 bedeutet keine Linie
- 1 gibt eine Stärke von 1/4 Punkt an
- 2 gibt eine Stärke von 1/2 Punkt an
- 3 gibt eine Stärke von 1 Punkt an
- 4 gibt eine Stärke von 2 Punkt an

Farbe ist die Farbe der Linie:

- Ist *Farbe* ein positiver Wert, gibt er eine bestimmte Farbe an.
- Ist *Farbe* gleich 0, gilt die Farbe schwarz.
- Ist *Farbe* gleich -1, soll nichts verändert werden.

Hinweis: Die Standardfarbe ist Schwarz.

Übergeben Sie eine ungültige Bereichsnummer, wird der Fehler -9850 generiert.
Übergeben Sie eine ungültige Spaltennummer, wird der Fehler -9852 generiert.
Übergeben Sie eine ungültige Zeilennummer, wird der Fehler -9853 generiert.
Übergeben Sie einen ungültigen Rahmenparameter, wird der Fehler -9854 generiert.
Übergeben Sie einen ungültigen Zeilenparameter, wird der Fehler -9855 generiert.

QR SET DESTINATION

QR SET DESTINATION (Bereich ; Typ {; Merkmale})

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Typ	Lange Ganzzahl	➡	Typ des Berichts
Merkmale	String, Variable	➡	Merkmale zum Ausgabetyp

Beschreibung

Der Befehl **QR SET DESTINATION** setzt die Ausgabeart *Typ* des Berichts mit der Referenz, die in *Bereich* übergeben wurde.

Im Parameter *Typ* können Sie eine Konstante aus dem Thema **QR Ausgabeart** übergeben. Der Inhalt des Parameters *Merkmale* richtet sich nach dem Wert von *Typ*. Nachfolgende Tabelle beschreibt die Werte für die Parameter *Typ* und *Merkmale*:

Konstante	Typ	Wert	Kommentar
_o_qr 4D Chart area	Lange Ganzzahl	4	***Konstante ist überholt***
qr 4D View area	Lange Ganzzahl	3	*Detail*: N.A.
qr HTML file	Lange Ganzzahl	5	*Detail*: Pfadname zur Datei
qr printer	Lange Ganzzahl	1	*Details*: "*" um kein Dialogfenster beim Drucken anzuzeigen
qr text file	Lange Ganzzahl	2	*Detail*: Pfadname zur Datei

qr printer (1): Übergeben Sie im Parameter *Merkmale* eine Zeichenkette mit Stern ("*"), erscheint während dem Drucken kein Dialogfenster und es werden automatisch die aktuellen Druckeinstellungen verwendet. Diese Einstellung ist erforderlich, wenn Sie den Bericht auf dem Server drucken wollen.

qr text file (2): Übergeben Sie im Parameter *Merkmale* einen leeren String, erscheint ein Sichern-Dialogfenster, ansonsten wird die Datei an der durch den Pfad angegebenen Stelle gesichert.
Der standardmäßige Begrenzer für Feld ist der Tabulator (Code 9). Der standardmäßige Begrenzer für Datensatz ist die Zeilenschaltung (Code 13). Sie können diese Vorgabe ändern, indem Sie Werte in den Systemvariablen *FldDelimit* und *RecDelimit* eingeben. Enthält *FldDelimit* unter Windows den Wert 13, wird nach der Zeilenschaltung Wert 10 für Zeilenvorschub eingetragen. Bedenken Sie jedoch, dass diese Variablen auch bei anderen Befehlen wie **IMPORT TEXT** Verwendung finden. Wenn Sie diese für den Schnellberichteditor anpassen, gilt die Änderung überall in der Anwendung.

qr 4D View area (3): Ist 4D View für den Benutzer aktiv, wird ein externes 4D View Fenster erstellt und mit den Ergebnissen der aktuellen Einstellungen im Bereich Schnellbericht gefüllt.

qr HTML file (5): Über die mit **QR SET HTML TEMPLATE** definierte Vorlage wird eine HTML Datei erstellt. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert. Ist der Zielwert in *Typ* ungültig, wird der Fehler -9852 generiert.

Beispiel

Nachfolgender Code setzt als Ziel die Textdatei "MeinDok.txt" fest und führt den Schnellbericht aus:

```
QR SET DESTINATION(MeinDok;qr text file;"MeinDok.txt")
QR RUN(MeinDok)
```

QR SET DOCUMENT PROPERTY

QR SET DOCUMENT PROPERTY (Bereich ; Eigenschaft ; Wert1...N)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Eigenschaft	Lange Ganzzahl	➡	1 = Druckdialog, 0 = Dokumenteinheit
Wert1...N	Lange Ganzzahl	➡	Wert für Eigenschaft

Beschreibung

Der Befehl **QR SET DOCUMENT PROPERTY** zeigt den Druckdialog an oder definiert die Einheit für das Dokument.

In *Eigenschaft* können Sie eine Konstante unter dem Thema **QR Dokumenteigenschaften** verwenden:

Konstante	Typ	Wert	Kommentar
qr printing dialog	Lange Ganzzahl	1	Druckdialog anzeigen: - Bei Wert = 0 wird der Druckdialog nicht vor dem Drucken angezeigt. - Bei Wert = 1 wird der Druckdialog vor dem Drucken angezeigt (Standardwert).
qr unit	Lange Ganzzahl	2	Maßeinheit für das Dokument: - Bei Wert = 0 ist die Einheit Punkte - Bei Wert = 1 ist die Einheit Zentimeter. - Bei Wert = 2 ist die Einheit Inches.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie einen ungültigen Wert in *Eigenschaft* oder *Wert*, wird der entsprechende Fehler generiert (-9852 oder -9853).

QR SET HEADER AND FOOTER

QR SET HEADER AND FOOTER (Bereich ; Selector ; TitelLinks ; TitelMitte ; TitelRechts ; Höhe {; Bild {; BildAusrichtung}})

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Selector	Lange Ganzzahl	➡	1 = Kopfteil, 2 = Fußteil
TitelLinks	String	➡	Text auf der linken Seite
TitelMitte	String	➡	Text in der Mitte
TitelRechts	String	➡	Text auf der rechten Seite
Höhe	Lange Ganzzahl	➡	Höhe des Kopf- oder Fußteils
Bild	Bild	➡	Anzuzeigendes Bild
BildAusrichtung	Lange Ganzzahl	➡	Attribut Ausrichtung für Bild

Beschreibung

Der Befehl **QR SET HEADER AND FOOTER** setzt Inhalt und Größe des Kopf- bzw. Fußteils.
Über *Selector* wählen Sie Kopf- oder Fußteil aus:

- Ist *Selector* gleich 1, betrifft es den Kopfteil
- Ist *Selector* gleich 2, betrifft es den Fußteil

TitelLinks, *TitelMitte* und *TitelRechts* sind die Werte für den Kopfteil/Fußteil Links, Mitte und Rechts.
Höhe gibt die Höhe des Kopfteils/Fußteils zurück, und zwar in der für den Bericht gewählten Einheit.
Bild ist ein Bild, das im Kopfteil bzw. Fußteil angezeigt wird.
BildAusrichtung ist das Attribut für die Ausrichtung des Bildes im Kopfteil/Fußteil.

- Bei *BildAusrichtung* gleich 1, ist das Bild linksbündig
- Ist *BildAusrichtung* gleich 2, ist das Bild zentriert
- Ist *BildAusrichtung* gleich 3, ist das Bild rechtsbündig

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie einen ungültigen Wert in *Selector*, wird der Fehler -9852 generiert.

Beispiel

Nachfolgender Code setzt den Titel "centerTitle" im Kopfteil des Schnellberichts in *MyArea* und setzt die Höhe des Kopfteils auf 200 Punkte:

```
QR SET HEADER AND FOOTER(MyArea;1;"";centerTitle;"";200)
```

QR SET HTML TEMPLATE

QR SET HTML TEMPLATE (Bereich ; Vorlage)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Vorlage	Text	➡	HTML Vorlage

Beschreibung

Der Befehl **QR SET HTML TEMPLATE** setzt die aktuell verwendete HTML Vorlage für den Schnellbericht in *Bereich*. Diese Vorlage wird beim Erstellen des Berichts in HTML Format verwendet.

Die Vorlage arbeitet mit einer Reihe von Tags zur Gestaltung der Daten, entweder um ein möglichst orginalgetreues Layout des Berichts zu erhalten oder um diese an Ihre eigene HTML Datei anzupassen.

Hinweis: Sie müssen zuerst den Befehl **QR SET DESTINATION** aufrufen, um die Ausgabe in eine HTML Datei umzuwandeln.

HTML Tags

<!--#4DQRheader--> ... <!--/#4DQRheader-->
Der HTML Inhalt zwischen diesen Tags stammt von den Spaltentiteln. Mit diesen Tags definieren Sie die Titelzeile des Berichts.

<!--#4DQRrow--> ... <!--/#4DQRrow-->
Der HTML Inhalt zwischen diesen Tags wird für jede Datenzeile wiederholt (inkl. Zeilen für Hauptteil und Zwischensummen).

<!--#4DQRcol--> ... <!--/#4DQRcol-->
Der HTML Inhalt zwischen diesen Tags wird für jede Spalte mit Daten innerhalb der Zeile wiederholt. Die Reihenfolge der Spalten bleibt wie im Bericht erhalten. In Verbindung mit *<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->*, the tags *<!--#4DQRcol--> ... <!--/#4DQRcol-->* werden nur die Spalten durchlaufen, deren Inhalt nicht über *<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->* eingefügt wurde.
Beispiel: Hat ein Bericht fünf Spalten, können Sie *<!--#4DQRcol;2--> ... <!--/#4DQRcol;2-->* verwenden, um Daten aus der zweiten Spalte einzufügen, *<!--#4DQRcol--> ... <!--/#4DQRcol-->* gehen für jede Zeile durch die Spalten 1, 3, 4 und 5. Diese letzten Tags ignorieren Spalten, deren Inhalt über *<!--#4DQRcol;2--> ... <!--/#4DQRcol;2-->* veröffentlicht wird.

<!--#4DQRcol;n--> ... <!--/#4DQRcol;n-->
Der HTML Inhalt zwischen diesen Tags wird aus den Spalten im Bericht mit der Nummer "n" entnommen. Wollen Sie zum Beispiel in einem dreispaltigen Bericht eine andere Spaltenanordnung in der HTML Ausgabe verwenden, können Sie folgende Tags benutzen:
<!--#4DQRrow--> <!--#4DQRcol;3--> ... <!--/#4DQRcol;3--><!--#4DQRcol;2--> ... <!--/#4DQRcol;2--><!--#4DQRcol;1--> ... <!--/#4DQRcol;1--> <!--/#4DQRrow-->
In diesem Beispiel werden die Spalten in entgegengesetzter Reihenfolge des Berichts eingefügt.

<!--#4DQRfont--> ... <!--/#4DQRfont-->
Der HTML Inhalt zwischen diesen Tags wird der Schriftart und -grösse der aktuellen Spalte oder Zelle zugewiesen.
<!--#4DQRfont--> wird ersetzt durch eine Schriftdefinition für HTML, *<!--/#4DQRfont-->* wird ersetzt durch ein passendes schließendes Tag (**).

<!--#4DQRface--> ... <!--/#4DQRface-->
Der HTML Inhalt zwischen diesen Tags wird dem Schriftstil der aktuellen Spalte oder Zelle zugewiesen.
<!--#4DQRface--> wird ersetzt durch eine Ansichtsdefinition für HTML, *<!--#4DQRface-->* wird ersetzt durch ein passendes schließendes Tag (*</face>*).

<!--#4DQRbgcolor-->
Dieses Tag wird ersetzt durch die aktuelle Farbe für die aktuelle Zelle.

<!--#4DQRdata-->
Dieses Tag wird ersetzt durch die aktuellen Daten für die aktuelle Zelle.

<!--#4DQRlHeader--><!--#4DQRdata--><!--/#4DQRlHeader-->
<!--#4DQRcHeader--><!--#4DQRdata--><!--/#4DQRcHeader-->
<!--#4DQRrHeader--><!--#4DQRdata--><!--/#4DQRrHeader-->
Diese Tags werden jeweils ersetzt durch die Daten im linken, rechten oder zentrierten Kopfteil.

<!--#4DQRlFooter--><!--#4DQRdata--><!--/#4DQRlFooter-->
<!--#4DQRcFooter--><!--#4DQRdata--><!--/#4DQRcFooter-->
<!--#4DQRrFooter--><!--#4DQRdata--><!--/#4DQRrFooter-->
Diese Tags werden jeweils ersetzt durch die Daten im linken, rechten oder zentrierten Fußteil.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

QR SET INFO COLUMN

QR SET INFO COLUMN (Bereich ; SpaltenNr ; Titel ; Objekt ; Ausblenden ; Größe ; Wiederholter Wert ; Format)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
SpaltenNr	Lange Ganzzahl	➡	Spaltennummer
Titel	String	➡	Titel der Spalte
Objekt	Feld, Variable	➡	Dieser Spalte zugewiesenes Objekt
Ausblenden	Lange Ganzzahl	➡	0 = Sichtbar, 1 = Ausgeblendet
Größe	Lange Ganzzahl	➡	Größe der Spalte
Wiederholter Wert	Lange Ganzzahl	➡	0 = Nicht wiederholt, 1 = Wiederholt
Format	String	➡	Format für die Daten

Beschreibung

Modus Liste

Der Befehl **QR SET INFO COLUMN** setzt die Parameter einer vorhandenen Spalte.

Bereich ist die Referenz auf den Schnellbericht Bereich.
SpaltenNr ist die Nummer der zu ändernden Spalte.
Titel ist der Titel, der im Kopfteil der Spalte erscheint.
Objekt ist das aktuelle Objekt der Spalte (Variable, Feld oder Formel).
Ausblenden gibt an, ob die Spalte ein- oder ausgeblendet ist.
- Bei *Ausblenden* gleich 0 wird die Spalte eingeblendet,
- Bei *Ausblenden* gleich 1 wird die Spalte ausgeblendet.

Größe ist die Größe in Pixel für die Spalte. Bei *Größe* gleich -1 ist sie automatisch.

Format ist das Anzeigeformat. Anzeigeformate sind 4D Formate, die mit den angezeigten Daten kompatibel sind.

WiederholterWert ist der Status für Datenwiederholung. Ändert sich zum Beispiel der Wert für ein Datenfeld oder eine Variable nicht von einem Datensatz zum nächsten, kann er wiederholt oder nicht wiederholt werden:
- Bei *WiederholterWert* gleich 0 werden die Werte nicht wiederholt,
- Bei *WiederholterWert* gleich 1 werden die Werte wiederholt.

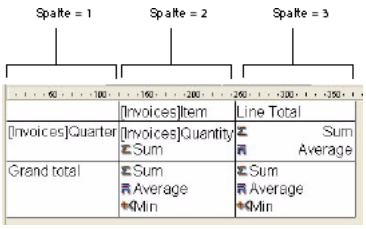
Die folgende Anweisung setzt den Titel der Spalte Nr 1 in Titel, den Inhalt des Hauptteils in *Feld2*, macht die Spalte mit der Breite 150 Pixel sichtbar und setzt das Format auf ###,##.

```
QR SET INFO COLUMN(area;1;"Titel";"[Tabelle 1]Feld2";0;150;0;"###,##")
```

Modus Kreuztabelle

Mit dem Befehl **QR SET INFO COLUMN** können Sie dieselben Parameter setzen, die Referenz auf die entsprechenden Bereiche ist jedoch anders und variiert je nach Parameter. In diesem Modus sind die Parameter *Titel*, *Ausblenden* und *WiederholterWert* ohne Bedeutung. Der Wert für *SpaltenNr* variiert, je nachdem, ob Sie Spaltengröße oder Datenquelle und Anzeigeformat erhalten wollen.

- Spaltengröße
  Dies ist ein visuelles Attribut, deshalb sind Spalten von links nach rechts, wie hier gezeigt, nummeriert:

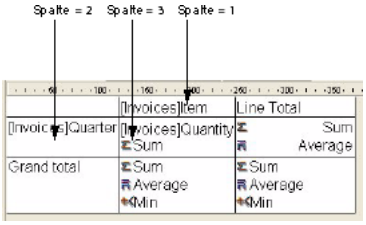


Die folgende Anweisung setzt die Größe für alle Spalten in einem Kreuztabellen-Bericht auf automatisch und lässt andere Elemente unverändert:

```
For($i;1;3)
   QR SET INFO COLUMN(qr_area;$i;$title;$obj;$hide;$size;$rep;$format)
   QR SET INFO COLUMN(qr_area;$i;$title;$obj;$hide;0;$rep;$format)
End for
```

Wollen Sie nur die Spaltengröße verändern, müssen Sie über den Befehl **QR GET INFO COLUMN** die Spalteneigenschaften aufrufen und bis auf die Spaltengröße unverändert in **QR SET INFO COLUMN** übernehmen.

- Datenquelle (Objekt) und Anzeigeformat
  In diesem Fall operiert die Spaltennummerierung wie folgt:

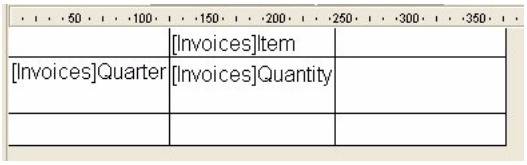


Sie werden feststellen, dass nicht alle Zellen über **QR SET INFO COLUMN** aufgerufen werden können. Die nicht nummerierten Zellen werden über den Befehl **QR SET TOTALS DATA** aufgerufen.

Nachfolgender Code weist den drei Zellen, die zum Erstellen eines einfachen Kreuztabellen-Berichts erforderlich sind, Datenquellen zu:

```
QR SET REPORT TABLE(qr_area;Table(->[Invoices]))
ALL RECORDS([Invoices])
QR SET REPORT KIND(qr_area;2)
QR SET INFO COLUMN(qr_area;1;"";->[Invoices]Item;1;-1;1;"")
QR SET INFO COLUMN(qr_area;2;"";->[Invoices]Quarter;1;-1;1;"")
QR SET INFO COLUMN(qr_area;3;"";->[Invoices]Quantity;1;-1;1;"")
```

Der Bereich des sich ergebenden Berichts sieht folgendermaßen aus:



Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie einen ungültigen Wert in *SpaltenNr*, wird der Fehler -9852 generiert.

QR SET INFO ROW

QR SET INFO ROW (Bereich ; Zeile ; Ausblenden)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den erstellten Bereich
Zeile	Lange Ganzzahl	➡	Anzeige der Zeile
Ausblenden	Lange Ganzzahl	➡	0 = Sichtbar, 1 = Ausblenden

Beschreibung

Der Befehl **QR SET INFO ROW** blendet die Zeile ein bzw. aus, deren Referenz in *Zeile* übergeben wurde.
Zeile zeigt an, für welche Zeile der Befehl gilt:

- Bei *Zeile* gleich -1 ist der Titel des Berichts betroffen
- Bei *Zeile* gleich -2 ist der Hauptteil des Berichts betroffen
- Bei *Zeile* gleich -3 ist die Gesamtsumme des Berichts betroffen
- Ist *Zeile* eine positive Ganzzahl, ist die Untersumme (Umbruchebene) betroffen.

Sie können den Zeilentyp über Konstanten zum Thema **QR Zeilen für Eigenschaften** bestimmen:

Konstante	Typ	Wert	Kommentar
qr detail	Lange Ganzzahl	-2	Detailbereich des Berichts
qr grand total	Lange Ganzzahl	-3	Bereich Gesamtsumme
qr title	Lange Ganzzahl	-1	Titel des Berichts

Ausblenden gibt an, ob die Zeile sichtbar (Null) oder ausgeblendet (1) ist.
Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie einen ungültigen Wert in *Zeile*, wird der Fehler -9852 generiert.

Beispiel

Folgende Anweisung blendet die Zeile Detail aus:

```
QR SET INFO ROW(area;qr detail;1)
```

QR SET REPORT KIND

QR SET REPORT KIND (Bereich ; Typ)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Typ	Lange Ganzzahl	➡	Typ des Berichts

Beschreibung

Der Befehl **QR SET REPORT KIND** setzt den Bericht *Typ*, dessen Referenz in *Bereich* übergeben wurde.

- Bei *Typ* gleich 1 ist der Bericht vom Typ Liste
- Bei *Typ* gleich 2 ist der Bericht vom Typ Kreuztabelle

Sie können auch Konstanten unter dem Thema **QR Berichtstypen** verwenden:

Konstante	Typ	Wert
qr cross report	Lange Ganzzahl	2
qr list report	Lange Ganzzahl	1

Setzen Sie einen neuen Typ für einen vorhandenen aktuellen Bericht, entfernt er die vorigen Einstellungen und erstellt einen neuen leeren Bericht.

QR SET REPORT TABLE

QR SET REPORT TABLE (Bereich ; Tabelle)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Tabelle	Lange Ganzzahl	➡	Tabellennummer

Beschreibung

Der Befehl **QR SET REPORT TABLE** setzt die aktuelle Tabelle für den Berichtteil, dessen Referenz in *Bereich* übergeben wurde, für die Tabelle mit der in *Tabelle* angegebenen Nummer.

Es ist wichtig, dass dem Bericht eine Tabelle zugewiesen ist, da der Berichteditor die aktuelle Auswahl für diese Tabelle verwendet, um die Daten anzuzeigen, Berechnungen durchzuführen und bei Bedarf Verknüpfungen zu ziehen.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie einen ungültigen Wert in *Tabelle*, wird der Fehler -9852 generiert.

QR SET SELECTION

QR SET SELECTION (Bereich ; Links ; Oben {; Rechts {; Unten}})

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Links	Lange Ganzzahl	➡	Linker Rand
Oben	Lange Ganzzahl	➡	Oberer Rand
Rechts	Lange Ganzzahl	➡	Rechter Rand
Unten	Lange Ganzzahl	➡	Unterer Rand

Beschreibung

Der Befehl **QR SET SELECTION** hebt eine Zelle, Zeile, Spalte oder den gesamten Bereich wie mit einem Mausklick hervor. Damit können Sie auch die aktuelle Auswahl abwählen.

Links gibt die Nummer des linken Rands zurück. Bei *Links* gleich 0 (Null), ist die ganze Zeile ausgewählt.

Oben gibt die Nummer des oberen Rands zurück. Bei *Oben* gleich 0 (Null) wird die gesamte Spalte ausgewählt.

Rechts ist die Nummer des rechten Rands.

Unten ist die Nummer des unteren Rands.

Hinweis: Sind *Links* und *Oben* gleich 0 (Null), wird der gesamte Bereich hervorgehoben.
Wollen Sie keine Auswahl, setzen Sie *Links*, *Oben*, *Rechts* und *Unten* auf -1.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

QR SET SORTS

QR SET SORTS (Bereich ; Spalten {; Reihenfolge})

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
Spalten	Array Zahl	➡	Spalten
Reihenfolge	Array Zahl	➡	Sortierreihenfolge

Beschreibung

Der Befehl **QR SET SORTS** setzt die Sortierreihenfolge für die Spalten des Berichts, dessen Referenz in *Bereich* übergeben wurde.

- *Spalten*
  In diesem Array müssen Sie die Nummern der Spalten speichern, die Sie mit einer Sortierreihenfolge versehen wollen.
- *Reihenfolge*
  Jedes Element dieses Array muss die Sortierreihenfolgen der zutreffenden Spalte im Array *Spalten* enthalten.
  - Bei *Reihenfolge{$i}* gleich 1 ist die Sortierreihenfolge aufsteigend.
  - Bei *Reihenfolge{$i}* gleich -1 ist die Sortierreihenfolge absteigend.

Modus Kreuztabelle

In diesem Modus kann das Array nicht mehr als zwei Einträge enthalten. Sie können nur Spalten (1) und Zeilen (2) sortieren. Die Daten, d.h. die Schnittstelle zwischen Spalten und zeilen lassen sich nicht sortieren.

Im folgenden sehen Sie den Code für eine Sortierung nach Zeilen in einem Kreuztabellen-Bericht.

```
ARRAY REAL($aColumns;1)
$aColumns{1}:=2
ARRAY REAL($aOrders;1)
$aOrders{1}:=-1  `Alphabetische Sortierung für Zeilen
QR SET SORTS(qr_area;$aColumns;$aOrders)
```

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.

QR SET TEXT PROPERTY

QR SET TEXT PROPERTY (Bereich ; SpaltenNr ; ZeilenNr ; Eigenschaft ; Wert1...N)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
SpaltenNr	Lange Ganzzahl	➡	Nummer der Spalte
ZeilenNr	Lange Ganzzahl	➡	Nummer der Zeile
Eigenschaft	Lange Ganzzahl	➡	Nummer der Eigenschaft
Wert1...N	Lange Ganzzahl, String	➡	Wert der gewählten Eigenschaft

Beschreibung

Der Befehl **QR SET TEXT PROPERTY** setzt die Textattribute für die Zelle, definiert über *SpaltenNr* und *ZeilenNr*.
Bereich ist die Referenz auf den Bereich des Schnellberichts.
SpaltenNr ist die Spaltennummer der Zelle.
ZeilenNr ist die Referenz auf die Zelle der Zeile. Sie können folgendes übergeben:

- Einen positiven Wert zur Angabe der entsprechenden Zwischensumme (Umbruch).
- Eine Konstante unter dem Thema **QR Zeilen für Eigenschaften**:

Konstante	Typ	Wert	Kommentar
qr detail	Lange Ganzzahl	-2	Detailbereich des Berichts
qr footer	Lange Ganzzahl	-5	Seite Fußteil
qr grand total	Lange Ganzzahl	-3	Bereich Gesamtsumme
qr header	Lange Ganzzahl	-4	Seite Kopfteil
qr title	Lange Ganzzahl	-1	Titel des Berichts

Hinweis: Übergeben Sie als *ZeilenNr* -4 oder -5, müssen Sie auch in *SpaltenNr* eine Spaltennummer übergeben, selbst wenn sie nicht benützt wird.
Hinweis: Im Modus Kreuztabelle ist das Prinzip ähnlich, außer dass die Werte für Zeilen immer positiv sind.
Eigenschaft ist der Wert oder die Konstante des zuzuweisenden Textattributs. Sie können die Konstanten unter dem Thema **QR Texteigenschaften** verwenden:

Konstante	Typ	Wert	Kommentar
_o_qr font	Lange Ganzzahl	1	Ab 4D v14 R3 überholt (qr font name verwenden)
qr alternate background color	Lange Ganzzahl	9	Nummer für wechselnde Hintergrundfarbe
qr background color	Lange Ganzzahl	8	Nummer für Hintergrundfarbe
qr bold	Lange Ganzzahl	3	Stilattribut für Fett (0 oder 1)
qr font name	Lange Ganzzahl	10	Schriftname, wie er z.B. vom Befehl **FONT LIST** zurückgegeben wird
qr font size	Lange Ganzzahl	2	Schriftgröße ausgedrückt in Punkten (9 bis 255)
qr italic	Lange Ganzzahl	4	Stilattribut für Kursiv (0 oder 1)
qr justification	Lange Ganzzahl	7	Attribut für Ausrichtung (0 = Standard, 1 = linksbündig, 2 = zentriert, 3 = rechtsbündig)
qr text color	Lange Ganzzahl	6	Attribut für Farbnummer (Lange Ganzzahl)
qr underline	Lange Ganzzahl	5	Stilattribut für Unterstrichen

Bei einer ungültigen Nummer für *Bereich* wird der Fehler -9850 generiert.
Ist der Parameter *SpaltenNr* inkorrekt, wird der Fehler -9852 generiert.
Ist der Parameter *ZeilenNr* inkorrekt, wird der Fehler -9853 generiert.
Ist der Parameter *Eigenschaft* inkorrekt, wird der Fehler -9854 generiert.

Beispiel

Diese Methode definiert mehrere Attribute für den Titel der ersten Spalte:

```
 //Weist die Schrift Times zu:
QR SET TEXT PROPERTY(qr_area;1;-1;qr font name;"Times")
 //Weist die Schriftgröße 10 Punkt zu:
QR SET TEXT PROPERTY(qr_area;1;-1;qr font size;10)
 //Weist das Attribut fett zu:
QR SET TEXT PROPERTY(qr_area;1;-1;qr bold;1)
 //Weist das Attribut kursiv zu:
QR SET TEXT PROPERTY(qr_area;1;-1;qr italic;1)
```

```
 //Weist das Attribut unterstrichen zu:
QR SET TEXT PROPERTY(qr_area;1;-1;qr underline;1)
 //Weist die Farbe hellgrün zu:
QR SET TEXT PROPERTY(qr_area;1;-1;qr text color;0x0000FF00)
```

QR SET TOTALS DATA

QR SET TOTALS DATA (Bereich ; SpaltenNr ; UmbruchNr ; Operator / Text)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	➡	Referenz auf den Bereich
SpaltenNr	Lange Ganzzahl	➡	Spaltennummer
UmbruchNr	Lange Ganzzahl	➡	Umbruchnummer
Operator / Text	Lange Ganzzahl, String	➡	Operatorwert für Zelle oder Zelleninhalt

Beschreibung

Hinweis: Dieser Befehl kann keine Zwischensumme erstellen.

Modus Liste

Der Befehl **QR SET TOTALS DATA** liefert Einzelheiten zu einem bestimmten Umbruch.

Bereich ist die Referenz auf den Bereich Schnellbericht.

SpaltenNr ist die Nummer der Spalte, deren Daten Sie setzen.

UmbruchNr ist die Nummer des Umbruchs, dessen Daten Sie setzen. Für eine Zwischensumme ist es die Zahl der Sortierung, für die Gesamtsumme die Nummer -3 oder die Konstante qr grand total.

Operator ist die Summe aller in der Zelle vorhandenen Operatoren. Sie können zum Bearbeiten des zurückgegebenen Werts eine Konstante unter dem Thema **QR Operatoren** verwenden:

Konstante	Typ	Wert
qr average	Lange Ganzzahl	2
qr count	Lange Ganzzahl	16
qr max	Lange Ganzzahl	8
qr min	Lange Ganzzahl	4
qr standard deviation	Lange Ganzzahl	32
qr sum	Lange Ganzzahl	1

Bei *Operator* gleich 0 gibt es keinen Operator

Text gibt den in *Zelle* zu setzenden Text zurück.

Hinweis: *Operator* und *Text* schließen sich gegenseitig aus. Sie können entweder einen Operator oder einen Text setzen.

Sie können folgende Werte übergeben:
- # für den Wert, der Umbruch oder Untersumme verwaltet
- ##S wird ersetzt durch die Summe
- ##A wird ersetzt durch den Durchschnitt
- ##C wird ersetzt durch den Zähler
- ##X wird ersetzt durch Max.
- ##N wird ersetzt durch Min.
- ##D wird ersetzt durch die Standardabweichung.
- ##xx, wobei xx eine Spaltennummer ist. Dies wird ersetzt durch den Wert dieser Spalte mit deren Formatierung.

Modus Kreuztabelle

Mit dem Befehl **QR SET TOTALS DATA** setzen Sie Einzelheiten zu einer bestimmten Zelle.

Bereich ist die Referenz auf den Bereich des Schnellberichts.

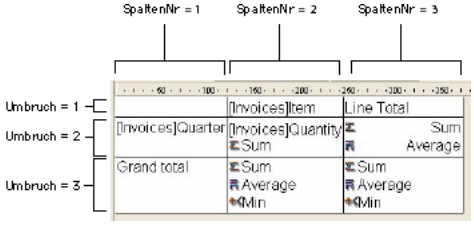
SpaltenNr ist die Spaltennummer der Zelle, deren Daten gesetzt werden.

UmbruchNr ist die Zeilennummer der Zelle, deren Daten gesetzt werden.

Operator ist die Summe aller in der Zelle vorhandenen Operatoren. Sie können zum Bearbeiten des zurückgegebenen Werts eine Konstante unter dem Thema **QR Operatoren** verwenden (siehe oben).

Text ist der in der Zelle zu setzende Text.

Nachfolgendes Schema zeigt, wie die Parameter in einer Kreuztabelle kombiniert werden:



Unterstützte Datentypen

Sie können zwei Arten vergeben:

- Titel
  Ein Titel wird über den Parameter *Text* vergeben. Der Wert ist ein String, der nur für folgende Zellen übergeben werden kann: *SpaltenNr=3 UmbruchNr=1* sowie *SpaltenNr=1* und *UmbruchNr=3*.
- Operator
  Einen Operator oder eine Kombination mehrerer Operatoren können Sie für folgende Zellen übergeben:
  Spalte=2, UmbruchNr=2
  Spalte=3, UmbruchNr=2

Spalte=2, UmbruchNr=3
Beachten Sie, dass die beiden letzten Werte auch die Zelle (Spalte 3; Zeile 3) beeinflussen. Wurde in Zelle (Spalte 2; Zeile 3) eine Berechnung durchgeführt, definiert der Inhalt dieser Zelle immer den Inhalt der Zelle (Spalte 3; Zeile 3).

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie eine ungültige Nummer in *SpaltenNr*, wird der Fehler -9852 generiert.
Übergeben Sie eine ungültige Nummer in *UmbruchNr*, wird der Fehler -9853 generiert.

QR SET TOTALS SPACING

QR SET TOTALS SPACING (Bereich ; ZwischensumNr ; Wert1...N)

Parameter	Typ		Beschreibung
Bereich	Lange Ganzzahl	⇒	Referenz auf den Bereich
ZwischensumNr	Lange Ganzzahl	⇒	Nummer der Spalte
Wert1...N	Lange Ganzzahl	⇒	0=Kein Abstand, 32000=Fügt Seitenumbruch ein, >0=Abstand unter Umbruch hinzugefügt, <0=proportionale Erhöhung

Beschreibung

Der Befehl **QR SET TOTALS SPACING** setzt einen Abstand unter die Zeile mit der Zwischensumme. Der Befehl gilt nur für den Listenmodus.

Bereich ist die Referenz auf den Bereich Schnellbericht.

ZwischensumNr ist die betroffene Umbruchebene oder Zwischensummenebene.

Wert gibt den Wert des Abstands an:

- Ist *Wert* gleich 0, wird kein Abstand hinzugefügt.
- Ist *Wert* gleich 32000, wird ein Seitenumbruch eingefügt.
- Ist *Wert* ein positiver Wert, gibt er den Abstand in Pixel an.
- Ist *Wert* ein negativer Wert, gibt er den Abstand als Prozentsatz der Zeile mit der Zwischensumme an. Zum Beispiel, -100 setzt einen Abstand von 100% unter die Zeile Zwischensumme.

Hinweis: Würde der Abstand unter der Zeile Zwischensumme diese Zeile auf die nächste Seite schieben, wird hier kein Abstand eingefügt.

Übergeben Sie eine ungültige Nummer in *Bereich*, wird der Fehler -9850 generiert.
Übergeben Sie eine ungültige Nummer in *ZwischensumNr*, wird der Fehler -9852 generiert.

SQL

- Einführung in SQL Befehle
- Datenbankmethode 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*

Einführung in SQL Befehle

4D bietet eine integrierte SQL Engine. Das Programm enthält auch einen SQL Server, den andere 4D Anwendungen oder third-party Anwendungen über den 4D ODBC Treiber abfragen können.
Die SQL Dokumentation in 4D besteht aus zwei Bereichen:

- **4D SQL Reference Guide** (**4D - SQL Reference**): Dieses Handbuch beschreibt die verschiedenen Wege, um auf die 4D SQL Engine zuzugreifen, die Konfiguration des SQL Server sowie die Befehle und Schlüsselwörter für SQL Anfragen, z.B. **SELECT** oder **UPDATE**. Hier finden Sie ausführliche Informationen zur Implementierung der SQL Sprache in 4D
- **Kapitel SQL im Handbuch 4D Programmiersprache** (**SQL**): Es beschreibt die verschiedenen 4D high level Befehle zur Verwendung von SQL in 4D:
  - SQL Server steuern: **START SQL SERVER** und **STOP SQL SERVER**
  - Direkt auf die integrierte SQL Engine zugreifen: **SET FIELD VALUE NULL**, **Is field value Null**, **QUERY BY SQL**.
  - Verbindungen an externe oder interne Datenquellen (SQL pass-through) verwalten: **GET DATA SOURCE LIST**, **Get current data source**, **SQL LOGIN**, **SQL LOGOUT**.
  - High-level Befehle, um Daten im Rahmen direkter SQL Verbindungen oder via ODBC zu verwalten: **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**.

Arbeitsweise der high-level SQL Befehle

Die integrierten SQL Befehle beginnen mit der Vorsilbe SQL. Sie führen folgendes aus:

- Sie können diese Befehle mit der 4D internen SQL Engine oder in einer externen Verbindung verwenden, die direkt oder via ODBC geöffnet wird. Über den Befehl **SQL LOGIN** können Sie die Art der zu öffnenden Verbindung angeben.
- Die Reichweite einer Verbindung ist der Prozess. Wollen Sie mehrere gleichzeitige Verbindungen verwalten, müssen Sie mit dem Befehl **SQL LOGIN** einen Prozess starten.
- Mit dem Befehl **ON ERR CALL** können Sie jeden ODBC Fehler abfangen, der während der Ausführung eines high-level SQL Befehls auftritt. Über den Befehl **SQL GET LAST ERROR** erhalten Sie zusätzliche Informationen.

Unterstützung von standard ODBC

Der ODBC Standard (Open DataBase Connectivity) spezifiziert eine Library mit standardisierten Funktionen. Damit kann eine Anwendung wie 4D via SQL Programmiersprache auf jedes ODBC-kompatible Datenmanagement-System, wie Datenbanken, Tabellenkalkulation, andere 4D Programme, etc. zugreifen.
4D ermöglicht auch, Daten in eine ODBC Quelle zu importieren bzw. daraus zu exportieren, entweder über die Befehle **IMPORT ODBC** oder **EXPORT ODBC** oder "manuell" im Designmodus. Weitere Informationen dazu finden Sie im Abschnitt **Import und Export über ODBC Datenquelle** des Handbuchs *4D Designmodus*.

Hinweis: Die high-level SQL Befehle von 4D eignen sich für einfache Vorgehensweisen, über die 4D Anwendungen mit ODBC Datenquellen kommunizieren können. Für eine komplexere Unterstützung von ODBC Standards benötigen Sie **4D ODBC Pro**,das "low level" ODBC Plug-In für 4D.

Entsprechung der Datentypen

Nachfolgende Tabelle zeigt die Entsprechungen, die 4D automatisch zwischen 4D und SQL Datentypen herstellt:

4D Typ	SQL Typ
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

Hinweis: Der SQL Kernel von 4D unterstützt **nicht** Daten vom Typ Objekt (**C_OBJECT**).

Referenz auf 4D Ausdrücke in SQL Anfragen setzen

4D bietet zwei Wege, um 4D Ausdrücke (Variablen, Arrays, Felder, Zeiger, gültige Ausdrücke) in SQL Anfragen einzufügen: Direkte Assoziation und Parameter setzen über den Befehl **SQL SET PARAMETER**.
Bei der direkten Assoziation sind 2 Wege möglich:

- Sie setzen den Namen des 4D Objekts im Anfragetext zwischen spitze Klammern
- Sie setzen einen Doppelpunkt vor die Referenz:

```
SQL EXECUTE("INSERT INTO emp (empnum,ename) VALUES (<<vEmpnum>>,<<vEname>>)")
SQL EXECUTE("SELECT age FROM People WHERE name= :vName")
```

Hinweis: Im kompilierten Modus können Sie keine Referenzen auf lokale Variablen verwenden (mit vorangestellten Symbol $).

In diesen Beispielen ersetzen die aktuellen Werte der 4D Variablen *vEmpnum*, *vEname* und *vName* die Parameter, wenn die Anfrage ausgeführt wird. Das funktioniert auf dieselbe Art mit 4D Feldern und Arrays.

Diese einfache Syntax hat jedoch den Nachteil, dass sie nicht mit den SQL Standards im Einklang ist und keine Ausgabeparameter zulässt. Um dem abzuhelfen, verwenden Sie den Befehl **SQL SET PARAMETER**. Damit integrieren Sie jedes 4D Objekt in eine Anfrage und definieren seine Verwendungsart (Eingabe, Ausgabe oder beides). Auf diese Weise wird eine standardmäßige Syntax erstellt.

Beispiele:

1. Dieses Beispiel führt eine SQL Anfrage aus, die direkt die zugewiesenen 4D Arrays verwendet:

```
ARRAY TEXT(MyTextArray;10)
ARRAY LONGINT(MyLongintArray;10)

For(vCounter;1;Size of array(MyTextArray))
   MyTextArray{vCounter}:="Text"+String(vCounter)
   MyLongintArray{vCounter}:=vCounter
End for
SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (<<MyTextArray>>, <<MyLongintArray>>)"
SQL EXECUTE(SQLStmt)
```

2. Dieses Beispiel führt eine SQL Anfrage aus, die direkt die zugewiesenen 4D Felder verwendet:

```
ALL RECORDS([Table 2])
SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (<<[Table 2]Field1>"+">,<<[Table 2]Field2>>)"
SQL EXECUTE(SQLStmt)
```

3. Dieses Beispiel führt eine SQL Anfrage aus, die direkt Variableninhalt über einen dereferenzierten Zeiger übergibt:

```
C_LONGINT($vLong)
C_POINTER($vPointer)
$vLong:=1
$vPointer:=->$vLong
SQL LOGIN("mysql";"root";"")
SQLStmt:="SELECT Col1 FROM TEST WHERE Col1=:$vPointer";
SQL EXECUTE(SQLStmt)
```

Lokale Variablen im kompilierten Modus verwenden

Im kompilierten Modus können Sie unter bestimmten Bedingungen in SQL Statements lokale Variablenreferenzen (mit vorangestelltem Zeichen $) verwenden:

- Sie können lokale Variablen innerhalb einer Sequenz **Begin SQL** / **End SQL** einsetzen, außer für den Befehl **EXECUTE IMMEDIATE**;
- Sie können lokale Variablen mit dem Befehl **SQL EXECUTE** einsetzen, wenn diese Variablen direkt im Parameter der SQL Anfrage und nicht über Referenzen verwendet werden.
  So funktioniert z.B. folgender Code im kompilierten Modus:

  ```
  SQL EXECUTE("select * from t1 into :$myvar")  // funktioniert im kompilierten Modus
  ```

  während folgender Code einen Fehler im kompilierten Modus generiert:

  ```
  C_TEXT(tRequest)
  tRequest:="select * from t1 into :$myvar"
  SQL EXECUTE(tRequest)  // Fehler im kompilierten Modus
  ```

Werte in 4D wiederfinden

In 4D gibt es zwei Möglichkeiten, Werte in der 4D Programmiersprache wiederzufinden, die sich aus SQL Abfragen ergeben:

- Über den zusätzlichen Parameter im Befehl **SQL EXECUTE** (empfohlene Lösung)
- Über die INTO Klausel in der SQL Anfrage selbst (Alternativ-Lösung für spezifische Fälle)

Ergebnis einer SQL-Anfrage in Listbox anzeigen

Sie können die Ergebnisse einer SQL-Anfrage direkt in ein Array vom Typ Listbox setzen. So können Sie sich rasch die Ergebnisse von SQL Anfragen ansehen. Es sind nur Suchläufe vom Typ **SELECT** möglich (siehe Handbuch *4D SQL Reference*).

Dieser Mechanismus lässt sich nicht mit einer externen SQL Datenbank verwenden.
Dabei gilt folgendes:

- Sie erstellen die Listbox zum Empfangen des Suchergebnisses. Die Datenquelle der Listbox muss vom Typ **Array** sein.
- Führen Sie eine SQL-Suche vom Typ **SELECT** aus und weisen Sie das Ergebnis der mit der Listbox verknüpften Variablen zu. Sie können die 4D Tags **Begin SQL**/**End SQL** verwenden.
- Listbox-Spalten lassen sich vom Benutzer sortieren oder ändern.
- Bei erneuter Ausführung einer Anfrage mit **SELECT** werden die Spalten neu gefüllt. Es ist nicht möglich, dieselbe Listbox mit mehreren **SELECT** Anfragen nacheinander zu füllen.
- Es wird empfohlen, der Listbox die gleiche Anzahl Spalten zu geben wie im Ergebnis der SQL Anfrage sind. Gibt es weniger Listbox-Spalten als für die **SELECT** Anfrage erforderlich, werden automatisch Spalten hinzugefügt. Gibt es mehr Spalten als für die **SELECT** Anfrage erforderlich, werden die nicht benötigten Spalten automatisch ausgeblendet.
  Hinweis: Den automatisch hinzugefügten Spalten werden Variablen vom Typ **Dynamische Variablen** zugewiesen, d.h. sie bestehen solange wie das Formular. Eine dynamische Variable wird auch für jeden Kopfteil erstellt. Beim Aufrufen des 4D Befehls **LISTBOX GET ARRAYS** enthält der Parameter *arrColVars* Zeiger auf die dynamischen Arrays und der Parameter *arrHeaderVars* Zeiger auf die dynamischen Kopfteilvariablen. Ist die hinzugefügte Spalte z.B. die fünfte Spalte, lautet ihr Name *sql_column5* und ihr Kopfteil *sql_header5*.
- Im interpretierten Modus lassen sich vorhandene Arrays, welche die Listbox verwendet, automatisch entsprechend der von der SQL-Anfrage gesendeten Daten erneut tippen.

Beispiel:
Wir wollen nach allen Datenfeldern der Tabelle [PEOPLE] suchen und den Inhalt mit der Variablen *vlistbox* in die Listbox setzen. Für die Objektmethode einer Schaltfläche (als eine Möglichkeit) schreiben Sie:

```
Begin SQL
   SELECT * FROM PEOPLE INTO <<vlistbox>>
End SQL
```

Datenbankmethode On SQL Authentication

Mit der **Datenbankmethode On SQL Authentication** können Sie Anfragen filtern, die an den integrierten SQL Server von 4D gesendet werden. Der Filter kann auf Name und Kennwort sowie optional auf der IP Adresse des Benutzers basieren. Entwickler können ihre eigene Benutzertabelle oder die der 4D Benutzer zum Prüfen der Identifizierer der Verbindung verwenden. Wurde die Anmeldung authentifiziert, muss der Befehl **CHANGE CURRENT USER** aufgerufen werden, um den Zugriff auf Anfragen innerhalb der 4D Datenbank zu steuern.

4D oder 4D Server rufen die **Datenbankmethode On SQL Authentication** - sofern vorhanden - bei jeder externen Anmeldung an den SQL Server automatisch auf. Deshalb wird das interne System zum Verwalten von 4D Benutzern nicht aktiviert. Die Verbindung wird nur angenommen, wenn die Datenbankmethode in $0 **Wahr** zurückgibt und der Befehl **CHANGE CURRENT USER** erfolgreich ausgeführt wurde. Ist eine dieser Bedingungen nicht erfüllt, wird die Anfrage verweigert.

Hinweis: Die Anweisung **SQL LOGIN**(SQL_INTERNAL;$Benutzer;$Kennwort) ruft nicht die **Datenbankmethode On SQL Authentication** auf, da es sich in diesem Fall um eine interne Anmeldung handelt.

Die Datenbankmethode empfängt von 4D bis zu drei Parameter vom Typ ($1, $2, $3) und gibt $0 als Boolean Wert zurück:

Parameter	Typ	Beschreibung
$1	Text	Benutzername
$2	Text	Kennwort
$3	Text	(optional) IP Adresse des Client am Ursprung der Anfrage (*)
$0	Boolean	Wahr = Anfrage angenommen, Falsch = Anfrage verweigert

(*) 4D gibt IPv4 Adressen in einem hybrid IPv6/IPv4 Format mit einem 96-bit Prefix zurück, z.B. ::ffff:192.168.2.34 für die IPv4 Adresse 192.168.2.34. Weitere Informationen dazu finden Sie im Abschnitt **Unterstützung von IPv6**.

Sie müssen diese Parameter folgendermaßen deklarieren:

```
  ` Datenbankmethode On Web Authentication
C_TEXT($1;$2;$3)
C_BOOLEAN($0)
  ` Code für Methode
```

Das Kennwort ($2) wird als Standardtext empfangen.

Die Identifizierer der SQL Verbindung müssen Sie über die **Datenbankmethode On SQL Authentication** prüfen. Sie können z.B. Name und Kennwort über eine eigene Benutzertabelle prüfen. Sind die Identifizierer gültig, übergeben Sie **Wahr** in $0, um die Verbindung zuzulassen.
Andernfalls übergeben Sie **Falsch** in $0; dann wird die Verbindung abgewiesen.

Hinweis: Gibt es keine **Datenbankmethode On SQL Authentication**, wird die Verbindung über das in 4D integrierte System zur Benutzerverwaltung bewertet (wenn es aktiviert ist, d.h. wenn dem Designer ein Kennwort zugewiesen ist). Ist das System nicht aktiviert, werden Benutzer mit Designer-Zugriffsrechten angemeldet, d.h. mit uneingeschränktem Zugriff.

Haben Sie in $0 Wahr übergeben, müssen Sie dann in der **Datenbankmethode On SQL Authentication** den Befehl **CHANGE CURRENT USER** erfolgreich aufrufen, damit die Anfrage angenommen wird und 4D eine SQL Sitzung für den Benutzer öffnet. Über den Befehl **CHANGE CURRENT USER** können Sie ein virtuelles Authentifizierungssystem einrichten mit doppeltem Vorteil: Es ermöglicht einerseits die Steuerung der Verbindungsaktionen und blendet andererseits die Identifizierer der Verbindung aus, so dass sie von außerhalb der 4D SQL Sitzung nicht einsehbar sind.

Das folgende Beispiel der **Datenbankmethode On SQL Authentication** prüft, ob die Anfrage der Verbindung vom internen Netzwerk stammt, bestätigt die Identifizierer und weist dann dem Benutzer "sql_user" die Zugriffsrechte für die SQL Sitzung zu.

```
C_TEXT($1;$2;$3)
C_BOOLEAN($0)
  `$1: Benutzer
  `$2: Kennwort
  `{$3: IP Adresse des Client}
ON ERR CALL("SQL_Fehler")
If(checkInternalIP($3))
  `Die Methode checkInternalIP prüft, ob die IP Adresse intern ist
   If($1="Viktor") & ($2="Hugo")
      CHANGE CURRENT USER("sql_Benutzer";"")
      If(OK=1)
         $0:=True
      Else
         $0:=False
      End if
   Else
      $0:=False
   End if
Else
   $0:=False
End if
```

Begin SQL

Begin SQL
Dieser Befehl benötigt keine Parameter

Beschreibung

Begin SQL ist ein Tag, mit dem Sie im Methodeneditor angeben, dass hier SQL Anweisungen beginnen, die von der aktuellen Datenquelle des Prozesses interpretiert werden müssen. Das kann die in 4D integrierte SQL Engine oder eine über den Befehl **SQL LOGIN** angegebene Quelle sein.

Eine Abfolge von SQL Befehlen muss immer mit **Begin SQL** starten und mit **End SQL** abschließen.

Diese Tags funktionieren folgendermaßen:

- Sie können in derselben Methode mehrere Blöcke mit **Begin SQL/End SQL** setzen. Sie können Methoden einrichten, die nur SQL Code enthalten oder 4D Code und SQL Code in derselben Methode miteinander mischen.
- Sie können mehrere SQL Anweisungen in einer Zeile schreiben oder in mehreren Zeilen, getrennt durch Strichpunkt ";". Beispiel:

```
Begin SQL
   INSERT INTO SALESREPS (NAME, AGE) VALUES (Henry,40);
   INSERT INTO SALESREPS (NAME, AGE) VALUES (Bill,35)
End SQL
```

oder:

```
Begin SQL
   INSERT INTO SALESREPS (NAME, AGE) VALUES (Henry,40);INSERT INTO SALESREPS (NAME, AGE)VALUES (Bill,35)
End SQL
```

Beachten Sie, dass der 4D Debugger (siehe **Fenster Debugger**) den SQL Code Zeile für Zeile analysiert. In bestimmten Fällen ist es vorteilhafter, mit mehreren Zeilen zu arbeiten.

End SQL

End SQL
Dieser Befehl benötigt keine Parameter

Beschreibung

End SQL ist ein Tag, das in einer Methode das Ende von SQL Anweisungen angibt.
SQL Anweisungen müssen innerhalb der Tags **Begin SQL** und **End SQL** stehen. Weitere Informationen dazu finden Sie in der Beschreibung zu **Begin SQL**.

Get current data source

Get current data source -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	String		Name der aktuell verwendeten Datenquelle

Beschreibung

Die Funktion **Get current data source** gibt den Namen der aktuellen Datenquelle der Anwendung zurück. Die aktuelle Datenquelle empfängt die SQL Anfragen, die innerhalb der Tags **Begin SQL/End SQL** ausgeführt werden.
Ist die aktuelle Datenquelle die lokale 4D Datenbank, gibt die Funktion den String ";DB4D_SQL_LOCAL;" zurück, der dem Wert der Konstante SQL_INTERNAL unter dem Thema **SQL** entspricht.

Mit dieser Funktion können Sie die aktuelle Datenquelle prüfen, z.B. vor Ausführen einer SQL Anfrage.

GET DATA SOURCE LIST

GET DATA SOURCE LIST (QuellTyp ; QuellNamenArr ; TreiberArr)

Parameter	Typ		Beschreibung
QuellTyp	Lange Ganzzahl	➡	Quelltyp:Benutzer oder System
QuellNamenArr	Array Text	⬅	Array mit Namen der Datenquellen
TreiberArr	Array Text	⬅	Array der Treiber für Quellen

Beschreibung

Der Befehl **GET DATA SOURCE LIST** gibt in den Arrays *QuellNamenArr* und *TreiberArr* die Namen und Treiber der Datenquellen vom Typ Quelltyp zurück, die im ODBC Manager des Betriebssystems definiert sind.

4D ermöglicht, sich per Programmiersprache direkt an externe ODBC Datenquellen anzumelden und SQL Abfragen innerhalb der Tags **Begin SQL/End SQL** auszuführen.
Das funktioniert folgendermaßen: Über den Befehl **GET DATA SOURCE LIST** erhalten Sie die Liste der auf dem Rechner verfügbaren Datenquellen. Über den Befehl **SQL LOGIN** bestimmen Sie die zu verwendende Datenquelle. Dann führen Sie in der aktuellen Quelle SQL Anfragen mit **Begin SQL/End SQL** aus. Um Anfragen mit der internen Engine erneut auszuführen, übergeben Sie einfach den Befehl **SQL LOGOUT**. Weitere Informationen dazu finden Sie im Handbuch **4D - SQL Reference**.

In *QuellTyp* übergeben Sie den zu suchenden Typ. Sie können dafür eine der Konstanten unter dem Thema **SQL**verwenden:

Konstante	Typ	Wert
System data source	Lange Ganzzahl	2
User data source	Lange Ganzzahl	1

Hinweis: Dieser Befehl berücksichtigt keine Datenquellen vom Typ File Data Source

Der Befehl füllt die Arrays *QuellNamenArr* und *TreiberArr* mit den entsprechenden Werten und passt ihre Größe an.

Hinweis: Wollen Sie sich über ODBC an eine externe 4D Datenquelle anmelden, muss auf Ihrem Rechner der 4D ODBC Treiber installiert sein. Weitere Informationen dazu finden Sie im Handbuch *4D ODBC Driver Installation*.

Beispiel

Beispiel mit einer Datenquelle Benutzer:

```
ARRAY TEXT(arrDSN;0)
ARRAY TEXT(arrDSNDrivers;0)
GET DATA SOURCE LIST(User data source;arrDSN;arrDSNDrivers)
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl hat die Systemvariable OK den Wert 1, sonst den Wert 0 und es wird ein Fehler generiert.

Is field value Null

Is field value Null (Feldname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Zu bewertendes Feld
Funktionsergebnis	Boolean	⮌	Wahr = Feld ist NULL, Falsch = Feld ist nicht NULL

Beschreibung

Die Funktion **Is field value Null** gibt Wahr zurück, wenn das durch den Parameter *Feldname* bestimmte Feld den Wert NULL enthält, sonst Falsch.

Die SQL Engine von 4D arbeitet mit NULL Werten. Weitere Informationen dazu finden Sie im Handbuch **4D - SQL Reference**.

Der von dieser Funktion zurückgegebene Wert ist nur von Bedeutung, wenn die Option **NULL Werten leere Werte zuordnen** in der Felddefinition des Strukturedi tors **nicht** markiert ist. Andernfalls gibt sie immer **Falsch** zurück .

QUERY BY SQL

QUERY BY SQL ({Tabellenname ;} sqlFormel)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, an die eine Datensatzauswahl zurückgegeben wird ohne diesen Parameter Standardtabelle
sqlFormel	String	➡	Gültige SQLSuchformel mit WHERE Klausel der Suche SELECT

Beschreibung

QUERY BY SQL ermöglicht, eine einfache Anfrage mit SELECT auszuführen. Dazu schreiben Sie folgende Anweisung:

```
SELECT *
   FROM table
   WHERE <sqlFormula>
```

Tabellenname ist der im ersten Parameter übergebene Name, *sqlFormel* der Suchstring der WHERE Klausel.
So führt z.B. die folgende Anweisung:

```
QUERY BY SQL([Employees];"name='smith'")
```

dasselbe aus wie die folgende SQL Anfrage:

```
SELECT*FROM Employees WHERE"name='smith'"
```

QUERY BY SQL sucht in der angegebenen Tabelle nach Datensätzen. Er verändert die aktuelle Auswahl der Tabelle für den aktuellen Prozess und macht den ersten Datensatz der neuen Auswahl zum aktuellen Datensatz.

Hinweis: QUERY BY SQL lässt sich nicht im Kontext einer externen SQL Verbindung einsetzen; denn er meldet sich direkt an die integrierte SQL Engine von 4D an.

QUERY BY SQL wendet *sqlFormel* auf jeden Datensatz in der Tabelle oder Auswahl an. *sqlFormel* ist ein Boolean Ausdruck, der den Wert WAHR oder FALSCH hat. Wie Sie sicherlich wissen, kann in SQL-2 Standard eine Suchbedingung das Ergebnis WAHR, FALSCH oder NULL hervorbringen. Die neue aktuelle Auswahl enthält als Suchergebnis nur die Datensätze (Zeilen), für die die Suchbedingung das Ergebnis WAHR hervorbringt.
Der Ausdruck *sqlFormel* kann einfach sein, wie z.B. Vergleich eines Feldes (Spalte) mit einem Wert; oder komplex, z.B. Ausführung einer Berechnung. Er kann, analog zu den Befehlen **QUERY BY FORMULA**, **QUERY BY SQL** eine Information in einer verknüpften Tabelle bewerten (siehe Beispiel 4). *sqlFormel* kann eine gültige SQL Anweisung sein, vorausgesetzt, sie beachtet die für SQL-2 geltenden Regeln sowie die Einschränkungen der derzeit in 4D implementierten SQL Engine. Weitere Informationen dazu finden Sie im Handbuch **4D - SQL Reference**.

Der Parameter *sqlFormel* kann Referenzen auf 4D Ausdrücke verwenden. Die Syntax dafür ist dieselbe wie für ODBC Befehle oder die Funktionen **Begin SQL**/**End SQL**, z.B.:
<<MyVar>> oder *:MyVar*.

Hinweis: Dieser Befehl ist kompatibel mit den Befehlen **SET QUERY LIMIT** und **SET QUERY DESTINATION**.

Achtung: Im kompilierten Modus können Sie keine Referenzen auf lokale Variablen benutzen. Weitere Informationen zur SQL Programmierung in 4D finden Sie im Abschnitt **Einführung in SQL Befehle**.

Über Verknüpfungen

QUERY BY SQL verwendet keine Verknüpfungen zwischen Tabellen, die im 4D Strukturedtor definiert wurden. Dafür müssen Sie eine Join Klausel hinzufügen.
Nehmen wir als Beispiel folgende Struktur mit einer Viele-zu-Eine Verknüpfung von [PEOPLE]City zu [CITIES]Name:

```
[People]
   Name
   City
[Cities]
   Name
   Population
```

Mit **QUERY BY FORMULA** schreiben Sie:

```
QUERY BY FORMULA([People];[Cities]Population>1000)
```

Mit **QUERY BY SQL** müssen Sie mit oder ohne Verknüpfung folgende Anweisung verwenden:

```
QUERY BY SQL([People];"people.city=cities.name AND cities.population>1000")
```

Hinweis: QUERY BY SQL verwaltet Eine-zu-Viele und Viele-zu-Viele Verknüpfungen anders als **QUERY BY FORMULA**.

Beispiel 1

Dieses Beispiel zeigt die Firmen mit Verkäufen über 100. Die SQL Anfrage lautet:

```
SELECT *
   FROM Offices
   WHERE Sales > 100
```

Mit **QUERY BY SQL** lautet sie:

```
C_STRING(30;$queryFormula)
$queryFormula:="Sales > 100"
QUERY BY SQL([Offices];$queryFormula)
```

Beispiel 2

Dieses Beispiel zeigt die Bestellungen im Bereich 3000 bis 4000. Die SQL Anfrage lautet:

```
SELECT *
   FROM Orders
   WHERE Amount BETWEEN 3000 AND 4000
```

Mit **QUERY BY SQL** lautet sie:

```
C_STRING(40;$queryFormula)
$queryFormula:="Amount BETWEEN 3000 AND 4000"
QUERY BY SQL([Orders];$queryFormula)
```

Beispiel 3

Dieses Beispiel zeigt, wie Sie das Suchergebnis über spezifische Suchkriterien erhalten. Die SQL Anfrage lautet:

```
SELECT *
   FROM People
   WHERE City ='Paris'
      ORDER BY Name
```

Mit **QUERY BY SQL** lautet sie:

```
C_STRING(40;$queryFormula)
$queryFormula:="City= 'Paris' ORDER BY Name"
QUERY BY SQL([People];$queryFormula)
```

Beispiel 4

Dieses Beispiel zeigt eine Suche mit verknüpften Tabellen in 4D. In SQL müssen Sie JOIN verwenden, um die 4D Verknüpfung zu simulieren. Wir gehen von zwei Tabellen in 4D aus:

```
[Invoices] mit folgenden Spalten (Feldern):
   ID_Inv: Longint
   Date_Inv: Date
   Amount: Real
[Lines_Invoices] mit folgenden Spalten (Feldern):
   ID_Line: Longint
   ID_Inv: Longint
   Code: Alpha (10)
```

Es gibt eine Viele-zu-Eine Verbindung von [Lines_Invoices]ID_Inv to [Invoices]ID_Inv. Mit **QUERY BY FORMULA** schreiben Sie:

```
QUERY BY FORMULA([Lines_Invoices];([Lines_Invoices]Code="FX-200") & (Month of([Invoices]Date_Inv)=4))
```

Die SQL Anfrage lautet:

```
SELECT ID_Line
   FROM Lines_Invoices, Invoices
   WHERE Lines_Invoices.ID_Inv=Invoices.ID_Inv
      AND Lines_Invoices.Code='FX-200'
      AND MONTH(Invoices.Date_Inv) = 4
```

Mit **QUERY BY SQL** lautet sie:

```
C_STRING(40;$queryFormula)
$queryFormula:="Lines_Invoices.ID_Inv=Invoices.ID_InvAND Lines_Invoices.Code='FX-200' AND MONTH(Invoices.Date_Inv)=4"
QUERY BY SQL([Lines_Invoices];$queryFormula)
```

Systemvariablen und Mengen

Bei korrekt formatierter Suchbedingung gibt die Systemvariable OK den Wert 1 zurück. Sonst wird sie auf 0 (Null) gesetzt; es wird eine leer Auswahl zurückgegeben und ein Fehler generiert, der sich mit einer Fehlerverwaltungsmethode abfangen lässt, die mit dem Befehl **ON ERR CALL** installiert wurde.

SET FIELD VALUE NULL

SET FIELD VALUE NULL (Feldname)

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Feld mit zugewiesenem NULL Wert

Beschreibung

Der Befehl **SET FIELD VALUE NULL** weist dem durch den Parameter *Feldname* bestimmten Feld den Wert NULL zu.
Die SQL Engine von 4D arbeitet mit NULL Werten. Weitere Informationen dazu finden Sie im Handbuch **4D - SQL Reference**.
Hinweise:

- Der NULL Wert für 4D Felder lässt sich auf der Strukturebene auch ausschließen. Weitere Informationen dazu finden Sie im Abschnitt **Eingabe von NULL verweigern** des Handbuchs *4D Designmodus*.
- **SET FIELD VALUE NULL** entfernt den Inhalt von Objektfeldern.

SQL CANCEL LOAD

SQL CANCEL LOAD
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **SQL CANCEL LOAD** beendet die aktuelle Anfrage SELECT und initialisiert die Parameter.
Mit diesem Befehl können Sie mehrere Anfragen SELECT innerhalb derselben Verbindung ausführen (z.B. derselbe Cursor), die über den Befehl **SQL LOGIN** gestartet wurde.

Beispiel

In diesem Beispiel werden zwei Anfragen in derselben Verbindung ausgeführt:

```
C_BLOB(Myblob)
C_TEXT(MyText)
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

 `Cursor erneut setzen
SQL CANCEL LOAD

SQLStmt:="SELECT Name FROM Employee"
SQL EXECUTE(SQLStmt;MyText)
While(Not(SQL End selection))
   SQL LOAD RECORD
End while
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl hat die Systemvariable OK den Wert 1, sonst den Wert 0 (Null).

SQL End selection

SQL End selection -> Funktionsergebnis

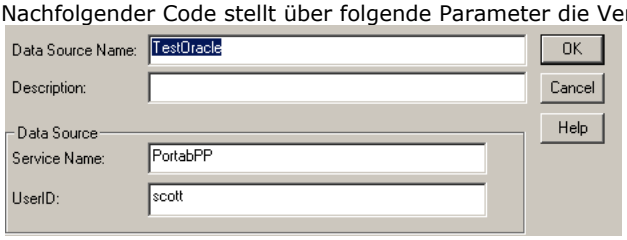
Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean	➲	Grenzen der Ergebnismenge sind erreicht

Beschreibung

Die Funktion **SQL End selection** gibt an, ob die Grenzen der Ergebnismenge erreicht sind.

Beispiel

Nachfolgender Code stellt über folgende Parameter die Verbindung zu einer externen Datenquelle (Oracle) her:




```
C_TEXT(vName)

SQL LOGIN("TestOracle";"schott";"tiger")
If(OK=1)
    SQL EXECUTE("SELECT ename FROM emp";vName)
    While(Not(SQL End selection))
        SQL LOAD RECORD
    End while
    SQL LOGOUT
End if
```

Dieser Code gibt in der 4D Variablen *vName* die Namen *ename* aus der Tabelle mit Namen **Emp** zurück.

SQL EXECUTE

SQL EXECUTE (SQLAnweisung {; GebundObj}{; GebundObj2 ; ... ; GebundObjN})

Parameter	Typ		Beschreibung
SQLAnweisung	Text	➡	Auszuführender SQL Befehl
GebundObj	Variable, Feld	⬅	Empfängt Ergebnis (wenn erforderlich)

Beschreibung

Der Befehl **SQL EXECUTE** führt einen SQL Befehl aus und verbindet das Ergebnis mit 4D Objekten (Arrays, Variablen oder Felder).

Der Befehl wird nur ausgeführt, wenn der aktuelle Prozess eine gültige Verbindung enthält.
Der Parameter *SQLAnweisung* enthält den auszuführenden SQL Befehl, *GebundObj* empfängt das Ergebnis.
Variablen werden in der Reihenfolge der Spalten gebunden, d.h. verbleibende Spalten der SQL Anweisung ohne zugeordnete Variablen bleiben außer Acht.

Warnung: Übergeben Sie 4D Felder im Parameter *GebundObj* und führen den Befehl **SELECT** aus, werden immer die Daten der remote 4D Quelle geändert. Um Daten aus der remote Quelle lokal wiederzufinden, müssen Sie intermediäre lokale Arrays verwenden und den Befehl **INSERT** aufrufen (siehe Beispiel 6).

Werden in *GebundObj* als Parameter 4D Felder übergeben, erstellt der Befehl Datensätze und sichert sie automatisch.
4D Felder müssen aus derselben Tabelle stammen. Sie können nicht im gleichen Aufruf ein Feld aus Tabelle 1, ein weiteres Feld aus Tabelle 2 übergeben. In diesem Fall wird ein Fehler generiert.
Übergeben Sie in *GebundObj* 4D Arrays oder Variablen, sollten Sie diese vor Aufruf des Befehls deklarieren, um den Typ der bearbeiteten Daten zu überprüfen. Arrays werden bei Bedarf automatisch angepasst.

Mit einer 4D Variablen wird zur selben Zeit ein Datensatz geholt. Andere Ergebnisse werden ignoriert.

Weitere Informationen dazu finden Sie im Abschnitt **Einführung in SQL Befehle**.

Beispiel 1

In diesem Beispiel erhalten wir die Spalte *ename* der Tabelle *emp* der Datenquelle. Das Ergebnis wird gespeichert in der 4D Tabelle [employee]Name. 4D Datensätze werden automatisch angelegt:

```
SQLStmt:="SELECT ename FROM emp"
SQL EXECUTE(SQLStmt;[Employee]Name)
SQL LOAD RECORD(SQL all records)
```

Beispiel 2

Um die Erstellung von Datensätzen zu prüfen, können Sie Code in eine Transaktion setzen und nur dann bestätigen, wenn die Operation zu einem zufriedenstellenden Ergebnis führt:

```
SQL LOGIN("mysql";"root";"")
SQLStmt:="SELECT alpha_field FROM app_testTable"
START TRANSACTION
SQL EXECUTE(SQLStmt;[Table 2]Field1)
While(Not(SQL End selection))
   SQL LOAD RECORD
   ... `Setze hier Code zur Bestätigung der Daten
End while
VALIDATE TRANSACTION  `Transaktion bestätigen
```

Beispiel 3

In diesem Beispiel erhalten wir die Spalte *ename* der Tabelle *emp* der Datenquelle. Das Ergebnis wird gespeichert im Array *aName*. Wir holen Datensätze in 10er Schritten.

```
ARRAY STRING(30;aName;20)
SQLStmt:="SELECT ename FROM emp"
SQL EXECUTE(SQLStmt;aName)
While(Not(SQL End selection))
   SQL LOAD RECORD(10)
End while
```

Beispiel 4

In diesem Beispiel erhalten wir die Spalten *ename* und *job* der Tabelle *emp* für eine spezifische ID (WHERE Klausel) der Datenquelle. Das Ergebnis wird gespeichert in den 4D Variablen *vName* und *vJob*. Es wird nur der erste Datensatz erwartet.

```
SQLStmt:="SELECT ename, job FROM emp WHERE id = 3"
SQL EXECUTE(SQLStmt;vName;vJob)
SQL LOAD RECORD
```

Beispiel 5

In diesem Beispiel erhalten wir die Spalte Blob_Field der Tabelle Test in der Datenquelle. Das Ergebnis wird gespeichert in einer BLOB Variable, deren Wert bei jedem Laden eines Datensatzes aktualisiert wird

```
C_BLOB(MyBlob)
SQL LOGIN
SQL EXECUTE("SELECT Champ_Blob FROM Test";MyBlob)
While(Not(SQL End selection))
  `Wir sichten die Ergebnisse
   SQL LOAD RECORD
  `Der Wert von MyBlob wird bei jedem Aufruf aktualisiert.
End while
```

Beispiel 6

Sie wollen Daten, die in einer remote 4D Server Datenbank gespeichert sind, lokal wiederfinden. Dazu müssen Sie dazwischen gesetzte Arrays verwenden:

```
 // An die remote Datenbank anmelden
SQL LOGIN("IP:192.168.18.15:19812";"Benutzer";"Kennwort";*)
If(OK=1)
 // Ab hier werden alle SQL Anfragen auf der remote Datenbank gemacht
   C_TEXT($LastName_value)  // 4D Variable, die in der Suchanweisung verwendet wird
   ARRAY TEXT($a_LastName;0)  // Temporäres Speichern der remote Werte für Nachname
   ARRAY TEXT($a_FirstName;0)  // Temporäres Speichern der remote Werte für Vorname
   C_BOOLEAN($UseSQL)  //Auswahl der Art für lokales Speichern von Daten aus der remote Datenbank
 // (nur Demo)

   $LastName_value:="Smith"  // Initialisierung der 4D Variable

 // Die Variable 4D $LastName_value dem ersten "?" in der SQL Anfrage zuordnen
   SQL SET PARAMETER($LastName_value;SQL param in)

 // Aus der remote Tabelle PERSONS die Werte der Felder LastName und FirstName finden
 // mit "LastName = Smith" und sie in den Arrays $a_LastName und $a_FirstName speichern
   SQL EXECUTE("SELECT LastName, FirstName FROM PERSONS WHERE LastName = ?";$a_LastName;$a_FirstName)
   If(Not(SQL End selection))  // Wird mindestens ein Datensatz gefunden

      SQL LOAD RECORD(SQL all records)  // Lädt alle Datensätze

      $UseSQL:=True  // Wählt die Art, wie die Daten integriert werden (nur Demo)

      If($UseSQL)  // SQL Anfragen verwenden
         SQL LOGOUT  // Von der remote Datenbank abmelden
         SQL LOGIN(SQL_INTERNAL;"user";"password")  // An die lokale Datenbank anmelden
 // Ab hier werden alle SQL Anfragen in der lokalen Datenbank gemacht
 // Die Arrays $a_LastName und $a_FirstName in der lokalen Tabelle PERSONS sichern
         SQL EXECUTE("INSERT INTO PERSONS(LastName, FirstName) VALUES (:$a_LastName, :$a_FirstName);")

      Else  // 4D Befehle verwenden
         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  // Die Verbindung schließen
End if
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl hat die Systemvariable OK den Wert 1, sonst den Wert 0 (Null).

SQL EXECUTE SCRIPT

SQL EXECUTE SCRIPT (SkriptPfad ; FehlerAktion {; attrName ; attrWert} {; attrName2 ; attrWert2 ; ... ; attrNameN ; attrWertN})

Parameter	Typ		Beschreibung
SkriptPfad	Text	➡	Kompletter Pfadname der Datei mit dem auszuführenden SQL Skript
FehlerAktion	Lange Ganzzahl	➡	Auszuführende Aktion bei einem Fehler während der Skript-Ausführung
attrName	Text	➡	Name des zu verwendenden Attributs
attrWert	Text	➡	Wert des Attributs

Beschreibung

Der Befehl **SQL EXECUTE SCRIPT** führt eine Reihe von SQL Anweisungen in der Skript-Datei aus, definiert in *SkriptPfad*. Dieser Befehl lässt sich nur auf einem lokalen Rechner ausführen (4D lokal oder Serverprozedur auf 4D Server). Er arbeitet mit der aktuellen Datenbank (intern oder externe Datenbank).

Hinweis: Dieser Befehl lässt sich nicht in einer externen Verbindung verwenden, die direkt oder über ODBC geöffnet wird.

Im Parameter *SkriptPfad* übergeben Sie den kompletten Pfadnamen der Textdatei mit den SQL Anweisungen, die in *SkriptPfad* ausgeführt werden sollen. Der Pfadname muss in der Syntax des aktuellen Systems ausgedrückt werden. Übergeben Sie einen leeren String ("") in *SkriptPfad*, erscheint ein Standard Öffnen-Dialog für Dokumente, so dass der Benutzer die auszuführende Skript-Datei auswählen kann.

Hinweis: Die Befehle **SQL EXPORT DATABASE** und **SQL EXPORT SELECTION** generieren diese Skript-Datei automatisch.

Mit dem Parameter *FehlerAktion* können Sie die Funktionsweise des Befehls einstellen, wenn während der Skript-Ausführung ein Fehler auftritt. Übergeben Sie eine der nachfolgenden Konstanten unter dem Thema **SQL**:

Konstante	Typ	Wert	Kommentar
SQL On error abort	Lange Ganzzahl	1	Bei einem Fehler stoppt 4D sofort die Ausführung des Skript.
SQL On error confirm	Lange Ganzzahl	2	Bei einem Fehler zeigt 4D ein Dialogfenster, das den Fehler beschreibt und dem Benutzer ermöglicht, die Skript-Ausführung abzubrechen oder weiter auszuführen.
SQL On error continue	Lange Ganzzahl	3	Bei einem Fehler ignoriert 4D diesen und fährt mit der Skript-Ausführung fort.

Die Parameter *attrName* und *attrWert* müssen als Paar übergeben werden. Damit geben Sie spezifische Attribute für die Skript-Ausführung an. In der aktuellen 4D Version kann in *attrName* ein einzelnes Attribut über die nachfolgende Konstante unter dem Thema **SQL** übergeben werden:

Konstante	Typ	Wert	Kommentar
SQL use access rights	Zeichenkette	SQL_Use_Access_Rights	Dient zur Einschränkung der Zugriffsrechte, die während der Ausführung von SQL Befehlen des Skript gelten sollen. Verwenden Sie dieses Attribut, müssen Sie in *attrWert* 0 oder 1 verwenden. • *attrWert* = 1: 4D verwendet die Zugriffsrechte des aktuellen 4D Benutzers. • *attrWert* = 0 (oder Attribut nicht angegeben): 4D schränkt den Zugriff nicht ein, es gelten die Designer-Rechte.

Ist das 4D Logbuch aktiviert (über die Selectoren 28 oder 45 des Befehls **SET DATABASE PARAMETER**), erzeugt jeder ausgeführte SQL Befehl eine Eingabe mit folgender Information:

- Art des SQL Befehls
- Anzahl der vom Befehl betroffenen Datensätze
- Dauer der Befehlsausführung
- Für jeden gefundenden Fehler:
  - den Fehler-Code
  - den Fehlertext, sofern vorhanden

Wird das Skript korrekt ausgeführt, d.h. ohne Fehler, wird die Systemvariable OK auf 1 gesetzt.
Tritt ein Fehler auf, wird die Systemvariable OK entweder auf 0 oder gemäß dem Parameter *FehlerAktion* gesetzt:

- Ist *FehlerAktion* SQL On error abort (Wert 1), *wird OK auf 0 gesetzt*.
- Ist *FehlerAktion* SQL On error confirm (Wert 2), wird OK auf 0 gesetzt, wenn der Benutzer die Operation stoppt; auf 1, wenn er die Operation fortsetzt.
- Ist *FehlerAktion* SQL On error continue (Wert 3), ist die Systemvariable OK immer 1.

Hinweis: Verwenden Sie diesen Befehl für speicherintensive Aktionen wie z.B. umfangreichen Datenimport, können Sie erwägen, den SQL Befehl **ALTER DATABASE** aufzurufen, um die SQL-Optionen temporär zu deaktivieren.

SQL EXPORT DATABASE

SQL EXPORT DATABASE (OrdnerPfad {; AnzDateien {; maxDateigröße {; maxFeldgröße}}})

Parameter	Typ		Beschreibung
OrdnerPfad	Text	➡	Pfadname des Exportordners oder "" um Auswahldialog für Ordner anzuzeigen
AnzDateien	Lange Ganzzahl	➡	Maximale Anzahl der Dateien pro Ordner
maxDateigröße	Lange Ganzzahl	➡	Maximale Größe der Datei Export.sql (in KB)
maxFeldgröße	Lange Ganzzahl	➡	max. Größe, bis zu der Inhalt eines Feldes vom Typ Text, BLOB oder Bild in die Hauptdatei integriert wird (in Bytes)

Beschreibung

Der Befehl **SQL EXPORT DATABASE** exportiert alle Datensätze von allen Tabellen der Datenbank in SQL Format. Diese globale Exportoperation wird in SQL "Dump" genannt.

Hinweis: Dieser Befehl lässt sich nicht mit einer externen Verbindung einsetzen, die direkt oder über ODBC geöffnet wurde.

Der Befehl erstellt für jede Tabelle eine Textdatei mit den SQL Anweisungen, die für den Import in eine andere Datenbank notwendig sind. Diese Datei lässt sich direkt vom Befehl **SQL EXECUTE SCRIPT** verwenden, um Daten in eine andere 4D Datenbank zu importieren.

Die Exportdateien werden in einem Ordner mit Namen "SQL Export" im Zielordner, definiert im Parameter *OrdnerPfad* abgelegt. Gibt es bereits einen Ordner mit Namen "SQL Export" an der angegebenen Stelle, ersetzt der Befehl diesen, ohne eine Meldung anzuzeigen.
Übergeben Sie in diesem Parameter einen leeren String, zeigt 4D ein Standard Dialogfenster, in welchem der Benutzer den Zielordner angeben kann. Das Dialogfenster zeigt standardmäßig den aktuellen Ordner des Benutzers, der die Sitzung geöffnet hat: "My Documents" unter Windows bzw. "Documents" auf Mac OS.

Der Befehl führt für jede exportierte Tabelle folgende Aktionen aus:

- im Zielordner wird ein Unterordner mit dem Tabellennamen erstellt.
- im Unterordner wird eine Textdatei mit Namen "Export.sql" erstellt. Diese Datei wird in UTF8 mit einer BOM (byte-order Marke) codiert. Sie enthält die SQL **INSERT** Aufrufe gemäß den exportierten Daten. Feldwerte werden durch Strichpunkte voneinander getrennt. Gibt es weniger Werte als Felder in der Tabelle vorhanden sind, werden die verbleibenden Felder als NULL gewertet.
- Enthält die Tabelle BLOB, Bild- oder Textfelder (extern gespeicherte Texte, d.h. außerhalb von Datensätzen), erstellt der Befehl standardmäßig neben der Datei "Export.sql" einen zusätzlichen Unterordner mit Namen "BLOBS". Innerhalb dieses Unterordners werden soviel Unterordner mit Namen "BlobsX" wie benötigt, erstellt. Diese Unterordner speichern den Inhalt aller Felder vom Typ BLOB, Bild oder externe Texte als separate Dateien. BLOB Dateien heißen "BlobXXXXX.BLOB" (wobei XXXXX eine von der Anwendung erzeugte einmalige Nummer ist.), Textdateien heißen "TEXTXXXXX.TXT"(wobei XXXXX eine von der Anwendung erzeugte einmalige Nummer ist.), Bilddateien heißen PICTXXXXX.ZZZZ (wobei XXXXX eine von der Anwendung erzeugte einmalige Nummer ist und ZZZZ die Endung). Bilder werden, soweit möglich, im Originalformat mit entsprechender Endung exportiert, z.B. .jpg, .png, etc.. Ist ein Export im native Format nicht möglich, werden die Bilder im internen 4D Format mit der Endung .4PCT exportiert.
  Dieses Standardverhalten lässt sich über den optionalen Parameter *maxFeldgröße* anpassen, der eine max. Größe für die im Feld enthaltenen Daten definiert.
  Hinweis: Dieses Verhalten ist anders, wenn Sie **SQL EXPORT DATABASE** mit 4D im remote Modus ausführen. In diesem Kontext werden die Daten, die extern gespeichert werden sollen automatisch in die Datei "Export.sql"eingebunden.

Übergeben Sie den Parameter *AnzDateien*, erstellt der Befehl soviele Unterordner "BlobsX" wie erforderlich, damit jeder nicht mehr als die in *AnzDateien* angebenen BLOB oder Bilddateien enthält. Wird dieser Parameter weggelassen, begrenzt der Befehl die Anzahl der Dateien standardmäßig auf 200. Übergeben Sie 0, enthält jeder Unterordner mindestens eine Datei.
Mit dem Parameter *maxDateigröße* definieren Sie die maximale Größe jeder auf der Festplatte angelegten Datei "Export.sql" (Größe in KB). Ist die in *maxDateigröße* definierte Größe erreicht, stoppt 4D das Schreiben von Datensätzen, schließt die Exportdatei und erstellt neben der ersten Datei eine weitere mit Namen "ExportX.sql" (wobei X die Sequenznummer ist). Beachten Sie, dass dies eine theoretische Begrenzung ist: Die aktuelle Größe der Dateien "ExportX.sql" ist größer als der im Parameter *maxDateigröße* gesetzte Wert, da die Datei erst geschlossen wird, wenn der beim Erreichen des Limits gerade exportierte Datensatz komplett geschrieben ist (der Inhalt der Datensätze wird nicht geteilt). Als Mindestwert ist 100, als Maximumwert (Standardwert) 100.000 (100 MB) zugelassen.

Der optionale Parameter *maxFeldgröße* setzt eine maximale Größe fest, bis zu der der Inhalt eines externen Feldes vom Typ BLOB, Bild oder Text in die Hauptdatei "Export.sql" eingebunden und nicht als separate Datei gesichert wird. Dieser Parameter dient zur Optimierung von Exportoperationen, denn er verringert die Anzahl an Unterordnern oder Dateien, die auf der Festplatte erstellt werden.
Der Wert dieses Parameters wird in Bytes ausgedrückt. Übergeben Sie z.B. 1000, werden alle externen Felder vom Typ BLOB, Bild oder Text mit einer Größe unter oder gleich 1000 Bytes in die Hauptdatei des Exports eingebunden.
Beachten Sie, dass binäre Daten (Felder vom Typ BLOB und Bild), die in die Exportdatei integriert werden, im hexadezimalen Format gespeichert werden, in Form von X'0f20' (standardmäßige hexadezimale Notation für SQL, siehe **literal**). Dieses Format wird automatisch von der 4D SQL Engine unterstützt.
Standardmäßig, d.h. ohne den Parameter *maxFeldgröße* werden externe Felder vom Typ BLOB, Bild und Text immer als externe Dateien exportiert, unabhängig von Ihrer Größe.

Die Exportdatei kann weniger Werte enthalten als Felder in der Tabelle vorhanden sind. In diesem Fall gelten die leeren Felder als NULL. Sie können in einem Feld auch den Wert NULL übergeben.

Wurde der Export korrekt ausgeführt, wird die Variable OK auf 1 gesetzt. Andernfalls hat sie den Wert 0.

Hinweis: Dieser Befehl unterstützt keine Felder vom Typ Objekt.

SQL EXPORT SELECTION

SQL EXPORT SELECTION (Tabellenname ; OrdnerPfad {; AnzDateien {; maxDateigröße {; maxFeldgröße}}})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, aus der die Auswahl exportiert wird oder Standardtabelle ohne Angabe
OrdnerPfad	Text	➡	Pfadname des Exportordners oder "", um Auswahldialog für Ordner anzuzeigen
AnzDateien	Lange Ganzzahl	➡	Maximale Anzahl der Dateien pro Ordner
maxDateigröße	Lange Ganzzahl	➡	Maximale Größe der Datei Export.sql (in KB)
maxFeldgröße	Lange Ganzzahl	➡	Maximale Größe (in Bytes), bis zu der der Inhalt eines Feldes vom Typ Text, BLOB oder Bild in die Hauptdatei eingebettet wird.

Beschreibung

Der Befehl **SQL EXPORT SELECTION** exportiert die Datensätze der aktuellen Auswahl der 4D Tabelle, definiert in *Tabellenname*, in SQL Format .

Dieser Befehl ist fast identisch mit dem Befehl **SQL EXPORT DATABASE**. Die generierte Datei lässt sich direkt vom Befehl **SQL EXECUTE SCRIPT** verwenden, um Daten in eine andere 4D Datenbank zu importieren. Der Unterschied ist, dass **SQL EXPORT SELECTION** nur die aktuelle Auswahl von *Tabellenname* exportiert, während **SQL EXPORT DATABASE** die gesamte Datenbank exportiert. Außerdem funktioniert der Befehl im Gegensatz zu **SQL EXPORT DATABASE** nicht mit externen SQL Datenbanken. Er lässt sich nur mit der Hauptdatenbank verwenden.

Die Funktionsweise und Parameter des Befehls werden ausführlich im Befehl **SQL EXPORT DATABASE** beschrieben.

Ist die aktuelle Auswahl leer, führt der Befehl nichts aus. Beachten Sie, dass in diesem Fall der Zielordner nicht geleert wird.

Wird der Export korrekt ausgeführt, wird die Variable OK auf 1 gesetzt. Andernfalls wird sie auf 0 gesetzt.

Hinweis: Dieser Befehl unterstützt keine Felder vom Typ Objekt.

SQL GET LAST ERROR

SQL GET LAST ERROR (FehlerCode ; FehlerText ; FehlerODBC ; FehlerSQLServer)

Parameter	Typ		Beschreibung
FehlerCode	Lange Ganzzahl	⇐	Fehlercode
FehlerText	Text	⇐	Fehlertext
FehlerODBC	Text	⇐	ODBC Fehlercode
FehlerSQLServer	Lange Ganzzahl	⇐	SQL Server native Fehlercode

Beschreibung

Der Befehl **SQL GET LAST ERROR** gibt Information zum Fehler zurück, der während der Ausführung eines ODBC Befehls auftritt. Dieser Fehler kann vom 4D Programm, vom Netzwerk, von einer ODBC Quelle, o.ä. stammen.

Dieser Befehl muss in der Regel im Rahmen einer Fehlerverwaltungsmethode aufgerufen werden, die über den Befehl **ON ERR CALL** aufgerufen wird.

- *FehlerCode* gibt den Fehlercode zurück.
- *FehlerText* gibt den Fehlertext zurück.

Die beiden letzten Parameter werden nur gefüllt, wenn der Fehler von der ODBC Quelle stammt; sonst werden sie leer zurückgegeben.

- *FehlerODBC* gibt den ODBC Fehlercode zurück (SQL Status).
- *FehlerSQLServer* gibt den native Fehlercode des SQL Server zurück.

SQL GET OPTION

SQL GET OPTION (Option ; Wert)

Parameter	Typ		Beschreibung
Option	Lange Ganzzahl	⇒	Nummer der Option
Wert	Lange Ganzzahl, Text	⇐	Wert der Option

Beschreibung

Der Befehl **SQL GET OPTION** gibt den aktuellen *Wert* für *Option* zurück.
Weitere Informationen zu den verschiedenen Optionen und den zugeordneteten Werten finden Sie unter dem Befehl **SQL SET OPTION**.

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl hat die Systemvariable OK den Wert 1, sonst den Wert 0 (Null).

SQL LOAD RECORD

SQL LOAD RECORD {(AnzDatensätze)}

Parameter	Typ		Beschreibung
AnzDatensätze	Lange Ganzzahl	➡	Anzahl der zu ladenden Datensätze

Beschreibung

Der Befehl **SQL LOAD RECORD** findet einen/mehrere Datensätze in 4D, die aus einer Datenquelle stammen, die in der aktuellen Verbindung geöffnet wurde.

Mit dem optionalen Parameter *AnzDatensätze* setzen Sie die Anzahl der zu findenden Datensätze:

- Geben Sie diesen Parameter nicht an, findet der Befehl den aktuellen Datensatz aus der Datenquelle. Das entspricht dem Finden von Daten in einer Schleife, wo jeweils ein Datensatz empfangen wird.
- Übergeben Sie in *AnzDatensätze* eine Zahl, findet der Befehl die Datensätze *AnzDatensätze*.
- Übergeben Sie die Konstante SQL All Records (Wert -1), findet der Befehl alle Datensätze in der Tabelle.

Hinweis: Die beiden letzten Parameter sind nur von Bedeutung, wenn die gefundenen Daten mit Arrays oder 4D Feldern verbunden sind.

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl hat die Systemvariable OK den Wert 1, sonst den Wert 0 (Null).

SQL LOGIN

SQL LOGIN {(Datenquelle ; Benutzername ; Kennwort ; *)}

Parameter	Typ		Beschreibung
Datenquelle	String	➡	Name der externen Datenbank oder IP Adresse der externen Datenbank oder Name der eingegebenen Datenquelle im ODBC Manager oder "", um Auswahldialog anzuzeigen
Benutzername	String	➡	Name des in der Datenquelle registrierten Benutzers
Kennwort	String	➡	Kennwort des in der Datenquelle registrierten Benutzers
*	Operator	➡	Anwenden auf Begin SQL/End SQL Ohne *: Nein (lokale Datenbank); Mit *: Ja

Beschreibung

Der Befehl **SQL LOGIN** öffnet eine Verbindung mit der SQL Datenquelle, angegeben im Parameter *Datenquelle*. Er bezeichnet das Ziel der SQL Anfragen, die nachfolgend im aktuellen Prozess ausgeführt werden:

- über den Befehl **SQL EXECUTE**
- über den Code innerhalb der Tags *Begin SQL / End SQL* (wenn der Parameter * übergeben ist)

Die SQL Datenquelle kann folgendes sein:

- Eine externe 4D Server Datenbank, auf die Sie direkt zugreifen können
- Eine externe ODBC Quelle
- Die lokale 4D Datenbank (interne Datenbank)

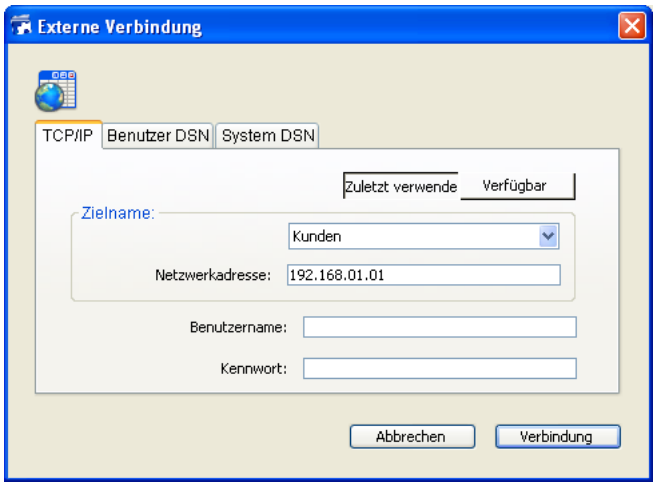
In *Datenquelle* können Sie einen der folgenden Werte übergeben: Eine IP Adresse, einen Namen zum Veröffentlichen der 4D Datenbank, einen Namen der ODBC Datenquelle, einen leeren String oder eine Konstante SQL_INTERNAL.

- **IP Adresse**
  Syntax: **IP:<IPAddress>{:<TCPPort>}**
  In diesem Fall öffnet der Befehl eine direkte Verbindung mit der auf dem Rechner ausgeführten 4D Server Datenbank mit der angegebenen IP Adresse. Auf dem Zielrechner muss der SQL Server gestartet sein. Übergeben Sie eine TCP Port Nummer, muss sie in der Zieldatenbank als Publikationsport für den SQL Server angegeben sein. Übergeben Sie keine TCP Portnummer, wird der standardmäßige Port verwendet (19812). In den Datenbank-Eigenschaften auf der Seite **SQL** können Sie die TCP Port Nummer des SQL Server verändern. Siehe hierzu die Beispiele 4 und 5.
  Haben Sie TLS für den "Ziel" SQL Server aktiviert (Option in den Datenbank-Eigenschaften), müssen Sie an die IP Adresse das Schlüsselwort ".ssl" und die TCP Portnummer (in diesem Fall zwingend) anhängen, damit der Server die Anfrage korrekt bearbeiten kann (siehe Beispiel 6)

- **Anzeigename der 4D Datenbank**
  Syntax: **4D:<Publication_Name>**
  In diesem Fall öffnet der Befehl eine direkte Verbindung mit der 4D Server Datenbank, deren Netzwerk-Anzeigename dem angegebenen Namen entspricht. Der Netzwerk-Anzeigename wird in den Datenbank-Eigenschaften auf der Seite **Client-Server** gesetzt. Siehe Beispiel 4.
  Hinweis: Die TCP Port Nummer des Zielserver 4D SQL (der die 4D Datenbank veröffentlicht) und die TCP Port Nummer des SQL Server der 4D Anwendung, welche die Verbindung öffnet, müssen identisch sein.

- **Gültiger Name der ODBC Datenquelle**
  Syntax: ODBC:<My_DSN> oder <My_DSN>
  In diesem Fall enthält der Parameter *Datenquelle* den Namen der Datenquelle, so wie er im ODBC Treiber Manager festgelegt wurde.
  Hinweise:
  - Um die Kompatibilität mit früheren 4D Versionen zu bewahren, können Sie die Vorsilbe "ODBC:" weglassen, zur besseren Lesbarkeit des Code empfehlen wir jedoch, diese Vorsilbe zu verwenden.
    Siehe Beispiel 2.
  - Unter Windows wird beim Namen der Datenquelle zwischen Klein- und Großschreibung unterschieden. Wurde die Datenquelle z.B. "4D_v16" genannt, funktioniert der Name "4D_V16" nicht.
  - Die Vorsilbe "ODBC:" muss unter Windows und auf Mac in Großbuchstaben eingegeben werden. Übergeben Sie "odbc:", schlägt die Verbindung fehl.

- **Leerer String**
  Syntax: ""
  In diesem Fall zeigt der Befehl den Verbindungsdialog an, so dass Sie die Datenquelle für die Verbindung manuell eingeben können:



Dieses Dialogfenster besteht aus mehreren Seiten. Die Seite **TCP/IP** enthält folgende Elemente:

- **Zielname:** Dieses DropDown-Menü enthält zwei Listen:
  - Die Liste der Datenbanken, die zuletzt in direkter Verbindung geöffnet wurden. Die Operation zum Aktualisieren dieser Liste ist dieselbe wie für die 4D Applikation. Lediglich der Ordner mit den .4DLink Dateien heißt anders. Er hat den Namen "Favorites SQL vXX" anstatt "Favorites vXX".
  - Die Liste der 4D Server Anwendungen mit gestartetem SQL Server, und dessen TCP Port für SQL Verbindungen derselbe wie der der Quellanwendung ist. Diese Liste wird bei jedem neuen Aufruf des Befehls SQL LOGIN ohne den Parameter *Datenquelle* dynamisch aktualisiert. Das Zeichen "^" vor dem Datenbanknamen gibt an, dass die Verbindung im gesicherten Modus via SSL ausgeführt wird.
- **Netzwerk Adresse:** Dieser Bereich zeigt die Adresse und evtl. den TCP Port der Datenbank, die im DropDown-Menü Zielname ausgewählt wurde. Sie können auch eine IP-Adresse eingeben und dann auf die Schaltfläche zur Verbindung klicken, um sich an die entsprechende 4D Server Datenbank anzumelden. Sie können den TCP Port auch mit Doppelpunkt (:) an die IP-Adresse anhängen, z.B.: 192.168.93.105:19855.
- **Benutzername und Kennwort:** In diese Bereiche können Sie die Kennung der Verbindung eingeben.
- Die Seiten **Benutzer DSN** und **System DSN** zeigen jeweils die Liste der Benutzer und die ODBC Datenquellen des Systems, die im ODBC Treiber des Rechners eingetragen sind. Auf diesen Seiten können Sie eine Datenquelle auswählen und die Kennung eingeben, um eine Verbindung mit einer externen ODBC Datenquelle zu öffnen.

Wird die Verbindung hergestellt, wird die Systemvariable OK auf 1 gesetzt. Andernfalls wird sie auf 0 gesetzt und ein Fehler erzeugt. Diesen Fehler können Sie über eine Fehlerverwaltungsmethode abfangen, die mit dem Befehl **ON ERR CALL** installiert wird.

Konstante SQL_INTERNAL
Syntax: **SQL_INTERNAL**
In diesem Fall leitet der Befehl nachfolgende SQL Anfragen an die interne 4D Datenbank.

Warnung: Die Vorsilben, die im Parameter *Datenquelle* verwendet werden, müssen großgeschrieben werden (IP, ODBC, 4D).

Benutzername enthält den Namen des Benutzers, der berechtigt ist, sich an die die externe Datenquelle anzumelden. Der Benutzername für Oracle kann z.B. “Schott” lauten.

Kennwort enthält das Kennwort des Benutzers, der berechtigt ist, sich an die die externe Datenquelle anzumelden. Das Kennwort für Oracle kann z.B.“Tiger” sein.

Hinweis: Übergeben Sie bei direkten Verbindungen in den Parametern *Benutzername* und *Kennwort* leere Strings, wird die Verbindung nur angenommen, wenn in der Zieldatenbank keine 4D Kennwörter aktiviert sind. Andernfalls wird die Verbindung zurückgewiesen.

Mit dem optionalen Parameter * können Sie das Ziel des SQL Code ändern, der innerhalb der Tags *Begin SQL/End SQL* ausgeführt wird. Übergeben Sie diesen Parameter NICHT, wird der Code innerhalb der Tags *Begin SQL/End SQL* weiterhin an die interne SQL Engine von 4D gesendet, ohne die im Befehl SQL LOGIN angegebene Konfiguration zu berücksichtigen. Übergeben Sie diesen Parameter, wird der innerhalb der Tags *Begin SQL/End SQL* ausgeführte Code an die Quelle gesendet, die im Parameter *Datenquelle* definiert ist.

Um die aktuelle Verbindung zu schließen und den Speicher freizumachen, führen Sie den Befehl **SQL LOGOUT** aus. Dann werden alle SQL Anfragen an die interne 4D SQL Datenbank gesendet.
Rufen Sie SQL LOGIN erneut auf und wurde die aktuelle Verbindung nicht explizit geschlossen, wird sie automatisch geschlossen.

Hinweis: Wenn ein externer Verbindungsversuch via SQL LOGIN fehlschlägt, wird die interne 4D Datenbank automatisch die aktuelle Datenquelle.

Diese Parameter sind optional; sind sie nicht angegeben, zeigt der Befehl das Dialogfenster für die ODBC Anmeldung an, in dem Sie die externe Datenquelle auswählen können.

Die Reichweite dieses Befehls ist der Prozess; wollen Sie zwei unterschiedliche Verbindungen ausführen, müssen Sie zwei Prozesse erstellen und jede Verbindung in einem eigenen Prozess ausführen.

Beispiel 1

Diese Anweisung bringt das Verwaltungsfenster des ODBC Treibers nach vorne:

```
SQL LOGIN
```

Beispiel 2

Eine Verbindung über das ODBC Protokoll mit der externen Datenquelle "MyOracle" öffnen. Über den Befehl **SQL EXECUTE** ausgeführte SQL Anfragen sowie Anfragen innerhalb der Tags *Begin SQL/End SQL* werden zu dieser Verbindung weitergeleitet:

```
SQL LOGIN("ODBC:MyOracle";"Schott";"tiger";*)
```

Beispiel 3

Diese Anweisung initialisiert eine Verbindung mit der 4D internen SQL Engine:

```
SQL LOGIN(SQL_INTERNAL;$user;$password)
```

Beispiel 4

Eine direkte Verbindung öffnen mit der 4D Server Anwendung, ausgeführt auf dem Rechner mit der IP Adresse 192.168.45.34, die auf dem standardmäßigen TCP Port antwortet. Über den Befehl **SQL EXECUTE** ausgeführte SQL Anfragen werden zu dieser Verbindung weitergeleitet, Anfragen innerhalb der Tags *Begin SQL/End SQL* werden nicht weitergeleitet.

```
SQL LOGIN("IP:192.168.45.34";"John";"azerty")
```

Beispiel 5

Eine direkte Verbindung öffnen mit der 4D Server Anwendung, ausgeführt auf dem Rechner mit der IP Adresse 192.168.45.34, die auf dem TCP Port 20150 antwortet. Über den Befehl **SQL EXECUTE** ausgeführte SQL Anfragen werden zu dieser Verbindung weitergeleitet, Anfragen innerhalb der Tags *Begin SQL/End SQL* werden an diese Verbindung weitergeleitet.

```
SQL LOGIN("IP:192.168.45.34:20150";"John";"azerty";*)
```

Beispiel 6

Auf dem Rechner mit der IP Adresse 192.168.45.34 mit der 4D Server Anwendung eine direkte Verbindung in TLS öffnen und auf dem Standard TCP Port antworten. Dazu müssen Sie TLS für den SQL Server auf der 4D Server Application aktiviert haben:

```
SQL LOGIN("IP:192.168.45.34:19812:ssl";"Admin";"sd156")  // Beachte ":ssl" am Ende von IP Adresse und TCP Port
```

Beispiel 7

Eine direkte Verbindung mit der 4D Server Application öffnen, die auf dem Rechner mit der IPv6 Adresse 2a01:e35:2e41:c960:dc39:3eb0:f29b:3747 ausgeführt wird und auf den TCP Port 20150 antwortet. Über den Befehl **SQL EXECUTE** ausgeführte SQL Anfragen werden zu dieser Verbindung weitergeleitet, Anfragen innerhalb der Tags **Begin SQL**/**End SQL** werden nicht weitergeleitet.

```
SQL LOGIN("IP:[2a01:e35:2e41:c960:dc39:3eb0:f29b:3747]:20150";"John";"qwerty")
```

Beispiel 8

Eine direkte Verbindung öffnen mit der 4D Server Application, die auf dem lokalen Netzwerk eine Datenbank mit Namen "Accounts_DB" veröffentlicht. Der für die SQL Server beider Datenbanken verwendete TCP Port (gesetzt in den Datenbank-Eigenschaften auf der Seite SQL) muss gleich sein (standardmäßig 19812). Über den Befehl **SQL EXECUTE** ausgeführte SQL Anfragen werden zu dieser Verbindung weitergeleitet, Anfragen innerhalb der Tags *Begin SQL/End SQL* werden nicht weitergeleitet.

```
SQL LOGIN("4D:Accounts_DB";"John";"azerty")
```

Beispiel 9

Dieses Beispiel zeigt die Möglichkeiten für Verbindungen über den Befehl **SQL LOGIN**:

```
ARRAY TEXT(aNames;0)
ARRAY LONGINT(aAges;0)
SQL LOGIN("ODBC:MyORACLE";"Mark";"azerty")
If(OK=1)
  `Die folgende Anfrage wird an die externe ORACLE Datenbank weitergeleitet
   SQL EXECUTE("SELECT Name, Age FROM PERSONS";aNames;aAges)
  `Die folgende Anfrage wird an die lokale 4D Datenbank weitergeleitet
   Begin SQL
```

```
        SELECT Name, Age
        FROM PERSONS
        INTO :aNames, :aAges;
    End SQL
  `Der folgende Befehl SQL LOGIN schließt die aktuelle Verbindung mit der externen ORACLE Datenbank und öffnet eine neue Verbindung mit einer externen MySQL Datenbank
    SQL LOGIN("ODBC:MySQL";"Jean";"qwerty";*)
    If(OK=1)
  `Die folgende Anfrage wird an die externe MySQL Datenbank weitergeleitet
        SQL EXECUTE("SELECT Name, Age FROM PERSONS";aNames;aAges)
  `Die folgende Anfrage wird auch an die externe MySQL Datenbank weitergeleitet
        Begin SQL
            SELECT Name, Age
            FROM PERSONS
            INTO :aNames, :aAges;
        End SQL
        SQL LOGOUT
  `Die folgende Anfrage wird an die lokale 4D Datenbank gesendet
        Begin SQL
            SELECT Name, Age
            FROM PERSONS
            INTO :aNames, :aAges;
        End SQL
    End if
End if
```

Systemvariablen und Mengen

Bei erfolgreicher Verbindung wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

SQL LOGOUT

SQL LOGOUT
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **SQL LOGOUT** schließt die Verbindung zu einer ODBC Quelle, die im aktuellen Prozess (falls vorhanden) offen ist. Ist keine ODBC Verbindung offen, führt der Befehl nichts aus.

Systemvariablen und Mengen

Bei korrekt ausgeführter Abmeldung hat die Systemvariable OK den Wert 1, sonst den Wert 0 (Null). Sie können diesen Fehler über eine Fehlerverwaltungsmethode abfangen, die über den Befehl **ON ERR CALL** installiert wird.

SQL SET OPTION

SQL SET OPTION (Option ; Wert)

Parameter	Typ		Beschreibung
Option	Lange Ganzzahl	➡	Nummer der zu setzenden Option
Wert	Lange Ganzzahl, String	➡	Neuer Wert von Option

Beschreibung

Der Befehl **SQL SET OPTION** ändert den in *Option* übergebenen *Wert*.
Für *Option* sind folgende vordefinierten Konstanten unter dem Thema **SQL** möglich:

Konstante	Typ	Wert	Kommentar
SQL asynchronous	Lange Ganzzahl	1	0 = Synchrone Verbindung (Standardwert), 1 (oder anderer Wert als 0) = Asynchrone Verbindung
SQL charset	Lange Ganzzahl	100	Textcodierung für Anfragen, die an externe Quellen gesendet werden (via SQL pass-through). Die Änderung wird für den aktuellen Prozess und die aktuelle Verbindung ausgeführt. Mögliche Werte: MIBEnum Identifier (siehe Hinweis 2) oder Wert -2 (siehe Hinweis 3) Standardwert: 106 (UTF-8)
SQL connection timeout	Lange Ganzzahl	5	Maximale Wartezeit für Antwort beim Ausführen des Befehls **SQL LOGIN**. Dieser Wert wird nur berücksichtigt, wenn er vor dem Öffnen der Verbindung gesetzt wurde. Mögliche Werte: Zeit in Sekunden Standardwert: Kein Timeout
SQL max data length	Lange Ganzzahl	3	Maximale Länge der zurückgegebenen Daten
SQL max rows	Lange Ganzzahl	2	Maximale Anzahl Zeilen in Ergebnisgruppe (verwendet für Vorschau)
SQL query timeout	Lange Ganzzahl	4	Maximale Wartezeit für Antwort beim Ausführen des Befehls **SQL EXECUTE**. Werte: Zeit in Sekunden Standardwert: kein Timeout

Hinweise:

1. Arbeiten Sie mit der 4D internen SQL Engine, entfällt die Konstante SQL Asynchronous, da diese Verbindung immer synchron ist.
2. **MIBEnum** Nummern finden Sie unter *http://www.iana.org/assignments/character-sets*.
3. Übergeben Sie -2 als *Wert* für SQL Charset, wird die vom 4D SQL Server verwendete Codierung automatisch an die laufende Plattform angepasst (keine UTF Codierung):
   - Unter Windows wird ISO8859-1 verwendet
   - Auf Mac OS wird MAC-ROMAN verwendet

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl hat die Systemvariable OK den Wert 1, sonst den Wert 0 (Null).

SQL SET PARAMETER

SQL SET PARAMETER (Objekt ; ParamTyp)

Parameter	Typ		Beschreibung
Objekt	4D Objekt	➡	4D Objekt (Variable, Array oder Feld)
ParamTyp	Lange Ganzzahl	➡	Parametertyp

Beschreibung

Der Befehl **SQL SET PARAMETER** ermöglicht, in SQL Anfragen eine 4D Variable, ein Array oder Wert eines Feldes zu verwenden.

Hinweis: Sie können den Namen des zu verwendenden 4D Objekts (Variable, Array oder Feld) im Text der Anfrage auch direkt zwischen << und >> einfügen (siehe Beispiel 1). Weitere Informationen dazu finden Sie im Abschnitt **Einführung in SQL Befehle**.

- Im Parameter *Objekt* übergeben Sie das gewünschte 4D Objekt (Variable, Array oder Feld) für die Anfrage.
- Im Parameter *ParamTyp* übergeben Sie den Parameter vom Typ SQL. Sie können einen Wert übergeben oder eine der folgenden Konstanten unter dem Thema **SQL** verwenden:

Konstante	Typ	Wert	Kommentar
SQL param in	Lange Ganzzahl	1	
SQL param in out	Lange Ganzzahl	2	Ist nur im Rahmen einer SQL Serverprozedur verwendbar (in-out Parameter definiert in Serverprozedur)
SQL param out	Lange Ganzzahl	4	Ist nur im Rahmen einer SQL Serverprozedur verwendbar (in-out Parameter definiert in Serverprozedur)

Der Wert des 4D Objekts ersetzt das Fragezeichen (?) in der SQL Anfrage (Standardsyntax).
Enthält die Anfrage mehr als ein Fragezeichen, muss der Befehl **SQL SET PARAMETER** mehrmals aufgerufen werden. Die Werte der 4D Objekte werden in der Anfrage sequentiell zugewiesen, und zwar in Übereinstimmung mit der Ausführungsreihenfolge der Befehle.

Warnung: Dieser Befehl dient zum Verwalten von Parametern, die in den SQL Anfragen übergeben werden. Sie können den Typ SQL param out nicht verwenden, um ein 4D Objekt dem *Ergebnis* einer SQL Anfrage zuzuordnen. SQL Anfrage-Ergebnisse werden z.B. über den Parameter *boundObj* des Befehls **SQL EXECUTE** gefunden (siehe unter **Einführung in SQL Befehle**). **SQL SET PARAMETER** dient hauptsächlich zum Setzen von Parametern für die Anfrage (SQL param in); Die Typen SQL param out und SQL param in out sind für die Verwendung im Rahmen von SQL Serverprozeduren reserviert, die Parameter zurückgeben können.

Beispiel 1

Dieses Beispiel führt eine SQL Anfrage aus, die direkt die zugewiesenen 4D Variablen aufruft:

```
C_TEXT(MyText)
C_LONGINT(MyLongint)

SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (<<MyText>>, <<MyLongint>>)"
For(vCounter;1;10)
   MyText:="Text"+String(vCounter)
   MyLongint:=vCounter
   SQL EXECUTE(SQLStmt)
   SQL CANCEL LOAD
End for
SQL LOGOUT
```

Beispiel 2

Dasselbe Beispiel wie oben, jedoch mit dem Befehl **SQL SET PARAMETER**:

```
C_TEXT(MyText)
C_LONGINT(MyLongint)

SQL LOGIN("mysql";"root";"")
SQLStmt:="insert into app_testTable (alpha_field, longint_field) VALUES (?,?)"
For(vCounter;1;10)
   MyText:="Text"+String(vCounter)
   MyLongint:=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
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl hat die Systemvariable OK den Wert 1, sonst den Wert 0 (Null).

START SQL SERVER

START SQL SERVER
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **START SQL SERVER** startet den integrierten SQL Server in der 4D Anwendung dort, wo er ausgeführt wurde. Ist er gestartet, kann der SQL Server auf SQL Anfragen antworten, die auf dem TCP Port empfangen wurden, der in den Einstellungen der Anwendung definiert wurde.

Hinweis: Dieser Befehl beeinträchtigt nicht die interne Funktionsweise der 4D SQL Engine. Sie ist für interne Anfragen stets verfügbar.

Systemvariablen und Mengen

Bei korrekten Start des SQL Server wird die Systemvariable OK auf 1 gesetzt, sonst auf 0.

STOP SQL SERVER

STOP SQL SERVER
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **STOP SQL SERVER** stoppt den integrierten SQL Server.

Ist der SQL Server gestartet, werden alle SQL Verbindungen unterbrochen, der Server akzeptiert keine externen SQL Anfragen mehr. Ist der SQL Server nicht gestartet, hat der Befehl keine Auswirkung.

Hinweis: Dieser Befehl beeinträchtigt nicht die interne Funktionsweise der 4D SQL Engine. Sie ist für interne Anfragen stets verfügbar.

_o_USE EXTERNAL DATABASE

_o_USE EXTERNAL DATABASE (QuellName {; Benutzer ; Kennwort})

Parameter	Typ		Beschreibung
QuellName	String	➡	Name der zu verwendenden ODBC Datenquelle
Benutzer	String	➡	Benutzername
Kennwort	String	➡	Benutzerkennwort

Hinweis zur Kompatibilität

Dieser Befehl wurde ab 4D Version 11.3 durch den Befehl **SQL LOGIN** ersetzt. Er sollte nicht mehr verwendet werden.

_o_USE INTERNAL DATABASE

_o_USE INTERNAL DATABASE
Dieser Befehl benötigt keine Parameter

Hinweis zur Kompatibilität

Dieser Befehl wurde ab 4D Version 11.3 durch den Befehl **SQL LOGOUT** ersetzt und sollte nicht mehr verwendet werden.

String

- Symbole für direkten Zeichenzugriff
- 4D Transformation Tags
- 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*

Symbole für direkten Zeichenzugriff

Einführung

Mit den Symbolen **[[...]]**
können Sie sich auf ein einzelnes Zeichen innerhalb einer Zeichenkette beziehen. So können Sie in einem Feld bzw. einer Variablen vom Typ Alpha bzw. Text einzelne Zeichen ansprechen.
Diese Syntax auf der linken Seite des Zuweisungsoperators (*:=*) weist der angegebenen Position in der Zeichenkette ein Zeichen zu. Beispiel: Ist *vsName* kein leerer String, setzt folgender Code das erste Zeichen von *vsName* in Großbuchstaben:

```
If(vsName#"")
   vsName[[1]]:=Uppercase(vsName[[1]])
End if
```

Diese Syntax mit dem gewünschten Zeichen innerhalb eines Ausdrucks gibt dieses als String mit einem Zeichen zurück:

```
 //Folgendes Beispiel prüft, ob das letzte Zeichen von vtText das jokerzeichen "@" ist
If(vtText#"")
   If(Character code(Substring(vtText;Length(vtText);1))=At sign)
      ...
   End if
End if
 //Mit den Sondersymbolen schreiben Sie einfacher:
If(vtText#"")
   If(Character code(vtText[[Length(vtText)]])=At sign)
      ...
   End if
End if
```

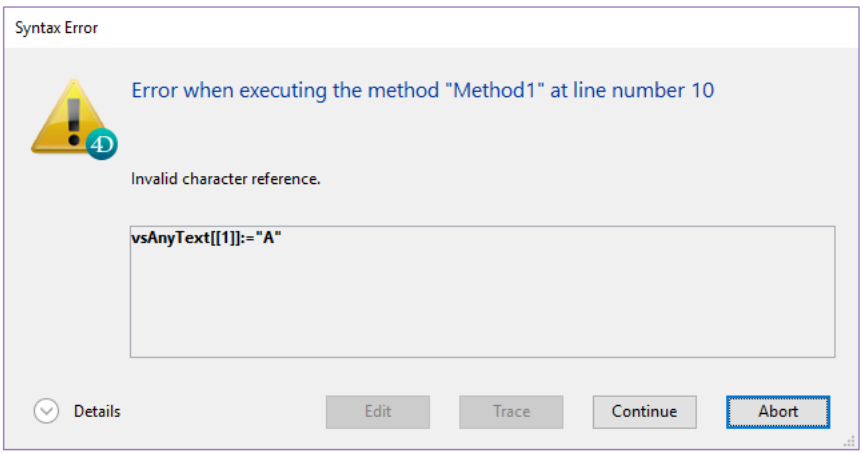
Ungültiger Zeichenzugriff

Beim Einsatz der Symbole für direkten Zeichenzugriff müssen Sie vorhandene Zeichen in einer Zeichenkette auf dieselbe Weise wie die Elemente in einem Array ansprechen. Sprechen Sie beispielsweise das 20. Zeichen einer Textvariablen an, **muss** diese Variable auch mindestens 20 Zeichen enthalten. Ist das nicht der Fall,

- erhalten Sie im interpretierten Modus keinen Syntaxfehler.
- kann das im kompilierten Modus (ohne Optionen) zum Überschreiben von anderweitig belegtem Speicher führen, wenn Sie z.B. ein Zeichen nach dem Ende einer Zeichenkette oder eines Textes schreiben.
- tritt im kompilierten Modus ein Fehler mit der Option Range Checking On auf. Bei dem Code:

```
 //Sehr schlecht und hässlich, buh!
vsAnyText:=""
vsAnyText[[1]]:="A"
```

löst folgenden Runtime-Fehler aus:



Beispiel

Folgende Projektmethode schreibt den ersten Buchstaben jedes Worts im Text groß, der als Parameter übergeben wurde und gibt den Text mit großen Anfangsbuchstaben zurück:

```
//Projektmethode Text mit Großbuchstaben
//Text mit Großbuchstaben( Text ) -> Text
//Text mit Großbuchstaben ( Quelltext ) -> Text mit Großbuchstaben
$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
```

Die Zeile:

```
ALERT(Capitalize text("hello, my name is jane doe and i'm running for president!"))
```

zeigt folgende Meldung:

4D Transformation Tags

4D bietet verschiedene Transformation Tags, um Referenzen auf 4D Variablen oder Ausdrücke einzufügen, oder verschiedene Vorgehensweisen innerhalb des Quelltextes, auch "template" genannt, auszuführen. Die Tags werden beim Ausführen des Quelltextes interpretiert und der Ausgabetext wird generiert.

Diese Funktionsweise nutzt der 4D Web Server zum Erzeugen von **Halbdynamische Seiten**.

Die Tags müssen in Form von HTML Kommentaren (*<!--#Tag Inhalt-->*) in den Quellcode eingefügt werden. Aber auch Kommentare wie *<!--Beginning of list-->* sind möglich. Es lassen sich auch verschiedene Tag Typen miteinander mischen. Hier ein Beispiel:

```
<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>
```

Vorlagen ausführen

Der Inhalt der "template" Seiten wird in zwei Kontexten ausgeführt:

- Über den Befehl **PROCESS 4D TAGS**: Er akzeptiert ein "template" als Eingabe, optional auch Parameter und gibt den Text als Ergebnis der Bearbeitung zurück.
- Über den in 4D integrierten HTTP Server: **Halbdynamische Seiten**, die über die Befehle **WEB SEND FILE** (.htm, .html, .shtm, .shtml), **WEB SEND BLOB** (text/html vom Typ BLOB) oder **WEB SEND TEXT** gesendet oder über URL aufgerufen werden. Im letzten Fall werden zwecks Optimierung Seiten mit den Endungen .htm und .html NICHT analysiert. Um hier das Analysieren der HTML Seiten zu erzwingen, müssen Sie die Endung .shtm oder shtml anfügen, z.B. http://www.server.com/dir/page.shtm. Weitere Informationen dazu finden Sie im Abschnitt **Halbdynamische Seiten** des Kapitels **Web Server**.

Tags Überblick

Nachfolgende Tabelle listet die verfügbaren 4D Transformation Tags mit Beispielen und Verwendungsweise:

Tag	Aktion	Beispiel	$ Syntax(*)	Kommentare
4DTEXT	Fügt 4D Variablen und Ausdrücke als Text ein	<!--#4DTEXT [Kunde]Name-->	X	Empfohlen bei externer Bearbeitung der Daten, um Injektion von böswilligem Code zu verhindern.
4DHTML	Fügt HTML Code ein	<!--#4DHTML -->	X	Nicht empfohlen bei externer Bearbeitung der Daten
4DEVAL	Bewertet jeden 4D Ausdruck	<!--#4DEVAL a:=20-->	X	Nicht empfohlen bei externer Bearbeitung der Daten
4DSCRIPT/	Führt eine 4D Methode mit einem Parameter aus	<!--#4DSCRIPT/MeineMethode/MeinParam-->		
4DINCLUDE	Fügt eine HTML Seite innerhalb einer anderen ein	<!--#4DINCLUDE Unterseite.html-->		
4DBASE	Bestimmt die von 4DINCLUDE verwendete Datei	<!--#4DBASE ../Datei/-->		
4DCODE	Fügt 4D Code ein	<!--#4DCODE ALERT(myVar)-->		Unterstützt CR, LF (4D Code Blöcke)
4DIF, 4DELSE, 4DELSEIF, 4DENDIF	Fügt Bedingungen in den Code in Form vom Tags ein	<!--#4DIF (myVar=1)-->		
4DLOOP, 4DENDLOOP	Fügt Schleifen in den Code in Form von Tags ein	<!--#4DLOOP [table]-->		Lässt sich mit Tabellen, Arrays, Methoden, Ausdrücken, ZeigerArrays verwenden

(*) Die Tags sollten generell als HTML Kommentare (*<!--#Tag Content-->*) in den Quelltext eingefügt werden. Die alternative Syntax mit $ eignet sich unter bestimmten Bedingungen für Tags, die Werte zurückgeben, damit sie XML konform sind. Weitere

Informationen dazu finden Sie unten im Abschnitt **Alternative Syntax für 4DTEXT, 4DHTML, 4DEVAL**.

Grundlagen zur Verwendung von Tags

Über Web auf 4D Methoden zugreifen

Für eine 4D Methode, die mit *4DTEXT*, *4DHTML*, *4DEVAL*, *4DSCRIPT*, *4DIF*, *4DELSEIF* oder *4DLOOP* über eine Web Anfrage läuft, muss in den Methodeneigenschaften die Eigenschaft "Zugang per 4D HTML Tags und URLs (4DACTION ...)" definiert sein. Ist diese Eigenschaft für die Methode nicht markiert, lässt sie sich nicht über eine Web Anfrage aufrufen. Weitere Informationen dazu finden Sie im Abschnitt **Zugang per 4D HTML Tags und URLs**.

Rekursive Durchführung

4D Tags werden rekursiv interpretiert: 4D versucht immer, das Ergebnis einer Transformation wieder zu interpretieren. Wenn eine neue Transformation stattgefunden hat, wird eine zusätzliche Interpretation durchgeführt und weitergeführt, bis das erhaltene Produkt keine weitere Transformation benötigt. Nehmen wir z.B. folgende Anweisung:

```
<!--#4DHTML [Mail]Letter_type-->
```

Enthält das Textfeld [Mail]Letter_type selbst ein Tag, z.B. *<!--#4DSCRIPT/m_Gender-->*, wird dieses Tag nach der Interpretation des 4DHTML Tag rekursiv bewertet.

Dieses Vorgehen erfüllt die meisten Anforderungen an Texttransformation. Beachten Sie jedoch, dass dies in manchen Fällen zum Einfügen von böswilligem Code führen kann.

Einfügen von böswilligem Code verhindern

4D Transformation Tags akzeptieren verschiedene Datentypen als Parameter: Text, Variablen, Methoden, Befehlsnamen, etc. Stammen diese Daten von Ihrem eigenen Code, besteht kein Risiko, dass böswilliger Code eingefügt wird, da Sie die Eingabe steuern. Jedoch arbeitet Ihr Datenbank Code oft mit Daten, die zu irgendeinem Zeitpunkt über eine externe Quelle dazugekommen sind, wie Benutzereingabe, Import, etc.
In diesem Fall raten wir, keine Transformation Tags wie 4DEVAL oder 4DSCRIPT zu verwenden, da sie die Parameter direkt mit dieser Art Daten bewerten.
Außerdem kann durch die rekursive Vorgehensweise (siehe voriger Abschnitt) böswilliger Code selbst Transformation Tags enthalten. Hier MUSS der Tag 4DTEXT verwendet werden.
Nehmen wir z.B. ein Feld im Webformular genannt "Name", in das Benutzer ihren Namen eingeben müssen. Dieser Name wird dann über ein Tag *<!--#4DHTML vName-->* auf der Seite angezeigt. Wird stattdessen Text vom Typ "*<!--#4DEVAL **QUIT 4D**-->*" eingefügt, führt die Interpretation dieses Tags zum Beenden der Anwendung.
Um dieses Risiko zu vermeiden, verwenden Sie einfach für solche Fälle systematisch das Tag 4DTEXT. Da dieses Tag die speziellen HTML Zeichen schützt, wird evtl. eingefügter böswilliger Code nicht erneut interpretiert. Das bedeutet für obiges Beispiel: Das Feld "Name" enthält in diesem Fall "*<!--#4DEVAL **QUIT 4D**-->*" es wird nicht transformiert.

Identifier mit Tokens

Um sicherzustellen, dass Ausdrücke unabhängig von der 4D Programmiersprache oder Version korrekt bewertet werden, empfehlen wir die Syntax mit Tokens bei Elementen, deren Name sich zwischen verschiedenen Versionen ändern kann (Befehle, Tabellen, Felder, Konstanten). Um beispielsweise die Funktion **Current time** einzufügen, geben Sie '**Current time:C178**' ein. Weitere Informationen dazu finden Sie im Abschnitt **Tokens in Formeln verwenden**.

Punkt "." als Dezimaltrenner verwenden

Ab v15 R4 verwendet 4D beim Bewerten eines numerischen Ausdrucks mit einem 4D Tag 4DTEXT, 4DVAR, 4DHTML, 4DHTMLVAR und 4DEVAL (bzw. den alten Tags 4DVAR und 4DHTMLVAR) immer den Punkt (.) als Dezimaltrenner. Regionale Einstellungen werden ignoriert.

Dieses Feature vereinfacht die Wartung von Code und Kompatibilität zwischen verschiedenen 4D Programmiersprachen und Versionen.

Hierzu ein Beispiel:

```
value:=10/4
input:="<!--#4DTEXT value-->"
PROCESS 4D TAGS(input;output)
  // ergibt immer 2.5, selbst wenn regionale Einstellungen ',' als Trenner verwenden
```

Hinweis zur Kompatibilität: Bewertet Ihr von einer früheren Version konvertierte Code numerische Ausdrücke über 4D Tags mit Berücksichtigung regionaler Einstellungen, müssen Sie ihn mit der Funktion **String** anpassen:

- Für *value* mit einem Punkt als Dezimalzeichen: *<!--#4DTEXT value-->*
- Für *value* mit einem Dezimalzeichen gemäß den regionalen Einstellungen: *<!--#4DTEXT String(value)-->*

4DTEXT

Syntax: *<!--#4DTEXT VarName-->* oder *<!--#4DTEXT 4DExpression-->*
Alternative Syntax: *$4DTEXT(VarName)* oder *$4DTEXT(4DExpression)* (siehe **Alternative Syntax für 4DTEXT, 4DHTML, 4DEVAL**)

Über *<!--#4DTEXT VarName-->* können Sie eine Referenz auf eine 4D Variable bzw. Ausdruck einfügen. Schreiben Sie zum Beispiel in einer HTML Seite:
<P>Welcome to *<!--#4DTEXT vtSiteName-->*!</P>,
wird der Wert der 4D Variablen *vtSiteName* in die HTML Seite eingefügt, wenn sie gesendet wird. Dieser Wert wird als einfacher Text eingefügt, spezifische HTML Zeichen wie ">" werden automatisch aufgelöst.

Über *4DTEXT* können Sie auch einen 4D Ausdruck einfügen. So können Sie den Inhalt eines Feldes, z.B. *<!--4DTEXT [Tabellenname]Feldname-->)*, ein Array Element, z.B. *<!--#4DTEXT tabarr{1}-->)* oder eine Methode, die einen Wert zurückgibt (*<!--#4DTEXT mymethod-->)* direkt einfügen. Ein Ausdruck wird genauso wie eine Variable konvertiert. Ein 4D Ausdruck muss darüberhinaus auch die Syntaxregeln von 4D berücksichtigen.

Bei einem Bewertungsfehler erscheint der eingefügte Text in Form von "*<!--#4DTEXT meinevar-->* : ## Fehler # Fehlercode".

Hinweise:

- Sie können mit Prozessvariablen arbeiten.
- Sie können den Inhalt eines Feldes vom Typ Bild anzeigen. Dagegen lässt sich der Inhalt eines Array-Elements vom Typ Bild nicht anzeigen.
- Sie können den Inhalt eines Feldes vom Typ Objekt über eine 4D Formel anzeigen. Sie können z.B. schreiben *<!--#4DTEXT **OB Get**([Rect]Desc;\"color\")-->*.
- Aus Sicherheitsgründen empfehlen wir, dieses Tag beim Bearbeiten von Daten zu verwenden, die von außerhalb kommen, um das Einfügen von bösartigem Code zu vermeiden. Weitere Informationen dazu finden Sie oben im Abschnitt **Einfügen von böswilligem Code verhindern**.
- In der Regel arbeiten Sie mit Textvariablen. Sie können aber auch Variablen vom Typ BLOB einsetzen. Dazu müssen Sie lediglich das BLOB im Modus *Text without length* generieren.

4DHTML

Syntax: *<!--4DHTML VarName-->* oder *<!--4DHTML 4D Expression-->*
Alternative Syntax: *$4DHTML(VarName)* oder *$4DHTML(4DExpression) (siehe **Alternative Syntax für 4DTEXT, 4DHTML, 4DEVAL**)*

Über dieses Tag können Sie, wie beim Tag ***4DTEXT**,* eine Variable oder einen 4D Ausdruck bewerten und in einen HTML Ausdruck einfügen. Dieser Tag löst jedoch NICHT spezielle HTML Zeichen auf. Hierzu ein Beispiel:

Wert von meinevar	Tags	Ergebnis
meinevar:="<B>"	*<!--#4DTEXT meinevar-->*	<B>
meinevar:="<B>"	*<!--#4DHTML meinevar-->*	<B>

Bei einem Bewertungsfehler erscheint der eingefügte Text in Form von "*<!--#4DHTML meinevar-->* : ## Fehler # Fehlercode".

Hinweis: Aus Sicherheitsgründen empfehlen wir, das Tag 4DTEXT beim Bearbeiten von Daten zu verwenden, die von außerhalb kommen, um das Einfügen von bösartigem Code zu vermeiden. Weitere Informationen dazu finden Sie oben im Abschnitt **Einfügen von böswilligem Code verhindern**.

4DEVAL

Syntax: *<!--#4DEVAL VarName--> oder <!--#4DEVAL 4DExpression-->*
Alternative Syntax: *$4DEVAL(VarName)* oder *$4DEVAL(4DExpression*) (siehe **Alternative Syntax für 4DTEXT, 4DHTML, 4DEVAL**)

Über das Tag 4DEVAL können Sie auf eine Variable oder einen 4D Ausdruck zugreifen. Wie 4DHTML löst auch 4DEVAL die HTML Zeichen im zurückgegebenen Text nicht auf. Im Gegensatz zu 4DHTML oder 4DTEXT kann 4DEVAL jedoch jede gültige 4D Anweisung ausführen, d.h. auch Anweisungen und Ausdrücke, die keinen Wert zurückgeben.

Sie können z.B. folgendes ausführen:

```
$input:="<!--#4DEVAL a:=42-->"  // Zuweisung ohne Ausgabe
$input:=$input+"<!--#4DEVAL a+1-->"  // Berechnung mit Ausgabe
PROCESS 4D TAGS($input;$output)
 //$output = "43"
```

Bei fehlerhafter Interpretation wird der Text in Form von <!--#4DEVAL expression--> : ## Fehler # Fehlernummer" eingefügt.

Hinweis: Aus Sicherheitsgründen empfehlen wir, das Tag **4DTEXT** beim Bearbeiten von Daten zu verwenden, die von außerhalb kommen, um das Einfügen von bösartigem Code zu vermeiden. Weitere Informationen dazu finden Sie oben im Abschnitt **Einfügen von böswilligem Code verhindern**.

4DSCRIPT/

Syntax: *<!--#4DSCRIPT/MeineMeth/MeinParam-->*

Mit *4DSCRIPT* können Sie beim Bearbeiten des Template 4D Methoden ausführen. Existiert *<!--#4DSCRIPT/MeineMeth/MeinParam-->* als HTML Kommentar, wird die Methode ***MeineMeth*** mit dem Parameter *MeinParam* als ein String in $1 ausgeführt. Wird das Tag im Rahmen eines Web Prozesses beim Laden der Seite aufgerufen, ruft 4D – sofern vorhanden – die **Datenbankmethode On Web Authentication** auf. Gibt diese *Wahr* zurück, führt 4D die Methode aus.

Die Methode muss in $0 Text zurückgeben. Beginnt der String mit dem Zeichencode 1, wird er als HTML betrachtet. Dasselbe Prinzip gilt auch für das Tag *4DHTML*.

Fügen Sie zum Beispiel den Kommentar "heute ist der *<!--#4DSCRIPT/MYMETH/MYPARAM-->*" in eine statische Seite ein. Beim Laden der Seite ruft 4D –sofern vorhanden– die **Datenbankmethode On Web Authentication** auf, dann die Methode ***MeineMeth*** und übergibt den String "/MeinParam" als Parameter $1.

Die Methode gibt in $0 Text zurück, zum Beispiel "31.12.14", der Ausdruck "heute ist der *<!--4DSCRIPT/MeineMeth/MeinParam--->*" also zu "Heute ist der 31.12.14".

Die Methode ***MeineMeth*** lautet:

```
 //MeineMeth
C_TEXT($0;$1) //Diese Parameter müssen immer angegeben werden
$0:=String(Current date)
```

Hinweis: Eine Methode, die über *4DSCRIPT* aufgerufen wird, darf keine Elemente der Oberfläche aufrufen, wie **DIALOG**, **ALERT**... .

Da 4D Methoden in ihrer eigenen Reihenfolge erscheinen, lässt sich eine Methode aufrufen, die den Wert von vielen Variablen setzt, auf die später im Dokument verwiesen wird. Sie können in ein Template beliebig viele *<!--4DSCRIPT...-->* Kommentare einfügen.

4DINCLUDE

Syntax: *<!--#4DINCLUDE Pfad-->*

Dieses Tag dient hauptsächlich dazu, um in eine HTML Seite den Hauptteil einer anderen HTML Seite einzufügen (angegeben im Parameter *Pfad*). Als Hauptteil einer HTML Seite gilt alles, was innerhalb der Ausdrücke <body> und </body> steht. Die Tags selbst werden nicht eingefügt. Auf diese Weise können Sie Konflikte mit den im Header enthaltenen Meta Tags vermeiden. Enthält die angegebene HTML Seite keine Tags <body> </body>, wird die komplette Seite integriert. Sie müssen selbst für die Konsistenz der Meta Tags sorgen.
Der Kommentar *<!--#4DINCLUDE-->* ist besonders hilfreich in Verbindung mit dem Test (*<!--#4DIF-->*) oder den Schleifen (*<!--#4DLOOP-->*). Er ist auch praktisch, um Tags nach einem bestimmten Kriterium oder zufällig einzufügen.
4D analysiert die aufgerufene Seite im Moment des Einfügens, unabhängig von der Endung des Dateinamens, und fügt dann den evtl. geänderten Inhalt in die Seite ein, von der der Aufruf *4DINCLUDE* ausgeht.

Eine Seite, die über den Kommentar *<!--#4DINCLUDE-->* eingefügt wurde, wird im Web Cache genauso gesetzt wie die Seiten, die über eine URL angefordert oder über den Befehl **WEB SEND FILE** gesendet wurden.
In *Pfad* übergeben Sie den Zugriffspfad des einzufügenden Dokuments. Beachten Sie, dass sich der Pfad bei Aufrufen mit *4DINCLUDE* nach dem Dokument richtet, das gerade analysiert wird, d.h. nach dem Hauptdokument. Verwenden Sie den Schrägstrich (/) als Trenner zwischen den Dokumenten und die zwei Punkte (..), um in der Hierarchie eine Ebene höher zu gehen (HTML Syntax).

Hinweise:

- Verwenden Sie das Tag 4DINCLUDE mit dem Befehl **PROCESS 4D TAGS**, ist der Standardordner der Ordner mit der Strukturdatei der Anwendung.
- Mit *<!--#4DBASE -->* können Sie den Standardordner, den *4DINCLUDE* in der aktuellen Seite verwendet, verändern (siehe unten)

Sie können *<!--#4DINCLUDE Pfad-->* innerhalb einer Seite beliebig oft verwenden. Die Aufrufe von *<!--#4DINCLUDE Pfad-->* sind jedoch nur auf einer Ebene möglich. Sie können also nicht den Kommentar *<!--#4DINCLUDE meindok3.html-->* in den Hauptteil der Seite *meindok2.html* einfügen, die wiederum von *<!--#4DINCLUDE meindok2-->* aufgerufen wird, welches in *meindok1.html* eingefügt ist. 4D prüft außerdem, dass die Einfügungen nicht rekursiv sind.

Bei einem Fehler erscheint der eingefügte Text in Form von "*<!--#4DINCLUDE Pfad-->*: Dokument kann nicht geöffnet werden".

Beispiele

```
<!--#4DINCLUDE Unterseite.html--> <!--#4DINCLUDE Ordner/Unterseite.html--> <!--#4DINCLUDE ../Ordner/Unterseite.html-->
```

4DBASE

Syntax: *<!--#4DBASE folderPath-->*

Das Tag *<!--#4DBASE -->* bezeichnet ein Arbeitsverzeichnis, das vom Tag *<!--#4DINCLUDE-->* verwendet wird.
Beim Aufrufen in einer Web Seite ändert das Tag *<!--#4DBASE -->* alle nachfolgenden Aufrufe von *<!--#4DINCLUDE-->* auf dieser Seite bis zum nächsten *<!--#4DBASE -->* - sofern vorhanden. Wird der Ordner *<!--#4DBASE -->* aus einer enthaltenden Datei heraus geändert, findet er deren ursprünglichen Wert aus der übergeordneten Datei.

Der Parameter *folderPath* muss einen Pfad relativ zur aktuellen Seite enthalten und dieser muss mit einen Schrägstrich (/) enden. Der angegebene Ordner muss innerhalb des Web Ordners liegen.
Übergeben Sie das Schlüsselwort WEBFOLDER, um den Standardpfad, d.h. relativ zur Seite, wiederherzustellen.

Der folgende Code aus 4D v12, der für jeden Aufruf einen relativen Pfad angibt:

<!--#4DINCLUDE subpage.html--> <!--#4DINCLUDE folder/subpage1.html--> <!--#4DINCLUDE folder/subpage2.html--> <!--#4DINCLUDE folder/subpage3.html--> <!--#4DINCLUDE ../folder/subpage.html-->

... lässt sich mit dem Tag *<!--#4DBASE -->* schreiben wie folgt:

<!--#4DINCLUDE subpage.html--> <!--#4DBASE folder/--> <!--#4DINCLUDE subpage1.html--> <!--#4DINCLUDE subpage2.html--> <!--#4DINCLUDE subpage3.html--> <!--#4DBASE ../folder/--> <!--#4DINCLUDE subpage.html--> <!--#4DBASE WEBFOLDER-->

Beispiel

Verzeichnis für die Home Page über das Tag *<!--#4DBASE -->* erstellen:

```
/* Index.html */ <!--#4DIF LangFR=True-->  <!--#4DBASE FR/--> <!--#4DELSE-->  <!--#4DBASE US/--> <!--#4DENDIF--> <!--#4DINCLUDE head.html--> <!--#4DINCLUDE body.html--> <!--#4DINCLUDE footer.html-->
```

In der Datei Head.html wird der aktuelle Ordner durch *<!--#4DBASE -->* geändert, ohne dass sich sein Wert in Index.htm ändert:

```
/* Head.htm */ /* das Arbeitsverzeichnis hier ist relativ zur enthaltenen Datei (FR/ oder US/) */ <!--#4DBASE Styles/--> <!--#4DINCLUDE main.css--> <!--#4DINCLUDE product.css--> <!--#4DBASE Scripts/--> <!--#4DINCLUDE main.js--> <!--#4DINCLUDE product.js-->
```

4DCODE

Mit dem Tag 4DCODE können Sie einen mehrzeiligen Block mit 4D Code in eine Vorlage einfügen.

Wird eine "<!--#4DCODE" Sequenz gefunden, die mit einem Abstand, einem Zeichen CR oder LF abschließt, interpretiert 4D alle Code Zeilen bis zur nächsten "-->" Sequenz. Innerhalb des Code Blocks kann es Zeilenschaltungen, Zeilenvorschub oder beides geben, dies wird von 4D sequentiell interpretiert.

Sie können z.B. mit dem Tag 4DCODE in einer Vorlage schreiben:

```
<!--#4DCODE
//Initialisierung der PARAMETER
C_OBJECT:C1216($graphParameters)
OB SET:C1220($graphParameters;"graphType";1)
$graphType:=1
//...Ihr Code hier
if (OB Is defined:C1231($graphParameters;"graphType"))
    $graphType:=OB GET:C1224($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
-->
```

Hinweis: In einem Tag 4DCODE muss immer die Englisch-US Programmiersprache verwenden werden. Deshalb ignoriert 4DCODE die Benutzereinstellung "Verwende regionale Systemeinstellungen" (siehe **Sprache für Befehle und Konstanten**).

Für den Tag 4DCODE gibt es folgende Vorgaben:

- Der Befehl **TRACE** wird unterstützt und aktiviert den 4D Debugger, so dass Sie Ihren Template Code debuggen können.
- Jeder Fehler zeigt den Standard Fehlerdialog, über den der Nutzer die Ausführung des Code stoppen oder in den Schrittmodus zum Debuggen gehen kann.
- Der Text zwischen <!--#4DCODE und --> ist in Zeilen aufgeteilt, die die verschiedenen Zeilenenden (cr, lf oder crlf) zulassen.
- Der Text wird im Kontext der Datenbank tokenisiert, die **PROCESS 4D TAGS** aufgerufen hat. Das ist zum Beispiel zum Erkennen von Projektmethoden wichtig.
  Hinweis: Die Methodeneigenschaft "Zugang per 4D Tags und URLs (4DAction)" wird nicht berücksichtigt (siehe auch unten **Hinweis zur Sicherheit**).
- Der 4D Code muss immer mit der Programmiersprache in Englisch-US geschrieben werden. Deshalb ignoriert 4DCODE die Benutzereinstellungen "Verwende regionale Systemeinstellungen" für die 4D Programmiersprache (siehe **Sprache für Befehle und Konstanten**).
- Auch wenn der Text immer Englisch-US verwendet, wird empfohlen, die Syntax *:Cxxx* und *:Kxxx* für Befehls- und Konstantennamen zu verwenden, um Umbenennung von Befehlen oder Konstanten von einer 4D Version zur nächsten aufzufangen.
  Hinweis: Weitere Informationen zur Syntax *:Cxxx* und *:Kxxx* finden Sie im Abschnitt **Tokens in Formeln verwenden**.

Hinweis zur Sicherheit: Die Tatsache, dass 4DCODE Tags einen beliebigen Befehl der 4D Programmiersprache oder Projektmethoden aufrufen kann, könnte als Sicherheitsrisiko eingestuft werden, besonders wenn die Datenbank über HTTP verfügbar ist. Da er jedoch server-seitigen Code aus Ihren eigenen Vorlagedateien ausführt, ist das Tag selbst kein Sicherheitsrisiko. In diesem Kontext wird die Sicherheit, wie für jeden Web Server, hauptsächlich auf der Ebene von externen Zugriffen auf die Server-Dateien verwaltet.

4DIF, 4DELSE, 4DELSEIF und 4DENDIF

Syntax: *<!--#4DIF expression--> {<!--#4DELSEIF expression2-->...<!--#4DELSEIF expressionN-->} {<!--#4DELSE-->} <!--#4DENDIF-->*

Der Kommentar <!--#4DIF expression--> ermöglicht in Verbindung mit den Kommentaren *<!--#4DELSE-->* (optional) und *<!--#4DENDIF-->*, Teile des Code auf bedingte Weise auszuführen.
Der Parameter *expression* kann jeden gültigen 4D Ausdruck enthalten, der einen booleschen Wert zurückgibt. Er muss in Klammern stehen und die Syntaxregeln von 4D berücksichtigen.

Die Blöcke *<!--#4DIF expression>* ... *<!--#4DENDIF-->* können ineinander verschachtelt sein. Analog zu 4D muss jeder mit *<!--#4DIF expression-->* geöffnete Kommentar mit *<!--#4DENDIF-->* geschlossen werden.

Bei einem Interpretationsfehler wird zwischen *<!--#4DIF -->* und *<!--#4DENDIF-->* anstelle des Inhalts der Text "*<!--#4DIF expression-->*: Ein boolescher Ausdruck wurde erwartet" eingefügt.
Entspricht nicht jedem *<!-#-4DENDIF-->* ein *<!--#4DIF -->*, wird anstelle des Inhalts der Text "*<!--#4DIF expression-->*: 4DENDIF wurde erwartet" eingefügt.

Über das Tag *<!--#4DELSEIF-->* können Sie eine unbegrenzte Anzahl von Bedingungen testen. Nur der Code, der auf die erste als Wahr gewertete Bedingung folgt, wird ausgeführt. Ist keine der Bedingungen wahr, wird keine Anweisung ausgeführt (wenn es kein abschließendes *<!--#4DELSE-->* gibt).
Sie können nach dem letzten Tag *<!--#4DELSEIF-->* ein Tag *<!--#4DELSE-->* verwenden. Sind alle Bedingungen falsch, werden die Anweisungen ausgeführt, die auf *<!--#4DELSE-->* folgen.

Die beiden nachfolgenden Codes sind gleichwertig:

- Code nur mit 4DELSE:

```
<!--#4DIF Condition1--> /* Condition1 ist wahr*/ <!--#4DELSE--> <!--#4DIF Condition2--> /* Condition2 ist wahr*/ <!--#4DELSE--> <!--#4DIF Condition3--> /* Condition3 ist wahr*/ <!--#4DELSE--> /*Keine der Bedingungen ist wahr*/ <!--#4DENDIF--> <!--#4DENDIF--> <!--#4DENDIF-->
```

- Ähnlicher Code mit dem Tag 4DELSEIF:

```
<!--#4DIF Condition1--> /* Condition1 ist wahr*/ <!--#4DELSEIF Condition2--> /* Condition2 ist wahr*/ <!--#4DELSEIF Condition3--> /* Condition3 ist wahr*/ <!--#4DELSE--> /* Keine der Bedingungen ist wahr*/ <!--#4DENDIF-->
```

Beispiel 1

Nachfolgendes Beispiel zeigt Code aus einer statischen HTML Seite, die je nach Ergebnis des Ausdrucks vname#"" eine andere Bezeichnung zeigt:

```
<BODY> ... <!--#4DIF (vname#"")--> Namen, die mit <!--#4DTEXT vname--> beginnen. <!--#4DELSE--> Es wurde kein Name gefunden. <!--#4DENDIF--> ... </BODY>
```

Beispiel 2

Dieses Beispiel fügt je nach angemeldetem Benutzer unterschiedliche Seiten ein:

```
<!--#4DIF LoggedIn=False--> <!--#4DINCLUDE Login.htm --> <!--#4DELSEIF User="Admin" --> <!--#4DINCLUDE AdminPanel.htm --> <!--#4DELSEIF User="Manager" --> <!--#4DINCLUDE SalesDashboard.htm --> <!--#4DELSE--> <!--#4DINCLUDE ItemList.htm --> <!--#4DENDIF-->
```

4DLOOP und 4DENDLOOP

Syntax: *<!--#4DLOOP Bedingung--> <!--#4DENDLOOP-->*

Mit diesem Kommentar können Sie den Teil eines HTML Codes so oft wiederholen wie die Bedingung erfüllt ist. Der Teil wird begrenzt durch *<!--#4DLOOP-->* und *<!--#4DENDLOOP-->*. Sie können ineinander verschachtelt sein. Analog zu 4D muss jeder mit *<!--4DLOOP Bedingung-->* geöffnete Kommentar mit *<!--#4DENDLOOP-->* geschlossen werden.

Es gibt fünf Bedingungstypen:

- **<!--#4DLOOP [Tabelle]-->**
  Diese Syntax führt für jeden Datensatz der aktuellen Auswahl von Tabelle eine Schleife im laufenden Prozess aus. Der Teil des HTML Codes zwischen den beiden Kommentaren wird für jeden Datensatz der aktuellen Auswahl wiederholt.
  Hinweis: Wird ein Tag 4DLOOP mit einer Tabelle verwendet, werden die Datensätze im Nur-Lesen Modus geladen.

  Dieser Code:

  ```
  <!--#4DLOOP [People]--> <!--#4DTEXT [People]Name--> <!--#4DTEXT [People]Surname--><BR> <!--#4DENDLOOP-->
  ```

  ... lässt sich in 4D Code folgendermaßen ausdrücken:

  ```
  FIRST RECORD([People])
  While(Not(End selec tion([People])))
     ...
     NEXT RECORD([People])
  End while
  ```

- **<!--#4DLOOP array-->**
  Diese Syntax führt für jedes Element des Array eine Schleife aus. Der laufende Zähler des Array erhöht sich mit jeder Wiederholung dieses Teils des HTML Codes.
  Hinweis: Diese Syntax ist für zweidimensionale Arrays nicht verwendbar. Die Lösung: eine Methode mit verschachtelten Schleifen kombinieren.

  Dieser Code:

  ```
  <!--#4DLOOP arr_names--> <!--#4DTEXT arr_names{arr_names}--><BR> <!--#4DENDLOOP-->
  ```

  ... lässt sich folgendermaßen in 4D Code übersetzen:

  ```
  For($Elem;1;Size of array(arr_names))
     arr_names:=$Elem
     ...
  End for
  ```

- **<!--#4DLOOP Methode-->**
  Diese Syntax führt eine Schleife aus, solange die Methode *Wahr* zurückgibt. Sie lässt einen Parameter vom Typ Lange Ganzzahl zu. Sie wird das erste Mal mit dem Wert 0 (Null) aufgerufen, damit evtl. eine Initialisierung durchgeführt werden kann. Dann wird sie sukzessiv mit den Werten 1, 2, 3... aufgerufen, solange sie Wahr zurückgibt.

  Aus Sicherheitsgründen kann die **Datenbankmethode On Web Authentication** einmal direkt vor der Initialisierung aufgerufen werden (Methodenausführung mit 0 als Parameter). Wird die Authentifizierung bestätigt, findet die Initialisierungsphase statt.

  Warnung: Damit die Datenbank kompilierbar ist, müssen innerhalb der Methode unbedingt die Compilerbefehle **C_BOOLEAN**($0) und **C_LONGINT**($1) deklariert sein.

  Dieser Code:

```
<!--#4DLOOP meine_methode--> <!--#4DTEXT var--> <BR> <!--#4DENDLOOP-->
```

  ... lässt sich folgendermaßen in 4D Code übersetzen:

```
If(AuthenticationWebOK)
   If(my_method(0))
      $counter:=1
      While(my_method($counter))
         ...
         $counter:=$counter+1
      End while
   End if
End if
```

  Die Methode ***meine_methode*** lautet:

```
C_LONGINT($1)
C_BOOLEAN($0)
If($1=0)  `Initialisation
   $0:=True
Else
   If($1<50)
      ...
      var:=...
      $0:=True
   Else
      $0:=False  `Stops the loop
   End if
End if
```

- **<!--#4DLOOP 4DAusdruck-->**
  Mit dieser Syntax führt das Tag 4DLOOP eine Schleife aus, solange der 4D Ausdruck *Wahr* zurückgibt. Das kann jeder gültige Boolean Ausdruck sein und dieser sollte in jeder Schleife einen variablen Part zur Bewertung haben, um unendliche Schleifen zu vermeiden.

  Dieser Code:

```
<!--#4DEVAL $i:=0--> <!--#4DLOOP ($i<4)--> <!--#4DEVAL $i--> <!--#4DEVAL $i:=$i+1--> <!--#4DENDLOOP-->
```

  ergibt folgendes:

```
0
1
2
3
```

- **<!--#4DLOOP ZeigerArray-->**
  In diesem Fall arbeitet das Tag 4DLOOP wie mit einem Array, d.h. es macht für jeden Eintrag im Array eine Schleife. Der aktuelle Eintrag des Array erhöht sich, wenn dieser Part im Code wiederholt wird.

  Diese Syntax ist hilfreich, wenn Sie im Befehl **PROCESS 4D TAGS** als Parameter einen Array Zeiger übergeben.

  Beispiel:

```
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"
```

Bei einem Interpretationsfehler wird zwischen *<!--#4DLOOP-->* und *<!--#4DENDLOOP-->* der Text "*<!--#4DLOOP expression-->*:Beschreibung" eingefügt.

Folgende Fehlerbeschreibungen sind möglich:

- Ein derartiger Ausdruck wurde nicht erwartet (generischer Fehler)
- Ungültiger Tabellenname (Fehler im Tabellennamen)
- Es wurde ein Array erwartet (die Variable ist kein Array bzw. ein zweidimensionales Array)
- Die Methode existiert nicht
- Syntaxfehler (beim Ausführen der Methode)
- Zugriffsfehler (Rechte nicht ausreichend für Zugriff auf Tabelle oder Methode)
- *4DENDLOOP* wird erwartet (die Anzahl von *<!--#4DENDLOOP-->* passt nicht zur Anzahl von *<!--#4DLOOP-->)*.

Alternative Syntax für 4DTEXT, 4DHTML, 4DEVAL

Einige vorhandene 4D Transformation Tags lassen sich jetzt mit der neuen Syntax mit $ schreiben:
Sie können **$4dtag (*expression*)** anstelle von **<!--#4dtag *expression*-->** verwenden

Diese alternative Syntax ist nur für Tags verfügbar, die bearbeitete Werte zurückgeben:

- 4DTEXT
- 4DHTML
- 4DEVAL

(andere Tags, wie 4DIF oder 4DSCRIPT, müssen die reguläre Syntax verwenden)

Sie können zum Beispiel schreiben:

```
$4DEVAL(UserName)
```

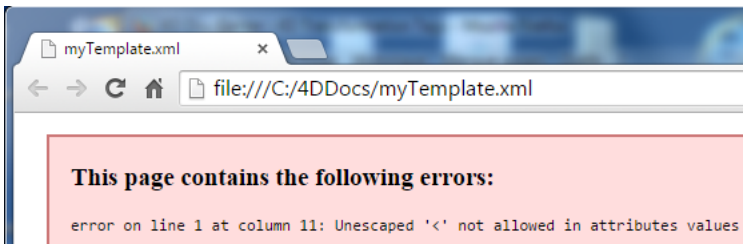
anstelle von:

```
<!--#4DEVAL(UserName)-->
```

Hauptvorteil dieser neuen Syntax ist,dass sie **für XML geeignete Vorlagen** ermöglicht. Einige 4D Entwickler müssen auf XML basierende Vorlagen erstellen und diese über standardmäßige XML Analyse Tools auswerten. Da das Zeichen "<" in einem XML Attributwert ungültig ist, ließ sich die Syntax "<!-- -->" von 4D Tags nicht ohne Brechen der Dokumentsyntax verwenden. Außerdem verhindert das auflösende Zeichen "<" die korrekte Interpretation von Tags durch 4D.

So würde der folgende Code wegen dem ersten "<" einen XML Parsing Fehler verursachen:

```
<line x1="<!--#4DEVAL $x-->" y1="<!--#4DEVAL $graphY1-->"/>
```



Mit der $ Syntax wird der nachfolgende Code vom Parser korrekt interpretiert:

```
<line x1="$4DEVAL($x)" y1="$4DEVAL($graphY1)"/>
```

Beachten Sie, dass *$4dtag* und *<--#4dtag -->* nicht exakt identisch sind: im Gegensatz zu *<--#4dtag -->* interpretiert ein *$4dtag* 4D Tags beim Bearbeiten nicht rekursiv. $ Tags werden immer einmal bewertet und das Ergebnis wird als Volltext gewertet.

Hinweis: Weitere Informationen dazu finden Sie im Abschnitt .

Grund für diesen Unterschied ist der Schutz vor Injektion von böswilligem Code. Wie bereits im Handbuchs *Programmiersprache* erläutert, wird dringend geraten,zum Schutz vor ungewollter Reinterpretation von Tags beim Verwalten von Benutzertext 4DTEXT Tags anstelle von 4DHTML Tags zu verwenden: Mit 4DTEXT werden Sonderzeichen wie "<" aufgelöst, so dass alle 4D Tags mit der Syntax *<!--#4dtag expression -->* ihre spezielle Bedeutung verlieren. Da 4DTEXT jedoch das $ Symbol nicht auflöst, haben wir entschieden, die rekursive Unterstützung aufzuheben, um jegliche Injektion von böswilligem Code über die Syntax *$4dtag (expression)* zu verhindern.

Nachfolgende Beispiele zeigen das Ergebnis der Bewertung je nach verwendeter Syntax und Tag:

```
 // Beispiel 1
myName:="<!--#4DHTML QUIT 4D-->"  //böswillige Injektion
input:="My name is: <!--#4DHTML myName-->"
PROCESS 4D TAGS(input;output)
 //4D beendet!
```

```
 // Beispiel 2
myName:="<!--#4DHTML QUIT 4D-->"  //böswillige Injektion
input:="My name is: <!--#4DTEXT myName-->"
PROCESS 4D TAGS(input;output)
 //Ausgabe ergibt "My name is:  & lt;!--#4DHTML QUIT 4D-->"
```

```
 // Beispiel 3
myName:="$4DEVAL(QUIT 4D)"  //böswillige Injektion
input:="My name is: <!--#4DTEXT myName-->"
PROCESS 4D TAGS(input;output)
 //Ausgabe ergibt "My name is: $4DEVAL(QUIT 4D)"
```

Beachten Sie, dass die Syntax *$4dtag* zusammenpassende Paare von eingebundenen Anführungszeichen oder Klammern unterstützt. Angenommen, Sie müssen (rein hypotetisch) folgenden komplexen String bewerten:

```
String(1) + "\"(hello)\""
```

Sie können schreiben:

```
input:="$4DEVAL( String(1)+\"\\\"(hello)\\\"\")"
PROCESS 4D TAGS(input;output)
-->Ausgabe ergibt 1"(hello)"
```

Change string

Change string (Quelle ; Neuer ; Beginn) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Quelle	String	➡	Text, in dem Teile ersetzt werden sollen
Neuer	String	➡	Neue Zeichen
Beginn	Lange Ganzzahl	➡	Beginn der Änderung
Funktionsergebnis	String	⮌	Geänderter Text

Beschreibung

Change string ersetzt eine Zeichengruppe im Text *Quelle* durch den Text *Neuer* und gibt diesen Text zurück. Der Befehl überschreibt *Quelle* ab dem in *Neuer* angegebenen Zeichen.

Ist *Neuer* ein leerer String (""), bleibt *Quelle* unverändert. **Change string** gibt immer einen String mit derselben Länge wie *Quelle* zurück. Ist *Neuer* kleiner als Eins oder länger als der Text *Quelle*, wird der ursprüngliche Text zurückgegeben.

Change string fügt im Gegensatz zu **Insert string** keine Zeichen ein, sondern überschreibt sie.

Beispiel

Folgendes Beispiel zeigt die Verwendung von **Change string**. Die Ergebnisse werden der Variablen *vtResult* zugewiesen.

```
vtResult:=Change string("Acme";"CME";2) ` vtResult ergibt "ACME"
vtResult:=Change string("November";"Dez";1) ` vtResult ergibt "Dezember"
```

Char

Char (CharCode) -> Funktionsergebnis

Parameter	Typ		Beschreibung
CharCode	Lange Ganzzahl	→	Zeichencode
Funktionsergebnis	String	↩	In CharCode dargestelltes Zeichen

Beschreibung

Die Funktion **Char** gibt das Zeichen zurück, dessen Code gleich *CharCode* ist.
In *CharCode* übergeben Sie einen UTF-16 Wert (zwischen 1 und 65535).

Tipp: Mit dieser Funktion werden hauptsächlich die Zeichen gesetzt, die sich beim Schreiben einer Methode nicht durch die Tastatur eingeben lassen oder im Methodeneditor als ein Bearbeitungsbefehl interpretiert werden könnten.

Beispiel

Folgendes Beispiel fügt im Text einer Fehlermeldung mit **Char** einen Zeilenumbruch ein:

```
ALERT("Angestellte: "+String(Records in table([Employees]))+Char(Carriage return)+"Drücke OK für weiter.")
```

Character code

Character code (Zeichen) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Zeichen	String	→	Zeichen, dessen Code gewünscht wird
Funktionsergebnis	Lange Ganzzahl	↩	Code für das Zeichen

Beschreibung

Die Funktion **Character code** gibt den Code des Zeichens in Unicode UTF-16 (1 bis 65535) zurück, das im Parameter *Zeichen* übergeben wurde.
Bei mehr als 1 Zeichen in *Zeichen*, gibt **Character code** nur den Code des ersten Zeichens zurück.
Die Funktion **Char** ist das Gegenstück zu **Character code**. Sie gibt das Zeichen zum entsprechenden UTF-16 Code zurück.

Beispiel 1

In einem Vergleich wird normalerweise nicht zwischen Klein- und Großschreibung unterschieden. Mit **Character code** können Sie diese Unterscheidung machen.
Folgende Zeile gibt TRUE zurück:

```
("A")=("a")
```

Folgende Zeile dagegen gibt FALSE zurück:

```
(Character code("A")=Character code("a"))
```

Beispiel 2

Dieses Beispiel gibt den Code des ersten Zeichens der Zeichenkette *"ABC"* zurück:

```
GetCode=Character code("ABC")  ` GetCode ergibt 65, den Character Code von A
```

Beispiel 3

Folgendes Beispiel prüft den Zeilenumbruch und Tabulatorzeichen:

```
For($vlChar;1;Length(vtText))
   Case of
      :(vtText[[$vlChar]]=Char(Carriage return))
   ` Führe etwas aus
      :(vtText[[$vlChar]]=Char(Tab))
   ` Führe wieder etwas aus
      :(...)
   ` ...
   End case
End for
```

Bei mehrmaliger Ausführung in langen Texten läuft der Text kompiliert schneller mit folgender Schreibweise ab:

```
For($vlCode;1;Length(vtText))
   $vlCode:=Character code(vtText[[$vlChar]])
   Case of
      :($vlCode=Carriage return)
   ` Führe etwas aus
      :($vlCode=Tab)
   ` Führe wieder etwas aus
      :(...)
   ` ...
   End case
End for
```

Der zweite Teil des Code läuft schneller, weil er pro Durchlauf nur auf ein Zeichen zugreift und für Zeilenumbruch und Tabulatorzeichen nicht Zeichenketten, sondern Lange Ganzzahlen miteinander vergleicht. Verwenden Sie diese Technik, wenn Sie

mit gängigen Codes wie CR und TAB arbeiten.

CONVERT FROM TEXT

CONVERT FROM TEXT (4Dtext ; Zeichensatz ; BLOBkonvertiert)

Parameter	Typ		Beschreibung
4Dtext	String	➡	Text im aktuellen Zeichensatz von 4D
Zeichensatz	String, Lange Ganzzahl	➡	Name oder Nummer des Zeichensatzes
BLOBkonvertiert	BLOB	⬅	BLOB mit konvertiertem Text

Beschreibung

Der Befehl **CONVERT FROM TEXT** wandelt einen Text im aktuellen Zeichensatz von 4D in einen Text mit einem anderen Zeichensatz um.
Im Parameter *4Dtext* übergeben Sie den zu konvertierenden Text. Dieser Text wird im Zeichensatz von 4D dargestellt, ab Version 11 verwendet 4D den Unicode Zeichensatz.
In *Zeichensatz* übergeben Sie den gewünschten Zeichensatz. Sie können eine Zeichenkette mit dem Standardnamen übergeben, z.B. "ISO-8859-1", "UTF-8" oder ihren MIBEnum Identifier.

Nachfolgend sind die Zeichenketten aufgeführt, welche die Routinen **CONVERT FROM TEXT** und **Convert to text** unterstützen:

MIBEnum	Name(n)
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
2024	Windows-31J
57	GB_2312-80
57	csISO58GB231280

Hinweis: Einige Zeilen haben den gleichen MIBEnum Identifier, da ein Zeichensatz mehr als einen Namen (Alias) haben kann.

Weitere Informationen über Namen von Zeichensätzen finden Sie im Internet unter *http://www.iana.org/assignments/character-sets* .

Nach Ausführung des Befehls wird der konvertierte Text in *BLOBkonvertiert* zurückgegeben. Dieses BLOB ist mit der Funktion **Convert to text** lesbar.

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

Convert to text

Convert to text (BLOB ; Zeichensatz) -> Funktionsergebnis

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	BLOB mit Text im angegebenen Zeichensatz
Zeichensatz	String, Lange Ganzzahl	➡	Name oder Nr. des BLOB Zeichensatzes
Funktionsergebnis	Text	⮌	Inhalt des BLOB in 4D Zeichensatz

Beschreibung

Die Funktion **Convert to text** konvertiert den Text im Parameter *BLOB* und gibt den Text im Zeichensatz von 4D zurück. 4D verwendet standardmäßig den UTF-16 Zeichensatz.

In *Zeichensatz* übergeben Sie den Zeichensatz des in *BLOB* enthaltenen Textes für die Konvertierung. Enthält das BLOB Text, der innerhalb 4D kopiert wurde, wird in der Regel der UTF-16 Zeichensatz verwendet.

Sie können eine Zeichenkette mit dem Standardnamen des Zeichensatzes übergeben oder eins seiner Aliase (z.B. "ISO-8859-1" oder "UTF-8") bzw. Identifier (Lange Ganzzahl). Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **CONVERT FROM TEXT**.

Convert to text unterstützt BOM (Byte Order Marks). Ist der angegebene Zeichensatz vom Typ Unicode (UTF-8, UTF-16 oder UTF-32), versucht 4D unter den ersten empfangenen Bytes ein BOM zu identifizieren. Wurde es gefunden, wird es aus dem Ergebnis gefiltert und 4D verwendet den hier definierten Zeichensatz anstelle des vorgegebenen.

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

Delete string

Delete string (Quelle ; Beginn ; Länge) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Quelle	String	➡	Text, in dem gelöscht wird
Beginn	Lange Ganzzahl	➡	Beginn für Löschen
Länge	Lange Ganzzahl	➡	Anzahl der zu löschenden Zeichen
Funktionsergebnis	String	⮌	Geänderter Text

Beschreibung

Die Funktion **Delete string** löscht die durch *Länge* bestimmte Anzahl der Zeichen ab *Beginn* und gibt diesen Text zurück. Der Text *Quelle* bleibt dabei unverändert.

Delete string gibt den Text aus *Quelle* zurück, wenn:

- Der Text *Quelle* leer ist.
- *Beginn* größer oder gleich der Länge von *Quelle* ist.
- *Länge* kleiner oder gleich 0 ist.

Ist *Beginn* kleiner oder gleich 0, werden die Zeichen ab Beginn des Textes gelöscht.

Ist *Beginn* plus *Länge* gleich oder länger als der Text *Quelle*, werden die Zeichen ab Beginn bis zum Ende von *Quelle* gelöscht.

Beispiel

Folgendes Beispiel zeigt die Verwendung von **Delete string**. Die Ergebnisse werden der Variablen *vtResult* zugewiesen.

```
vtResult:=Delete string("Lamborghini";6;6)  ` vtResult ergibt "Lambo"
vtResult:=Delete string("Haut";2;1)  ` vtResult ergibt "Hut"
vtResult:=Delete string(vtOtherVar;3;32000)
  ` vtResult ergibt die ersten beiden Zeichen von vtOtherVar
```

Get localized string

Get localized string (ResName) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ResName	String	➡	Name des Attributs Resname
Funktionsergebnis	String	⮌	Wert des String definiert durch ResName der aktuellen Sprache

Beschreibung

Get localized string gibt den Wert der Zeichenkette zurück, definiert durch das Attribut *ResName* für die aktuelle Sprache.
Diese Funktion funktioniert nur innerhalb einer XLIFF Architektur. Weitere Informationen dazu finden Sie im **PICTURE TO BLOB** des Handbuchs *4D Designmodus*.
Hinweis: Mit der Funktion **Get database localization** können Sie die für die Anwendung verwendete Sprache herausfinden.
In *ResName* übergeben Sie den Ressourcennamen der Zeichenkette, die in die aktuelle Zielsprache übersetzt werden soll.
Beachten Sie, dass XLIFF diakritisch ist.

Beispiel

Hier der Auszug einer .xlf Datei:

```
<file source-language="en-US" target-language="fr-FR">  <trans-unit resname="Show on disk">  <source>Show on disk</source>  <target>Montrer sur disque</target>  </trans-unit>
```

Nach Ausführen der Anweisung:

```
$value:=Get localized string("Show on disk")
```

.....wenn die aktuelle Sprache Französisch ist, enthält $value “Montrer sur disque”.

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt. Wird *ResName* nicht gefunden, gibt die Funktion einen leeren String zurück und die Variable OK wird auf 0 (Null) gesetzt.

GET TEXT KEYWORDS

GET TEXT KEYWORDS (Text ; arrKeywords {; *})

Parameter	Typ		Beschreibung
Text	Text	➡	Originaltext
arrKeywords	Array Text	⬅	Array mit Schlüsselwörtern
*	Operator	➡	Mit Stern: einmalige Wörter

Beschreibung

Der Befehl **GET TEXT KEYWORDS** zerteilt alle Parameter *Text* in einzelne Wörter und erstellt für jedes Wort einen Eintrag im Array *arrKeywords*.

4D verwendet zum Aufteilen von Text in einzelne Wörter denselben Algorithmus wie zum Erstellen von **Volltext-Index**. Er basiert auf der ICU Library. Weitere Informationen dazu finden Sie unter folgender Adresse: http://userguide.icu-project.org/boundaryanalysis.
Hinweis: Auf vielfachen Wunsch haben wir für Französisch und Italienisch eine Ausnahme eingeführt: Folgt auf den Apostroph (') ein Vokal oder der Buchstabe "h", gilt er als Worttrenner. Zum Beispiel: die Zeichenketten "L'homme" oder "l'arbre" werden geteilt in "L'"+"homme" und "l'"+"arbre".

Der verwendete Algorithmus variiert je nachdem, ob in den Datenbank-Eigenschaften die Option **Nur nicht alphanumerische Zeichen als Schlüsselwörter ansehen** markiert ist (siehe Abschnitt **Seite Datenbank/ Datenspeicherung** im Handbuch *4D Designmodus*).

Im Parameter *Text* übergeben Sie den Originaltext, der in Wörter aufgeteilt werden soll. Bei formatiertem Text werden Stil-Tags ignoriert.

Der Befehl füllt das Text Array *arrKeywords* mit den aus dem Text entnommenen Wörtern.

Übergeben Sie den optionalen Parameter *, speichert der Befehl jedes unterschiedliche Schlüsselwort nur einmal in *arrKeywords*. Dieser Parameter ist standardmäßig nicht angegeben, d.h. alle aus dem Text entnommenen Wörter werden im Array gespeichert, selbst wenn sie mehrmals vorkommen.

Dieser Befehl bietet Ihnen einen einfachen Weg, nach Datensätzen mit umfangreichem Text zu suchen, und garantiert, dass dieselben Schlüsselwörter wie in 4D verwendet werden. Nehmen wir beispielsweise den Text "10.000 Hans-Peter BC45". Wird dieser Text in die Schlüsselwörter "10.000" + "Hans" + "Peter" + "BC45" aufgeteilt, enthält das Array 4 Elemente. Dieses Array lässt sich mit dem Operator % leicht per Programmierung durchlaufen, um die Datensätze zu finden, die eins oder mehrere dieser Schlüsselwörter enthalten (siehe Beispiele).

Beispiel 1

In einem Formular mit einem Suchbereich kann der Benutzer ein oder mehrere Wörter eingeben. Bestätigt er dieses Formular, suchen wir nach Datensätzen, in denen das Datenfeld MyField mindestens eins der vom Benutzer eingegebenen Wörter enthält.

```
 // vSearch ist die Variable des Suchbereichs im Formular
GET TEXT KEYWORDS(vSearch;arrSearch;*)
 //* falls der Benutzer das gleiche Wort mehr als einmal eingibt
CREATE SET([MyTable];"Totalfound")
$n:=Size of array(arrSearch)
For($i;1;$n)
   QUERY([MyTable];[MyTable]MyField % arrSearch{$i})
   CREATE SET([MyTable];"found")
   UNION("Totalfound";"found";"Totalfound")
End for
USE SET("Totalfound")
```

Beispiel 2

Im gleichen Formular wie oben suchen wir nach Datensätzen, wo das Feld *MyField* alle vom Benutzer eingegebenen Wörter enthält.

```
 // vSearch ist die Variable des Suchbereichs im Formular
GET TEXT KEYWORDS(vSearch;arrSearch;*)
$n:=Size of array(arrSearch)
QUERY([MyTable];[MyTable]MyField >=0;*)
 // Suche initialisieren = alle Datensätze
For($i;1;$n)
   QUERY([MyTable];&;[MyTable]MyField % arrSearch{$i};*)
 // Kriterium hinzufügen
```

```
End for
QUERY([MyTable]) //search
```

Beispiel 3

Wörter in einem Text zählen:

```
GET TEXT KEYWORDS(vText;arrWords) // Alle Wörter
$n:=Size of array(arrWords)
GET TEXT KEYWORDS(vText;arrWords;*) // verschiedene Wörter
$m:=Size of array(arrWords)
ALERT("Dieser Text enthält "+String($n)+" separate Wörter unter "+String($m))
```

Insert string

Insert string (Quelle ; Einfügen ; Beginn) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Quelle	String	→	Text, in den eingefügt wird
Einfügen	String	→	Einzufügender String
Beginn	Lange Ganzzahl	→	Beginn des Einfügens
Funktionsergebnis	String	↩	Geänderter Text

Beschreibung

Die Funktion **Insert string** setzt den Text *Einfügen* in den Text *Quelle* ab dem Wert *Beginn* ein und gibt diesen Text zurück. Der Text *Quelle* bleibt dabei unverändert. Die Zeichen hinter *Beginn* werden an das Ende des Textes verschoben.

Ist der Text *Einfügen* ein leerer Text (""), gibt **Insert string** den ursprünglichen Text zurück.

Ist der Wert *Beginn* kleiner oder gleich 0, wird der Text *Einfügen* am Anfang des Textes *Quelle* eingefügt. Ist der Wert *Beginn* größer oder gleich der Länge des Textes *Quelle*, wird der Text *Einfügen* am Ende des Textes *Quelle* eingefügt.
Ist der Text *Quelle* ein leerer Text, wird nur der Text *Einfügen* zurückgegeben.

Insert string überschreibt im Gegensatz zu **Change string** nicht die Zeichen, sondern fügt neue ein.

Beispiel

Folgendes Beispiel zeigt die Verwendung von **Insert string**. Die Ergebnisse werden in der Variablen *vtResult* zugewiesen.

```
vtResult:=Insert string("Der Baum";" grüne";4)  ` vtResult ergibt "Der grüne Baum"
vtResult:=Insert string("Hut";"a";2)  ` vtResult ergibt "Haut"
vtResult:=Insert string("Schrank";"en";8)  ` vtResult ergibt "Schranken"
```

Length

Length (String) -> Funktionsergebnis

Parameter	Typ		Beschreibung
String	String	➡	Text, dessen Länge Sie berechnen wollen
Funktionsergebnis	Lange Ganzzahl	➲	Textlänge

Beschreibung

Die Funktion **Length** gibt die Anzahl Zeichen zurück, aus denen *String* besteht.

Hinweis: Wollen Sie im Unicode Modus prüfen, ob ein String Zeichen enthält, inkl. ignorierbarer Zeichen, müssen Sie den Test **If**(Length(vtAnyText)=0) und nicht **If**(vtAnyText="") verwenden. Enthält der String z.B. **Char**(1), was ein ignorierbares Zeichen ist, gibt **Length**(vtAnyText) 1 zurück, vtAnyText dagegen *Wahr*.

Beispiel

Dieses Beispiel zeigt die Verwendung von **Length**. Die Ergebnisse werden der Variablen *vlResult* zugewiesen.

```
vlResult:=Length("Stadt") ` vlResult ergibt 5
vlResult:=Length("Einwohner") ` vlResult ergibt 9
```

Lowercase

Lowercase (Quelltext {; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Quelltext	String	➡	In Kleinbuchstaben umzuwandelnder String
*	Operator	➡	Mit *: Akzente beibehalten
Funktionsergebnis	String	⮌	String in Kleinbuchstaben

Beschreibung

Die Funktion **Lowercase** gibt den in *Quelltext* übergebenen Text als Kleinbuchstaben zurück.
Mit dem optionalen Parameter * werden die in *Quelltext* enthaltenen Zeichen mit Akzent beibehalten.
Dieser Parameter ist standardmäßig nicht angegeben, d.h. die Zeichen verlieren ihren Akzent beim Umwandeln in Kleinbuchstaben.

Beispiel 1

Folgende Projektmethode schreibt den als Parameter übergebenen Text oder String groß. So ergibt Caps ("jasmin") "Jasmin".

```
` Projektmethode Caps
` Caps ( String ) -> String
` Caps ( Beliebiger Text oder String ) -> Großschreibung

$0:=Lowercase($1)
If(Length($0)>0)
   $0[[1]]:=Uppercase($0[[1]])
End if
```

Beispiel 2

Dieses Beispiel vergleicht die erhaltenen Ergebnisse danach, ob der optionale Parameter * übergeben wurde oder nicht:

```
$thestring:=Lowercase("DÉJÀ VU")  ` $thestring ist “deja vu”
$thestring:=Lowercase("DÉJÀ VU";*)  ` $thestring ist “déjà vu”
```

Match regex

Match regex (Muster ; spezString ; Start {; Pos_gefunden ; Länge_gefunden}{; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Muster	Text	→	Regulärer Ausdruck
spezString	Text	→	String, in dem die Suche ausgeführt wird
Start	Lange Ganzzahl	→	Position im String, wo die Suche beginnt
Pos_gefunden	Array Lange Ganzzahl, Variable Lange Ganzzahl	←	Position des Vorkommens
Länge_gefunden	Array Lange Ganzzahl, Variable Lange Ganzzahl	←	Länge des Vorkommens
*	Operator	→	Mit *: Nur Suche an der angegebenen Position
Funktionsergebnis	Boolean	↩	True = Suche hat 1 Vorkommen gefunden; sonst False.

Match regex (Muster ; Zeichenkette) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Muster	Text	→	regulärer Ausdruck (komplette Übereinstimmung)
Zeichenkette	Text	→	Zeichenkette, in der eine Suche ausgeführt wird
Funktionsergebnis	Boolean	↩	True = Suche hat ein Vorkommen gefunden, sonst False

Beschreibung

Die Funktion **Match regex** prüft, ob eine Zeichenkette mit den analysierenden Regeln einer Meta-Sprache, genannt „regular expression" oder „rational expression", übereinstimmt. Diese werden in der Regel mit dem Begriff regex abgekürzt.

In *Muster* übergeben Sie den regulären Ausdruck für die Suche. Das ist eine Zeichenkette, die den Ausdruck mit spezifischen Zeichen beschreibt.
In *spezString* übergeben Sie den String für die Suche nach dem regulären Ausdruck.
In *Start* geben Sie die Position an, ab der die Suche starten soll.

Sind die Parameter *Pos_gefunden* und *Länge_gefunden* Variablen, gibt die Funktion darin Position und Länge des Vorkommens zurück. Bei Arrays gibt die Funktion in Element Null der Arrays Position und Länge des Vorkommens zurück, in den darauffolgenden Elementen Position und Länge der Gruppen, die der reguläre Ausdruck gefunden hat.

Mit dem optionalen Parameter * legen Sie fest, dass die Suche nur an der in *Start* angegebenen Position ausgeführt werden soll, ohne weiter zu suchen, falls hier nichts Passendes gefunden wird.

Die Funktion gibt *True* zurück, wenn die Suche ein Vorkommen gefunden hat.

Weitere Informationen über Regex finden Sie im Internet unter:
http://en.wikipedia.org/wiki/Regular_expression

Weitere Informationen zur Syntax regulärer Ausdrücke im Parameter *Muster* finden Sie im Internet unter:
http://www.icu-project.org/userguide/regexp.html

Beispiel 1

Nach kompletter Übereinstimmung suchen (einfache Syntax):
*vfound:=*Match regex*(pattern;mytext)*

```
QUERY BY FORMULA([Employees];Match regex(".*smith.*";[Employees]name))
```

Beispiel 2

In Text nach Start suchen:
*vfound:=*Match regex*(pattern;mytext; start; pos_found; length_found)*
Beispiel zur Anzeige aller Tags in $1:

```
$start:=1
Repeat
   vfound:=Match regex("<.*>";$1;$start;pos_found;length_found)
   If(vfound)
      ALERT(Substring($1;pos_found;length_found))
      $start:=pos_found+length_found
   End if
Until(Not(vfound))
```

Beispiel 3

Suche mit Unterstützung von "capture groups":
*vfound:=*Match regex*(pattern;mytext; start; pos_found_array; length_found_array)*

```
ARRAY LONGINT(pos_found_array;0)
ARRAY LONGINT(length_found_array;0)
```

```
vfound:=Match regex("(.*)stuff(.*)";$1;1;pos_found_array;length_found_array)
If(vfound)
    $group1:=Substring($1;pos_found_array{1};length_found_array{1})
    $group2:=Substring($1;pos_found_array{2};length_found_array{2})
End if
```

Beispiel 4

Suche beschränkt auf das Muster an der angegebenen Position:
Fügen Sie am Ende einer der beiden vorigen Syntaxarten einen Stern hinzu.

```
vfound:=Match regex("a.b";"---a-b---";1;$pos_found;$length_found)
  `returns True
vfound:=Match regex("a.b";"---a-b---";1;$pos_found;$length_found;*)
  `returns False
vfound:=Match regex("a.b";"---a-b---";4;$pos_found;$length_found;*)
  `returns True
```

Hinweis: Die zurückgegebenen Positionen und Längen sind nur im Modus Unicode von Bedeutung oder wenn der Text in Bearbeitung vom 7-bit ASCII Typ ist.

Fehlerverwaltung

Tritt ein Fehler auf, erzeugt die Funktion einen Fehler, den Sie über eine mit dem Befehl **ON ERR CALL** installierte Methode abfangen können.

Num

Num (Ausdruck {; Trenner}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Ausdruck	String, Boolean, Lange Ganzzahl	→	String, der in eine Zahl umgewandelt werden soll oder Boolean, um 0 oder 1 zurückzugeben, oder numerischer Ausdruck
Trenner	String	→	Dezimaltrenner
Funktionsergebnis	Zahl	↩	Zahlenform des Parameters Ausdruck

Beschreibung

Die Funktion **Num** gibt den Zahlenwert des String, Boolean oder numerischen Ausdrucks zurück, definiert in *Ausdruck*. Mit dem optionalen Parameter *Trenner* können Sie einen spezifischen Dezimaltrenner für einen Zahlenwert in *Ausdruck* übergeben.

Ausdrücke vom Typ Text

Enthält *Ausdruck* nur Buchstaben, gibt **Num** den Wert 0 zurück. Enthält *Ausdruck* Buchstaben und Zahlen, ignoriert der Befehl die Buchstaben. So wandelt er die Zeichenkette "a1b2c3" um in die Zahl 123.

Es gibt drei reservierte Zeichen, die **Num** besonders behandelt: Der im System definierte Dezimaltrenner (wenn der Parameter *Trenner* nicht übergeben ist), der Bindestrich "-" und "*e*" *oder* "*E*". Diese Zeichen werden als numerische Formate interpretiert.

- Das Dezimaltrennzeichen, z.B. "." gilt als Trennzeichen für Dezimalwerte. Es muss innerhalb einer Zahl erscheinen. Die Funktion verwendet standardmäßig das vom Betriebssystem vorgegebenen Trennzeichen. Mit dem Parameter *Trenner* können Sie ein anderes Zeichen festlegen.
- Der Bindestrich gilt als Minuszeichen. Er muss vor der Zahl erscheinen oder nach dem "e" für einen Exponenten. Steht innerhalb einer Zahl ein Bindestrich, gibt **Num** den Wert Null (0) zurück.
  Beispiel: **Num**(123-456) ergibt 0, **Num**(-9) ergibt -9.
- Das e bzw. E gilt als Exponent eines Zahlenwertes. Es muss innerhalb einer Zahl erscheinen. Beispiel: **Num**("123e–2") ergibt 1,23.

Die Funktion verwendet standardmäßig den Dezimaltrenner, der im Betriebssystem definiert ist. Wird der zu bewertende String mit einem Dezimaltrenner dargestellt, der sich vom Trenner des Systems unterscheidet, gibt die Funktion ein falsches Ergebnis zurück.
Mit dem optionalen Parameter *Trenner* können Sie eine korrekte Bewertung erzielen. Ist dieser Parameter übergeben, berücksichtigt die Funktion nicht die Dezimaltrenner des Systems. Sie können ein oder mehrere Zeichen übergeben.

Hinweis: Der Befehl **GET SYSTEM FORMAT** ermöglicht, den aktuellen Dezimaltrenner sowie andere landesspezifische Systemparameter zu finden.

Ausdrücke vom Typ Boolean

Übergeben Sie einen Boolean Ausdruck, gibt **Num** den Wert 1 zurück, wenn der Ausdruck wahr ist; sonst den Wert Null (0).

Numerische Ausdrücke

Übergeben Sie in *Ausdruck* einen numerischen Ausdruck, gibt **Num** den Zahlenwert wie eingetragen zurück. Das ist besonders bei generischer Programmierung mit Zeigern hilfreich.

Undefinierte Ausdrücke

Wird *Ausdruck* als undefiniert gewertet, gibt **Num** 0 (Null) zurück. Das ist hilfreich, wenn als Ergebnis eines Ausdrucks (z.B. ein Objektattribut) eine Zahl erwartet wird, auch wenn sie undefiniert ist.

Beispiel 1

Folgendes Beispiel zeigt, wie **Num** bei Ausdrücken vom Typ Text arbeitet. Jede Zeile weist der Variablen *vResult* eine Zahl zu:

```
vResult:=Num("ABCD")  ` vResult ergibt 0
vResult:=Num("A1B2C3")  ` vResult ergibt 123
vResult:=Num("123")  ` vResult ergibt 123
vResult:=Num("123,4")  ` vResult ergibt 123,4
vResult:=Num("–123")  ` vResult ergibt –123
vResult:=Num("–123e2")  ` vResult ergibt –12300
```

Beispiel 2

Im Folgenden wird *[Client]Debt* mit *$1000* verglichen. **Num** gibt entweder 1 oder 0 zurück. Die Multiplikation mit 1 bzw. 0 wiederholt den Text bzw. gibt einen leeren Text zurück. *[Client]Risk* erhält als Ergebnis entweder "Gut" oder "Schlecht":

```
  ` Schuldet der Kunde unter 1000, gutes Risiko.
  ` Schuldet der Kunde über 1000, schlechtes Risiko.
[Client]Risk:=("Gut"*Num([Client]Debt<1000))+("Schlecht"*Num([Client]Debt>=1000))
```

Beispiel 3

Dieses Beispiel vergleicht die erhaltenen Ergebnisse abhängig vom "aktuellen" Trenner:

```
$thestring:="33,333.33"
$thenum:=Num($thestring)
  ` Standardmäßig entspricht $thenum 33,33333 auf einem deutschen System
$thenum:=Num($thestring;".")
  ` $thenum wird korrekt bewertet unabhängig vom System;
  ` zum Beispiel 33333,33 auf einem deutschen System
```

Position

Position (Suchtext ; Quelltext {; Start {; Längegefunden}}{; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Suchtext	String	➡	Zu suchender Text
Quelltext	String	➡	
Start	Lange Ganzzahl	➡	Stelle in String, wo die Suche beginnt
Längegefunden	Lange Ganzzahl	⬅	Länge des gefundenen String
*	Operator	➡	Mit Stern: Vergleich basiert auf Zeichen-Codes
Funktionsergebnis	Lange Ganzzahl	↩	Position des ersten Auftretens

Beschreibung

Die Funktion **Position** gibt die Position des ersten Auftretens von *Suchtext* in *Quelltext* zurück. Enthält String nicht *Suchtext*, gibt der Befehl den Wert 0 zurück.

Findet **Position** ein Vorkommen von *Suchtext*, gibt sie die Position des ersten Zeichens in *Quelltext* zurück.
Fragen Sie innerhalb eines leeren String nach einer Position, gibt **Position** den Wert Null (0) zurück.

Die Suche beginnt standardmäßig mit dem ersten Zeichen von *Quelltext*. Mit dem Parameter *Start* können Sie das erste Zeichen angeben, mit dem die Suche in *Quelltext* beginnen soll.

Der Parameter *Längegefunden* gibt die Länge der Zeichenkette zurück, die aktuell durch die Suche gefunden wird. Dieser Parameter ist zum korrekten Verwalten von Buchstaben mit einem oder mehr Zeichen notwendig, z.B. ß und ss, ae und æ, etc. Ist der optionale Parameter * übergeben, gelten diese Buchstaben als ungleich (ae # æ). In diesem Modus ist *Längegefunden* immer identisch mit der Länge von *Suchtext* (wenn ein Auftreten gefunden wurde).

Der Befehl führt standardmäßig globale Vergleiche aus, die linguistische Besonderheiten und Buchstaben, die als ein oder mehr Zeichen geschrieben werden (z.B. æ = ae) berücksichtigt. Dagegen ist er nicht diakritisch (a=A, a=á, etc.) und berücksichtigt nicht "ignorierbare" Zeichen (Unicode Spezifikation). Das sind alle Zeichen im Satz C0 Steuerung (U+0000 bis U+001F, Ascii Zeichen Steuerungssatz) mit Ausnahme der druckbaren Zeichen (U+0009 TAB, U+0010 LF, U+0011 VT, U+0012 FF und U+0013 CR)
Wollen Sie diese Funktionsweise ändern, übergeben Sie * als letzten Parameter. Dann basieren Vergleiche auf Zeichen-Codes. Sie müssen * übergeben, um:

- Sonderzeichen zu berücksichtigen, die z.B. als Begrenzer dienen: Zeilenschaltung, *Char(1)*, etc.
- Groß- und Kleinschreibung und Zeichen mit Akzenten unterschiedlich zu werten: a#A, a#à, etc. Beachten Sie, dass der Vergleich in diesem Modus keine Variation in der Schreibweise von Wörtern behandelt.

Hinweise:

- In bestimmten Fällen kann der Parameter * die Ausführung des Befehls signifikant beschleunigen.
- Der Joker (@) kann mit **Position** nicht benutzt werden. Übergeben Sie zum Beispiel in *Suchtext* "abc@", sucht die Funktion nach "abc@" und nicht nach "abc" plus beliebigen Zeichen.

Beispiel 1

Dieses Beispiel zeigt die Verwendung von **Position**. Die in den Kommentaren beschriebenen Ergebnisse werden der Variablen *vlResult* zugewiesen.

```
vlResult:=Position("ll";"Wille")  ` vlResult ergibt 3
vlResult:=Position(vtText1;vtText2)
  ` Übergibt das erste Auftreten von vtText1 in vtText2
v1Result:=Position("Tag";"Montag ist der erste Tag";1)  `gibt 4 zurück
v1Result:=Position("Tag";"Montag ist der erste Tag";8)  `gibt 22 zurück
v1Result:=Position("TAG;"Montag ist der erste Tag";1;*)  `gibt 0 zurück
v1Result:=Position("œ";"Bœuf";1;$stringlength)
  `v1Result gibt 2 und $stringlength 1 zurück
```

Beispiel 2

Im folgenden Beispiel werden mit *Längegefunden* alle Vorkommen von aegis in einem Text gefunden, egal ob es "ægis" oder "aegis" geschrieben wird.

```
$start:=1
Repeat
   v1Result:=Position("aegis";$text;$start;$stringlength)
   $start:=$start+$stringlength
Until(v1Result=0)
```

Replace string

Replace string (Quelle ; AlterString ; NeuerString {; Wieviele}{; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Quelle	String	➡	Original String
AlterString	String	➡	Zu ersetzende Zeichen
NeuerString	String	➡	String zum Ersetzen (ist String leer, werden alle Vorkommen gelöscht)
Wieviele	Lange Ganzzahl	➡	Wieviele Mal soll ersetzt werden Ohne Angabe werden alle Vorkommen ersetzt
*	Operator	➡	Mit Stern: Bewertung basiert auf Zeichen-Codes
Funktionsergebnis	String	↩	Geänderter Text

Beschreibung

Die Funktion **Replace string** ersetzt im Text *Quelle* den Text *AlterString* durch den Text *NeuerString*. Ist *NeuerString* ein leerer String (""), ersetzt **Replace string** alle Vorkommen des Textes *AlterString* in *Quelle*. Ist der Parameter *Wieviele* angegeben, ersetzt der Befehl nur die angegebene Anzahl der Vorkommen von *AlterString*, beginnend mit dem ersten Zeichen von *Quelle*. Geben Sie diesen Parameter nicht an, ersetzt **Replace string** alle Vorkommen von *AlterString*.

Ist *AlterString* ein leerer String (""), bleibt *Quelle* unverändert.

Der Befehl führt standardmäßig globale Vergleiche aus, die linguistische Besonderheiten und Buchstaben, die als ein oder mehr Zeichen geschrieben werden (z.B. æ = ae) berücksichtigt. Dagegen ist er nicht diakritisch (a=A, a=á, etc.) und berücksichtigt nicht "ignorierbare" Zeichen, z.B. Zeichen mit Code < 9 (Unicode Spezifikation).

Wollen Sie diese Funktionsweise ändern, übergeben Sie * als letzten Parameter. Dann basieren Vergleiche auf Zeichen-Codes. Sie müssen * übergeben, um:

- Sonderzeichen zu berücksichten, die z.B. als Begrenzer dienen: **Char**(1), o.ä.
- Groß- und Kleinschreibung und Zeichen mit Akzenten unterschiedlich zu werten: a#A, a#à, etc. Beachten Sie, dass der Vergleich in diesem Modus keine Variation in der Schreibweise von Wörtern verwaltet.

Hinweis: Ab 4D v15 R3 gibt es eine Optimierung: Über einen neuen internen Algorithmus läuft die Ausführung von [#cmd id="233"/] signifikant schneller ab, wenn Sie einen String durch einen String mit anderer Länge ersetzen, unabhängig von der verwendeten Syntax.

Beispiel 1

Folgendes Beispiel zeigt die Anwendung von **Replace string**. Die Ergebnisse werden der Variablen *vtResult* zugewiesen.

```
vtResult:=Replace string("Bindfaden";" ndf";"ldl")  ` vtResult ergibt "Bildladen"
vtResult:=Replace string("Haut";"a";"")  ` vtResult ergibt "Hut"
vtResult:=Replace string(vtOtherVar;Char(Tab);",";*)  ` Ersetzt alle Tabulatoren in vtOtherVar durch Kommas
```

Beispiel 2

Folgendes Beispiel löscht Zeilenumbrüche und Tabulatoren aus dem Text in *vtResult*:

```
vtResult:=Replace string(Replace string(vtResult;Char(Carriage return);"";*);Char(Tab);"";*)
```

Beispiel 3

Dieses Beispiel zeigt die Verwendung des Parameters * bei Berücksichtigung diakritischer Zeichen

```
vtResult:=Replace string("Crème brûlée";"Brulee";"caramel")  `Ergebnis ist "Crème caramel"
vtResult:=Replace string("Crème brûlée";"Brulee";"caramel";*)  `Ergebnis ist "Crème brûlée"
```

Split string

Split string (StringSplitten ; Trenner {; Optionen}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
StringSplitten	Text	➡	Wert des Strings zum Splitten
Trenner	Text	➡	String, bei dem stringToSplit splittet. Bei Leerstring ("") ist jedes Zeichen von stringToSplit ein Unterstring
Optionen	Lange Ganzzahl	➡	Option(en) zu Leerstrings und Leerzeichen
Funktionsergebnis	Collection	↩	Collection mit Unterstrings

Beschreibung

Die Funktion **Split string** gibt eine Collection mit Strings zurück, erstellt durch Aufteilen von *StringSplitten* in Unterstrings an den Grenzen, die im Parameter *Separator* definiert sind. Die Unterstrings in der zurückgegebenen Collection enthalten nicht den *Trenner* selbst.

Wird in *StringSplitten* kein Trenner gefunden, gibt **Split string** eine Collection mit einem einzigen Element *StringSplitten* zurück. Haben Sie einen Leerstring in *Trenner* übergeben, gibt **Split string** eine Collection jedes Zeichens von *StringSplitten* zurück.

Im Parameter *Option* können Sie eine Konstante unter dem Thema **Strings** übergeben oder die Konstanten kombinieren:

Konstante	Typ	Wert	Kommentar
sk ignore empty strings	Lange Ganzzahl	1	Leere Strings aus der resultierenden Collection entfernen (sie werden ignoriert)
sk trim spaces	Lange Ganzzahl	2	Leerzeichen am Anfang und Ende von Unterstrings kürzen

Beispiel 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"]
```

Beispiel 2

Der Parameter *Trenner* kann ein String aus mehreren Zeichen sein:

```
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 (Ausdruck {; Format {; plusZeit}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Ausdruck	Ausdruck	➡	Ausdruck, der in Text umgewandelt werden soll (vom Typ numerisch, Datum, Zeit, String, Text, Boolean oder undefiniert)
Format	String, Lange Ganzzahl	➡	Anzeigeformat
plusZeit	Zeit	➡	hinzuzufügende Zeit, wenn Ausdruck ein Datum ist
Funktionsergebnis	String	⮌	Zeichenkette des Ausdrucks

Beschreibung

Der Befehl **String** gibt eine Zeichenkette zurück für die in *Ausdruck* übergebenen Ausdrücke. Sie können vom Typ numerisch, Datum, Zeit, String oder Boolean sein.

Format ist optional. Geben Sie diesen Parameter nicht an, wird das entsprechende Standardformat gewählt. Mit *Format* können Sie ein bestimmtes Format vorgeben.

Der optionale Parameter *plusZeit* fügt in einem kombinierten Format an ein Datum eine Zeit hinzu. Er ist nur verwendbar, wenn der Parameter *Ausdruck* ein Datum ist (siehe unten).

Numerischer Ausdruck

Ist *Ausdruck* vom Typ numerisch (Zahl, Ganzzahl, Lange Ganzzahl), können Sie ein optionales Textformat übergeben. Hierzu einige Beispiele:

Beispiel	Ergebnis	
String(2^15)	32768	Standardformat
String(2^15;"###.##0 Einwohner")	32.768 Einwohner	
String(1/3;"##0,00000")	0,33333	
String(1/3)	0,33333333333333330000	Standardformat (*)
String(Arctan(1)*4)	3,141592653589790000	Standardformat (*)
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	Immer "." als Dezimaltrenner
String(Num(1=1);"True;;False")	Wahr	
String(Num(1=2);"True;;False")	Falsch	
String(Log(-1))	""	Undefinierte Nummer
String(1/0)	"INF"	Positive infinite Nummer
String(-1/0)	"-INF"	Negative infinite Nummer

(*) Ab 4D v14 R3 basiert der Algorithmus zum Konvertieren von Daten des Typs Zahl auf 13 signifikanten Stellen (es waren bisher 15 Stellen).

Sie definieren das Format genauso wie für ein numerisches Datenfeld in einem Formular. Weitere Informationen dazu finden Sie im Abschnitt **Zahlenformate** des Handbuchs *4D Designmodus*. In *Format* können Sie auch eigene Stilvorlagen übergeben. Sie müssen dann dem Namen das Zeichen "|" voranstellen.

Hinweis: Die Funktion **String** ist nicht kompatibel mit Feldern vom Typ "Ganzzahl 64 bits" im kompilierten Modus.

Ausdruck vom Typ Datum

Ist *Ausdruck* vom Typ Datum, erhalten Sie als Zeichenkette das vom Betriebssystem definierte Standardformat. Im Parameter *Format* können Sie eine der unten beschriebenen Konstanten unter dem Thema **Datum Anzeigeformate** übergeben. In diesem Fall können Sie auch im Parameter *plusZeit* eine Zeit übergeben. Dieser Parameter ermöglicht, Datum und Zeit zu kombinieren, um Zeitstempel in Übereinstimmung mit aktuellen Standards zu erstellen (Konstanten ISO Date GMT und Date RFC 1123). Diese Formate sind besonders hilfreich im Kontext von XML und Web Processing. Der Parameter *plusZeit* ist nur verwendbar, wenn der Parameter *Ausdruck* ein Datum ist.

Konstante	Typ	Wert	Kommentar
Blank if null date	Lange Ganzzahl	100	"" statt 0
Date RFC 1123	Lange Ganzzahl	10	Fri, 10 Sep 2010 13:07:20 GMT
Internal date abbreviated	Lange Ganzzahl	6	29. Dez. 2006
Internal date long	Lange Ganzzahl	5	29. Dezember 2006
Internal date short	Lange Ganzzahl	7	29.12.2006
Internal date short special	Lange Ganzzahl	4	29.12.06 (aber 29.12.1896 oder 29.12.2096)
ISO Date	Lange Ganzzahl	8	2006-29-12T00:00:00 (überholt)
ISO Date GMT	Lange Ganzzahl	9	2010-09-13T16:11:53Z
System date abbreviated	Lange Ganzzahl	2	So, 29. Dez. 2006
System date long	Lange Ganzzahl	3	Sonntag, 29. Dezember 2006
System date short	Lange Ganzzahl	1	29.12.2006

Hinweis: Die Formate können je nach Systemeinstellungen variieren.

Hier ein paar Beispiele für einfache Formate. Sie gehen vom aktuellen Datum 5.3.09 aus:

```
$vsResult:=String(Current date)  //$vsResult ergibt "05.03.09"
$vsResult:=String(Current date;Internal date long)  // $vsResult ergibt "5. März 2009"
$vsResult:=String(Current date;ISO Date GMT)  // $vsResult ergibt "2009-03-04T23:00:00" in Deutschland
```

Hinweise für kombinierte Datum/Zeitformate:

- Das Format ISO Date GMT entspricht dem Standard ISO8601. Dieses Format enthält Datum und Zeit unter Berücksichtigung der Zeitzone (GMT).

  ```
  $mydate:=String(Current date;ISO Date GMT;Current time)  // gibt z.B. 2010-09-13T16:11:53Z zurück
  ```

  Beachten Sie, dass das Zeichen "Z" am Ende das GMT Format angibt.
  Übergeben Sie nur ein Datum, gibt der Befehl das Datum um Mitternacht (lokale Zeit) in GTM Zeit zurück. Das kann je nach der lokalen Zeitzone bewirken, dass das Datum vor- oder zurückbewegt wird:

  ```
  $mydate:=String(!13/09/2010!;ISO Date GMT)  // gibt 2010-09-12T22:00:00Z in Deutschland zurück
  ```

- Das Format ISO Date ist ähnlich zum Format ISO Date GMT, mit dem Unterschied, dass es Datum und Zeit ohne Berücksichtigung der Zeitzone ausdrückt. Beachten Sie, dass dieses Format nicht dem Standard ISO8601 entspricht. Es sollte nur für ganz bestimmte Zwecke verwendet werden.

  ```
  $mydate:=String(!13/09/2010!;ISO Date)  // gibt   2010-09-13T00:00:00 zurück unabhängig von der Zeitzone.
  $mydate:=String(Current date;ISO Date;Current time)  // gibt 2010-09-13T18:11:53 zurück
  ```

- Das Format Date RFC 1123 formatiert eine Datum-/Zeitkombination gemäß dem von RFC 822 und 1123 definierten Standard. Sie benötigen dieses Format zum Beispiel, um für Cookies in einem HTTP Header das Ablaufdatum zu setzen.

  ```
  $mydate:=String(Current date;Date RFC 1123;Current time)  // gibt z.B. Fr, 10 Sep 2010 13:07:20 GMT zurück
  ```

  Die dargestellte Zeit berücksichtigt die Zeitzone (GMT Zone). Übergeben Sie nur ein Datum, gibt der Befehl das Datum um Mitternacht (lokale Zeit) ausgedrückt in GMT Zeit zurück. Das kann je nach der lokalen Zeitzone bewirken, dass das Datum vor- oder zurückbewegt wird:

  ```
  $mydate:=String(Current date;Date RFC 1123)  // gibt Do, 09 Sep 2010 22:00:00 GMT
  ```

Ausdruck vom Typ Zeit
Ist *Ausdruck* vom Typ Zeit, erhalten Sie als Zeichenkette das Standardformat HH:MM:SS. Im Parameter *Format* können Sie eine Konstante unter dem Thema **Zeit Anzeigeformate** übergeben:

Konstante	Typ	Wert	Kommentar
Blank if null time	Lange Ganzzahl	100	"" statt 0
HH MM	Lange Ganzzahl	2	
HH MM AM PM	Lange Ganzzahl	5	
HH MM SS	Lange Ganzzahl	1	
Hour min	Lange Ganzzahl	4	1 Stunde 2 Minuten
Hour min sec	Lange Ganzzahl	3	1 Stunde 2 Minuten 3 Sekunden
ISO time	Lange Ganzzahl	8	
Min sec	Lange Ganzzahl	7	62 Minuten 3 Sekunden
MM SS	Lange Ganzzahl	6	
System time long	Lange Ganzzahl	11	1:02:03 AM HNEC (nur Mac OS)
System time long abbreviated	Lange Ganzzahl	10	1•02•03 AM (nur Mac OS)
System time short	Lange Ganzzahl	9	

Hinweise:

- Das Format ISO Time entspricht dem Standard ISO8601 und enthält theoretisch ein Datum und eine Zeit.Da dieses Format keine Kombination Datum/Zeit unterstützt, wird der Datumsteil mit Nullen gefüllt. Dieses Format stellt die lokale Zeit dar.
- Die Konstante Blank if null muss zum Format hinzugefügt werden; sie gibt an, dass 4D im Falle von einem Nullwert anstelle von Nullen einen leeren String zurückgibt.

Diese Beispiele gehen von der aktuellen Zeit 5:30 PM und 45 Sekunden aus:

```
$vsResult:=String(Current time)  ` $vsResult ergibt "17:30:45"
$vsResult:=String(Current time;Hour Min Sec)  ` $vsResult ergibt "17 Stunden 30 Minuten 45 Sekunden"
```

Ausdruck vom Typ String
Ist *Ausdruck* vom Typ String oder Text, gibt die Funktion den Wert wie eingetragen zurück. Das ist besonders bei generischer Programmierung mit Zeigern hilfreich. In diesem Fall wird der Parameter *Format* ignoriert.

Ausdruck vom Typ Boolean
Ist *Ausdruck* vom Typ Boolean, gibt die Funktion den String "True" oder "False" in der Sprache der Anwendung zurück, also "Wahr" oder "Falsch" in der deutschen Version. In diesem Fall wird der Parameter *Format* ignoriert.

Undefinierte Ausdrücke
Wird *Ausdruck* als undefiniert gewertet, gibt **String** einen leeren String zurück. Das ist hilfreich, wenn als Ergebnis eines

Ausdrucks (z.B. ein Objektattribut) ein String erwartet wird, auch wenn er undefiniert ist.

Substring

Substring (Quelle ; Von {; Länge}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Quelle	String	➡	Anfangstext
Von	Lange Ganzzahl	➡	Anfangstext
Länge	Lange Ganzzahl	➡	Zahl der gewünschten Zeichen
Funktionsergebnis	String	⮌	Teilstring des Ursprungstextes

Beschreibung

Die Funktion **Substring** gibt eine Zeichenkette aus den Zeichen des Textes *Quelle* ab dem Zeichen *Von* mit der Länge *Länge* zurück.

Von zeigt das erste zurückzugebene Zeichen der Zeichenkette an, Länge definiert die Anzahl der zurückzugebenden Zeichen.

Ist *Von* plus *Länge* größer als die Anzahl Zeichen der Zeichenkette oder ist *Länge* nicht angegeben, gibt **Substring** die Zeichen ab dem Wert *Von* zurück. Ist *Von* größer als die Anzahl Zeichen der Zeichenkette, gibt **Substring** einen leeren String zurück.

Warnung: Bei Verwendung in Text mit Mehrfachstil müssen Sie unter Windows alle Zeichen für Zeilenende in Form von ('\r\n') in einfache Zeichen ('\r') umwandeln, damit die Bearbeitung gültig bleibt. Das ist notwendig, da 4D die Zeilenenden vereinheitlicht, um die Plattformkompatibilität für Texte sicherzustellen. Weitere Information dazu finden Sie im Abschnitt **Automatische Vereinheitlichung von Zeilenenden**.

Beispiel 1

Dieses Beispiel zeigt die Verwendung von **Substring**. Die Ergebnisse werden der Variablen *vsResult* zugewiesen.

```
vsResult:=Substring("08/04/62";4;2)  ` vsResult ergibt "04"
vsResult:=Substring("Hilfestellung";1;5)  ` vsResult ergibt "Hilfe"
vsResult:=Substring(var;2)  ` vsResult ergibt alle Zeichen außer dem ersten
```

Beispiel 2

Folgende Projektmethode hängt die im Text gefundenen Abschnitte (als erster Parameter übergeben) in einer Tabelle vom Typ alpha oder Text an (der als zweiter Parameter übergebene Zeiger):

```
  ` EXTRACT PARAGRAPHS
  ` EXTRACT PARAGRAPHS (Text ; Zeiger )
  ` EXTRACT PARAGRAPHS (Zu übertragender Text; -> Array der ¶-Zeichen )

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 (Quelltext {; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Quelltext	String	→	In Großbuchstaben umzuwandelnder String
*	Operator	→	Mit *: Akzente beibehalten
Funktionsergebnis	String	←	Text in Großbuchstaben

Beschreibung

Die Funktion **Uppercase** gibt *Quelltext* in Großbuchstaben zurück.

Mit dem optionalen Parameter * legen Sie fest, dass alle Zeichen mit Akzent in *Quelltext* als Großbuchstaben mit Akzent zurückgegeben werden müssen.
Dieser Parameter ist standardmäßig nicht angegeben, d.h. die Zeichen verlieren ihren Akzent beim Umwandeln in Großbuchstaben.

Beispiel 1

Dieses Beispiel vergleicht die erhaltenen Ergebnisse danach, ob der optionale Parameter * übergeben wurde oder nicht:

```
$thestring:=Uppercase("hélène") ` $thestring is "HELENE"
$thestring:=Uppercase("hélène";*) ` $thestring is "HÉLÈNE"
```

Beispiel 2

Siehe Beispiel für **Lowercase**.

_o_Convert case

_o_Convert case
Dieser Befehl benötigt keine Parameter

Beschreibung

Dieser Befehl ist veraltet und darf nicht mehr verwendet werden..

_o_ISO to Mac

_o_ISO to Mac (Text) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Text	String	➡	Text, dargestellt im Standard Web Zeichensatz
Funktionsergebnis	String	⮌	Text, dargestellt im Mac OS ASCII Zeichensatz

Hinweis zur Kompatibilität

Diese Funktion führt im Unicode Modus nichts aus, d.h. der String *Text* wird ohne Änderung zurückgegeben. Die Funktion ist seit 4D Version 11 überholt und sollte nicht mehr verwendet werden. Wir empfehlen, Zeichenketten mit den Routinen **CONVERT FROM TEXT** oder **Convert to text** zu konvertieren.

_o_Mac to ISO

_o_Mac to ISO (Text) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Text	String	➡	Text, dargestellt mit dem MacOS ASCII Zeichensatz
Funktionsergebnis	String	➲	Text, dargestellt mit dem Standard Web Zeichensatz

Hinweis zur Kompatibilität

Die Funktion ist ab 4D Version 11 überholt und sollte nicht mehr verwendet werden. Sie führt nichts aus, d.h. der String *Text* wird ohne Änderung zurückgegeben. Wir empfehlen, Zeichenketten mit den Routinen **CONVERT FROM TEXT** oder **Convert to text** zu konvertieren.

_o_Mac to Win

_o_Mac to Win (Text) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Text	String	➡	Text, dargestellt mit dem Mac OS ASCII Code
Funktionsergebnis	String	⮌	Text, dargestellt mit dem Windows ANSI Code

Hinweis zur Kompatibilität

Die Funktion ist ab 4D Version 11 überholt und sollte nicht mehr verwendet werden. Sie führt nichts aus, d.h. der String *Text* wird ohne Änderung zurückgegeben. Wir empfehlen, Zeichenketten mit den Routinen **Convert to text** oder **CONVERT FROM TEXT** zu konvertieren.

_o_Win to Mac

_o_Win to Mac (Text) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Text	String	➡	Text, dargestellt mit Windows ANSI Zeichensatz
Funktionsergebnis	String	⮌	Text, dargestellt mit Macintosh ASCII Zeichensatz

Hinweis zur Kompatibilität

Im Unicode Modus führt diese Funktion nichts aus, d.h. der String *Text* wird ohne Änderung zurückgegeben. Sie ist seit 4D Version 11 veraltet und sollte nicht mehr verwendet werden. Wir empfehlen, Zeichenketten mit den Routinen **Convert to text** oder **CONVERT FROM TEXT** zu konvertieren.

Strukturzugriff

- Einführung in Strukturzugriff
- 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

Einführung in Strukturzugriff

Die Befehle und Funktionen dieses Kapitels geben eine Beschreibung der Datenbankstruktur zurück. Damit erhalten Sie die Anzahl der Tabellen, die Anzahl der Felder in jeder Tabelle, die Namen der Tabellen und Felder sowie Typ und Eigenschaften jedes Feldes. Über Hilfsbefehle können Sie fehlende Tabellen ausfindig machen und erneut generieren, und so "Phantom" Daten wieder zugänglich machen.

Die Festlegungen der Datenbankstruktur sind sehr hilfreich, wenn Sie Gruppen von Projektmethoden und Formularen entwickeln und einsetzen, die dann in verschiedene Datenbanken kopiert werden können.

Mit der Möglichkeit, die Datenbankstruktur zu lesen, können Sie übertragbaren Code entwickeln und einsetzen.

Hinweis: Sie können 4D Felder und Tabellen über den in 4D integrierten SQL Kernel per Programmierung erstellen und ändern, z.B. mit **CREATE TABLE** oder **ALTER TABLE**. Weitere Informationen dazu finden Sie im Handbuch **4D - SQL Reference**.

Tabellen und Felder zählen

Sie können Tabellen und Datenfelder löschen. Diese Möglichkeit müssen Sie in den Algorithmen zum Zählen von Tabellen und Datenfeldern berücksichtigen. Sie müssen Algorithmen durch Kombinieren der Funktionen **Get last table number** und **Get last field number**, sowie **Is table number valid** und **Is field number valid** verwenden. Hier ein Beispiel für diese Art von Algorithmus:

```
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))
            ...\\Das Feld existiert und ist gültig
         End if
      End for
   End if
End for
```

CREATE INDEX

CREATE INDEX (Tabellenname ; FelderArray ; IndexTyp ; IndexName {; *})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle für Erstellen des Index
FelderArray	Array Zeiger	➡	Zeiger auf Feld(er) zur Indizierung
IndexTyp	Lange Ganzzahl	➡	Art des anzulegenden Index: -1 = Keywords 0 = Standard,1 = Standard BTree, 3 = Cluster BTree
IndexName	Text	➡	Name des anzulegenden Index
*	Operator	➡	Mit * = Asynchrone Indizierung

Beschreibung

Der Befehl **CREATE INDEX** erstellt zwei Indexarten:

- Einen Standard Index auf ein oder mehrere Datenfelder (zusammengesetzter Index)
- Index nach Schlüsselwörtern auf ein Feld

Der Index für die Tabelle wird in einem oder mehreren Feldern, definiert im Array *FelderArray* erstellt. Das Array enthält eine einzelne Zeile für einen einfachen Index und zwei oder mehr Zeilen für einen zusammengesetzten Index - mit Ausnahme des Index mit Schlüsselwörtern. Bei zusammengesetzten Indizes ist die Reihenfolge der Felder im Array beim Einrichten des Index von Bedeutung.

Im Parameter *IndexTyp* definieren Sie den Indextyp. Sie können eine der nachfolgenden Konstanten unter dem Thema **Indextyp** übergeben:

Konstante	Typ	Wert	Kommentar
Cluster BTree index	Lange Ganzzahl	3	Index vom Typ B-Baum mit Clustern. Dieser Typ ist optimal für Indizes mit wenigen Schlagwörtern, z.B. wenn dieselben Werte in den Daten sich häufig wiederholen.
Default index type	Lange Ganzzahl	0	4D definiert den Indextyp (ausgenommen Index nach Schlüsselwörtern), der gemäß dem Feldinhalt am besten passt.
Keywords index	Lange Ganzzahl	-1	Volltext-Index, der die Indizierung des Feldinhalts Wort für Wort ermöglicht. Dieser Indextyp lässt sich nur für Datenfelder vom Typ Text, alphanumerisch oder Bild verwenden. **Achtung:** Volltext-Indizes können nicht zusammengesetzt sein.
Standard BTree index	Lange Ganzzahl	1	Index vom Typ Standard B-Baum. Dieser vielfältige Indextyp wird in den bisherigen Versionen von 4D verwendet.

Hinweis: Ein Index vom Typ B-Tree, der einem Datenfeld vom Typ Text zugeordnet ist, speichert maximal die ersten 1024 Zeichen des Datenfeldes. Deshalb funktionieren in diesem Kontext Suchläufe nach Strings mit mehr als1024 Zeichen nicht.

Im Parameter *IndexName* übergeben Sie den Namen des zu erstellenden Index. Sie müssen einen Namen übergeben, wenn dem gleichen Feld mehrere verschiedene Indizes zugewiesen werden können und wenn Sie diese über den Befehl **DELETE INDEX** einzeln löschen möchten. Gibt es den in *IndexName* bezeichneten Index bereits, führt der Befehl nichts aus.

Mit dem optionalen Parameter * können Sie die Indizierung asynchron ausführen. In diesem Modus wird die Originalmethode nach Aufrufen aus dem Befehl weiter ausgeführt, unabhängig ob die Indizierung abgeschlossen ist oder noch läuft.

Stößt **CREATE INDEX** auf gesperrte Datensätze, werden sie nicht indiziert. Der Befehl wartet darauf, dass sie entsperrt werden. Tritt während der Ausführung des Befehls ein Problem auf, wie z.B. nicht-indiziertes Feld, wird ein Fehler generiert. Sie können ihn mit einer Fehlerverwaltungsmethode abfangen.

Beispiel 1

Zwei Standard-Indizes auf die Felder "Last Name" und "Telephone" in der Tabelle [Customers] erstellen:

```
ARRAY POINTER(fieldPtrArr;1)
fieldPtrArr{1}:=->[Customers]LastName
CREATE INDEX([Customers];fieldPtrArr;Standard BTree Index;"CustLNameIdx")
fieldPtrArr{1}:=->[Customers]Telephone
CREATE INDEX([Customers];fieldPtrArr;Standard BTree Index;"CustTelIdx")
```

Beispiel 2

Einen Index nach Schlüsselwörtern auf das Feld "Observations" in der Tabelle [Customers] erstellen:

```
ARRAY POINTER(fieldPtrArr;1)
fieldPtrArr{1}:=->[Customers]Observations
CREATE INDEX([Customers];fieldPtrArr;Keywords Index;"CustObsIdx")
```

Beispiel 3

Einen zusammengesetzten Index auf die Felder "City" und "Zipcode" der Tabelle [Customers] erstellen:

```
ARRAY POINTER(fieldPtrArr;2)
fieldPtrArr{1}:=->[Customers]City
fieldPtrArr{2}:=->[Customers]Zipcode
CREATE INDEX([Customers];fieldPtrArr;Standard BTree Index;"CityZip")
```

DELETE INDEX

DELETE INDEX (FeldPtr | IndexName {; *})

Parameter	Typ		Beschreibung
FeldPtr | IndexName	Zeiger, String	➡	Zeiger auf Feld, dessen Indizes gelöscht werden sollen, oder Name des zu löschenden Index
*	Operator	➡	Mit * = asynchrone Ausführung

Beschreibung

Der Befehl **DELETE INDEX** löscht einen oder mehrere vorhandene Indizes aus der Datenbank.
Sie können entweder einen Zeiger auf ein Feld oder den Namen eines Index im Parameter übergeben:

- Übergeben Sie in *FeldPtr* einen Zeiger, werden alle dem Feld zugewiesene Indizes gelöscht. Das können Standardindizes oder Indizes nach Schlüsselwörtern sein. Ist das Feld dagegen in einem oder mehreren zusammengesetzten Indizes enthalten, werden diese nicht gelöscht (Sie müssen einen Indexnamen übergeben).
- Übergeben Sie in *IndexName* den Namen eines Index, wird nur der angegebene Index gelöscht. Das können Indizes nach Schlüsselwörtern, Standardindizes oder zusammengesetzte Indizes sein.

Mit dem optionalen Parameter * können Sie das Entfernen der Indizierung asynchron ausführen. In diesem Modus wird die Originalmethode nach Aufrufen aus dem Befehl weiter ausgeführt, unabhängig ob das Löschen des Index abgeschlossen ist oder noch läuft.

Gibt es keinen Index, der zu *FeldPtr* oder *IndexName* passt, führt der Befehl nichts aus.

Beispiel

Dieses Beispiel zeigt die beiden Syntaxarten des Befehls:

```
  `Alle Indizes zum Feld "LastName" löschen
DELETE INDEX(->[Customers]LastName)
  `Index mit Namen “CityZip” löschen
DELETE INDEX("CityZip")
```

EXPORT STRUCTURE

EXPORT STRUCTURE (xmlStructure)

Parameter	Typ		Beschreibung
xmlStructure	Textvariable	⇐	Export der XML Definition der 4D Datenbankstruktur

Beschreibung

Der Befehl **EXPORT STRUCTURE** exportiert in *xmlStruktur* die aktuelle Struktur der 4D Datenbank im XML Format. Er verwendet dieselben Mechanismen wie der Menübefehl **Exportieren > Strukturdefinition in XML Datei...** im Designmodus (siehe **Strukturdefinitionen exportieren und importieren**).

Im Parameter *xmlStruktur* übergeben Sie eine Textvariable zum Füllen mit der Strukturdefinition. Die exportierte Definition enthält Tabellen, Felder, Indizes und Verknüpfungen zusammen mit ihren Attributen und verschiedenen Merkmalen, die für eine komplette Definition der Struktur erforderlich sind. Unsichtbare Elemente sind enthalten, gelöschte Elemente werden dagegen nicht exportiert.

Der interne Aufbau von 4D Strukturdefinitionen wird durch DTD Dateien dokumentiert — die auch zum Prüfen von XML Dateien dienen. Die von 4D verwendeten DTD Dateien liegen im **DTD** Ordner, der neben der 4D Applikation liegt. Die Dateien **base_core.dtd** und **common.dtd** dienen zur Definition der Struktur. Sie enthalten auch weitere Informationen und Kommentare zum Definieren der 4D Struktur.

Beispiel

Die Datenbankstruktur in ein Textdokument exportieren:

```
C_TEXT($vTStruc)
EXPORT STRUCTURE($vTStruc)
TEXT TO DOCUMENT("myStructure.xml";$vTStruc)
```

Field

Field (TabelleNum ; FeldNum) -> Funktionsergebnis

Parameter	Typ		Beschreibung
TabelleNum	Lange Ganzzahl	→	Tabellennummer auf Feld
FeldNum	Lange Ganzzahl	→	Feldnummer bei Tabellennummer
Funktionsergebnis	Zeiger	↩	Feld Zeiger

Field (FeldPtr) -> FeldNum

Parameter	Typ		Beschreibung
FeldPtr	Zeiger	→	Zeiger auf Feld
FeldNum	Lange Ganzzahl	↩	Feldnummer

Beschreibung

Die Funktion **Field** hat zwei Formen:

- Übergeben Sie in *TabelleNum* eine Tabellennummer und in *FeldNum* eine Feldnummer, gibt **Field** einen Zeiger auf das Feld zurück.
- Übergeben Sie in *FeldPtr* einen Zeiger auf das Feld, gibt **Field** die Feldnummer des Feldes zurück.

Beispiel 1

Folgendes Beispiel setzt die Variable *FeldPtr* in einen Zeiger auf das zweite Feld in der dritten Tabelle:

```
FeldPtr:=Field(3;2)
```

Beispiel 2

Übergeben Sie *FeldPtr* (einen Zeiger auf das zweite Feld der Tabelle) in **Field**, wird die Zahl zwei zurückgegeben. Folgende Zeile setzt *FeldNum* auf 2:

```
FeldNum:=Field(FeldPtr)
```

Beispiel 3

Folgendes Beispiel setzt die Variable *FeldNum* auf die Feldnummer von [Tabelle3]Feld2:

```
FeldNum:=Field(->[Tabelle3]Feld2)
```

Field name

Field name (FeldPtr | TabelleNum {; FeldNum}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
FeldPtr | TabelleNum	Zeiger, Lange Ganzzahl	→	Zeiger auf Feld oder Tabellennummer
FeldNum	Lange Ganzzahl	→	Feldnummer bei übergebener Tabellennummer
Funktionsergebnis	String	↩	Name des Feldes

Beschreibung

Die Funktion **Field name** gibt den Namen des Feldes mit dem Zeiger *FeldPtr* oder der Tabellen- und Feldnummer *TabelleNum* und *FeldNum* zurück.

Beispiel 1

Dieses Beispiel setzt das zweite Element des Array *FeldArray*{1} auf den Namen des zweiten Feldes in der ersten Tabelle. *FeldArray* ist ein zweidimensionales Array:

```
FeldArray{1}{2}:=Field name(1;2)
```

Beispiel 2

Dieses Beispiel setzt das zweite Element des Array *FeldArray*{1} auf den Namen des Feldes *[MeineTabelle]MeinFeld*. *FeldArray* ist ein zweidimensionales Array:

```
FeldArray{1}{2}:=Field name(->[MeineTabelle]MeinFeld)
```

Beispiel 3

Dieses Beispiel zeigt eine Meldung. Diese Methode übergibt einen Zeiger auf ein Feld:

```
ALERT("Die Kennummer für das Feld "+Field name($1)+" in der Tabelle "+Table name(Table($1))+" muss mehr als fünf Zeichen haben.")
```

Get external data path

Get external data path (Feld) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Feld	Text, BLOB, Bild	➡	Feld, dessen Speicherort Sie erhalten wollen
Funktionsergebnis	Text	⮌	Kompletter Pfadname der externen Speicherdatei

Beschreibung

Die Funktion **Get external data path** gibt für den aktuellen Datensatz den kompletten Pfadnamen der externen Speicherdatei für die Daten des Feldes zurück, das im Parameter *Feld* übergeben ist. Das müssen Felder vom Typ Text, BLOB oder Bild sein. Die Funktion gibt den Pfadnamen der Speicherdatei auch dann zurück, wenn die Datei nicht existiert oder nicht zugänglich ist.

Über diese Funktion können Sie insbesondere die externe Datei erneut kopieren.

Hinweis: Weitere Informationen zum externen Speichern finden Sie im Abschnitt **Daten extern speichern** des Handbuchs *4D Designmodus*.

Diese Funktion gibt in folgenden Fällen einen leeren String zurück:

- Das Feld wird nicht außerhalb der Datendatei gespeichert.
- Das Feld hat einen Nullwert, d.h. die externe Datei wurde nicht angelegt.
- Der Befehl wird von einem remote 4D ausgeführt.

GET FIELD ENTRY PROPERTIES

GET FIELD ENTRY PROPERTIES (FeldPtr | TabelleNum {; FeldNum}; Liste ; Zwingend ; NichtEingebbar ; NichtÄnderbar)

Parameter	Typ		Beschreibung
FeldPtr | TabelleNum	Zeiger, Lange Ganzzahl	➡	Zeiger auf Feld oder Tabellennummer
FeldNum	Lange Ganzzahl	➡	Feldnummer, wenn Tabellenummer als erster Parameter übergeben wird
Liste	String	⬅	Name der zugeordneten Auswahlliste oder leerer String
Zwingend	Boolean	⬅	Wahr = Zwingend, Falsch = Optional
NichtEingebbar	Boolean	⬅	Wahr = Nicht eingebbar, Falsch = Eingebbar
NichtÄnderbar	Boolean	⬅	Wahr = Nicht änderbar, Falsch = Änderbar

Beschreibung

Der Befehl **GET FIELD ENTRY PROPERTIES** gibt die Eigenschaften bei der Dateneingabe für das Datenfeld zurück, das in *TabelleNum* bzw. *FeldNum* angegeben ist.
Sie übergeben entweder:

- In *TabelleNum* und *FeldNum* Tabellen- und Feldnummer oder
- In *FeldPtr* einen Zeiger auf das Feld.

Hinweis: Dieser Befehl gibt die Eigenschaften auf der Ebene des Strukturfensters zurück. Auf der Formularebene lassen sich ähnliche Eigenschaften definieren.

Sobald der Befehl ausgeführt wurde:

- Gibt der Parameter *Liste* - sofern vorhanden - den Namen der Auswahlliste zurück, die dem Datenfeld zugeordnet wurde. Folgende Feldtypen lassen eine solche Liste zu: String, Text, Zahl, Ganzzahl, Lange Ganzzahl, Datum, Zeit und Boolean. Ist dem Datenfeld keine Auswahlliste zugeordnet, wird ein leerer String zurückgegeben ("").
- Der Parameter *Zwingend* gibt Wahr zurück, wenn das Datenfeld "Zwingend" ist; sonst Falsch. Das Attribut Zwingend lässt sich allen Feldtypen zuordnen, mit Ausnahme von BLOB.
- Der Parameter *NichtEingebbar* gibt Wahr zurück, wenn das Datenfeld "Nicht Eingebbar" ist, sonst Falsch. Ein nicht eingebbares Feld ist nur lesbar, hier lassen sich keine Daten eingeben. Das Attribut Nicht Eingebbar lässt sich allen Feldtypen zuordnen, mit Ausnahme von BLOB.
- Der Parameter *NichtÄnderbar* gibt Wahr zurück, wenn das Datenfeld "NichtEingebbar" ist, sonst Falsch. Ein nicht eingebbares Datenfeld kann nur einmal eingegeben und dann nicht mehr verändert werden. Das Attribut Nicht Änderbar lässt sich allen Feldtypen zuordnen, mit Ausnahme von BLOB.

GET FIELD PROPERTIES

GET FIELD PROPERTIES (FeldPtr | TabelleNum {; FeldNum}; FeldTyp {; FeldLänge {; Indiziert {; Einmalig {; Ausgeblendet}}}})

Parameter	Typ		Beschreibung
FeldPtr | TabelleNum	Zeiger, Lange Ganzzahl	➡	Zeiger auf Feld oder Tabellennummer
FeldNum	Lange Ganzzahl	➡	Feldnummer, wenn Tabellennummer der erste Parameter ist
FeldTyp	Lange Ganzzahl	⬅	Feldtyp
FeldLänge	Lange Ganzzahl	⬅	Feldlänge, wenn alphanumerisch
Indiziert	Boolean	⬅	Wahr = Indiziert, Falsch = Nicht indiziert
Einmalig	Boolean	⬅	Wahr = Einmalig, Falsch = Nicht einmalig
Ausgeblendet	Boolean	⬅	Wahr = Unsichtbar, Falsch = Sichtbar

Beschreibung

Der Befehl **GET FIELD PROPERTIES** gibt Information über das Feld zurück, adressiert durch *FeldPtr* bzw. *TabelleNum* und *FeldNum*.

Sie übergeben entweder:

- Die Tabellen- und Feldnummern in *TabelleNum* und *FeldNum* oder
- Einen Zeiger auf das Feld in *FeldPtr*

Nach dem Aufruf:

- gibt *FeldTyp* den Feldtyp zurück. Dieser Parameter für Variablen kann einen Wert aus folgenden vordefinierten Konstanten unter dem Thema **Feld und Variablentypen** annehmen:

Konstante	Typ	Wert
Is alpha field	Lange Ganzzahl	0
Is BLOB	Lange Ganzzahl	30
Is Boolean	Lange Ganzzahl	6
Is date	Lange Ganzzahl	4
Is float	Lange Ganzzahl	35
Is integer	Lange Ganzzahl	8
Is integer 64 bits	Lange Ganzzahl	25
Is longint	Lange Ganzzahl	9
Is object	Lange Ganzzahl	38
Is picture	Lange Ganzzahl	3
Is real	Lange Ganzzahl	1
Is subtable	Lange Ganzzahl	7
Is text	Lange Ganzzahl	2
Is time	Lange Ganzzahl	11

- Der Parameter *FeldLänge* gibt die Feldlänge zurück, wenn das Feld alphanumerisch ist (z.B. FeldTyp=Is Alpha Field). Der Wert von *FeldLänge* ist für die anderen Feldtypen ohne Bedeutung.
- Der Parameter *Indiziert* gibt WAHR zurück, wenn das Feld indiziert ist, FALSCH wenn es nicht indiziert ist. Indiziert ist nur von Bedeutung für Felder vom Typ alphanumerisch, Ganzzahl, Lange Ganzzahl, Zahl, Datum, Zeit und Boolean.
- Der Parameter *Einmalig* gibt Wahr zurück, wenn das Feld als Attribut "Einmalig " hat, sonst Falsch.
- Der Parameter *Ausgeblendet* gibt Wahr zurück, wenn das Feld als Attribut "Unsichtbar" hat, sonst Falsch. Mit diesem Attribut lässt sich im 4D Standardeditor (Etiketten, Diagramme, ...) ein bestimmtes Feld ausblenden.

Beispiel 1

Dieses Beispiel setzt die Variablen *vTyp*, *vLänge*, *vIndex*, *vEinmalig* und *vUnsichtbar* auf die Attribute des dritten Feldes der ersten Tabelle:

```
GET FIELD PROPERTIES(1;3;vTyp;vLänge;vIndex,vEinmalig;vUnsichtbar)
```

Beispiel 2

Dieses Beispiel setzt die Variablen *vTyp*, *vLänge*, *vIndex*, *vEinmalig* und *vUnsichtbar* auf die Attribute des Feldes [Tabelle3]Feld2:

```
GET FIELD PROPERTIES(-[Tabelle3]Feld2;vTyp;vLänge;vIndex;vEinmalig;vUnsichtbar)
```

Get last field number

Get last field number (TabelleNum | TabellePtr) -> Funktionsergebnis

Parameter	Typ		Beschreibung
TabelleNum | TabellePtr	Lange Ganzzahl, Zeiger	➡	Tabellennummer oder Zeiger auf die Tabelle
Funktionsergebnis	Lange Ganzzahl	⮌	Höchste Feldnummer in der Tabelle

Beschreibung

Die Funktion **Get last field number** gibt die höchste Feldnummer unter den Feldern in der Tabelle mit der Nummer *TabelleNum* oder dem Zeiger *TabellePtr* zurück.

Felder werden in der Reihenfolge der Erstellung numeriert. Wurde kein Feld in der Tabelle gelöscht, gibt die Funktion die Anzahl der Felder in der Tabelle zurück. Bei Schleifen auf Feldnummern der Tabelle müssen Sie die Funktion **Is field number valid** verwenden, um zu prüfen, ob das Feld gelöscht wurde oder nicht.

Beispiel

Folgende Projektmethode erstellt das Array *asFields*. Es enthält die Feldnamen der Tabelle, deren Zeiger als erster Parameter empfangen wird:

```
$vlTable:=Table($1)
ARRAY STRING(31;asFields;Get last field number($vlTable))
For($vlField;Size of array(asFields);1;-1)
   If(Is field number valid($vlTable;$vlField))
      asFields{$vlTable}:=Field name($vlTable;$vlField)
   Else
      DELETE FROM ARRAY(asFields;$vlField)
   End if
End for
```

Get last table number

Get last table number -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Höchste Tabellennummer in der Datenbank

Beschreibung

Die Funktion **Get last table number** gibt die höchste Tabellennummer unter den Tabellen der Datenbank zurück. Die Tabellen werden in der Reihenfolge ihrer Erstellung nummeriert. Wurde in der Datenbank keine Tabelle gelöscht, gibt die Funktion die Anzahl der Tabellen in der Datenbank zurück. Bei Schleifen auf Tabellennummern der Datenbank müssen Sie die Funktion **Is table number valid** verwenden, um zu prüfen, ob die Tabelle gelöscht wurde oder nicht.

Beispiel

Folgendes Beispiel erstellt ein Array mit Namen *asTables*, das die Namen der in der Datenbank definierten Tabellen enthält. Dieses Array lässt sich als DropDown-Liste, Registerkarte, rollbarer Bereich usw. einsetzen, um die Liste der Tabellen in einem Formular anzuzeigen:

```
ARRAY STRING(31;asTables;Get last table number)
If(Get last table number>0)  `Wenn die Datenbank Tabellen enthält
   For($vlTable;1;Size of array(asTables);1;-1)
      If(Is table number valid($v1Tables))
         asTables{$vlTables}:=Table name($vlTables)
      Else
         DELETE FROM ARRAY(asTables;$vlTables)
      End if
   End for
End if
```

GET MISSING TABLE NAMES

GET MISSING TABLE NAMES (FehlTabellen)

Parameter	Typ		Beschreibung
FehlTabellen	Array Text	⇐	Namen der fehlenden Tabellen in der Anwendung

Beschreibung

Der Befehl **GET MISSING TABLE NAMES** gibt die Namen aller fehlenden Tabellen der aktuellen Anwendung im Array *FehlTabellen* zurück.
Fehlende Tabellen sind Tabellen, deren Daten in der Datendatei vorhanden sind, die aber nicht in der aktuellen Struktur existieren. Das kann passieren, wenn eine Datendatei mit unterschiedlichen Versionen der Struktur geöffnet wird.
Hierzu ein typischer Fall:

- Der Entwickler liefert eine Struktur mit den Tabellen A, B, C.
- Der Benutzer fügt die eigenen Tabellen D und E hinzu; z.B. über die integrierten SQL Befehle von 4D und speichert Daten in diesen Tabellen
- Der Entwickler liefert eine neue Version der Struktur. Sie enthält nicht die Tabellen D und E.

Die Benutzerversion der Anwendung enthält weiter Daten aus den Tabellen D und E, sie sind aber nicht zugänglich. Der Befehl **GET MISSING TABLE NAMES** gibt die Namen D und E zurück.
Haben Sie die fehlenden Tabellen der Anwendung identifiziert, können Sie diese über den Befehl **REGENERATE MISSING TABLE** regenerieren.

Hinweis: Die Daten fehlender Tabellen werden entfernt, wenn die Datendatei komprimiert wird, außer die Tabellen wurden erneut generiert.

GET RELATION PROPERTIES

GET RELATION PROPERTIES (FeldPtr | TabelleNum {; FeldNum}; EineTabelle ; EineFeld {; Auswahlfeld {; AutoEine {; AutoViele}}})

Parameter	Typ		Beschreibung
FeldPtr | TabelleNum	Zeiger, Lange Ganzzahl	➡	Zeiger auf Feld oder Tabellennummer
FeldNum	Lange Ganzzahl	➡	Feldnummer, wenn Tabellennummer als erster Parameter übergeben wird
EineTabelle	Lange Ganzzahl	⬅	Nummer der Eine-Tabelle oder 0, wenn von dieser Tabelle keine Verknüpfung ausgeht
EineFeld	Lange Ganzzahl	⬅	Nummer des Eine-Felds oder 0, wenn von diesem Feld keine Verknüpfung ausgeht
Auswahlfeld	Lange Ganzzahl	⬅	Nummer des Auswahlfeldes oder 0, wenn kein Auswahlfeld vorhanden ist
AutoEine	Boolean	⬅	Wahr = Automatische Eine-Verknüpfung, Falsch = Manuelle Eine-Verknüpfung
AutoViele	Boolean	⬅	Wahr = Automatische Eine-zu-Viele, Falsch = Manuell Eine-Zu-Viele

Beschreibung

Der Befehl **GET RELATION PROPERTIES** gibt - sofern vorhanden - die Eigenschaften der Verknüpfung zurück, ausgehend vom Quellfeld, das in den Parametern *TabelleNum* und *FeldNum* oder *FeldPtr* definiert wird.
Sie übergeben entweder:

- In *TabelleNum* und *FeldNum* Tabellen- und Feldnummer oder,
- In *FeldPtr* einen Zeiger auf das Feld.

Sobald der Befehl ausgeführt ist:

- Enthalten die Parameter *EineTabelle* und *EineFeld* jeweils die Tabellennummer und die Feldnummer, auf welche die Verknüpfung zeigt. Startet vom Quellfeld keine Verknüpfung, geben diese Parameter 0 (Null) zurück.
- Der Parameter *Auswahlfeld* enthält die Nummer des Auswahlfeldes der Zieltabelle für diese Verknüpfung. Gibt es für diese Verknüpfung kein Auswahlfeld oder startet vom Quellfeld keine Verknüpfung, gibt dieser Parameter 0 (Null) zurück.
- Die Parameter *AutoEine* und *AutoViele* geben Wahr zurück, wenn für diese Verknüpfung das Kontrollkästchen "Automatische Verknüpfung" markiert ist, sonst Falsch.

Hinweis: Die Parameter *AutoEine* und *AutoViele* geben auch Wahr zurück, wenn vom Quellfeld keine Verknüpfung startet (Sie geben nicht-signifikante Werte zurück). Anhand der Werte der Parameter *EineTabelle* und *EineFeld* können Sie sicherstellen, dass eine Verknüpfung vorhanden ist.

GET TABLE PROPERTIES

GET TABLE PROPERTIES (TabellePtr | TabelleNum ; Ausgeblendet {; TrigSichNeu {; TrigSichDatens {; TrigLöschDatens {; TrigLadeDatens}}}})

Parameter	Typ		Beschreibung
TabellePtr | TabelleNum	Zeiger, Lange Ganzzahl	➡	Zeiger auf Tabelle oder Tabellennummer
Ausgeblendet	Boolean	⬅	Wahr = Unsichtbar, Falsch = Sichtbar
TrigSichNeu	Boolean	⬅	Wahr = Trigger "Neuer Datensatz sichern" aktiviert, sonst Falsch
TrigSichDatens	Boolean	⬅	Wahr = Trigger "Datensatz ändern" aktiviert, sonst Falsch
TrigLöschDatens	Boolean	⬅	Wahr = Trigger "Datensatz löschen" aktiviert, sonst Falsch
TrigLadeDatens	Boolean	⬅	***Nicht verwenden (überholt) ***

Beschreibung

Der Befehl **GET TABLE PROPERTIES** gibt die Eigenschaften für die Tabelle zurück, die in *TabellePtr* oder *TabelleNum* definiert wurde. Die Tabellennummer oder der Zeiger auf die Tabelle kann als erster Parameter übergeben werden.

Sobald der Befehl ausgeführt ist:

- Gibt der Parameter *Ausgeblendet* Wahr zurück, wenn das Attribut "Unsichtbar" für die Tabelle definiert wurde, sonst Falsch. So können Sie die Tabelle beim Verwenden von 4D Standardeditoren (Etiketten, Diagramme...) ausblenden.
- Gibt der Parameter *TrigSichNeu* Wahr zurück, wenn für die Tabelle der Trigger "Neuer Datensatz sichern" gesetzt wurde, sonst Falsch.
- Gibt der Parameter *TrigSichDatens* Wahr zurück, wenn für die Tabelle der Trigger "Datensatz ändern" gesetzt wurde, sonst Falsch.
- Gibt der Parameter *TrigLöschDatens* Wahr zurück, wenn für die Tabelle der Trigger "Datensatz löschen" gesetzt wurde, sonst Falsch.

IMPORT STRUCTURE

IMPORT STRUCTURE (xmlStructure)

Parameter	Typ		Beschreibung
xmlStructure	Text	➡	XML definition of 4D database structure

Beschreibung

Der Befehl **IMPORT STRUCTURE** importiert die im Parameter *xmlStruktur* angegebene 4D XML Strukturdefinition in die aktuelle Anwendung.

Der Parameter *xmlStruktur* muss eine gültige 4D Strukturdefinition im XML Format enthalten. Dafür gibt es folgende Möglichkeiten:

- Sie führen den Befehl **EXPORT STRUCTURE** aus,
- Sie wählen im Designmodus den Menübefehl **Exportieren > Strukturdefinition in XML Datei** (siehe **Strukturdefinitionen exportieren und importieren**),
- Sie erstellen oder bearbeiten eine eigene XML Datei, die auf DTD Dateien im Ordner "DTD" der 4D Applikation basieren.

Die importierte Strukturdefinition wird zur aktuellen hinzugefügt und erscheint im standardmäßigen 4D Struktureditor zwischen den Tabellen (falls vorhanden). Hat eine importierte Tabelle denselben Namen wie eine bereits vorhandene Tabelle, wird ein Fehler generiert und die Importoperation abgebrochen.

Sie können die Struktur auch in eine leere Datenbank importieren und so eine neue Anwendung erstellen.

Ist die Struktur im kompilierten oder Nur-Lesen Modus, wird ein Fehler generiert. Sie lässt sich auch nicht über eine 4D remote Anwendung aufrufen.

Beispiel

Eine gespeicherte Strukturdefinition in die aktuelle Anwendung importieren:

```
$struc:=Document to text("c:\\4DStructures\\Employee.xml")
IMPORT STRUCTURE($struc)
```

Is field number valid

Is field number valid (TabelleNum | TabellePtr ; FeldNum) -> Funktionsergebnis

Parameter	Typ		Beschreibung
TabelleNum | TabellePtr	Lange Ganzzahl, Zeiger	→	Tabellennummer oder Zeiger auf Tabelle
FeldNum	Lange Ganzzahl	→	Feldnummer
Funktionsergebnis	Boolean	↩	Wahr = Feld existiert in Tabelle Falsch = Feld existiert nicht in Tabelle

Beschreibung

Die Funktion **Is field number valid** gibt Wahr zurück, wenn das Feld, dessen Nummer im Parameter *FeldNum* übergeben ist, in der Tabelle vorhanden ist, deren Nummer oder Zeiger im Parameter *TabelleNum* oder *TabellePtr* übergeben ist. Gibt es das Feld nicht, gibt die Funktion Falsch zurück. Beachten Sie, dass die Funktion Falsch zurückgibt, wenn die Tabelle mit dem Feld im Papierkorb des Explorers ist.

Über diese Funktion finden Sie gelöschte Felder, die zu Lücken in der Reihenfolge der Feldnummern führen.

Is table number valid

Is table number valid (TabelleNum) -> Funktionsergebnis

Parameter	Typ		Beschreibung
TabelleNum	Lange Ganzzahl	→	Tabellennummer
Funktionsergebnis	Boolean	↩	Wahr = Tabelle existiert in der Datenbank Falsch = Tabelle existiert nicht in der Datenbank

Beschreibung

Die Funktion **Is table number valid** gibt Wahr zurück, wenn die Tabelle, deren Nummer im Parameter *TabelleNum* übergeben ist, in der Datenbank vorhanden ist, sonst Falsch. Beachten Sie, dass die Funktion Falsch zurückgibt, wenn die Tabelle im Papierkorb des Explorers ist.

Über diese Funktion finden Sie gelöschte Tabellen, die zu Lücken in der Reihenfolge der Tabellennummern führen.

PAUSE INDEXES

PAUSE INDEXES (Tabellenname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Anhalten der Indizes

Beschreibung

Der Befehl **PAUSE INDEXES** deaktiviert vorübergehend alle Indizes von Tabellenname, außer dem Index des Primärschlüssels..

Die Indizes werden nicht physisch aus den Daten (.4DIndx Datei) oder der Strukturdatei der Anwendung (_USER_INDEXES, siehe **System Tables**) gelöscht, sie werden jedoch ungültig gemacht und so nicht mehr aktualisiert. Sind die Indizes deaktiviert, verwenden alle in *Tabellenname* ausgeführten Operationen, wie Suchen, Sortieren, Datensätze hinzufügen, ändern oder löschen die Indizes nicht mehr.
Dieser Befehl ist besonders hilfreich beim Importieren oder Ändern großer Datenmengen in Tabellen mit mehreren Indizes. Da 4D die Indizes jedes Mal aktualisieren muss, wenn ein Datensatz bestätigt wird, kann diese Operation u.U. geraume Zeit dauern. Wurden dagegen die Indizes zuvor deaktiviert, kann das die Operation signifikant beschleunigen.
Um die Indizes nach Beenden der Operation wieder zu reaktivieren, rufen Sie für *Tabellenname* einfach den Befehl **RESUME INDEXES** auf.

Hinweis: Ein ähnliches Ergebnis erhalten Sie mit den Befehlen **CREATE INDEX** und **DELETE INDEX**, jedoch mit folgenden Unterschieden:

- Sie müssen **DELETE INDEX** / **CREATE INDEX** für jeden Index von *Tabellenname* aufrufen.
- Die Befehle **DELETE INDEX** / **CREATE INDEX** verändern die interne Nummer des Index. Das passiert nicht mit **PAUSE INDEXES** / **RESUME INDEXES**. Bei einer veränderten Indexnummer werden die Daten automatisch neu indiziert, wenn sich die Datenmenge ändert.

Rufen Sie den Befehl **PAUSE INDEXES** für eine Tabelle auf und beenden dann die Anwendung ohne **RESUME INDEXES** aufzurufen, werden alle Indizes für diese Tabelle beim Neustart der Anwendung automatisch neu erstellt.

Hinweis: Dieser Befehl lässt sich nicht über ein 4D remote aufrufen.

Beispiel

Beispiel für eine Methode zum Importieren großer Datenmengen:

```
PAUSE INDEXES([Articles])
IMPORT DATA("HugeImport.txt") //Importieren
RESUME INDEXES([Articles])
```

REGENERATE MISSING TABLE

REGENERATE MISSING TABLE (TabellenName)

Parameter	Typ		Beschreibung
TabellenName	Text	➡	Name der fehlenden Tabelle, die erneut generiert werden soll.

Beschreibung

Der Befehl **REGENERATE MISSING TABLE** regeneriert die fehlende Tabelle, deren Name im Parameter *TabellenName* übergeben ist. Die regenerierte Tabelle wird im Strukturedtor sichtbar und die darin enthaltenen Daten sind wieder zugänglich.

Fehlende Tabellen sind Tabellen, deren Daten in der Datendatei vorhanden sind, die jedoch nicht auf der Strukturebene existieren.
Über den Befehl **GET MISSING TABLE NAMES** können Sie fehlende Tabellen, die u.U. in der Anwendung enthalten sind, identifizieren.
Ist die Tabelle, angegeben im Parameter *TabellenName*, keine fehlende Tabelle der Anwendung, führt der Befehl nichts aus.

Beispiel

Nachfolgende Methode regeneriert alle fehlenden Tabellen, die evtl. in der Anwendung vorhanden sind:

```
ARRAY TEXT($arrMissingTables;0)
GET MISSING TABLE NAMES($arrMissingTables)
$SizeArray:=Size of array($arrMissingTables)
If($SizeArray#0)
 //Füllt das Array mit den Namen aller Tabellen in der Anwendung
   ARRAY TEXT(arrTables;Get last table number)
   If(Get last table number>0)  //Gibt es derzeit Tabellen
      For($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("Regeneriere die Tabelle"+$arrMissingTables{$i}+"?")
         If(OK=1)
            REGENERATE MISSING TABLE($arrMissingTables{$i})
         End if
      Else
         ALERT("Tabelle "+$arrMissingTables{$i}+" lässt sich nicht regenerieren, da die Anwendung bereits eine Tabelle mit diesem Namen enthält.")
      End if
   End for
 Else
   ALERT("Keine Tabellen zu regenerieren.")
 End if
```

RELOAD EXTERNAL DATA

RELOAD EXTERNAL DATA (Feld)

Parameter	Typ		Beschreibung
Feld	Text, BLOB, Bild, Objekt	➡	Feld, dessen Speicherort Sie setzen wollen

Beschreibung

Der Befehl **RELOAD EXTERNAL DATA** lädt den Inhalt einer externen Speicherdatei, die einem Datenfeld vom Typ BLOB, Bild oder Text zugeordnet ist, erneut in den Speicher.

Dieser Befehl ist hilfreich, wenn ein bereits geladenes Feld eines Datensatzes durch ein anderes Programm auf der Festplatte geändert wird.
Zur Erinnerung: Externe Speicherdateien für Felder sind immer im Schreibmodus verfügbar. Beispielsweise lässt sich ein Bild in einem Feld vom Typ Bild über ein Grafikprogramm ändern und anschließend auf der Festplatte speichern.

Sie müssen dann die Daten über den Befehl **RELOAD EXTERNAL DATA** erneut laden, um den Inhalt des Feldes zu aktualisieren, wenn es in einem Formular angezeigt wird.

Hinweis: RELOAD EXTERNAL DATA funktioniert nur in einem lokalen 4D oder auf 4D Server. In remote 4D können Sie ein einzelnes Feld nicht erneut laden. Hier müssen Sie alle Datenfelder erneut laden, z.B. mit dem Befehl **LOAD RECORD**.

RESUME INDEXES

RESUME INDEXES (Tabellenname {; *})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zum Reaktivieren der Indizes
*	Operator	➡	Mit * = asynchrone Indizierung

Beschreibung

Der Befehl **RESUME INDEXES** reaktiviert alle Indizes von Tabellenname, wenn sie zuvor über den Befehl **PAUSE INDEXES** angehalten wurden. Wurden die Indizes von *Tabellenname* nicht angehalten, führt dieser Befehl nichts aus.

In den meisten Fällen löst das Ausführen dieses Befehls den erneuten Aufbau der Indizes für *Tabellenname* aus.
Mit dem optionalen Parameter * werden die Indizes im asynchronen Modus aufgebaut, d.h. die aufrufende Methode fährt mit der Ausführung fort, egal, ob die Indizierung abgeschlossen ist oder noch läuft. Lassen Sie diesen Parameter weg, blockiert der Neuaufbau der Indizes die Ausführung der Methode, bis diese Operation abgeschlossen ist.

Der Befehl **RESUME INDEXES** lässt sich nur auf 4D Server oder 4D Einzelplatz aufrufen. Beim Ausführen auf einem Rechner mit remote 4D wird der Fehler -10513 generiert. Dieser Fehler lässt sich mit einer Methode abfangen, die über den Befehl **ON ERR CALL** installiert wurde.

SET EXTERNAL DATA PATH

SET EXTERNAL DATA PATH (Feld ; Pfad)

Parameter	Typ		Beschreibung
Feld	Text, BLOB, Bild, Objekt	➡	Feld, dessen Speicherort gesetzt werden soll
Pfad	Text, Lange Ganzzahl	➡	Pfadname und Dateiname zum externen Speichern oder 0 = Verwende Strukturdefinition 1 = Verwende Standardordner

Beschreibung

Der Befehl **SET EXTERNAL DATA PATH** setzt oder ändert den externen Speicherort für das im Parameter *Feld* übergebene Feld für den aktuellen Datensatz.

In 4D lassen sich Datenfelder vom Typ Text, BLOB, Bild und Objekt außerhalb der Datendatei speichern. Weitere Informationen dazu finden Sie im Abschnitt **Daten extern speichern** des Handbuchs *4D Designmodus*.

Die über diesen Befehl definierten Einstellungen gelten nur, wenn der aktuelle Datensatz auf der Festplatte gespeichert wird. Speicherparameter, die in der Struktur der Anwendung gesetzt wurden, werden nicht verändert. Wird der aktuelle Datensatz abgebrochen, führt der Befehl nichts aus.

In *Pfad* können Sie entweder einen eigenen Pfadnamen übergeben, oder eine Konstante, die einen automatischen Speicherort angibt:

- **Eigener Pfadname zur Datei**
  In diesem Fall nutzen Sie das externe Speichern im "eigenen Modus". Hier sind einige 4D Datenbankfunktionen nicht automatisch verfügbar (siehe Handbuch *4D Designmodus*). Insbesondere das Erstellen oder Ändern der Dateien müssen Sie selbst verwalten.
  Sie können entweder einen Pfad in Bezug auf die Datendatei oder einen absoluten Pfad übergeben, er muss Name und Endung der Speicherdatei enthalten. Sie müssen die Syntax des Systems verwenden. Für einen relativen Pfad übergeben Sie "../" (Windows) oder "..:" (OS X) am Textanfang. Sie können einen beliebigen Ordner festlegen, das kann auch der Standardordner für externe Dateien der Datenbank sein (*DatenbankName.ExternalData*) - In diesem Fall sind die Dateien beim Sichern der Datenbank enthalten.
  Die im Parameter *Pfad* angegebene Datei muss vorhanden und zugänglich sein, wenn der Befehl ausgeführt wird. Beachten Sie, dass bei einem ungültigen Pfad (Datei oder Ordner fehlt) ein Fehler nur bei einem als absolut definierten Pfad zurückgegeben wird. Ist ein relativer Pfad angegeben, müssen Sie selbst für die Gültigkeit sorgen, da kein Fehler generiert wird, wenn die Datei nicht gefunden wird.
  Sichern Sie die externe Datei im gleichen Ordner wie die Datendatei oder einen ihrer Unterordner, bewertet 4D den angegebenen Pfad als relativ zur Datendatei und behält den Link bei, selbst wenn der Datendateiordner bewegt oder umbenannt wird.
  Folglich ist es möglich, dass mehrere Datensätze die gleiche externe Datei gemeinsam nutzen können. Alle Änderungen in dieser Datei sind dann in allen Datensätzen verfügbar. Wenn in solchen Fällen mehrere Prozesse dieselben Felder gleichzeitig im Schreibmodus verwenden können, müssen Sie konkurrierende Zugriffe über Semaphoren steuern, um die evtl. Beeinträchtigung externer Dateien zu vermeiden.

- **Automatischer Speicherort**
  Sie können zwei automatische Speicherorte über folgende Konstanten unter dem Thema **Datendatei Wartung** angeben:

Konstante	Typ	Wert	Kommentar
Use default folder	Lange Ganzzahl	1	Die Daten des Feldes, das als Parameter übergeben wird, werden im Standardordner mit Namen DatenbankName.ExternalData gesichert und neben die Datendatei gelegt. In diesem Modus verwaltet 4D externe Daten wie Daten innerhalb der Datendatei.
Use structure definition	Lange Ganzzahl	0	4D verwendet die Parameter, die in der Struktur zum Speichern des Felds angegeben sind (siehe Handbuch *4D Designmodus*). Wechseln Sie vom externen Speichern zum internen Speichern, wird die externe Datei nicht gelöscht.

Wurde dieser Befehl einmal ausgeführt, behält 4D den Link zwischen dem Datenfeld des Datensatzes und der Datei auf der Festplatte automatisch bei. Sie müssen den Befehl nicht erneut ausführen - außer, der Pfad ändert sich. Kann 4D nicht mehr auf die Daten im Feld zugreifen (Speicherdatei wurde umbenannt oder gelöscht, Pfad wurde gelöscht, o.ä.), is das Feld leer, es wird jedoch kein Fehler generiert.

Hinweis: SET EXTERNAL DATA PATH lässt sich nur auf einem lokalen 4D oder 4D Server ausführen. Bei Ausführung auf einem remote 4D wird der Befehl ignoriert.

Beispiel

Den Inhalt eines Bildfeldes außerhalb der Datendatei, in Ordner neben der Strukutrdatei speichern:

```
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" muss neben der Strukturdatei vorhanden sein
SAVE RECORD([Photos])
```

SET INDEX

SET INDEX (Feldname ; Index {; Modus} {; *})

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Datenfeld für Erstellen oder Löschen des Index
Index	Boolean, Ganzzahl	➡	Wahr=Index anlegen, Falsch =Index löschen oder Indextyp erstellen: -1=Schlüsselwort, 0=Standard, 1=B-Tree standard, 3=B-Tree cluster
Modus	Lange Ganzzahl	➡	Überholt (Parameter wird ignoriert)
*		➡	Mit *: Asynchrone Indizierung

Beschreibung

Hinweis zur Kompatibilität: Dieser Befehl wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen, die Befehle **CREATE INDEX** und **DELETE INDEX** zu verwenden, um Indizes per Programmierung zu verwalten.

Der Befehl **SET INDEX** erlaubt zwei Syntaxarten:

- Übergeben Sie in *Index* einen Boolean Wert, erstellt oder entfernt der Befehl den Index für das in *Feldname* angegebene Datenfeld.
- Übergeben Sie in *Index* eine Ganzzahl, erstellt der Befehl den angegebenen Indextyp.

Index = Boolean
Wollen Sie das Datenfeld indizieren, übergeben Sie in *Index* WAHR. Der Befehl legt einen Index vom Typ Standard an. Ist der Index bereits vorhanden, hat der Aufruf keine Auswirkung.
Übergeben Sie in *Index* FALSCH, löscht der Befehl alle Standard Indizes, z.B. nicht-zusammengesetzt und ohne Schlüsselwort, die dem Feld zugewiesen sind. Ist kein Index vorhanden, hat der Aufruf keine Auswirkung.

Index = Ganzzahl
In diesem Fall erstellt der Befehl einen bestimmten Indextyp. Sie können eine der nachfolgenden Konstanten unter dem Thema **Indextyp** übergeben:

Konstante	Typ	Wert	Kommentar
Cluster BTree Index	Lange Ganzzahl	3	Index vom Typ B-Baum mit Clustern. Dieser Typ ist optimal für Indizes mit wenigen Schlagwörtern, z.B. wenn dieselben Werte in den Daten sich häufig wiederholen.
Default Index Type	Lange Ganzzahl	0	4D definiert den Indextyp (ausgenommen Index nach Schlüsselwörtern), der gemäß dem Feldinhalt am besten passt.
Keywords Index	Lange Ganzzahl	-1	Index nach Schlüsselwörtern, der die Indizierung des Feldinhalts Wort für Wort ermöglicht. Dieser Indextyp lässt sich nur für Datenfelder vom Typ Text oder alphanumerisch verwenden.
Standard BTree Index	Lange Ganzzahl	1	Index vom Typ Standard B-Baum. Dieser vielfältige Indextyp wird in den bisherigen Versionen von 4D verwendet.

Hinweis: Ein Index vom Typ B-Tree, der einem Datenfeld vom Typ Text zugeordnet ist, speichert maximal die ersten 1024 Zeichen des Datenfeldes. Deshalb funktionieren in diesem Kontext Suchläufe nach Strings mit mehr als1024 Zeichen nicht.

SET INDEX indiziert keine gesperrten Datensätze; er wartet ab, bis der Datensatz nicht mehr gesperrt ist.

Der Parameter *Modus* hat ab Version 11 keine Bedeutung mehr und wird ignoriert, wenn er übergeben wurde.

Der optionale Parameter * erlaubt eine asynchrone (simultane) Indizierung. Mit asynchroner Indizierung kann eine aufgerufene Methode weiter ausgeführt werden, auch wenn die Indizierung noch nicht abgeschlossen ist. Die Ausführung wird jedoch bei einem Befehl gestoppt, der auf den Index zugreift.

Hinweise:

- Mit diesem Befehl erstellte Indizes haben keinen Namen. Folglich lassen sie sich nicht über den Befehl **DELETE INDEX** löschen, da er eine auf Namen basierende Syntax verwendet.
- Dieser Befehl kann keine zusammengesetzten Indizes erstellen oder löschen.
- Dieser Befehl erlaubt nicht das Löschen von Indizes nach Schlüsselwörtern, die mit dem Befehl **CREATE INDEX** angelegt wurden.

Beispiel 1

Folgendes Beispiel indiziert das Feld *[Customers]ID*:

```
UNLOAD RECORD([Customers])
SET INDEX([Customers]ID;True)
```

Beispiel 2

Sie wollen das Feld *[Customers]Name* im asynchronen Modus indizieren:

```
SET INDEX([Customers]Name;True;*)
```

Beispiel 3

Dieses Beispiel erstellt einen Volltext-Index:

```
SET INDEX([Books]Summary;Keywords Index)
```

Table

Table (TabelleNum | Ptr) -> Funktionsergebnis

Parameter	Typ		Beschreibung
TabelleNum | Ptr	Lange Ganzzahl, Zeiger	➡	Tabellennummer oder Zeiger auf Tabelle, oder Zeiger auf Feld
Funktionsergebnis	Lange Ganzzahl, Zeiger	➲	Zeiger auf Tabelle bei Tabellennummer Tabellennummer bei Zeiger auf Tabelle bzw. Feld

Beschreibung

Die Funktion **Table** hat drei Formen:

- Übergeben Sie in *TabelleNum* eine Zahl, gibt **Table** einen Zeiger auf die Tabelle zurück.
- Übergeben Sie in *Ptr* einen Zeiger auf die Tabelle, gibt **Table** die Tabellennummer der Tabelle zurück.
- Übergeben Sie in *Ptr* einen Zeiger auf das Feld, gibt **Table** die Tabellennummer des Feldes zurück.

Beispiel 1

Dieses Beispiel setzt die Variable *TabellePtr* in einen Zeiger auf die dritte Tabelle der Datenbank:

```
TabellePtr:=Table(3)
```

Beispiel 2

Übergeben Sie *TablePtr* (ein Zeiger auf die dritte Tabelle) in **Table**, wird die Zahl 3 zurückgegeben. Folgende Zeile setzt *TabelleNum* auf 3:

```
TabelleNum:=Table(TablePtr)
```

Beispiel 3

Dieses Beispiel setzt die Variable *TabelleNum* auf die Tabellennummer von *[Table3]*:

```
TabelleNum:=Table(->[Table3])
```

Beispiel 4

Dieses Beispiel setzt die Variable *TabelleNum* auf die Tabellennummer der Tabelle, zu der das Feld *[Table3]Field1* gehört:

```
TabelleNum:=Table(->[Table3]Field1)
```

Table name

Table name (TabelleNum | TabellePtr) -> Funktionsergebnis

Parameter	Typ		Beschreibung
TabelleNum | TabellePtr	Lange Ganzzahl, Zeiger	→	Tabellennummer oder Zeiger auf Tabelle
Funktionsergebnis	String	↩	Name der Tabelle

Beschreibung

Die Funktion **Table name** gibt den Namen der Tabelle zurück, deren Nummer oder Zeiger in *TabelleNum* oder *TabellePtr* übergeben wurde.

Beispiel

In Folgenden sehen Sie eine generische Methode, die die Datensätze einer Tabelle anzeigt. Die Referenz auf die Tabelle wird als Zeiger übergeben. Über **Table name** wird der Tabellenname in die Titelleiste des Fensters integriert:

```
` Projektmethode SHOW CURRENT SELECTION
` SHOW CURRENT SELECTION (Zeiger)
` SHOW CURRENT SELECTION (->[Tabelle])

SET WINDOW TITLE("Datensätze für "+Table name($1)) `Setzt den Fenstertitel
DISPLAY SELECTION($1->) `Zeigt die Auswahl
```

Suchen

- 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 (Status)

Parameter	Typ		Beschreibung
Status	Boolean	➡	Wahr=Aktiviere interne Suchanalyse Falsch=Deaktiviere interne Suchanalyse

Beschreibung

Der Befehl **DESCRIBE QUERY EXECUTION** aktiviert oder deaktiviert den Modus Suchanalyse für den aktuellen Prozess. Er funktioniert nur im Rahmen der Suchbefehle der 4D Programmiersprache, wie z.B. **QUERY**.
Hat der Parameter *Status* den Wert Wahr, wird der Modus Suchanalyse aktiviert. In diesem Modus speichert die 4D Engine intern zwei spezifische Informationsteile für die nachfolgenden Suchen innerhalb der Daten.

- Eine ausführliche interne Beschreibung der Suche unmittelbar vor der Ausführung, mit anderen Worten, was zur Ausführung der Suche geplant ist (der Suchplan)
- Eine ausführliche interne Beschreibung der gerade ausgeführten Suche (der Suchpfad)

Die aufgezeichnete Information enthält die Art der Suche (indiziert, sequentiell), die Anzahl der gefundenen Datensätze und die Zeit, die zur Ausführung aller Suchkriterien benötigt wird.
Mit den Funktionen **Get last query plan** und **Get last query path** können Sie diese Information dann auslesen.

Die Beschreibung des Suchplans und des Suchpfads sind in der Regel gleich. Unterschiede können sich ergeben, da 4D während der Suchausführung dynamische Optimierungen implementieren kann, um eine bessere Performance zu erzielen. So wird z.B. eine indizierte Suche bei Bedarf in eine sequentielle Suche umgewandelt, wenn 4D davon ausgeht, dass dies schneller ist. Das ist z.B. der Fall, wenn die Anzahl der zu durchsuchenden Datensätze relativ gering ist.
Übergeben Sie Falsch im Parameter *Status*, wenn Sie die Suchläufe nicht mehr analysieren wollen. Die Suchanalyse kann nämlich die Ausführung des Programms verlangsamen.

Beispiel

Dieses Beispiel zeigt die Art der Info, die diese Befehle liefern:

```
C_TEXT($vResultPlan;$vResultPath)
DESCRIBE QUERY EXECUTION(True)  //analysis mode
QUERY([Employees];[Employees]LastName="T@";*)  // Search for employees whose last name starts with T...
QUERY([Employees]; & ;[Companies]Name="H@";*)  // that work for a company whose name starts with H
QUERY([Employees]; & ;[Employees]Salary>2500;*)  // whose salary is > 2500
QUERY([Employees]; & ;[Cities]Pop<50000)  // that live in a city with less than 50,000 inhabitants
$vResultPlan:=Get last query plan(Description in text format)
$vResultPath:=Get last query path(Description in text format)
DESCRIBE QUERY EXECUTION(False)  //End of analysis mode
```

Nach Ausführen dieses Code enthalten *$vResultPlan* und *$vResultPath* Beschreibungen der ausgeführten Suchen, zum Beispiel:

```
$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 records found in 1 ms)
```

Ist in der Funktion **Get last query path** die Konstante Description in XML Format übergeben, zeigt *$vResultPath* die Beschreibung der Suche im XML Format:

```
$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 (ZielFeld ; Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ZielFeld	Feld	➡	Feld zum Ausführen der Suche
Wert	Feld, Variable	➡	Zu suchender Wert
		⬅	Gefundener Wert
Funktionsergebnis	Lange Ganzzahl	➲	Nummer des gefundenen Datensatzes oder -1, wenn kein Datensatz gefunden wurde

Beschreibung

Die Funktion **Find in field** gibt die Nummer des ersten Datensatzes zurück, dessen Feld *ZielFeld* gleich *Wert* ist. Werden keine Datensätze gefunden, gibt **Find in field** den Wert -1 zurück.

Nach Aufrufen dieser Funktion enthält *Wert* den gefundenen Wert. Damit können Sie auch Suchen mit dem Jokerzeichen (“@”) in Feldern vom Typ Alpha durchführen und dann den gefundenen Wert wiederfinden.

Hinweis: Nach diesem Prinzip können Sie in *Wert keine Parameter* $1, $2, etc. verwenden, da dies im kompilierten Modus zu Funktionsstörungen führen würde. Übergeben Sie ein Feld in *Wert*, beachten Sie, dass dieser Wert bei erfolgreicher Suche erneut zugewiesen wird (insbesondere der Befehl **Modified record** gibt **Wahr** für den aktuellen Datensatz der Tabelle zurück).

Find in field verändert weder die aktuelle Auswahl noch den aktuellen Datensatz.

Diese Funktion ist schnell und hilfreich, um Doppeleinträge während der Dateneingabe zu verhindern.

Historischer Hinweis: In früheren 4D Versionen hieß diese Funktion **Find index key** und funktionierte nur mit indizierten Feldern. Diese Einschränkung ist ab 4D v11 SQL weggefallen und der Name wurde geändert.

Beispiel 1

Sie möchten in einer Audio CD Datenbank während der Dateneingabe prüfen, ob der Name eines Sängers bereits in der Datenbank vorhanden ist. Da derselbe Name öfters vorkommen kann, wollen Sie das Feld [Sänger]Name nicht einmalig machen. So schreiben Sie im Eingabeformular in der Objektmethode für das Feld [Sänger]Name folgenden Code:

```
If(Form event=On Data Change)
   $RecNum:=Find in field([Sänger]Name;[Sänger]Name)
   If($RecNum #-1)  ` Wurde dieser Name bereits eingegeben
      CONFIRM("Es gibt bereits einen Sänger mit demselben Namen. Wollen Sie den Datensatz sehen?";"Ja";"Nein")
      If(OK=1)
         GOTO RECORD([Sänger];$RecNum)
      End if
   End if
End if
```

Beispiel 2

Mit diesem Beispiel können Sie prüfen, ob ein Wert vorhanden ist:

```
C_LONGINT($id;$1)
$id:=$1
If(Find in field([MyTable]MyID;$id)>=0)
   $0:=True
Else
   $0:=False
End if
```

Die Symbole >= decken alle Fälle ab. Die Funktion gibt eine Datensatznummer zurück und der erste Datensatz hat die Nummer 0.

Get last query path

Get last query path (Format) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Format	Lange Ganzzahl	→	Format der Beschreibung (Text oder XML)
Funktionsergebnis	String	↩	Beschreibung des zuletzt ausgeführten Suchpfads

Beschreibung

Die Funktion **Get last query path** gibt die ausführliche interne Beschreibung des aktuellen Pfads der letzten in den Daten ausgeführten Suche zurück. Weitere Informationen dazu finden Sie unter dem Befehl **DESCRIBE QUERY EXECUTION**.

Die Beschreibung wird im Format Text oder XML zurückgegeben, je nachdem, welcher Wert im Parameter *Format* definert ist. Sie können eine der folgenden Konstanten unter dem Thema **Suchen** übergeben:

Konstante	Typ	Wert
Description in text format	Lange Ganzzahl	0
Description in XML format	Lange Ganzzahl	1

Diese Funktion gibt nur einen sinnvollen Wert zurück, wenn der Befehl **DESCRIBE QUERY EXECUTION** während der aktuellen Sitzung ausgeführt wurde.

Für Optimierungszwecke lässt sich die Beschreibung des letzten Suchpfads mit der Beschreibung des Suchplans vergleichen, der für die letzte Suche erstellt wurde. Dazu rufen Sie die Funktion **Get last query plan** auf.

Get last query plan

Get last query plan (Format) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Format	Lange Ganzzahl	→	Format der Beschreibung (Text oder XML)
Funktionsergebnis	String	↩	Beschreibung des zuletzt ausgeführten Suchplans

Beschreibung

Die Funktion **Get last query plan** gibt die interne Beschreibung des Suchplans für die zuletzt in den Daten ausgeführte Suche zurück. Weitere Informationen dazu finden Sie unter dem Befehl **DESCRIBE QUERY EXECUTION**.

Die Beschreibung wird im Format Text oder XML zurückgegeben, je nachdem, welcher Wert im Parameter *Format* definiert ist. Sie können eine der folgenden Konstanten unter dem Thema **Suchen** übergeben:

Konstante	Typ	Wert
Description in text format	Lange Ganzzahl	0
Description in XML format	Lange Ganzzahl	1

Diese Funktion gibt nur einen sinnvollen Wert zurück, wenn der Befehl **DESCRIBE QUERY EXECUTION** während der Sitzung ausgeführt wurde.

Für Optimierungszwecke lässt sich die Beschreibung des letzten Suchplans mit der Beschreibung des aktuellen Pfads der letzten Suche vergleichen (erhalten über die Funktion **Get last query plan**).

GET QUERY DESTINATION

GET QUERY DESTINATION (Zieltyp ; Zielobjekt ; Zielzeiger)

Parameter	Typ		Beschreibung
Zieltyp	Lange Ganzzahl	⇐	0=aktuelle Auswahl, 1=Menge, 2=temporäre Auswahl, 3=Variable
Zielobjekt	String	⇐	Name der Menge, temporären Auswahl oder leerer String
Zielzeiger	Zeiger	⇐	Zeiger auf die lokale Variable, wenn Zieltyp=3

Beschreibung

Der Befehl **GET QUERY DESTINATION** gibt das aktuelle Ziel der Suchergebnisse für den laufenden Prozess zurück. Standardmäßig verändern die Suchergebnisse die aktuelle Auswahl. Das können Sie jedoch über den vorhandenen Befehl **SET QUERY DESTINATION** anders einstellen. Er wurde dafür in v13 entsprechend angepasst (siehe unten).

Im Parameter *Zieltyp* gibt 4D einen Wert für das aktuelle Ziel der Suchläufe an, im Parameter *Zielobjekt* den Namen des Ziels (falls zutreffend). Sie können *Zieltyp* mit den Konstanten unter dem Thema **Suchen** vergleichen:

Konstante	Typ	Wert
Into current selection	Lange Ganzzahl	0
Into named selection	Lange Ganzzahl	2
Into set	Lange Ganzzahl	1
Into variable	Lange Ganzzahl	3

Der in *Zielobjekt* zurückgegebene Wert richtet sich nach dem in *Zieltyp* angegebenen Wert:

Parameter Zieltyp	Parameter Zielobjekt
0 (current selection)	*Zielobjekt* ist ein leerer String
1 (set)	*Zielobjekt* enthält den Namen der Menge
2 (named selection)	*Zielobjekt* enthält den Namen der Auswahl
3 (variable)	*Zielobjekt* ist ein leerer String (Parameter *ZielZeiger* verwenden)

Ist das Suchziel eine lokale Variable (Zieltyp gibt 3 zurück), gibt 4D im Parameter *ZielZeiger* einen Zeiger auf diese Variable zurück.

Beispiel

Das Suchziel temporär ändern und dann die vorigen Parameter wiederherstellen:

```
GET QUERY DESTINATION($vType;$vName;$ptr)
 //aktuelle Parameter wiederfinden
SET QUERY DESTINATION(Into set;"$temp")
 //Ziel temporär ändern
QUERY(...)  //Suchen
SET QUERY DESTINATION($vType;$vName;$ptr)
 //Parameter wiederherstellen
```

Get query limit

Get query limit -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Limit für Anzahl der Datensätze 0 = unbegrenzte Anzahl

Beschreibung

Die Funktion **Get query limit** gibt das Limit für die Anzahl der Datensätze zurück, die eine Suche im aktuellen Prozess finden kann.

Dieses Limit setzen Sie mit dem Befehl **SET QUERY LIMIT**.

Standardmäßig, d.h. wenn kein Limit gesetzt ist, gibt die Funktion 0 zurück.

ORDER BY

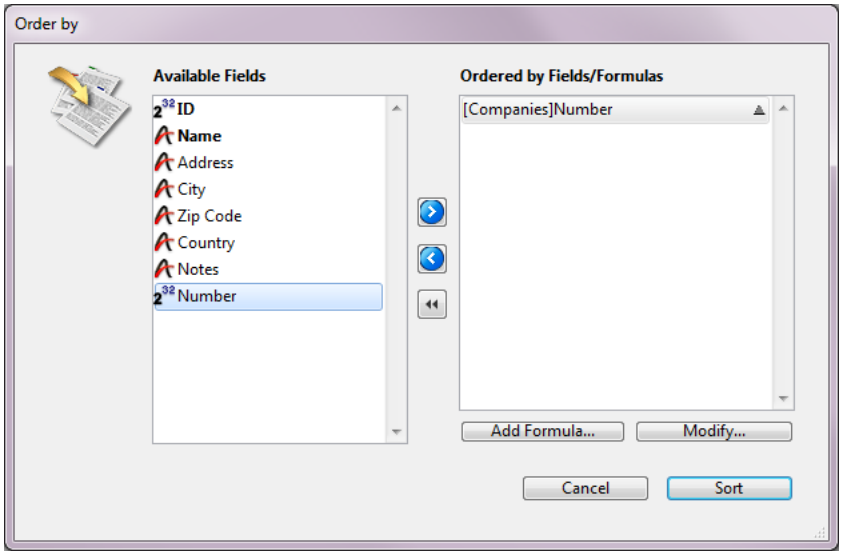
ORDER BY ({Tabellenname ;}{ Feldname }{; >oder< }{; Feldname2 ; >oder<2 ; ... ; FeldnameN ; >oder<N}{; *})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren ausgewählte Datensätze sortiert werden sollen Ohne Angabe Haupttabelle
Feldname	Feld	➡	Feld, nach dem sortiert werden soll
>oder<	Operator	➡	Sortierrichtung für jede Ebene: > aufsteigend, < absteigend
*	Operator	➡	Sortiermodus beibehalten

Beschreibung

ORDER BY sortiert die aktuelle Auswahl des laufenden Prozesses von *Tabellenname*. Ist die Sortierung abgeschlossen, wird der erste Datensatz der Auswahl zum aktuellen Datensatz.

Geben Sie den Parameter *Tabellenname* nicht an, bezieht sich der Befehl auf die Standardtabelle. Gibt es keine Standardtabelle, tritt ein Fehler auf.
Geben Sie die Parameter *Feldname*, >oder< bzw. * nicht an, zeigt **ORDER BY** für *Tabellenname* den Sortiereditor. Er sieht folgendermaßen aus:



Weitere Informationen dazu finden Sie im Abschnitt **Sortiereditor** des Handbuchs *4D Designmodus*.

Geben Sie *Datenfeld* und *>oder<* an, erscheint nicht der Sortiereditor, die Sortierung erfolgt per Programmierung. Sie können die Auswahl nach einer oder mehreren Ebenen sortieren. Geben Sie für jede Ebene *Datenfeld* und die Sortierreihenfolge *>oder<* an. "Größer als" (>) sortiert in aufsteigender Reihenfolge, "Kleiner als" (<) in absteigender Reihenfolge.
Ohne Festlegung wird in aufsteigender Reihenfolge sortiert.

Ist nur ein Datenfeld angegeben (1 Sortierebene) und ist dieses indiziert, wird mit Index sortiert. Ist das Datenfeld nicht indiziert bzw. gibt es mehrere Datenfelder, wird sequentiell sortiert (mit Ausnahme von zusammengesetzten Indizes). Das Datenfeld kann zur Tabelle der Auswahl oder zu einer Eine-Tabelle gehören, die über eine automatische Verknüpfung mit *Tabellenname* verbunden ist. In diesem Fall ist die Sortierung immer sequentiell.
Sind die sortierten Felder in einem zusammengesetzten Index integriert, verwendet **ORDER BY** den Index der Sortierung.

Für mehrfache Sortierung (Sortierung mehrerer Felden) rufen Sie **ORDER BY** so oft, wie notwendig auf und geben den optionalen Parameter * an, mit Ausnahme des letzten Aufrufs, denn der startet die aktuelle Sortierung. Damit können Sie in individuell erstellten Benutzerumgebungen die Mehrfachsortierung verwalten.
Achtung: Mit dieser Syntax können Sie pro Aufruf von **ORDER BY** nur eine Sortierebene (Feld) übergeben.

Bei länger andauernden Sortierungen zeigt 4D automatisch eine Meldung mit Ablaufbalken an. Mit den Befehlen **MESSAGES ON**, **MESSAGES OFF** können Sie diese Anzeige an- bzw. abschalten. Erscheint der Ablaufbalken, kann der Benutzer die Sortierung mit der Schaltfläche **Stop** unterbrechen.

Läuft die Sortierung ohne Unterbrechung, nimmt die Systemvariable **OK** den Wert 1 an. Klickt der Benutzer auf die Schaltfläche **Abbrechen**, wird die Sortierung annulliert, die Systemvariable OK nimmt dann den Wert 0 (Null) an.

Hinweis: Dieser Befehl unterstützt keine Felder vom Typ Objekt.

Beispiel 1

Folgende Zeile zeigt den Sortiereditor für die Tabelle [Products]:

```
ORDER BY([Products])
```

Beispiel 2

Folgende Zeile zeigt den Sortiereditor für die Standardtabelle (sofern vorhanden)

```
ORDER BY
```

Beispiel 3

Folgende Zeile sortiert die Auswahl von [Products] nach Name in aufsteigender Reihenfolge:

```
ORDER BY([Products];[Products]Name;>)
```

Beispiel 4

Folgende Zeile sortiert die Auswahl von [Products] nach Name in absteigender Reihenfolge:

```
ORDER BY([Products];[Products]Name;<)
```

Beispiel 5

Folgende Zeile sortiert die Auswahl von [Products] nach Typ und Preis, beide Male in aufsteigender Reihenfolge:

```
ORDER BY([Products];[Products]Type;>;[Products]Price;>)
```

Beispiel 6

Folgende Zeile sortiert die Auswahl von [Products] nach Typ und Preis, beide Male in absteigender Reihenfolge:

```
ORDER BY([Products];[Products]Type;<;[Products]Price;<)
```

Beispiel 7

Folgende Zeile sortiert die Auswahl von [Products] nach Typ in aufsteigender Reihenfolge, nach Preis in absteigender Reihenfolge:

```
ORDER BY([Products];[Products]Type;>;[Products]Price;<)
```

Beispiel 8

Folgende Zeile sortiert die Auswahl von [Products] nach Typ in absteigender Reihenfolge, nach Preis in aufsteigender Reihenfolge:

```
ORDER BY([Products];[Products]Type;<;[Products]Price;>)
```

Beispiel 9

Folgende Zeile führt eine indizierte Sortierung aus, wenn [Products]Name indiziert ist:

```
ORDER BY([Products];[Products]Name;>)
```

Beispiel 10

Folgende Zeile sortiert die Auswahl von [Products] nach Name in aufsteigender Reihenfolge:

```
ORDER BY([Products];[Products]Name)
```

Beispiel 11

Folgende Zeile führt eine sequentielle Sortierung aus, egal ob die Datenfelder indiziert sind oder nicht:

```
ORDER BY([Products];[Products]Type;>;[Products]Price;>)
```

Beispiel 12

Folgende Zeile führt eine sequentielle Sortierung mit verknüpften Datenfeld aus:

```
ORDER BY([Invoices];[Companies]Name;>)
  ` Rechnungen werden alphabetisch nach dem Datenfeld Firmenname sortiert.
```

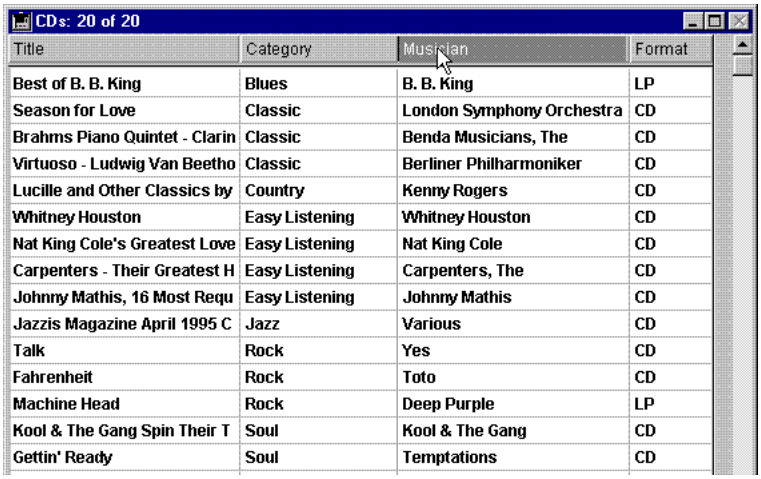
Beispiel 13

Folgendes Beispiel führt eine indizierte Sortierung auf zwei Ebenen aus, wenn in der Anwendung der zusammengesetzte Index [Contacts]LastName + [Contacts]FirstName angegeben ist.

```
ORDER BY([Contacts];[Contacts]LastName;>;[Contacs]FirstName;>)
```

Beispiel 14

In einem Ausgabeformular in der Anwendungsumgebung richten Sie ein, dass der Benutzer durch Klick in den Spaltentitel diese Spalte in aufsteigender Reihenfolge sortieren kann.
Klickt er bei gedrückter **Umschalttaste** in andere Spaltentitel, wird in mehreren Ebenen sortiert:



Title	Category	Musician	Format
Best of B. B. King	Blues	B. B. King	LP
Season for Love	Classic	London Symphony Orchestra	CD
Brahms Piano Quintet - Clarin	Classic	Benda Musicians, The	CD
Virtuoso - Ludwig Van Beetho	Classic	Berliner Philharmoniker	CD
Lucille and Other Classics by	Country	Kenny Rogers	CD
Whitney Houston	Easy Listening	Whitney Houston	CD
Nat King Cole's Greatest Love	Easy Listening	Nat King Cole	CD
Carpenters - Their Greatest H	Easy Listening	Carpenters, The	CD
Johnny Mathis, 16 Most Requ	Easy Listening	Johnny Mathis	CD
Jazzis Magazine April 1995 C	Jazz	Various	CD
Talk	Rock	Yes	CD
Fahrenheit	Rock	Toto	CD
Machine Head	Rock	Deep Purple	LP
Kool & The Gang Spin Their T	Soul	Kool & The Gang	CD
Gettin' Ready	Soul	Temptations	CD

Jeder Spaltentitel hat eine hervorgehobene Schaltfläche mit folgender Objektmethode:

```
MULTILEVEL(->[CDs]Titel)  `Name der Schaltfläche für Spaltentitel
```

Jede Schaltfläche ruft die Projektmethode MULTILEVEL mit einem Zeiger auf das entsprechende Spaltenfeld auf. Die Projektmethode lautet:

```
  ` Projektmethode MULTILEVEL
  ` MULTILEVEL (Zeiger)
  ` MULTILEVEL (->[Tabelle]Feld)

C_POINTER($1)  `Sortierebene (Feld)
C_LONGINT($lEbeneNr)

  `Erhalte Sortierebenen
If(Not(Shift down))  `Einfaches Sortieren (eine Ebene))
   ARRAY POINTER(aPtrSortFeld;1)
   aPtrSortFeld{1}:=$1
Else
   $lEbeneNr:=Find in array(aPtrSortFeld;$1)  `Ist dieses Feld schon sortiert?
   If($lEbeneNr<0)  `Wenn nicht
      INSERT ELEMENT(aPtrSortFeld;Size of array(aPtrSortFeld)+1;1)
      aPtrSortFeld{Size of array(aPtrSortFeld)}:=$1
   End if
End if
  `Führe Sortierung aus
$lEbeneNr:=Size of array(aPtrSortFeld)
If($lEbeneNr>0)  `Es gibt mindestens eine Sortierebene
   For($i;1;$lEbeneNr)
      ORDER BY([CDs];(aPtrSortFeld{$i})->;>;*)  `Erstelle eigene Sortierung
   End for
   ORDER BY([CDs])  `Ohne * endet die eigene Sortierung und die aktuelle Sortierung startet
End if
```

ORDER BY ATTRIBUTE

ORDER BY ATTRIBUTE ({Tabellenname ;} Objektfeld ; Attributpfad ; > oder < {; Objektfeld2 ; Attributpfad2 ; > oder <2 ; ... ; ObjektfeldN ; AttributpfadN ; > oder <N} {; *})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren ausgewählte Datensätze sortiert werden sollen Ohne Angabe Haupttabelle
Objektfeld	Objektfeld	➡	Objektfeld mit dem Attribut zum Sortieren
Attributpfad	String	➡	Name oder Pfad des Attributs zum Setzen der Sortierrichtung für jede Ebene
> oder <	Operator	➡	Sortierrichtung für jede Ebene: > aufsteigend, < absteigend
*	Operator	➡	Sortiermodus beibehalten

Beschreibung

Der Befehl **ORDER BY ATTRIBUTE** sortiert (reorders) die Datensätze der aktuellen Auswahl von *Tabellenname* für den aktuellen Prozess nach dem Inhalt des *Attributpfads* von *ObjektFeld*. Ist die Sortierung abgeschlossen, wird der erste Datensatz der Auswahl zum aktuellen Datensatz.

Geben Sie den Parameter *Tabellenname* nicht an, bezieht sich der Befehl auf die Standardtabelle, falls angegeben. Sonst verwendet 4D die Tabelle des ersten Feldes, das als Parameter übergeben ist.

In *Objektfeld* übergeben Sie das Objektfeld, dessen Attribut sie für die Sortierung verwenden wollen. Dieses Feld kann zu *Tabellenname* gehören oder zu einer damit verknüpften Tabelle mit automatischer oder manueller Verknüpfung. In diesem Fall ist die Sortierung immer sequentiell.

In *Attributpfad* übergeben Sie den Pfad des Attributs, dessen Werte Sie zum Sortieren der Datensätze verwenden wollen, z.B. "children.girls.age".

Hinweise:

- Nur Attribute mit skalaren Werten (Zahl, Text, Boolean, Datum) lassen sich sortieren. Andere Werte (Objekt, Bild, etc.) werden als Nullwerte gewertet.
- In *Attributpfad* können Sie kein Element eines Arrrays übergeben (in diesem Fall wird ein Fehler zurückgegeben).
- Beachten Sie, dass Attributnamen zwischen Groß- und Kleinschreibung unterscheiden: So wird im gleichen Datensatz zwischen "MyAtt" und "myAtt" unterschieden.

Enthält das Feldattribut Werte in verschiedenen Datentypen (z.B. Zahlen, Strings, Booleans), werden sie zuerst nach Typ, dann nach Wert gruppiert.

Ist der Wert des Feldattributs für einige Datensätze **Null** (z.B. der Attributwert ist Null oder *Attributpfad* existiert nicht im Feld) gilt folgendes:

- Bei **aufsteigender** Sortierung (>) werden Datensätze mit dem Wert **Null** an den Anfang der Auswahl gesetzt.
- Bei **absteigender** Sortierung (<) werden Datensätze mit dem Wert **Null** an das Ende der Auswahl gesetzt.

Sie können die Auswahl nach einer oder mehreren Ebenen sortieren. Für jede Sortierebene geben Sie ein *Objektfeld*, einen *Attributpfad* und die Sortierrichtung *> oder <* an. Mit dem Symbol "größer als" (>) wird in aufsteigender Richtung sortiert, mit dem Symbol "kleiner als" (<) in absteigender Richtung. Geben Sie keine Sortierrichtung an, wird standardmäßig in aufsteigender Richtung sortiert.
Ist nur ein Feld angegeben (eine Sortierebene) und es ist indiziert, wird der Index für die Sortierung verwendet. Ist das Feld nicht indiziert oder gibt es mehrere Felder, wird die Sortierung sequentiell ausgeführt.

Bei Mehrfachsortierung (Sortierung über mehrere Felder) können Sie **ORDER BY ATTRIBUTE** so oft wie erforderlich aufrufen und den optionalen Parameter * setzen, außer für den letzten Aufruf von **ORDER BY ATTRIBUTE**, der dann den Sortiervorgang startet. Dieses Feature ist hilfreich zum Verwalten von Mehrfachsortierungen in angepassten Benutzeroberflächen. Sie können auch Aufrufe von **ORDER BY ATTRIBUTE** mit Aufrufen von **ORDER BY** kombinieren.

Hinweis: Mit dieser Syntax können Sie nur eine Sortierebene (Feld) pro Aufruf von **ORDER BY ATTRIBUTE** übergeben.

Unabhängig von der Art der Sortierung zeigt 4D automatisch eine Meldung mit einem Ablaufbalken, wenn der Sortiervorgang eine Zeit lang andauert. Diese Meldung lässt sich über die Befehle **MESSAGES ON** und **MESSAGES OFF** an- und abschalten. Erscheint der Ablaufbalken, kann der Benutzer auf die Schaltfläche **Stop** klicken, um die Sortierung zu unterbrechen. Ist die Sortierung abgeschlossen, wird OK auf 1 gesetzt. Wird sie unterbrochen, wird OK auf 0 (Null) gesetzt.

Beispiel

Die aktuelle Auswahl nach Alter (absteigend) und dann nach Name (aufsteigend) sortieren. Die Eingabereihenfolge ist:

```
// [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}
```

Mit dieser Anweisung:

```
ORDER BY ATTRIBUTE([Customer];[Customer]OB_Info;"age";<;[Customer]OB_Info;"LastName";>)
```

werden Datensätze in folgender Reihenfolge sortiert:

{"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}, //Stringwerte in Alter {"LastName":"Belami","age":"Forty-six","client":true}, //werden separat verwaltet {"LastName":"Mikken","age":"Forty-six","client":true} {"LastName":"Gordini","client":true}, //stehen am Ende, da {"LastName":"Martin","client":true}, //Alter null ist (fehlt)

ORDER BY FORMULA

ORDER BY FORMULA (Tabellenname ; Formel {; >oder<}{; Formel2 ; >oder<2 ; ... ; FormelN ; >oder<N})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren aktuelle Auswahl sortiert werden soll
Formel	Ausdruck	➡	Sortierformel für jede Ebene
>oder<	Operator	➡	Sortierrichtung für jede Ebene: > aufsteigend, < absteigend

Beschreibung

Der Befehl **ORDER BY FORMULA** sortiert die aktuelle Auswahl des laufenden Prozesses von *Tabellenname*. Ist die Sortierung abgeschlossen, lädt 4D den ersten Datensatz der Auswahl in den Hauptspeicher. Er wird zum aktuellen Datensatz. **Hinweis:** Sie müssen *Tabellenname* angeben. Sie können nicht mit der Haupttabelle arbeiten.

Sie können die Auswahl nach einer oder mehreren Ebenen sortieren. Für jede Ebene geben Sie einen Ausdruck in *Formel* und die Sortierreihenfolge *> oder <* an. "Größer als" (>) sortiert in aufsteigender Reihenfolge, "Kleiner als" (<) in absteigender Reihenfolge. Ohne Festlegung wird in aufsteigender Reihenfolge sortiert.

Der Parameter *Formel* kann vom Typ Alphanumerisch, Zahl, Ganzzahl, Lange Ganzzahl, Datum, Zeit oder Boolean sein.

4D zeigt Ihnen den Sortierablauf während der Suche an. Wird das Ablaufthermometer angezeigt, kann der Benutzer die Sortierung mit der Schaltfläche **Stop** unterbrechen. Mit den Befehlen **MESSAGES ON**, **MESSAGES OFF** lässt sich diese Anzeige an- bzw. abschalten. Bricht der Anwender die Sortierung ab, wird OK auf 0 gesetzt, sonst auf 1.

4D Server: Dieser Befehl läuft auf dem Server, was seine Ausführung optimiert. Beachten Sie jedoch, dass die Sortierung bei direkt über *Formel* aufgerufenen Variablen mit dem Wert der Variablen des Client-Rechners berechnet werden.
Beispiel: die Anweisung **ORDER BY FORMULA([mytable];[mytable]myfield*myvariable)** wird auf dem Server ausgeführt, jedoch mit dem Inhalt der Variablen *myvariable* des Client-Rechners.

Hinweis zur Kompatibilität: Bis 4D Server v11 wurde **ORDER BY FORMULA** auf dem Client-Rechner ausgeführt. Zur Wahrung der Kompatibilität wird diese Arbeitsweise für konvertierte Datenbanken beibehalten. Es gibt jedoch eine Einstellung zur Kompatibilität oder einen Selektor im Befehl **SET DATABASE PARAMETER**, um für konvertierte Datenbanken die server-seitige Ausführung zu übernehmen.

Beispiel

Dieses Beispiel sortiert die Tabelle [People] in absteigender Reihenfolge nach der Länge der Nachnamen. Der längste Name erscheint in der aktuellen Auswahl an erster Stelle:

```
ORDER BY FORMULA([People];Length([People]Last Name);<)
```

QUERY

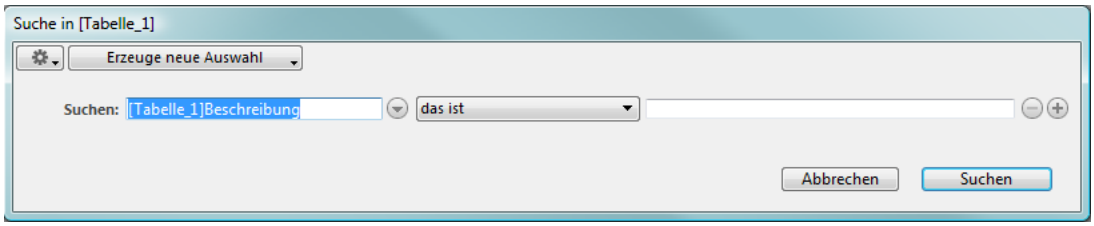
QUERY ({Tabellenname }{;}{ Suchbegriff {; *}})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, in der gesucht werden soll Ohne Angabe Haupttabelle
Suchbegriff	Ausdruck	➡	Operatoren und Suchbegriffe
*	Operator	➡	Weiterer Suchbegriff folgt

Beschreibung

Der Befehl **QUERY** sucht nach Datensätzen, welche die in *Suchbegriff* angegebenen Suchkriterien erfüllen und gibt die ausgewählten Datensätze für *Tabellenname* zurück. **QUERY** ändert die aktuelle Auswahl *Tabellenname* für den laufenden Prozess und macht den ersten Datensatz der neuen Auswahl zum aktuellen Datensatz.
Tabellenname ist optional. Wird dieser Parameter nicht angegeben, bezieht sich *Tabellenname* auf die Haupttabelle. Gibt es keine Haupttabelle, erhalten Sie eine Fehlermeldung.

Geben Sie *Suchbegriff* oder den optionalen Parameter * nicht an, zeigt **QUERY** den Sucheditor für *Tabellenname*, außer es ist die letzte Zeile einer komplexen Suche (siehe Beispiel 2). Der Sucheditor sieht folgendermaßen aus:



Weitere Informationen dazu finden Sie im Abschnitt **Sucheditor** des Handbuchs *4D Designmodus*.

Der Benutzer baut die Suche auf und klickt dann auf die Schaltfläche **Suche** oder wählt im DropDown-Menü links oben den Eintrag **Suche in Auswahl**, um die Suche auszuführen. Läuft die Suche ohne Unterbrechung, wird die Systemvariable OK auf 1 gesetzt. Klickt der Benutzer auf die Schaltfläche **Abbrechen**, beendet **QUERY** ohne eine Suche auszuführen und setzt die Systemvariable OK auf 0 (Null).

Beispiel 1

Folgende Zeile zeigt den Sucheditor für die Tabelle [Products] an:

```
QUERY([Products])
```

Beispiel 2

Folgende Zeile zeigt den Sucheditor für die Haupttabelle an (sofern vorhanden)

```
QUERY
```

Geben Sie den Parameter *Suchbegriff* an, erscheint nicht der Standardsucheditor, die Suche wird per Programmierung definiert. Für einfache Suchläufe (Suche nach nur einem Datenfeld) rufen Sie **QUERY** einmal mit *Suchbegriff* auf. Für Suchsequenzen (Suche nach mehreren Datenfeldern oder Kriterien) rufen Sie **QUERY** mit *Suchbegriff* so oft wie nötig auf und geben jedes Mal den optionalen Parameter * an, außer für den letzen Aufruf von **QUERY**. Dieser startet dann den Suchlauf. Weitere Informationen zum Parameter *Suchbegriff* finden Sie nachfolgend.

Beispiel 3

Folgende Zeile sucht in der Tabelle [People] nach Nachnamen, die mit "a" beginnen:

```
QUERY([People];[People]Last name="a@")
```

Beispiel 4

Folgende Zeile sucht in der Tabelle [People] nach Namen, die mit "a" oder "b" beginnen:

```
QUERY([People];[People]Name="a@";*)  ` * weist auf weitere Suchkriterien hin
QUERY([People];l;[People]Name="b@")
  ` Ohne * endet die Definition der Suche, der Suchlauf startet
```

Hinweis: Die Interpretation des Zeichens @ in Suchläufen lässt sich über eine Option in den Voreinstellungen ändern. Weitere Informationen dazu finden Sie im Abschnitt **Vergleichsoperatoren**.

Suchbegriff definieren

Einen Suchbegriff definieren Sie nach folgendem Modell:

{Operator ;} Datenfeld Vergleichsoperator Wert

Der Operator ist das Verbindungsglied zwischen zwei Suchbegriffen. Bei einem einzelnen Suchbegriff benötigen Sie keinen Operator. Es gibt folgende Operatoren:

Operator	**Symbol**
UND	&
ODER	|
Außer	#

Der Operator ist optional und wird bei einer einfachen oder mehrfachen Suche nicht für den ersten Aufruf von **QUERY** verwendet. Lassen Sie ihn in einer mehrfachen Suche weg, wird standardmäßig UND (&) verwendet.

- *Datenfeld* ist das Datenfeld, auf das sich die Suche bezieht. Das Feld kann aus einer anderen Tabelle stammen, wenn es zur Eine-Tabelle gehört, die über eine automatische oder manuelle Verknüpfung mit *Tabellenname* verknüpft ist.
- *Vergleichsoperator* vergleicht den Wert im Datenfeld mit dem angegebenen Wert. Es gibt folgende Vergleichsoperatoren:

Vergleichsoperator	**Symbol**
ist gleich	=
ist nicht gleich	#
ist größer als	>
ist größer als oder gleich	>=
ist kleiner als	<
ist kleiner als oder gleich	<=
Enthält Schlüsselwort	%

Sie können den Vergleichsoperator auch als alphanumerischen Ausdruck anstatt als Symbol definieren. In diesem Fall müssen Sie die einzelnen Ausdrücke in der Suchkette durch Strichpunkt voneinander trennen. Auf diese Weise können Sie z.B. durch Variieren des Vergleichsoperators konfigurierbare Suchläufe erstellen oder individuell auf den Benutzer abgestimmte Suchoberflächen. Weitere Informationen dazu finden Sie im Beispiel 21.

Wert wird mit dem Wert in Datenfeld verglichen. Der Wert kann jeder Ausdruck sein, der denselben Datentyp wie Datenfeld bewertet. Der Typ von *Wert* wird nur einmal, und zwar bei Beginn der Suche bewertet und nicht für jeden Datensatz. Suchen Sie eine Zeichenkette, die eine bestimmte Buchstabenkombination enthält (eine Suche mit "enthält"), setzen Sie die Buchstaben zwischen zwei Klammeraffen @, beispielsweise @mburg@. Beachten Sie, dass solch eine Suche wegen der kompakten Datenspeicherung nur teilweise vom Index profitiert.
Eine Enthält-Suche ist nur für Datenfelder vom Typ Alpha oder Text verfügbar. Weitere Informationen dazu finden Sie im Abschnitt **Vergleichsoperatoren**.

Folgende Regeln müssen Sie bei der Programmierung einer Suche beachten:

- In der ersten Suchzeile darf kein Operator stehen.
- In den folgenden Zeilen kann der Suchbegriff mit einem Operator beginnen. Lassen Sie ihn weg, wird standardmäßig UND (&) verwendet.
- Alle Zeilen, mit Ausnahme der letzten, müssen mit * beendet werden. * zeigt an, dass noch weitere Suchbegriffe folgen. Das Programm sucht erst, wenn nach einem Suchbegriff kein Stern angegeben ist.
- Die Definition der Suchbegriffe darf sich über mehrere Datenfelder in mehreren Tabellen erstrecken. Der Tabellenname muss immer im ersten Parameter angegeben sein.
  Der Befehl **QUERY** ohne bzw. mit *Tabellenname* als einzigem Argument ruft den Standardsucheditor auf. Wurde zuvor **QUERY** mit * aufgerufen, wird diese nun ausgeführt. Dadurch können mehrere Suchabläufe in einer Schleife zusammengesetzt werden, die dann mit **QUERY** ohne * abgeschlossen werden.

Hinweis: Jede Tabelle behält ihre eigene aktuell festgelegte Suche. So können Sie gleichzeitig komplexe Suchläufe erstellen, und zwar eine pro Tabelle. Zum Festlegen der Tabelle müssen Sie den Parameter *Tabellenname* verwenden oder die Standardtabelle setzen.

Egal, wie eine Suche definiert wurde, gilt folgendes:

- Bei länger dauernden Suchläufen zeigt 4D automatisch eine Meldung mit einem Ablaufbalken. Mit den Befehlen **MESSAGES ON**, **MESSAGES OFF** können Sie diese Anzeige an- bzw. abschalten. Wird das Ablaufthermometer angezeigt, kann der Benutzer die Suche mit der Schaltfläche **Stop** unterbrechen. Ist die Suche abgeschlossen, wird OK auf 1 gesetzt. Wird die Suche abgebrochen, wird OK auf 0 (Null) gesetzt.
- Sind indizierte Datenfelder angegeben, wird zuerst nach diesen Datenfeldern gesucht. So läuft die Suche bei indizierten Feldern optimiert ab. d.h. sie benötigt am wenigsten Zeit. Bei Suchläufen mit AND (&) verwendet der Befehl zusammengesetzte Indizes.

Beispiel 5

Sie suchen in der Tabelle [People] alle Personen mit dem Namen Maier. Das Datenfeld Name ist indiziert. Die Suche verläuft daher sehr schnell:

```
QUERY([People];[People]Last Name="Maier")
```

Zur Erinnerung: Diese Suche unterscheidet nicht zwischen “Maier”, “maier” und “MAIER”, etc. Wollen Sie Groß- und Kleinschreibung voneinander unterscheiden, verwenden Sie die Zeichen-Codes für eine genauere Suche.

Beispiel 6

Dieses Beispiel sucht in der Tabelle [People] alle Personen mit dem Namen Franz Maier. Das Datenfeld Last Name ist indiziert, das Datenfeld First Name nicht.

```
QUERY([People];[People]Last Name="maier";*) ` Finde alle Personen mit dem Nachnamen Maier
QUERY([People]; & ;[People]First Name="franz") ` und dem Vornamen Franz
```

Beim Ausführen der Suche erfolgt eine indizierte Suche auf das Datenfeld Last Name, die Datensatzauswahl wird auf die Personen mit Namen Maier eingeschränkt. Dann wird in dieser Auswahl sequentiell nach dem Vornamen Franz gesucht. Der Zeitverlust ist daher sehr gering.

Beispiel 7

Dieses Beispiel nutzt automatisch die Vorteile des zusammengesetzten Indexes aus den Feldern *[People]First Name*+ *[People]Last Name* (falls vorhanden) zur Suche der Datensätze für alle Personen mit Namen Franz Maier.

```
QUERY([People];[People]First Name="franz";*) ` Suche alle Personen mit Vorname Franz
QUERY([People];&;[People]Last Name="maier") ` mit Maier als Nachname
```

Weitere Informationen dazu finden Sie unter **Zusammengesetzter Index**.

Beispiel 8

Dieses Beispiel sucht nach den Namen Maier oder Huber. Das Datenfeld *Last Name* ist indiziert und wird für beide Suchläufe verwendet. Die Ergebnisse werden in interne Mengen gelegt und evtl. über eine Vereinigung kombiniert.

```
QUERY([People];[People]Last Name="maier";*)
  ` Finde alle Personen mit Nachnamen Maier···
QUERY([People];|;[People]Last Name="huber") ` ...oder Huber
```

Beispiel 9

Folgendes Beispiel findet alle Personen ohne Firmenname. Gesucht wird nach Eingaben mit leeren Datenfeldern (leerer String).

```
QUERY([People];[People]Company="") ` Finde alle Personen ohne Firma
```

Beispiel 10

Folgendes Beispiel findet alle Personen mit dem Nachnamen Maier, die in einer Firma in München arbeiten. Der zweite Suchlauf kann ein Datenfeld aus einer anderen Tabelle verwenden, da die Tabelle [People] durch eine Viele-zu-Eine Verknüpfung mit der Tabelle [Company] verbunden ist:

```
QUERY([People];[People]Last Name="Maier";*)
  ` Finde alle Personen mit Nachnamen Maier,
QUERY([People]; & ;[Company]City="München")
  ` die in einer Firma in München arbeiten.
```

Beispiel 11

Folgendes Beispiel findet alle Personen mit einem Nachnamen zwischen A (inkl.) und M (inkl.):

```
QUERY([People];[People]Name<"n") ` Finde alle Personen von A bis M
```

Beispiel 12

Folgendes Beispiel findet alle Personen, die im Raum München oder Nürnberg wohnen (Postleitzahlengebiet 8 oder 9):

```
QUERY([People];[People]PLZ Code ="8@";*)
  ` Finde alle Personen im Raum München···
QUERY([People];|;[People]PLZ Code ="9@") ` ...oder Nürnberg
```

Beispiel 13

Volltextsuche: Folgendes Beispiel such in der Tabelle [Products] nach Datensätzen, die im Feld Beschreibung das Wort “einfach” enthalten:

```
QUERY([Products];[Products]Description%"einfach")
  ` Finde Produkte, deren Beschreibung das Wort einfach enthalten.
```

Beispiel 14

Folgendes Beispiel sucht nach einer bestimmten Rechnungsnummer:

```
vFind:=Request("Finde Rechnungsnummer:")
  ` Erhalte Rechnungsnummer vom Benutzer
[If (OK=1) ` Hat der Benutzer auf OK geklickt,
QUERY([Invoice];[Invoice]Number=vFind)
  ` Finde Rechnungsnummer der Variablen vFind
End if
```

Beispiel 15

Folgendes Beispiel findet alle 1997 ausgestellten Rechnungen. Gesucht wird nach den Datensätzen, die nach dem 31.12.96 und vor dem 1.1.98 eingegeben wurden:

```
QUERY([Invoice];[Invoice]In Date>!31.12.96!;*)
  ` Finde Rechnungen nach 31.12.96…
QUERY([Invoice]; & ;[Invoice]In Date<!1.1.98!) ` und vor 1.1.98
```

Beispiel 16

Folgendes Beispiel findet alle Beschäftigten, deren Gehalt zwischen 1.000 Euro und 3.000 Euro liegt. Personen mit einem Gehalt gleich 1.000 Euro sollen in der Auswahl enthalten sein. Personen, deren Gehalt gleich 3.000 Euro ist, sollen nicht mehr enthalten sein:

```
QUERY([Employee];[Employee]Salary >=1000;*)
  ` Finde Beschäftige mit einem Gehalt zwischen…
QUERY([Employee]; & ;[Employee]Salary <3000) ` … 1.000 Euro und 3.000 Euro
```

Beispiel 17

Folgendes Beispiel findet alle Beschäftigten in der Marketing-Abteilung, deren Gehalt über 4.000 Euro liegt. Es wird zuerst im Datenfeld Salary gesucht, das es indiziert ist. Die zweite Suche verwendet ein Datenfeld aus einer anderen Tabelle. Das ist möglich, da die Tabellen [Employee] und [Dept]über eine automatische Viele-zu-Eine Verknüpfung verbunden sind:

```
QUERY([Employee];[Employee]Salary >4000;*)
  ` Finde Beschäftigte mit einem Gehalt über 4.000 Euro und…
QUERY([Employee]; & ;[Dept]Name="Marketing") ` … die in der Marketingabteilung sind.
```

Beispiel 18

Es gibt drei Tabellen, die über eine Viele-zu-Eine Verknüpfung verbunden sind: [City] -> [Department] -> [Region]. Die folgende Suche findet alle Regionen mit Städten, deren Name mit "Sankt" beginnt:

```
QUERY([Region];[City]Name="Sankt@") ` Finde alle Regionen mit Städten, die mit "Sankt" beginnen.
```

Beispiel 19

Folgendes Beispiel sucht nach Informationen aus der Variablen *myVar*.

```
QUERY([Laws];[Laws]Text =myVar) ` Finde alle in myVar enthaltenen Gesetze
```

Die Suche kann, je nach dem Wert von *myVar*, ganz verschiedene Ergebnisse liefern. Zum Beispiel:

- Ist *myVar* gleich*"Copyright@"*, enthält die Auswahl alle Gesetze, die mit Copyright beginnen.
- Ist *myVar* gleich *"@Copyright@"*, enthält die Auswahl alle Gesetze, die das Wort Copyright enthalten.

Beispiel 20

Dieses Beispiel fügt je nach Wert der Variablen in einer komplexen Suche Zeilen hinzu oder nicht. So werden nur die für die Suche gültigen Kriterien berücksichtigt:

```
QUERY([Invoice];[Invoice]Paid=False;*)
If($city#"") ` Wurde eine Stadt angegeben.
```

```
    QUERY([Invoice];[Invoice]Delivery_city=$city;*)
End if
If($zipcode#"") ` Wurde eine Postleitzahl angegeben.
    QUERY([Invoice];[Invoice]ZipCode=$zipcode;*)
End if
QUERY([Invoice]) ` Ausführung der Suche nach dem angegebenen Kriterium
```

Beispiel 21

Dieses Beispiel zeigt die Verwendung des Vergleichsoperators als alphanumerischen Ausdruck. Sein Wert wird über ein PopUp-Menü definiert, das in einem eigenen Suchdialog angelegt wurde:

```
C_TEXT($oper)
$oper:=_popup_operator{_popup_operator} `$oper entspricht z.B. "#" oder "="
If(OK=1)
    QUERY(Invoice];[Invoice]Amount;$oper;$amount)
End if
```

Beispiel 22

Der Einsatz von Volltext-Indizes für Bilder kann Ihre Anwendungen signifikant beschleunigen:

```
QUERY([PICTURES];[PICTURES]Photos %"Katzen") // Suche nach Fotos mit dem Schlüsselwort "Katzen"
```

Systemvariablen und Mengen

Wurde die Suche korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt.
Die Variable OK wird auf 0 (Null) gesetzt, wenn:

- Der Benutzer im Suchdialog auf die Schaltfläche **Abbrechen** klickt,
- Die Suche im Modus 'Suchen und Sperren' (siehe Befehl **SET QUERY AND LOCK**) mindestens einen gesperrten Datensatz gefunden hat. In diesem Fall wird auch die Systemmenge **LockedSet** aktualisiert.

QUERY BY ATTRIBUTE

QUERY BY ATTRIBUTE ({Tabellenname}{;}{KonjOp ;} ObjektFeld ; AttributPfad ; SuchOp ; Wert {; *})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zur Rückgabe einer Auswahl Datensätze Ohne Angabe Haupttabelle
KonjOp	Operator	➡	Verbindungsparameter bei mehrfacher Suche
ObjektFeld	Feld	➡	Objektfeld zum Suchen von Attributen
AttributPfad	String	➡	Name oder Pfad des Attributs
SuchOp	String, Operator	➡	Suchoperator (Vergleichsoperator)
Wert	Text, Zahl, Datum, Zeit	➡	Zu vergleichender Wert
*	Operator	➡	Weiterer Suchbegriff folgt

Beschreibung

QUERY BY ATTRIBUTE sucht nach Datensätzen, die zum Suchstring passen, definiert über die Parameter *ObjektFeld*, *AttributPfad*, *SuchOp* und *Wert*, und gibt eine Datensatzauswahl für *Tabellenname* zurück.

QUERY BY ATTRIBUTE ändert die aktuelle Auswahl von *Tabellenname* für den aktuellen Prozess und macht den ersten Datensatz der neuen Auswahl zum aktuellen Datensatz. Wird der Parameter *Tabellenname* weggelassen, gilt der Befehl für die Standardtabelle. Ist keine Standardtabelle vorhanden, tritt ein Fehler auf.

Der optionale Parameter *KonjOp* verbindet **QUERY BY ATTRIBUTE** Aufrufe in mehrfachen Suchen. Die logischen Operatoren sind dieselben wie im Befehl **QUERY**:

Konjunktion	Symbol für QUERY BY ATTRIBUTE
UND	&
ODER	|
Außer	#

Der Parameter *KonjOp* wird nicht bei einer einfachen Suche bzw. für den ersten Aufruf von **QUERY BY ATTRIBUTE** in einer mehrfachen Sucheverwendet. Lassen Sie ihn in einer mehrfachen Suche weg, wird standardmäßig der Operator UND (&) verwendet.

In *ObjektFeld* übergeben Sie das Objektfeld, nach dessen Attribut(en) Sie suchen wollen. Gehört es zu einer Eine-Tabelle, definiert in *Tabellenname* mit einer automatischen oder manuellen Verknüpfung, kann *ObjektFeld* zu einer anderen Tabelle gehören.
QUERY BY ATTRIBUTE unterstützt eigene Attribute in 4D Write Pro, wenn Dokumente in Objektfeldern gespeichert sind. Weitere Informationen dazu finden Sie im Abschnitt **4D Write Pro Dokumente in 4D Objektfeldern speichern**.

In *AttributPfad* übergeben Sie den Pfad des Attributs, dessen Werte Sie für jeden Datensatz vergleichen wollen, z.B. "Kinder.weiblich.Alter". Übergeben Sie eine einzelne Bezeichnung, z.B. "Ort", geben Sie das entsprechende Attribut an, das auf der ersten Ebene des Objektfelds gefunden wird.
Ist ein Attribut "x" ein Array, sucht **QUERY BY ATTRIBUTE** nach Datensätzen mit einem Attribut "x", in dem mindestens ein Element zum Suchkriterium passt. Zur Suche in Array Attributen muss im Befehl **QUERY BY ATTRIBUTE** angegeben werden, dass Attribut "x" ein Array ist, d.h. an den Namen in *AttributPfad* wird "[]" angefügt (siehe Beispiel 3).
Sie können einen Buchstaben in Klammern hinzufügen, z.B. [b] um Argumente zu verlinken (siehe nachfolgenden Abschnitt **Suchkriterien mit Array Attributen verlinken**).

Hinweise:

- Beachten Sie, dass Attributnamen Groß- und Kleinschreibung berücksichtigen. Im gleichen Datensatz wird also zwischen dem Attributnamen "MeinAtt" und "meinAtt" unterschieden.
- Mehrere Attributnamen werden zusammengezogen. Beispiel: " mein erstes Attribut .mein zweites Attribut " wird interpretiert als "mein erstes Attribut.mein zweites Attribut".

Der Parameter *SuchOp* ist der Vergleichsoperator zwischen *ObjektFeld* und *Wert*. Sie können folgende Symbole übergeben:

Vergleich	Symbol für QUERY BY ATTRIBUTE
Ist gleich	=
Ist ungleich *	#
Kleiner als	<
Größer als	>
Kleiner als oder gleich	<=
Größer als oder gleich	>=

(*) Bei Verwendung mit Array Elementen bedeutet der Operator # "enthält kein Element".

Hinweis: Sie können den Vergleichsoperator auch in Textform angeben. Weitere Informationen dazu finden Sie unter dem Befehl **QUERY**.

Der Inhalt von *Wert* wird mit *AttributPfad* verglichen. *Wert* kann jeder Ausdruck sein, der denselben Datentyp wie *AttributPfad* bewertet. Er wird einmal zu Beginn der Suche bewertet und nicht für jeden Datensatz. Für eine Suche nach einem String innerhalb eines anderen (Suche mit "enthält") setzen Sie in *Wert* den Joker (@), um den gesuchten String zu isolieren, also z.B. "@Schmid@". In diesem Fall profitiert die Suche jedoch nur teilweise von dem Index (Kompaktheit der Datenspeicherung).

Die Struktur einer Suche nach Attribut lautet wie folgt:

```
QUERY BY ATTRIBUTE([Tabellenname];[Tabellenname]ObjektFeld;"Attribut1.Attribut2";=;Wert)
```

Beachten Sie, dass immer davon ausgegangen wird, dass das Feld ein nicht-leeres Objekt mit festgelegten Attributen enthält. Nachfolgendes Beispiel gibt alle Datensätze zurück, in denen *[Tabellenname]ObjektFeld* ein Objekt mit einem Attribut *Attribut1* enthält, das wiederum ein Objekt mit einem Attribut *Attribut2* ist, dessen Wert ungleich *Wert* ist:

```
QUERY BY ATTRIBUTE([Tabellenname];[Tabellenname]ObjektFeld;"Attribut1.Attribut2";#;Wert)
```

Deshalb gibt diese Suche nicht Objekte zurück, die weder *Attribut1* noch *Attribut2* enthalten.

Operator # verwenden (Unterstützung für Nullwerte)

Suchen nach Attribut mit dem Operator "#" können unterschiedliche Ergebnisse haben, je nachdem ob für das Objektfeld die Eigenschaft markiert ist:

- **NULL Werten leere Werte zuweisen** ist markiert (Standardeinstellung, wird für die meisten Fälle empfohlen).
  In diesem Fall wählt der Operator # Datensätze aus, für die kein Attribut des Feldes auf den angegebenen Wert zutrifft. In ähnlicher Weise bewertet 4D:
  - Felder, deren Wert des Attributs unterschiedlich ist vom angegebenen Wert
  - Felder, deren Wert des Attributs nicht vorhanden ist (oder einen Null Wert enthält)

  Beispiel 1: Diese Suche gibt Datensätze mit den Personen zurück, die einen Hund haben, dessen Name nicht Rex ist und Datensätze mit Personen, die keinen Hund oder einen Hund ohne Namen haben:

  ```
  QUERY BY ATTRIBUTE([People];[People]Animals;"dog.name";#;"Rex")
  ```

  Beispiel 2: Diese Suche gibt Datensätze zurück, für die *[Table]ObjectField* ein Objekt mit einem Attribut *Attribute1* enthält, das selbst ein Objekt mit einem Attribut *attribute2* ist, dessen Wert ungleich *value* ist und Datensätze, deren Objektfelder nicht *attribute1* oder *attribute2* enthalten:

  ```
  QUERY BY ATTRIBUTE([Table] ;[Table]ObjectField ;"attribute1.attribute2";#;value)
  ```

  Beispiel 3: Dieses Prinzip gilt auch bei Array Attributen. Diese Suche gibt Datensätze der Personen zurück, die keine Adresse in Paris haben:

  ```
  QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[].city";#;"paris")
  ```

  Wollen Sie spezifisch Datensätze mit undefiniertem Attribut erhalten, können Sie ein leeres Objekt verwenden (siehe unten Beispiel 2). Beachten Sie jedoch, dass die Suche nach NULL Werten in Array Elementen nicht unterstützt wird.

- **NULL Werten leere Werte zuweisen** ist nicht markiert ("SQL" Modus).
  In diesem Fall werden undefinierte Attribute (im Feld nicht vorhandene Attribute oder mit dem Wert Null) standardmäßig als leere Werte gewertet. Dann geben Suchen vom Typ "Attribut A unterschiedlich von Attribut B" keine Datensätze mit undefiniertem Attribut A zurück.
  Nehmen wir dasselbe Beispiel wie oben, wenn die Eigenschaft **NULL Werten leere Werte zuweisen** für das Feld [People]Animals nicht markiert ist, liefert die Suche nur Datensätze für Personen, die einen Hund haben, dessen Attribut "Name" nicht Rex enthält. Es werden keine Datensätze mit Personen, die keinen Hund oder einen Hund ohne Namen haben, zurückgegeben.

  ```
  QUERY BY ATTRIBUTE([People];[People]Animals;"dog.name";#;"Rex")
  ```

  Diese Operation ist der SQL Logik näher und für spezifische Zwecke reserviert.

Mehrfache Suchen einrichten

Für mehrfache Suchen nach Attributen gelten folgende Regeln:

- Das erste Suchkriterium darf keinen Verknüpfungsoperator enthalten.
- Jedes nachfolgende Kriterium kann mit einem Verknüpfungsoperator beginnen. Lassen Sie ihn weg, wird standardmäßig der Operator UND (&) verwendet.
- Die einzelnen Suchaufrufe müssen den Parameter * verwenden. Der letzte Aufruf benötigt keinen *.
- **QUERY BY ATTRIBUTE** kann mit **QUERY** Befehlen gemischt werden.
- Um die Suche auszuführen, lassen Sie im letzten Aufruf von **QUERY BY ATTRIBUTE** den Parameter * weg. Als Alternative können Sie auch den Befehl **QUERY** nur mit der Tabelle ausführen, d.h. ohne die anderen Parameter (der Sucheditor erscheint nicht; an seiner Stelle wird die gerade definierte mehrfache Suche ausgeführt).

Hinweis: Jede Tabelle besitzt ihre eigenen aktuellen Suchläufe. Sie können also mehrere Suchläufe gleichzeitig anlegen, d.h. eine pro Tabelle. Für die eindeutige Zuordnung müssen Sie den Parameter *Tabellenname* angeben oder die Standardtabelle setzen.

Egal, auf welche Weise eine Suche definiert wurde, gilt folgendes:

- Dauert die Ausführung der aktuellen Suchoperation etwas länger, zeigt 4D automatisch eine Meldung mit einem Ablaufbalken. Sie lässt sich über die Befehle **MESSAGES ON** und **MESSAGES OFF** ein- und ausschalten. Erscheint ein Ablaufbalken, kann der Benutzer bei Bedarf auf die Schaltfläche Stopp klicken, um die Suche zu unterbrechen. Ist die Suche abgeschlossen, wird OK auf 1 gesetzt; wird sie unterbrochen, wird OK auf 0 (Null) gesetzt.
- Sind indizierte Objektfelder spezifiziert, wird die Suche, immer wenn möglich, optimiert (nach indizierten Feldern wird zuerst gesucht) und reduziert so den Zeitaufwand.

Datumswerte in Objekt

Datumsangaben werden in Objekten gemäß den Einstellungen der Datenbank gespeichert; standardmäßig wird die Zeitzone berücksichtigt (siehe Selector JSON use local time unter dem Befehl **SET DATABASE PARAMETER**).

!1973-05-22! -> "1973-05-21T23:00:00.000Z"

Diese Einstellung wird auch beim Suchen berücksichtigt. Sie müssen sich nicht darum kümmern, wenn Sie Ihre Anwendung immer am gleichen Ort verwenden und die Einstellungen auf allen Rechnern, die auf die Daten zugreifen, gleich sind. Folgende Suche gibt dann z.B. Datensätze mit dem Attribut Geburtsdatum ist gleich !1973-05-22! (gesichert als "1973-05-21T23:00:00.00Z") korrekt zurück. Dieses Beispiel gilt für Deutschland GMT Winterzeit - 1 h: 22.Mai 1973 0:00 ist somit 21.Mai 1973 23:00:

```
QUERY BY ATTRIBUTE([Persons];[Persons]OB_Info;"Geburtsdatum";=;!1973-05-22!)
```

Wollen Sie nicht die GMT Einstellungen verwenden, schreiben Sie folgende Anweisung:

```
SET DATABASE PARAMETER(JSON use local time;0)
```

Beachten Sie, dass die Reichweite dieser Einstellung nur der Prozess ist. Führen Sie diese Anweisung aus, wird 1. Oktober 1965 auch als "1965-10-01T00:00:00.000Z" gespeichert. Sie müssen denselben Parameter jedoch vor dem Starten Ihrer Suchen setzen:

```
SET DATABASE PARAMETER(JSON use local time;0)
QUERY BY ATTRIBUTE([Persons];[Persons]OB_Info;"Geburtsdatum";=;!1976-11-27!)
```

Virtuelle Eigenschaft "length"

Mit diesem Befehl können Sie die virtuelle Eigenschaft "length" verwenden. Sie ist automatisch für alle Attribute vom Typ Array verfügbar und gibt die Größe des Array zurück, z.B. die Anzahl der darin enthaltenen Elemente. Sie lässt sich beim Ausführen des Befehls **QUERY BY ATTRIBUTE** verwenden (siehe Beispiel 4).

Suchkriterien mit Array Attributen verlinken

(Neu in 4D v16 R2) Beim Suchen in Array Attributen mit mehreren Suchkriterien, verbunden durch den Operator UND, möchten Sie sicherstellen, dass nur Datensätze mit Elementen, auf die alle Kriterien zutreffen, zurückgegeben werden. Dazu müssen Sie die Suchkriterien mit Array Elementen *verlinken*, damit nur die Elemente mit verlinkten Kriterien gefunden werden.

Nehmen wir als Beispiel folgende beiden Datensätze:

Datensatz 1:

```
[People]name: "martin"
[People]OB_Field:
   "locations" : [ {
            "kind":"home",[People]
            "city":"paris"
         } ]
```

Datensatz 2:

```
[People]name: "smith"
[People]OB_Field:
   "locations" : [ {
            "kind":"home",
            "city":"lyon"
         } , {
            "kind":"office",
            "city":"paris"
         } ]
```

Sie möchten Personen finden mit der Adressenart "home" in der Stadt "paris". Schreiben Sie:

```
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[].city";=;"paris";*)
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[].kind";=;"home")
```

... gibt die Suche "martin" und "smith" zurück, da "smith" ein Element "locations" hat mit "kind" ist gleich "home" und ein Element "locations" mit "city" ist gleich "paris", selbst wenn die Elemente unterschiedlich sind.

Wollen Sie nur Datensätze erhalten mit den passenden Kriterien im gleichen Element, müssen Sie die **Kriterien verlinken**. Gehen Sie folgendermaßen vor:

- Sie setzen einen Buchstaben zwischen die [] im ersten Pfad, um den gleichen Buchstaben in allen verbundenen Kriterien zu verlinken und zu wiederholen. Zum Beispiel: **locations[a].city** und **locations[a].kind**. Sie können jeden beliebigen Klein- oder Großbuchstaben des lateinischen Alphabeths verwenden.
- Um ein weiteres verbundenes Kriterium in derselben Suche hinzuzufügen, verwenden Sie einen anderen Buchstaben (siehe Beispiele unten). Sie können in einer Suche bis zu 26 Kombinationen verwenden.

Wieder mit den o.a. Datensätzen schreiben Sie nun:

```
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[a].city";=;"paris";*)
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[a].kind";=;"home")
```

... gibt die Suche nur "martin" zurück, da sie ein Element "locations" hat mit "kind" ist gleich "home" und "city" ist gleich "paris". Die Suche gibt nicht "smith" zurück, da die Werte "home" und "paris" nicht im gleichen Array Element sind. Das Beispiel 3 veranschaulicht dieses Feature.

Hinweis: Eine verbundene Syntax in einer einzelnen Suchzeile führt zum gleichen Ergebnis wie eine Standardsuche, außer beim Verwenden des Operators "#": Da dies zu ungültigen Ergebnissen führen kann, wird diese spezifische Syntax nicht unterstützt.

Beispiel 1

In diesem Beispiel ist das Attribut "Alter" entweder ein String oder eine Ganzzahl. Wir wollen Personen im Alter 20 bis 29 finden. Die beiden ersten Zeilen suchen das Attribut als Ganzzahl (>=20 und < 30), die letzte Zeile sucht das Feld als String (startet mit "2", ist aber ungleich "2")

```
QUERY BY ATTRIBUTE([Persons];[Persons]OB_Info;"Alter";>=;20;*)
QUERY BY ATTRIBUTE([Persons]; & ;[Persons]OB_Info;"Alter";<;30;*)
QUERY BY ATTRIBUTE([Persons];I;[Persons]OB_Info;"Alter";=;"2@";*)
QUERY BY ATTRIBUTE([Persons]; & ;[Persons]OB_Info;"Alter";#;"2") // zuletzt kein * setzen, um die Ausführung zu starten
```

Beispiel 2

Mit dem Befehl **QUERY BY ATTRIBUTE** können Sie auch nach Datensätzen suchen, die bestimmte Attribute haben bzw. nicht haben. Dazu müssen Sie ein leeres Objekt verwenden:

```
//Finde Datensätze, für die im Objektfeld email definiert ist
C_OBJECT($undefined)
QUERY BY ATTRIBUTE([Persons];[Persons]Info;"email";#;$undefined)
```

```
//Finde Datensätze, für die im Objektfeld PLZ NICHT definiert ist
C_OBJECT($undefined)
QUERY BY ATTRIBUTE([Persons];[Persons]Info;"PLZ";=;$undefined)
```

Hinweis: Diese spezifische Syntax wird nicht in Attributen vom Typ Array unterstützt. Die Suche nach NULL Werten in Array Elementen liefert ein ungültiges Ergebnis.

Beispiel 3

Nach einem Feld mit Array Attributen suchen. Mit den beiden folgenden Datensätzen:

Datensatz 1:

```
[People]name: "martin"
[People]OB_Field:
   "locations" : [ {
            "kind":"office",
            "city":"paris"
         } ]
```

Datensatz 2:

```
[People]name: "smith"
[People]OB_Field:
   "locations" : [ {
            "kind":"home",
            "city":"lyon"
         } , {
            "kind":"office",
            "city":"paris"
         } ]
```

... findet **QUERY BY ATTRIBUTE** Personen mit einer Adresse in "paris". Dazu schreiben Sie folgende Anweisung:

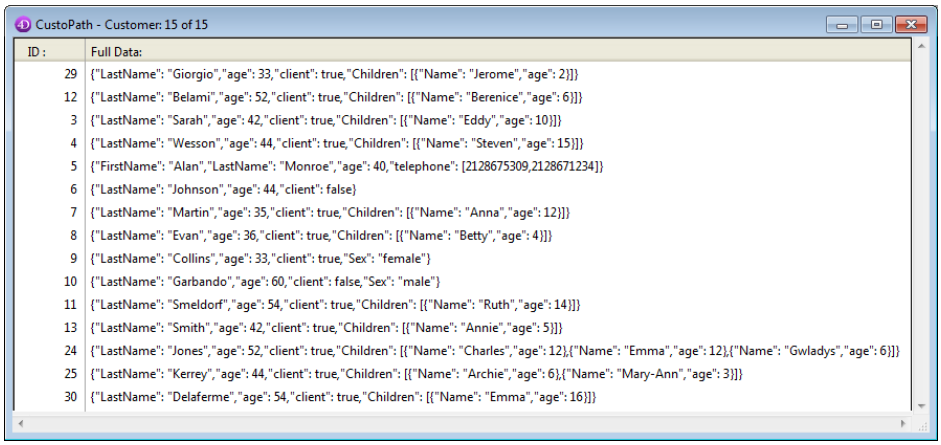
```
//Das Array Attribut mit der Syntax "[]"
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[].city";=;"paris")
//wählt "martin" und "smith"
```

Hinweis: Haben Sie im selben Array Attribut mehrere Kriterien definiert, wird das passende Kriterium nicht zwingend auf dasselbe Array Element angewendet. Im folgenden Beispiel gibt die Suche "smith" zurück, weil es ein Element "locations" mit "kind" ist gleich "home" hat und ein Element "locations" mit "city" ist gleich "paris", selbst wenn es nicht dasselbe Element ist:

```
QUERY BY ATTRIBUTE([People];[People]OB_Field;"locations[].kind";=;"home";*)
QUERY BY ATTRIBUTE([People]; & ;[People]OB_Field;"locations[].city";=;"paris")
//wählt "smith"
```

Beispiel 4

Dieses Beispiel zeigt die Verwendung der virtuellen Eigenschaft "length". Ihre Anwendung hat ein Objektfeld [Customer]full_Data mit folgenden Daten:



Sie wollen die Datensätze aller Kunden mit zwei oder mehr Kindern erhalten. Dafür schreiben Sie folgenden Code:

```
QUERY BY ATTRIBUTE([Customer];[Customer]full_Data;"Children.length";>=;2)
```

Beispiel 5

Hier sehen Sie die verschiedenen möglichen Kombinationen für verlinkte Argumente von Arrays. Wir gehen von folgenden Datensätzen aus:

Datensatz1:

```
[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"
      } ]
    } ]
```

Datensatz2:

```
[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"
      } ]
    } ]
```

Datensatz3:

```
[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"
  } ]
 } ]
```

Die Suche nach Personen mit einem Kind, genannt "Betty" und 15 Jahre alt:

```
QUERY BY ATTRIBUTE([Person];[Person]ObjectField;"Children[a].Name";=;"Betty";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[a].Age";=;"15")
 //ergibt "Victor"

QUERY BY ATTRIBUTE([Person];[Person]ObjectField;"Children[].Name";=;"Betty";*)
QUERY BY ATTRIBUTE([Person];&;[Person]ObjectField;"Children[].Age";=;"15")
 //ergibt "Sam", "Louis" und "Victor"
```

Die Suche nach Personen mit einem Kind, genannt "Betty", 15 Jahre alt und mit einem Kind, genannt "Harry", 9 Jahre alt:

```
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")
 //ergibt "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")
 //ergibt "Sam" und "Victor"
```

Die Suche nach Personen mit einem 15 jährigen Kind, genannt "Harry" mit dem Spielzeug "blue car" toy (Suche in einem Array von 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")
 //ergibt "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")
 //ergibt "Sam" und "Louis"
```

Systemvariablen und Mengen

Wurde die Suche korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt.
Die Variable OK wird auf 0 (Null) gesetzt, wenn:

- Der Benutzer im Suchdialog auf die Schaltfläche **Abbrechen** klickt,
- Die Suche im Modus 'Suchen und Sperren' (siehe Befehl **SET QUERY AND LOCK**) mindestens einen gesperrten Datensatz gefunden hat. In diesem Fall wird auch die Systemmenge **LockedSet** aktualisiert.

QUERY BY EXAMPLE

QUERY BY EXAMPLE ({Tabellenname}{;}{*})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, in der gesucht werden soll Ohne Angabe Haupttabelle
*	Operator	➡	Keine Rollbalken im Suchformular

Beschreibung

Der Befehl **QUERY BY EXAMPLE** hat die gleiche Funktionsweise wie der Menübefehl **Nach Beispiel suchen** in der Anwendungsumgebung. *Tabellenname* ist optional. Wird der Parameter nicht angegeben, wird der Befehl auf die Haupttabelle angewandt. **QUERY BY EXAMPLE** benutzt immer das aktuelle Eingabeformular als Suchfenster. In *Tabellenname* wird nach den Daten gesucht, die der Benutzer im Suchfenster eingibt. Die Suche wird optimiert, da automatisch indizierte Felder verwendet werden.

Der Parameter * ist optional. Geben Sie ihn an, wird das Suchformular ohne Rollbalken geöffnet.

Weitere Informationen zum Menübefehl **Nach Beispiel suchen** finden Sie im Abschnitt **Nach Beispiel suchen** des Handbuchs *4D Designmodus*.

Beispiel

Die Methode in diesem Beispiel zeigt dem Benutzer das Formular MyQuery. Bestätigt der Benutzer das Formular und führt die Suche aus (d.h., wenn die Systemvariable OK den Wert 1 annimmt), werden die Datensätze angezeigt, die die Suchkriterien erfüllen:

```
FORM SET INPUT([People];"MyQuery") ` Wechsle in Suchformular
QUERY BY EXAMPLE([People]) ` Zeige Formular und führe Suche aus
If(OK=1) ` Hat der Benutzer die Suche festgelegt
   DISPLAY SELECTION([People]) ` Zeige die Datensätze
End if
```

Systemvariablen und Mengen

Klickt der Anwender auf die Schaltfläche **Bestätigen** oder die **Eingabetaste**, startet die Suche und die Systemvariable OK nimmt den Wert 1 an. Klickt der Anwender auf die Schaltfläche **Abbrechen** oder die Tastenkombination für Abbrechen (unter Windows **Strg+Punkt**, auf Macintosh **Befehlstaste+Punkt**), wird die Systemvariable OK auf 0 gesetzt, die Suche wird annulliert.

QUERY BY FORMULA

QUERY BY FORMULA (Tabellenname {; Formel})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, in der eine Datensatzauswahl zurückgegeben wird
Formel	Boolean	➡	Suchformel

Beschreibung

Der Befehl **QUERY BY FORMULA** sucht nach Datensätzen in *Tabellenname*. **QUERY BY FORMULA** ändert die aktuelle Auswahl *Tabellenname* für den aktuellen Prozess. Der erste Datensatz der neuen Auswahl ist nun der aktuelle Datensatz. Der Datensatz wird geladen, falls er nicht bereits dem bisherigen aktuellen Datensatz entspricht.

QUERY BY FORMULA und **QUERY SELECTION BY FORMULA** haben dieselbe Arbeitsweise. Der Unterschied liegt darin, dass **QUERY BY FORMULA** jeden Datensatz in der Tabelle durchsucht, während **QUERY SELECTION BY FORMULA** nur die Datensätze der aktuellen Auswahl durchsucht.
Beide Befehle wenden *Formel* auf jeden Datensatz der Tabelle bzw. der Auswahl an. *Formel* ist ein Boolean Ausdruck, der den Wert WAHR oder FALSCH hat. Bei WAHR wird der Datensatz in die neue Auswahl aufgenommen.

Formel kann einfach sein, z.B. ein Datenfeld mit einem Wert vergleichen; oder komplex, z.B. eine Berechnung durchführen oder eine Information in einer verknüpften Tabelle bewerten. *Formel* kann ein 4D Befehl sein oder eine selbst erstellte Methode bzw. ein Ausdruck. Bei Feldern vom Typ Text oder alphanumerisch können Sie auch mit dem Joker-Zeichen @ arbeiten und mit dem Operator "enthält Schlüsselwort" (%) für Volltextsuche. Weitere Informationen dazu finden Sie unter dem Befehl **QUERY**.

Wird der Parameter *Formel* nicht angegeben, erscheint der Suchdialog. (Um eine Zeile mit Formel hinzufügen, kann der Benutzer die Schaltfläche **[+]** mit gedrückter **Alt-Taste** anklicken.)

Ist die Suche ausgeführt, wird der erste Datensatz der neuen Auswahl von der Festplatte geladen und zum aktuellen Datensatz.

Diese Befehle führen je nach Suchtyp die gleichen Suchläufe wie mit dem Befehl **QUERY** aus. So führt z.B. die Anweisung **QUERY BY FORMULA***([mytable]; [mytable]myfield=value)* dasselbe aus wie **QUERY***([mytable]; [mytable]myfield=value)*, d.h. sie kann auch den Index verwenden. 4D kann auch Suchläufe mit nicht-optimierbaren Teilen optimieren. Dazu werden zuerst die optimierbaren Teile ausgeführt und die Ergebnisse dann mit der restlichen Suche kombiniert. Beispiel: Die Anweisung **QUERY BY FORMULA***([mytable]; Length(myfield)=value)* wird nicht optimiert. Die Anweisung **QUERY BY FORMULA***([mytable]; Length(myfield)=value1 | myfield=value2)* wird dagegen teilweise optimiert.

Diese Befehle führen standardmäßig "joins" wie SQL aus, wenn Sie Felder aus unterschiedlichen Tabellen vergleichen. Das heißt, es muss nicht zwingend eine automatische Verknüpfung zwischen den Tabellen existieren. Sie können z.B. eine Anweisung vom Typ **QUERY BY FORMULA([Table_A];([Table_A]field_X = [Table_B]field_Y) & ([Table_B]field_Y = "abc"))** verwenden (siehe Beispiel 3). Der erste Teil der Formel (*[Table_A]field_X = [Table_B]field_Y*) definiert die Verbindung (join) zwischen den beiden Feldern, der zweite Teil (*[Table_B]field_Y = "abc"*) definiert die Suchkriterien (es muss mindestens eins gesetzt sein).

Sind automatische Verknüpfungen zwischen den Tabellen vorhanden, werden sie in der Regel nicht verwendet, sondern nur in folgenden Fällen:

- Wenn SuchFormel sich nicht in der Form {Feld; Vergleichsoperator; Wert} aufschlüsseln lässt.
- Wenn zwei Felder der gleichen Tabelle miteinander verglichen werden.

Hinweis: Zur Wahrung der Kompatibililät ist es möglich, die "joins" Mechanismen zu deaktivieren, entweder global über die Voreinstellungen (nur konvertierte Datenbanken) oder pro Prozess über den Befehl **SET DATABASE PARAMETER**.

4D Server: Dieser Befehl wird auf dem Server ausgeführt, was seine Ausführung optimiert. Beachten Sie jedoch, dass die Suche bei direkt über Formel aufgerufenen Variablen mit dem Wert der Variablen des Client-Rechners berechnet werden.
Beispiel: Die Anweisung **QUERY BY FORMULA***([mytable]);[mytable]myfield=myvariable)* läuft auf dem Server, jedoch mit dem Inhalt der Variablen *myvariable* des Client-Rechners.

Hinweis zur Kompatibilität: Bis 4D Server v11 wurde **QUERY BY FORMULA** auf dem Client-Rechner ausgeführt. Zur Wahrung der Kompatibilität wird diese Arbeitsweise für konvertierte Datenbanken beibehalten. Es gibt jedoch eine Einstellung zur Kompatibilität und einen Selektor im Befehl **SET DATABASE PARAMETER**, um die server-seitige Ausführung für konvertierte Datenbanken zu aktivieren.

Beispiel 1

Dieses Beispiel findet die Datensätze für alle Rechnungen, die - unabhängig vom Jahr - im Monat Dezember eingegeben wurden. Dazu wird die Funktion **Month of** auf jeden Datensatz angewandt. Diese Suche könnte sonst nur ausgeführt werden, wenn für Monat ein eigenes Feld eingerichtet würde:

```
QUERY BY FORMULA([Invoice];Month of([Invoice]Entered)=12)
` Finde Rechnungen, die im Dezember eingegeben wurden
```

Beispiel 2

Dieses Beispiel findet die Datensätze für alle Personen, deren Name mehr als 10 Zeichen enthält:

```
QUERY BY FORMULA([People];Length([People]Name)>10)
` Finde Namen, die mehr als 10 Zeichen enthalten
```

Beispiel 3

Dieses Beispiel aktiviert SQL joins für eine spezifische Suche nach Formel:

```
$currentval:=Get database parameter(QUERY BY FORMULA Joins)
SET DATABASE PARAMETER(QUERY BY FORMULA Joins;2)  `SQL joins aktivieren
  `Alle Zeilen der Kundenrechnungen "ACME" suchen,
  `auch wenn die Tabellen nicht verknüpft sind.
QUERY BY FORMULA([invoice_line];([invoice_line]invoice_id=[invoice]id  &  [invoice]client="ACME"))
SET DATABASE PARAMETER(QUERY BY FORMULA Joins;$currentval)#
  `Die aktuellen Einstellungen wiederherstellen
```

QUERY SELECTION

QUERY SELECTION ({Tabellenname }{;}{ Suchbegriff {; *}})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, in der gesucht werden soll, ohne Angabe Haupttabelle
Suchbegriff	Ausdruck	➡	Operatoren und Suchbegriffe
*	Operator	➡	Weiterer Suchbegriff folgt

Beschreibung

Der Befehl **QUERY SELECTION** sucht nach Datensätzen in *Tabellenname*. **QUERY SELECTION** ändert die aktuelle Auswahl *Tabellenname* für den laufenden Prozess. Der erste Datensatz der neuen Auswahl ist nun der aktuelle Datensatz. Der Datensatz wird geladen, falls er nicht bereits dem bisherigen aktuellen Datensatz entspricht.

QUERY SELECTION hat die gleiche Funktionsweise und die gleiche Syntax wie der Befehl **QUERY**. Der Geltungsbereich ist jedoch unterschiedlich:

- **QUERY** sucht in allen Datensätzen der Tabelle nach Datensätzen.
- **QUERY SELECTION** sucht nur in der aktuellen Auswahl der Tabelle nach Datensätzen.

Weitere Informationen dazu finden Sie unter dem Befehl **QUERY**.

QUERY SELECTION ist hilfreich, wenn es nicht möglich ist, eine Folge von **QUERY** Aufrufen verbunden mit dem Parameter * zu definieren. Das ist beispielsweise der Fall, wenn Sie in einer aktuellen Auswahl suchen wollen, die nicht aus einer vorigen Suche resultiert, sondern aus einem Befehl wie **USE SET**.

Beispiel

Sie wollen die Datensätze durchsuchen, die ein Benutzer zuvor in einem Listenformular markiert hat. Der Code dafür lautet:

```
USE SET("UserSet")  //die aktuelle Auswahl mit den markierten Datensätzen ersetzen
QUERY SELECTION([Company];[Company]City="New York City";*)
QUERY SELECTION([Company]Type Business="Stock Exchange")
```

Sie finden in der Ausgangsmenge des Benutzers alle Firmen in New York City, die Verkauf ab Lager machen.

QUERY SELECTION BY ATTRIBUTE

QUERY SELECTION BY ATTRIBUTE ({Tabellenname}{;}{KonjOp ;} ObjektFeld ; AttributPfad ; SuchOp ; Wert {; *})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle zur Rückgabe einer Auswahl Datensätze Ohne Angabe Haupttabelle
KonjOp	Operator	➡	Verbindungsparameter bei mehrfacher Suche
ObjektFeld	Feld	➡	Objektfeld zum Suchen von Attributen
AttributPfad	String	➡	Name oder Pfad des Attributs
SuchOp	Operator, String	➡	Suchoperator (Vergleichsoperator)
Wert	Text, Zahl, Datum, Zeit	➡	Zu vergleichender Wert
*	Operator	➡	Weiterer Suchbegriff folgt

Beschreibung

QUERY SELECTION BY ATTRIBUTE funktioniert wie **QUERY BY ATTRIBUTE** und führt die gleichen Aktionen durch. Die Befehle unterscheiden sich allein durch die Reichweite der Suche:

- **QUERY BY ATTRIBUTE** sucht in allen Datensätzen der Tabelle nach Datensätzen
- **QUERY SELECTION BY ATTRIBUTE** sucht in der aktuellen Datensatzauswahl der Tabelle nach Datensätzen

QUERY SELECTION BY ATTRIBUTE ändert die aktuelle Auswahl von *Tabellenname* für den aktuellen Prozess und macht den ersten Datensatz der neuen Auswahl zum aktuellen Datensatz.

Weitere Informationen dazu finden Sie unter dem Befehl **QUERY BY ATTRIBUTE**.

QUERY SELECTION BY ATTRIBUTE ist hilfreich, wenn sich eine Suche nicht als Kombination von **QUERY BY ATTRIBUTE** (sowie Aufrufe von **QUERY**) mit dem Parameter * definieren lässt. Das ist beispielsweise der Fall, wenn Sie in einer aktuellen Auswahl suchen wollen, die sich nicht aus einer vorangegangenen Suche ergibt, sondern über einen Befehl wie **USE SET**.

Beispiel

In Datensätzen, die der Benutzer vorher markiert hat, Personen im Alter zwischen 20 und 30 finden:

```
USE SET("UserSet") //erstellt eine neue aktuelle Auswahl
QUERY SELECTION BY ATTRIBUTE([Persons];[Persons]OB_Info;"age";>;20;*)
QUERY SELECTION BY ATTRIBUTE([Persons];&;[Persons]OB_Info;"age";<;30) //löst die Suche aus
```

QUERY SELECTION BY FORMULA

QUERY SELECTION BY FORMULA (Tabellenname {; Formel})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, in der gesucht werden soll Ohne Angabe Haupttabelle
Formel	Boolean	➡	Suchformel

Beschreibung

Der Befehl **QUERY SELECTION BY FORMULA** sucht nach Datensätzen in *Tabellenname*. **QUERY SELECTION BY FORMULA** ändert die aktuelle Auswahl *Tabellenname* für den aktuellen Prozess. Der erste Datensatz der neuen Auswahl ist nun der aktuelle Datensatz. Der Datensatz wird geladen, falls er nicht bereits dem bisherigen aktuellen Datensatz entspricht.

QUERY SELECTION BY FORMULA hat dieselbe Arbeitsweise wie **QUERY BY FORMULA** mit folgendem Unterschied:

- **QUERY BY FORMULA** durchsucht jeden Datensatz in der Tabelle
- **QUERY SELECTION BY FORMULA** durchsucht nur die Datensätze der aktuellen Auswahl.

Weitere Informationen finden Sie in der Beschreibung zum Befehl **QUERY BY FORMULA**.

QUERY SELECTION WITH ARRAY

QUERY SELECTION WITH ARRAY (ZielFeld ; ArrayName)

Parameter	Typ		Beschreibung
ZielFeld	Feld	➡	Feld zum Vergleichen der Werte
ArrayName	Array	➡	Array der gesuchten Werte

Beschreibung

Der Befehl **QUERY SELECTION WITH ARRAY** sucht die Tabelle des Feldes, das als erster Parameter für die Datensätze übergeben wurde, wo der Wert von *ZielFeld* mindestens mit einem Wert der Elemente in *ArrayName* gleich ist. Die gefundenen Datensätze werden die neue aktuelle Auswahl.

QUERY SELECTION WITH ARRAY funktioniert genauso wie **QUERY WITH ARRAY**. Die Befehle unterscheiden sich nur in ihrer Suchreichweite:

- **QUERY WITH ARRAY** sucht alle Datensätze der Tabelle, die *ZielFeld* enthalten.
- **QUERY SELECTION WITH ARRAY** sucht nur die Datensätze der aktuellen Auswahl der Tabelle, die *ZielFeld* enthalten.

QUERY WITH ARRAY

QUERY WITH ARRAY (ZielFeld ; ArrayName)

Parameter	Typ		Beschreibung
ZielFeld	Feld	➡	Feld zum Vergleichen der Werte
ArrayName	Array	➡	Array der gesuchten Werte

Beschreibung

Der Befehl **QUERY WITH ARRAY** sucht alle Datensätze, deren Wert *ZielFeld* mindestens zu einem der Werte in *ArrayName* gleich ist. Die gefundenen Datensätze werden die neue aktuelle Auswahl.

Mit **QUERY WITH ARRAY** können Sie schnell und einfach eine Suche auf mehrfache Werte einrichten.

Hinweise:

- Dieser Befehl lässt sich nicht einsetzen für Felder vom Typ Bild oder BLOB.
- *ZielFeld* und *ArrayName* müssen vom selben Datentyp sein. Ausnahme: Für ein Feld vom Typ Zeit können Sie ein Array vom Typ Lange Ganzzahl verwenden.

Beispiel

Mit folgendem Beispiel finden Sie die Datensätze von Kunden, die sowohl französisch als auch amerikanisch sind:

```
ARRAY STRING(2;SearchArray;2)
SearchArray{1}:="FR"
SearchArray{2}:="US"
QUERY WITH ARRAY([Clients]Country;SearchArray)
```

SET QUERY AND LOCK

SET QUERY AND LOCK (Sperren)

Parameter	Typ		Beschreibung
Sperren	Boolean	➡	Wahr = Durch Suchläufe gefundene Datensätze sperren Falsch = Gefundene Datensätze nicht sperren

Beschreibung

Der Befehl **SET QUERY AND LOCK** sperrt die Datensätze, die durch Suchläufe, ausgelöst durch Aufrufen dieses Befehls in der aktuellen Transaktion, gefunden werden. Das bedeutet, dass die Datensätze zwischen der Suche und dem Bearbeiten des Suchergebnisses nur im aktuellen Prozess verändert werden können.
Standardmäßig sind die durch eine Suche gefundenen Datensätze nicht gesperrt. Übergeben Sie im Parameter *Sperren* Wahr, wird das Sperren aktiviert.

Dieser Befehl muss innerhalb einer Transaktion verwendet werden. Bei Aufruf außerhalb dieses Kontexts wird ein Fehler erzeugt. So lässt sich das Sperren von Datensätzen besser steuern.
Die gefundenen Datensätze bleiben gesperrt, bis die Transaktion durch Bestätigen oder Abbrechen beendet wird. Dann sind wieder alle Datensätze freigegeben.

Wurde **SET QUERY AND LOCK**(Wahr) ausgeführt, führen die Suchbefehle, wie z.B. **QUERY** spezifische Operationen aus, wenn beim Suchen Datensätze gefunden werden, die bereits gesperrt sind:

- Die Suche wird gestoppt und die Systemvariable OK wird auf 0 (Null) gesetzt,
- Die aktuelle Auswahl wird aufgehoben
- Die Systemmenge *LockedSet* enthält die beim Suchen gefundenen gesperrten Datensätze.

Folglich müssen Sie in diesem Zusammenhang bei einer ergebnislosen Suche (aktuelle Auswahl leer bzw. Variable OK auf 0 gesetzt) die Menge *LockedSet* testen, um den Grund für die fehlgeschlagene Suche herauszufinden.

Die Datensätze werden für alle Tabellen in der aktuellen Transaktion gesperrt. Rufen Sie **SET QUERY AND LOCK**(Falsch) auf, um diese Operation anschließend wieder aufzuheben.

SET QUERY AND LOCK verändert nur das Verhalten von Suchbefehlen, also für:

- **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**

Er hat keine Auswirkung auf andere Befehle, welche die aktuelle Auswahl wie **ALL RECORDS**, **RELATE MANY**, etc.

Beispiel

In diesem Beispiel lässt sich zwischen den Befehlen **QUERY** und **DELETE SELECTION** ein Kunde, der von Kategorie C in Kategorie A übertragen wird, nicht verändern.

```
START TRANSACTION
SET QUERY AND LOCK(True)
QUERY([Customers];[Customers]Category=C)
  `In diesem Moment werden die gefundenen Datensätze automatisch für alle anderen Prozesse gesperrt.
DELETE SELECTION([Customers])
SET QUERY AND LOCK(False)
VALIDATE TRANSACTION
```

Fehlerverwaltung

Wird der Befehl außerhalb einer Transaktion aufgerufen, wird ein Fehler generiert.

SET QUERY DESTINATION

SET QUERY DESTINATION (Zieltyp {; Zielobjekt {; Zielzeiger}})

Parameter	Typ		Beschreibung
Zieltyp	Lange Ganzzahl	➡	0=aktuelle Auswahl, 1=Menge, 2=temporäre Auswahl, 3=Variable
Zielobjekt	String, Variable	➡	Name der Menge, temporären Auswahl oder Variablen
Zielzeiger	Zeiger	➡	Zeiger auf die lokale Variable, wenn Zieltyp=3

Beschreibung

Der Befehl **SET QUERY DESTINATION** gibt an, wo das Ergebnis einer nachfolgenden Suche für den aktuellen Prozess abgelegt werden kann.

Im Parameter *Zieltyp* geben Sie den Typ an. 4D bietet unter dem Thema **Suchen** folgende vordefinierten Konstanten:

Konstante	Typ	Wert
Into current selection	Lange Ganzzahl	0
Into named selection	Lange Ganzzahl	2
Into set	Lange Ganzzahl	1
Into variable	Lange Ganzzahl	3

Im optionalen Parameter *Zielobjekt* geben Sie das Ziel der Suche gemäß folgender Tabelle an:

Zieltyp	Zielobjekt
Parameter	**Parameter**
0 (aktuelle Auswahl)	Parameter weglassen
1 (Menge)	Den Namen einer Menge übergeben (bestehend oder neu)
2 (temp. Auswahl)	Den Namen einer temporären Auswahl übergeben (bestehend oder neu)
3 (Variable)	Eine numerische Variable (bestehend oder neu) oder einen leeren String "" zum Verwenden des Parameters *Zielzeiger* übergeben.

Beispiele:

```
SET QUERY DESTINATION(Into current selection)
```

legt alle Datensätze eines Suchlaufes in eine neue aktuelle Auswahl der betreffenden Tabelle.

```
SET QUERY DESTINATION(Into set;"mySet")
```

legt alle Datensätze eines Suchlaufes in die Menge *mySet*. Die aktuelle Auswahl und der aktuelle Datensatz der betreffenden Tabelle bleiben unverändert.

```
SET QUERY DESTINATION(Into named selection;"myNamedSel")
```

legt alle Datensätze eines Suchlaufes in die temporäre Auswahl *myNamedSel*. Die aktuelle Auswahl und der aktuelle Datensatz der betreffenden Tabelle bleiben unverändert.

Hinweise:

- Existiert bisher keine temporäre Auswahl, wird sie automatisch am Ende des Suchlaufs angelegt.
- Dieser Befehl behandelt temporäre Auswahlen wie der Befehl **CUT NAMED SELECTION**, d.h. es werden nur Referenzen beibehalten. Wurde die temporäre Auswahl verwendet, besteht sie nicht länger.

```
SET QUERY DESTINATION(Into variable;$vlResult)
```

Oder:

```
SET QUERY DESTINATION(Into variable;"";->$vlResult)
```

legt die Anzahl der Datensätze in die Variable *$vlResult*. Die aktuelle Auswahl und der aktuelle Datensatz der betreffenden Tabelle bleiben unverändert.

Hinweis: Die zweite Syntax vereinfacht die Verwendung dieses Befehls zusammen mit **GET QUERY DESTINATION**

Warnung: SET QUERY DESTINATION beeinflusst alle nachfolgenden Suchläufe im aktuellen Prozess. Sie MÜSSEN deshalb einen Aufruf von **SET QUERY DESTINATION** (Zieltyp#0) immer mit dem Aufruf von **SET QUERY DESTINATION**(0) gegensteuern, damit der normale Suchmodus wiederhergestellt wird.

SET QUERY DESTINATION ändert nur das Verhalten der Suchbefehle:

- **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**

SET QUERY DESTINATION hat keinen Einfluss auf andere Befehle, die die aktuelle Auswahl verändern können, wie z.B. **ALL RECORDS**, **RELATE MANY** usw..

Beispiel 1

Sie erstellen ein Formular, das die Datensätze aus eine Tabelle *[Phone Book]* anzeigt, sowie eine Registerkarte *asRolodex* (mit den 26 Buchstaben des Alphabets) und ein Unterformular, das die Datensätze *[Phone Book]* anzeigt. Wählen Sie nun einen Reiter aus der Registerkarte, erscheinen die Datensätze, die mit dem entsprechenden Buchstaben beginnen.

Die Tabelle *[Phone Book]* enthält eine ganze Reihe statischer Daten, so dass Sie nicht jedes Mal, wenn Sie ein Tab auswählen, einen Suchlauf starten müssen. So können Sie wertvolle Zeit für die Datenbank-Engine einsparen.

Sie legen die Suchläufe in temporären Auswahlen ab, die Sie bei Bedarf wiederverwenden können. Für die Registerkarte *asRolodex* schreiben Sie folgende Objektmethode:

```
  ` Objektmethode asRolodex
Case of
   :(Form event=On Load)
` Bevor das Formular auf dem Bildschirm erscheint,
` initialisiere Rolodex und ein Boolean Array, das
` uns mitteilt, ob eine Suche für den entsprechenden Buchstaben
` ausgeführt wurde oder nicht.
      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)
` Bei einem Klick auf die Registerkarte prüfe, ob die
` entsprechende Suche ausgeführt wurde oder nicht.
      If(Not(abQueryDone{asRolodex}))
` Wenn nicht, leite die nächste Suche zur temporären Auswahl
         SET QUERY DESTINATION(Into named selection;"temp")
` Führe die Suche aus
         QUERY([Phone Book];[Phone Book]Last name=asRolodex{asRolodex}+"@")
` Stelle den normalen Suchmodus wieder her
         SET QUERY DESTINATION(Into current selection)
` Verwende die gefundenen Datensätze
         USE NAMED SELECTION("temp")
         COPY NAMED SELECTION([Phone book];"Rolodex+asRolodex{asRolodex})
` Wird dieser Buchstabe wieder gewählt, wird die Suche nicht erneut ausgeführt
         abQueryDone{asRolodex}:=True
      Else
` Verwende die vorhandene temporäre Auswahl, um die Datensätze zum gewählten Buchstaben anzuzeigen
         USE NAMED SELECTION("Rolodex"+asRolodex{asRolodex}
      End if

   :(Form event=On Unload)
` Erscheint das Formular nicht auf dem Bildschirm
` Lösche die angelegten temporären Auswahlen
      For($vlElem;1;26)
         If(abQueryDone{$vlElem})
            CLEAR NAMED SELECTION("Rolodex"+asRolodex{$vlElem})
         End if
      End for
` Lösche die zwei nicht mehr benötigten Arrays
      CLEAR VARIABLE(asRolodex)
      CLEAR VARIABLE(abQueryDone)
End case
```

Beispiel 2

Mit der Projektmethode ***Unique values*** in diesem Beispiel können Sie die Einmaligkeit der Werte für jede Datensatzzahl in einer Tabelle prüfen. Der aktuelle Datensatz kann ein vorhandener oder ein neu erstellter Datensatz sein.

```
  ` Projektmethode Unique values
  ` Unique values ( Pointer ; Pointer { ; Pointer... } ) -> Boolean
  ` Unique values ( ->Table ; ->Field { ; ->Field2... } ) -> Ja oder Nein

C_BOOLEAN($0)
C_POINTER($1{1})
C_LONGINT($vlField;$vlNbFields;$vlFound;$vlCurrentRecord)
$vlNbFields:=Count parameters-1
$vlCurrentRecord:=Record number($1->)
If($vlNbFields>0)
   If($vlCurrentRecord#-1)
      If($vlCurrentRecord<0)
  ` Aktueller Datensatz ist ein ungesicherter neuer Datensatz(Datensatznr ist -3)
  ` so können wir die Suche stoppen, sobald mindestens ein Datensatz gefunden wurde
         SET QUERY LIMIT(1)
      Else
  ` Aktueller Datensatz ist ein vorhandener Datensatz,
  ` so können wir die Suche stoppen, sobald mindestens zwei Datensätze gefunden wurden
         SET QUERY LIMIT(2)
      End if
  ` Die Suche gibt das Ergebnis in $vlFound zurück
  ` ohne den aktuellen Datensatz oder die aktuelle Auswahl zu ändern.
      SET QUERY DESTINATION(Into variable;$vlFound)
  ` Führe Suche gemäß der festgelegten Anzahl Datensätze aus
      Case of
         :($vlNbFields=1)
            QUERY($1->;$2->=$2->)
         :($vlNbFields=2)
            QUERY($1->;$2->=$2->;*)
            QUERY($1->;&;$3->=$3->)
         Else
            QUERY($1->;$2->=$2->;*)
            For($vlField;2;$vlNbFields-1)
               QUERY($1->;&;${1+$vlField}->=${1+$vlField}->;*)
            End for
            QUERY($1->;&;${1+$vlNbFields}->=${1+$vlNbFields}->)
      End case
      SET QUERY DESTINATION(Into current selection)
  ` Stelle normalen Suchmodus wieder her
      SET QUERY LIMIT(0)  ` Suchläufe sind nicht mehr begrenzt
  ` Bearbeite Suchergebnis
      Case of
         :($vlFound=0)
            $0:=True  ` Keine doppelten Werte
         :($vlFound=1)
            If($vlCurrentRecord<0)
               $0:=False  ` Vorhandener Datensatz mit denselben Werten wie ungesicherter neuer Datensatz wurde gefunden
            Else
               $0:=True  ` Keine doppelten Werte, es wurde nur genau derselbe Datensatz gefunden
            End if
         :($vlFound=2)
            $0:=False  ` Egal aus welchem Grund, die Werte wurden dupliziert
      End case
   Else
      If(◇DebugOn)  ` Macht keinen Sinn, signalisiere, wenn Entwicklerversion
         TRACE  ` WARNUNG! Unique values wird NICHT mit aktuellem Datensatz aufgerufen.
      End if
      $0:=False  ` Keine Garantie für korrektes Ergebnis
   End if
Else
   If(◇DebugOn)  ` Macht keinen Sinn, signalisiere, wenn Entwicklerversion
      TRACE  ` WARNUNG! Unique values wird OHNE Suchbedingung aufgerufen
   End if
   $0:=False  ` Keine Garantie für korrektes Ergebnis
End if
```

Haben Sie diese Projektmethode in Ihre Anwendung integriert, können Sie schreiben:

```
  ` ...
If(Unique values(->[Contacts];->[Contacts]Company);->[Contacts]Last name;->[Contacts]First name)
```

```
    ` Führe geeignete Aktionen für diesen Datensatz mit einmaligen Werten aus.
Else
    ALERT("Es gibt bereits einen Contact mit diesem Namen in dieser Firma.")
End if
    ` ...
```

SET QUERY LIMIT

SET QUERY LIMIT (Begrenzung)

Parameter	Typ		Beschreibung
Begrenzung	Lange Ganzzahl	➡	Anzahl der Datensätze oder 0 ohne Begrenzung

Beschreibung

Der Befehl **SET QUERY LIMIT** kann jede nachfolgende Suche für den aktuellen Prozess stoppen, sobald die in *Begrenzung* angegebene Anzahl der Datensätze gefunden wurde.

Haben Sie z.B. in *Begrenzung* den Wert 1 übergeben, stoppt jede nachfolgende Suche das Durchlaufen eines Index bzw. einer Datendatei, sobald ein Datensatz mit den zutreffenden Suchbedingungen gefunden wurde.

Wollen Sie wieder unbegrenzte Suchläufe herstellen, rufen Sie erneut **SET QUERY LIMIT** auf und setzen *Begrenzung* auf 0.

Warnung: SET QUERY LIMIT beeinflusst alle nachfolgenden Suchläufe im aktuellen Prozess. Sie MÜSSEN deshalb einen Aufruf von **SET QUERY LIMIT** *(Zieltyp#0)* immer mit dem Aufruf von **SET QUERY LIMIT***(0)* gegensteuern, damit der normale Suchmodus wiederhergestellt wird.

SET QUERY LIMIT ändert nur das Verhalten der Suchbefehle:

- **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**

SET QUERY LIMIT beeinflusst nicht andere Befehle, die die aktuelle Auswahl verändern können, wie z.B. **ALL RECORDS**, **RELATE MANY** usw..

Beispiel 1

Sie wollen jeweils nach beliebigen zehn Kunden mit einem Verkaufsvolumen über 1 Million suchen:

```
SET QUERY LIMIT(10)
QUERY([Customers];[Customers]Verkaufsvolumen>1000000)
SET QUERY LIMIT(0)
```

Beispiel 2

Siehe zweites Beispiel zum Befehl **SET QUERY DESTINATION**.

SVG

- Überblick über SVG Befehle
- 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

Überblick über SVG Befehle

SVG (Scalable Vector Graphics) ist ein auf XML basierendes Dateiformat (Endung .svg), das zum Beschreiben von Vektorgrafiken dient. SVG wird in den meisten Fällen zum Veröffentlichen von statistischen oder kartografischen Daten verwendet.

Diese Dateien lassen sich in Web Browsern entweder nativ oder über Plug-Ins betrachten. 4D enthält eine SVG rendering Engine, über die sich SVG Dateien in Bildfeldern oder Variablen anzeigen lassen.
Der Befehl **SVG EXPORT TO PICTURE** ermöglicht, ein Bild in 4D anhand einer SVG Beschreibung zu generieren.
Beachten Sie auch, dass der Befehl **GRAPH** die in 4D integrierte SVG Engine verwendet.

Weitere Informationen zu diesem Format finden Sie unter http://www.w3.org/Graphics/SVG/.

SVG EXPORT TO PICTURE

SVG EXPORT TO PICTURE (ElementRef ; BildVar {; ExportTyp})

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf Root XML Element
BildVar	Bild	➡	Bildvariable zum Empfangen des XML Baums (SVG Bild)
ExportTyp	Lange Ganzzahl	➡	0 = Datenquelle nicht speichern, 1 = Datenquelle kopieren 2 (Standard) = Eigene Datenquelle

Beschreibung

Der Befehl **SVG EXPORT TO PICTURE** ermöglicht, ein Bild im SVG-Format innerhalb eines XML Baums in einem Bildfeld oder einer Variablen im Parameter *BildVar* zu speichern.

Hinweis: Weitere Informationen zum SVG Format finden Sie im Abschnitt **Überblick über XML Tools**.

Im Parameter *ElementRef* übergeben Sie die Referenz auf das XML Element mit dem Bild im Format SVG.
In *Bildvar* übergeben Sie das 4D Bildfeld oder die Variable mit dem Bild im Format SVG. Das Bild wird in seinem native Format exportiert (XML Beschreibung) und über die SVG rendering Engine beim Anzeigen neu gezeichnet.
Mit dem optionalen Parameter *ExportTyp* können Sie definieren, wie der Befehl die XML Datenquelle verwaltet.

Sie können eine der nachfolgenden Konstanten unter dem Thema **XML** übergeben:

Konstante	Typ	Wert	Kommentar
Copy XML data source	Lange Ganzzahl	1	4D behält eine Kopie des DOM Baumes mit dem Bild bei, d.h. das Bild lässt sich in einem Bildfeld der Datenbank sichern und dann jederzeit erneut anzeigen oder exportieren.
Get XML data source	Lange Ganzzahl	0	4D liest die XML Datenquelle nur; sie wird nicht mit dem Bild beibehalten. Das steigert die Ausführungsgeschwindigkeit des Befehls signifikant; da jedoch der DOM Baum nicht beibehalten wird, lässt sich das Bild weder speichern noch exportieren.
Own XML data source	Lange Ganzzahl	2	4D exportiert den DOM Baum mit Bild. Es lässt sich speichern oder exportieren und die Ausführung des Befehls ist schnell. Dann ist jedoch die XML Referenz *ElementRef* für andere 4D Befehle nicht mehr verwendbar. Dies ist der Standardmodus für Exportieren, wenn der Parameter *ExportTyp* nicht angegeben ist.

Beispiel

Das folgende Beispiel zeigt „Hello World" in einem 4D Bild an:

```
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 ({* ;} Bildobjekt ; x ; y) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Bildobjekt ein Objektname (String) Ohne * ist Bildobjekt ein Feld oder eine Variable
Bildobjekt	Bild	➡	Mit *: Objektname, ohne *: Feld oder Variable
x	Lange Ganzzahl	➡	X Koordinate in Pixel
y	Lange Ganzzahl	➡	Y Koordinate in Pixel
Funktionsergebnis	String	⮌	ID des gesuchten Elements an der Stelle X, Y

Beschreibung

Die Funktion **SVG Find element ID by coordinates** gibt die ID (Attribut "id" oder "xml:id") des XML Elements zurück, das an der durch die Koordinaten (x,y) bestimmten Stelle im SVG Bild, angegeben durch den Parameter *Bildobjekt*, gefunden wird. Diese Funktion dient insbesondere zum Erstellen interaktiver grafischer Oberflächen mithilfe von SVG Objekten.

Hinweis: Weitere Informationen über das SVG Format finden Sie im Abschnitt **Überblick über XML Tools**.

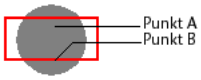
Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Bildobjekt* ein Objektname (String) ist. Ohne diesen Parameter geben Sie an, dass *Bildobjekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie keinen String, sondern eine Variablenreferenz (nur Objekt vom Typ Feld oder Variable).

Beachten Sie, dass ein Bild nicht zwingend in einem Formular angezeigt werden muss. In diesem Fall ist die Syntax vom Typ Objektname nicht gültig und Sie müssen einen Feld- oder Variablennamen übergeben.

Die in den Parametern x und y übergebenen Koordinaten müssen in Pixel, ausgehend von der oberen linken Ecke des Bildes (0,0), angegeben werden. Wird ein Bild in einem Formular angezeigt, können Sie die Werte verwenden, die von den Systemvariablen MouseX und MouseY zurückgegeben werden. Diese Variablen werden in den Formularereignissen On Clicked, On Double Clicked und On Mouse Up aktualisiert, sowie in den Formularereignissen On Mouse Enter und On Mouse Move.

Hinweis: Im Bildkoordinatensystem geben MouseX und MouseY immer denselben Punkt des Bildes an, unabhängig vom Anzeigeformat des Bildes. Davon ausgenommen ist das Format "Replicated", auch bei gescrolltem oder gezoomtem Bild.

Berücksichtigt wird der zuerst erreichte Punkt. Im hier gezeigten Beispiel gibt die Funktion die ID des Kreises zurück, wenn die Koordinaten von Punkt A übergeben wurden und die ID des Rechtecks, wenn die Koordinaten von Punkt B übergeben wurden:



Entsprechen die Koordinaten übereinanderliegenden oder zusammengesetzten Objekten, gibt die Funktion die ID des ersten Objekts mit einem gültigen ID Attribut zurück und geht dabei bei Bedarf in den verwandten Elementen zurück.

Die Funktion gibt einen leeren String zurück, wenn:

- der Root erreicht wird, ohne dass ein "id" Attribut gefunden wurde,
- der Koordinatenpunkt zu keinem Objekt gehört,
- das Attribut "id" ein leerer String ist.

Hinweis: Dieser Befehl findet keine Objekte mit einem Sättigungswert unter 0,1. (Attribut "fill-opacity")

Systemvariablen und Mengen

Enthält *Bildobjekt* kein gültiges SVG Bild, gibt der Befehl einen leeren String zurück und die Systemvariable OK wird auf 0 gesetzt. Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt.

SVG Find element IDs by rect

SVG Find element IDs by rect ({* ;} Bildobjekt ; PositionX ; PositionY ; Breite ; Höhe ; arrIDs) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Bildobjekt ist ein Objektname (String) Ohne Stern: Bildobjekt ist eine Variable
Bildobjekt	Bild	➡	Mit *: Objektname, ohne *: Feld oder Variable
PositionX	Lange Ganzzahl	➡	Horizontale Koordinate der oberen linken Ecke des Auswahlrechtecks
PositionY	Lange Ganzzahl	➡	Vertikale Koordinate der oberen linken Ecke des Auswahlrechtecks
Breite	Lange Ganzzahl	➡	Breite des Auswahlrechtecks
Höhe	Lange Ganzzahl	➡	Höhe des Auswahlrechtecks
arrIDs	Array Text	⬅	IDs der Elemente, deren begrenzendes Rechteck sich mit dem Auswahlrechteck überschneidet.
Funktionsergebnis	Boolean	↩	Wahr = mindestens ein Element wird gefunden

Beschreibung

Die Funktion **SVG Find element IDs by rect** füllt das Array vom Typ Text oder alphanumerisch mit den IDs (Attribute "id" oder "xml:id") der XML Elemente, deren begrenzendes Rechteck sich mit dem Auswahlrechteck überschneidet, angegeben in den Parametern *PositionX* und *PositionY*.

Die Funktion gibt wahr zurück, wenn mindestens ein Element gefunden wird, d.h. das Array *arrIDs* ist nicht leer. Andernfalls gibt sie wahr zurück.

Diese Funktion eignet sich besonders zum Verwalten von interaktiven grafischen Oberflächen. Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Bildobjekt* ein Objektname ist (String). Ohne diesen Parameter geben Sie an, dass *Bildobjekt* ein Feld oder eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz (nur Objektfeld oder -variable) anstelle eines String.

Arbeiten Sie mit einem Bildfeld bzw. einer -variablen, verwendet die Funktion das Originalbild, das der Datenquelle entspricht. Arbeiten Sie dagegen mit einem Formularobjekt, verwendet die Funktion das aktuelle Bild, das u.U. über den Befehl **SVG SET ATTRIBUTE** verändert wurde und mit den Eigenschaften des Formularobjekts beibehalten wird.

Die in den Parametern *PositionX* und *PositionY* übergebenen Koordinaten müssen in Pixel in Bezug auf die linke obere Ecke des Bildes (0,0) ausgedrückt werden. Sie können die Werte verwenden, die von den Systemvariablen MouseX und Mouse Y zurückgegeben werden. Sie werden von den Formularereignissen On Clicked und On Double Clicked sowie den Formularereignissen On Mouse Enter und On Mouse Move aktualisiert.

Hinweis: Im System der Bildkoordinaten geben [x;y] immer denselben Punkt an, und zwar unabhängig vom Anzeigeformat des Bildes mit Ausnahme des Formats "Replicated".

Alle Elemente, deren begrenzendes Rechteck sich mit dem Auswahlrechteck überschneidet, werden berücksichtigt, auch wenn sie unter anderen Elementen liegen.

SVG GET ATTRIBUTE

SVG GET ATTRIBUTE ({* ;} Bildobjekt ; Element_ID ; attrName ; attrWert)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: Bildobjekt ist ein Objektname (String) Ohne Stern: Bildobjekt ist eine Variable
Bildobjekt	Bild	➡	Mit *: Objektname, ohne *: Variable
Element_ID	Text	➡	ID des Elements, dessen Attribut Sie erhalten wollen
attrName	String	➡	Attribut, dessen Wert Sie erhalten wollen
attrWert	String, Lange Ganzzahl	⬅	Aktueller Wert des Attributs

Beschreibung

Der Befehl **SVG GET ATTRIBUTE** liefert den aktuellen Wert des Attributs *attrName* in einem Objekt oder einem SVG Bild.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Bildobjekt* ein Objektname (String) ist. In diesem Fall gibt der Befehl den Wert für das Attribut des gerenderte Bildes, das dem Objekt zugewiesen ist. Dieser Wert kann u.U. verändert worden sein, z.B. mit dem Befehl **SVG SET ATTRIBUTE**.

Ohne den Parameter * geben Sie an, dass der Parameter *Bildobjekt* eine Variable oder ein Feld ist. Dann übergeben Sie anstelle eines String eine Referenz auf die Variable (nur Objektvariable) oder das Feld. In diesem Fall gibt der Befehl den Wert für das Attribut des ursprünglichen gerenderten Bildes an, das der Datenquelle der Variablen entspricht.

Hinweis: Diese Vorgehensweise gilt auch für die Funktion **SVG Find element ID by coordinates**.

Der Parameter *Element_ID* setzt die ID ("id" oder "xml:id" Attribut) des Elements, dessen Attributwert Sie erhalten wollen.

Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **SVG SET ATTRIBUTE**. Hier ist eine Liste der 4D Attribute, die für Animation reserviert und bestimmt sind:

Attribut	Zugriff	Kommentare
4D-text	Lesen/Schreiben	Ersetzt/liest den Inhalt des Textknotens. Ist mit den Elementen 'text' 'tspan' und 'textArea' verwendbar
4D-NachVorneSetzen	Schreiben	Bei 'wahr' Knoten vor die Geschwisterknoten setzen. Nur mit dem Befehl **SVG SET ATTRIBUTE** verwendbar
4D-isOfClass-{IDENT [[S|COMMA] IDENT]*}	Lesen	Gibt 'wahr' zurück, wenn das geerbte Klassenattribut alle Klassennamen enthält; sonst wird 'falsch' zurückgegeben. Gibt z.B. wahr zurück, wenn für "4D-isOfClass-land" die geerbte Klasse des Knotens "land department01" ist
4D-enableD2D	Lesen/Schreiben	Deaktiviert bei 'falsch' Direct2D für die SVG Rendering Engine. SVG Filter werden ja auch nicht in Direct2D gerendert, sondern sie sind in GDI/GDIPlus. Mit dieser Option können Sie SVG Filter verwenden, selbst wenn die Anwendung in Direct2D ist. Beachten Sie, dass diese Option nur berücksichtigt wird, wenn das Bild bereits in *Bildobjekt* geladen wurde. Da diese Option jedoch global auf die Engine angewendet wird, müssen Sie sie nur einmal pro Sitzung setzen (z.B. beim Starten der Anwendung mit einem einfachen in den Speicher geladenen SVG aus einer Textvariablen).

SVG SET ATTRIBUTE

SVG SET ATTRIBUTE ({* ;} Bildobjekt ; Element_ID ; attrName ; attrWert {; attrName2 ; attrWert2 ; ... ; attrNameN ; attrWertN} {; *})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: BildObjekt ist ein Objektname (String) Ohne Stern: BildObjekt ist eine Variable
Bildobjekt	Bild	➡	Mit *: Objektname, ohne *: Variable
Element_ID	Text	➡	ID des Elements, wo ein bzw. mehrere Attribute gesetzt sind
attrName	String	➡	Zu definierendes Attribut
attrWert	String, Lange Ganzzahl	➡	Neuer Wert des Attributs
*	Operator	➡	Mit *: Internen DOM Baum des SVG Bildes ändern (nur Variable)

Beschreibung

Der Befehl **SVG SET ATTRIBUTE** ändert den Wert eines vorhandenen Attributs im SVG Rendering Baum eines angezeigten Bildes oder im internen DOM Baum eines Bildes.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Bildobjekt* ein Objektname (String) ist. In diesem Fall gilt der Befehl für die Parameter des gerenderten Bildes, das dem Objekt zugewiesen ist. Beachten Sie, dass die Parameter und folglich auch das gerenderte Bild des Objekts nur erstellt werden, wenn **SVG SET ATTRIBUTE** mindestens einmal aufgerufen wird. Ohne den Parameter * geben Sie an, dass der Parameter *Bildobjekt* eine Variable oder ein Feld ist. Dann übergeben Sie eine Variablen- oder Feldreferenz (nur Objektvariable) anstelle eines String. In diesem Fall gilt der Befehl für das gerenderte Bild für alle Objekte, welche die Variable oder das Feld benutzen.

Standardmäßig gelten die durch diesen Befehl gemachten Änderungen nur für die gerenderten Bilder; sie werden nicht in der Datenquelle gespeichert (interner DOM Baum) und gehen verloren, wenn das Bild per Programmierung entfernt wird oder wenn das Formular geschlossen wird. Sie können diese Änderungen jedoch in den internen DOM Baum des Bildes übertragen, wenn der Parameter *Bildobjekt* auf eine Variable verweist: Dazu müssen Sie nur den zweiten Stern als letzten Parameter übergeben. Auf diese Weise können Sie die Änderungen direkt beibehalten.

Hinweise:

- Änderungen in den internen DOM Baum lassen sich nicht übertragen, wenn der Parameter *Bildobjekt* auf ein Objekt verweist.
- Zum Übertragen von Änderungen muss die SVG Variable über ein DOM Dokument erstellt worden sein (über den Befehl **DOM EXPORT TO VAR**). Wurde die SVG Variable über eine Datei erstellt und übergeben Sie den zweiten Stern, führt der Befehl nichts aus und ein Fehler wird erzeugt, weil die Datenquelle in diesem Fall kein änderbares DOM Dokument enthält.
- Um die Datenquelle eines SVG Bildes zu ändern, können Sie auch die **XML DOM** Befehle oder die **4D SVG component** verwenden.

Der Parameter *Element_ID* gibt die ID ("id" oder "xml:id" Attribut) des Elements an, dessen Attribut(e) Sie ändern wollen.

In den Parametern *attrName* und *attrWert* übergeben Sie jeweils das zu schreibende Attribut und seinen Wert, wie Variablen, Felder oder tatsächliche Werte. Sie können beliebig viele Paare Attribut/Wert übergeben.

Mit dem Befehl **SVG SET ATTRIBUTE** können Sie die meisten der SVG Attribute ändern (jedoch nicht hinzufügen oder löschen), wie z.B. 'fill', 'opacity', 'font-family', etc. Die komplette Übersicht der SVG Attribute finden Sie in den entsprechenden Dokumenten im Internet, z.B. http://www.w3.org/TR/SVG11/attindex.html. Das gerenderte Bild wird sofort aktualisiert; die Änderungen werden werden unmittelbar aktualisiert; sie werden in einer Reihe der Kindelemente für vererbte Stilarten fortgeführt.

Beachten Sie, dass sich aus technischen Gründen die Attribute bestimmter Elemente sowie bestimmte Attribute nicht verändern lassen. Nachfolgende Tabelle zeigt eine Übersicht:

Elemente, deren Attribute sich ändern lassen

svg	Einschränkung: - "Breite" und "Höhe" sind nicht modifizierbar (1) - "viewBox" lässt sich nur ändern, wenn "Breite" und "Höhe" im Originaldokument angegeben sind.
g	
defs	
use	
Filter	Einschränkung: Kindelements fe_xxx lässt sich nicht ändern
Kreis	
Ellypse	
Linie	
polylinie	
Polygon	
Pfad	
rect	
Text, tspan, TextBereich	Das spezifische Attribut "4d-text" wird verwendet, um Text im Element "Text", "tspan" oder "TextBereich" zu ändern (siehe Beispiel)
Bild	

Elemente, deren Attribute sich nicht ändern lassen

linearerGradient, radialerGradient, Stop, Vollfarbe, Marker, Symbol, clipPath, Filter und Elemente, die mit fe beginnen, Stil, Muster	Diese Gruppe bezeichnet alle Elemente, auf die verwiesen werden kann oder die in einem Element enthalten sind, auf das verwiesen werden kann. D.h., es ist nicht möglich, zum Beispiel die Attribute eines Gradienten zu ändern (es ist jedoch möglich, den verwendeten Gradienten zu ändern). Um z.B. einen Marker von schwarz auf rot zu ändern, müssen beide Marker im SVG Dokument definiert werden (einer schwarz und einer rot) und einen davon auswählen. Es ist auch nicht möglich, die Farbe eines Rechtecks zu ändern, wenn das Elternelement ein Symbol oder Marker ist.

Attribute, die sich nicht ändern lassen

id oder xml:id	
lang oder xml:lang	
class oder xml:class	
Breite, Höhe	Betrifft nur die Attribute des 'svg' Elements (1)

(1) Diese Attribute lassen sich nicht ändern, da sie das Ergebnisbild definieren und strukturieren. Mit den Attributen Breite und Höhe des svg Elements können Sie die ursprünglichen Ausmaße des Bildes in 4D definieren. Diese Maße müssen nach Erstellen des Bildes konstant bleiben. (Es ist jedoch möglich, die Ausmaße des Ergebnisbildes über den 4D Befehl **TRANSFORM PICTURE** zu ändern).

Unter dem Befehl **SVG GET ATTRIBUTE** finden Sie die Liste der 4D Attribute, die für Animation reserviert und vorgesehen sind.

Versuchen Sie, das Attribut eines Elements zu verändern, das nicht unterstützt wird oder zu einem Kindelement gehört, führt der Befehl nichts aus und es wird kein Fehler erzeugt.

Wird dieser Befehl nicht im Kontext eines Formulars ausgeführt, oder wurde ein ungültiger Parameter *Bildobjekt* übergeben, wird die Systemvariable *OK* auf 0 gesetzt. Wurde der Befehl korrekt ausgeführt, wird sie auf 1 gesetzt.

Beispiel

Den Inhalt eines Elements vom Typ Text verändern:

```
SVG SET ATTRIBUTE(*;picture_object_name;text_element_ID;"4d-text";"This is a text")
```

Hinweis: Es gibt keinen Namensbereich, so dass das Attribut ohne Risiko eines Namenskonflikts in einem CSS Style Sheet verwendbar ist.

SVG SHOW ELEMENT

SVG SHOW ELEMENT ({* ;} Bildobjekt ; ID {; Rand})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit Stern: BildObjekt ist ein Objektname (String) Ohne Stern: BildObjekt ist eine Variable
Bildobjekt	Bild	➡	Mit *: Objektname, ohne *: Variable
ID	Text	➡	ID Attribut des anzuzeigenden Elements
Rand	Lange Ganzzahl	➡	Marge für Sichtbarkeit (standardmäßig in Pixel)

Beschreibung

Der Befehl **SVG SHOW ELEMENT** bewegt das SVG Element *Bildobjekt*, um das Element mit dem Attribut "ID", definiert im Parameter *ID*, anzuzeigen.

Mit dem optionalen Parameter * geben Sie an, dass der Parameter *Bildobjekt* ein Objektname ist (String). In diesem Fall gilt der Befehl für das gerenderete Bild, das dem Objekt angehängt ist. Ohne diesen Parameter geben Sie an, dass *Bildobjekt* eine Variable ist. In diesem Fall übergeben Sie eine Variablenreferenz (nur Objektvariable) anstelle eines String. Der Befehl gilt dann für die gerenderten Bilder aller Objekte, die diese Variable verwenden, mit Ausnahme des gerenderten Ausgangsbildes.

Der Befehl bewegt das SVG Dokument, so dass das gesamte Objekt, definiert durch das begrenzende Rechteck, sichtbar ist. Mit dem Parameter *Rand* konfigurieren Sie die Amplitude der Bewegung durch Angabe des Abstands, der zwischen Objekt und Rändern des Dokuments liegen muss. Mit anderen Worten, das begrenzende Rechteck wird in Breite und Höhe um die in Rand angegebenen Pixel vergrößert. Standardmäßig sind 4 Pixel vorgegeben.

Dieser Befehl wirkt sich nur im Anzeigemodus "oben links" aus (mit Rollbalken).

Wird dieser Befehl nicht im Kontext eines Formulars ausgeführt, oder wurde ein ungültiger Parameter *Bildobjekt* übergeben, wird die Systemvariable *OK* auf 0 gesetzt. Wurde der Befehl korrekt ausgeführt, wird sie auf 1 gesetzt.

Systemdokumente

- Einführung in Systemdokumente
- 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*

Einführung in Systemdokumente

Einleitung

Alle Dokumente und Anwendungen auf Ihrem Rechner sind in Dateien auf einer bzw. mehreren Festplatten gespeichert, die auf Ihrem Rechner, Ihr(en) Laufwerk(en) oder anderen Speichermedien **angemeldet** oder **angeschlossen** sind. Innerhalb von 4D verwenden wir die Begriffe **Datei** oder **Dokument** für diese Dokumente und Anwendungen. In diesem Abschnitt jedoch arbeiten wir überwiegend mit dem Begriff Dokument, da die meisten Befehle für den Zugriff von Dokumenten und weniger für Anwendungen oder Systemdateien eingesetzt werden.

Eine Festplatte kann als eine oder mehrere Partitions formatiert sein, eine Partition heißt **Volume**. Es spielt keine Rolle, ob auf derselben Festplatte zwei Volumes präsent sind, auf 4D Ebene behandeln Sie diese normalerweise als getrennte und gleiche Einheiten.

Ein Volume kann auf einer Festplatte liegen, die direkt an Ihren Rechner angeschlossen oder über das Netzwerk mit einem File Sharing Protokoll angemeldet ist, wie z.B. TCP/IP, AFP oder SMB auf Macintosh. Wenn Sie Befehle für Systemdokumente auf 4D Ebene einsetzen, behandeln Sie all diese Volumes immer gleich - außer in speziellen Fällen, wenn Sie z.B. mit Plug-Ins die Leistung Ihrer Anwendung in diesem Bereich erweitern.

Jedes Volume hat einen **Volume-Namen**. Unter Windows werden Volumes mit einem Buchstaben gefolgt von einem Doppelpunkt gekennzeichnet. Mit C: und D: werden im Normalfall Volumes zum Booten Ihres Systems bezeichnet - außer, Sie haben Ihren PC anders konfiguriert. Die Buchstaben E: bis Z: werden für zusätzliche Volumes verwendet, die an Ihrem Rechner angeschlossen oder angemeldet werden (USB Treiber, zusätzliche Treiber, Netzwerk Treiber, usw.). Auf Macintosh haben die Volumes Namen, die Sie auf Ihrem Schreibtisch auf der Finder-Ebene sehen können.

Sie unterteilen Ihre Dokumente im allgemeinen in **Ordner**, die wiederum Unterordner enthalten können. Sammeln Sie nicht hunderte von Dateien auf derselben Ebene eines Volumes an, denn das ist verwirrend und verlangsamt Ihr System. Unter Windows heißt ein Ordner auch **Verzeichnis**; auf Macintosh wird schon immer der Begriff Ordner verwendet.

Um ein Dokument eindeutig zu identifizieren, müssen Sie den Namen des Volumes wissen, den Namen des/der Ordner, wo das Dokument liegt und den Dokumentnamen selbst. Diese aneinandergereihten Namen ergeben den **Pfadnamen** des Dokuments. Die Namen werden durch ein spezielles Zeichen, das **Trennungssymbol für Ordner** voneinander getrennt. Unter Windows ist es der umgekehrte Schrägstrich (\); auf Macintosh der Doppelpunkt (:), in POSIX syntax der Schrägstrich (/).

Hierzu ein Beispiel: Sie haben das Dokument **Wichtige Notiz.txt** im Ordner **Notizen**, der im Ordner **Dokumente** liegt, der wiederum im Ordner **Aktuelle Projekte** liegt.

Unter Windows ergibt sich folgender Pfadname, wenn alles auf dem Laufwerk C: (Volume) liegt:

C:\Aktuelle Projekte\Dokumente\Notizen\Wichtige Notiz.txt

Hinweis: Das Zeichen \ wird auch vom 4D Methodeneditor für Escape Sequenzen (Steuerzeichen) verwendet. Um Probleme bei der Interpretation zu vermeiden, wandelt der Editor Pfadnamen wie **C:\Festplatte** automatisch um in **C:\\Festplatte**. Weitere Informationen dazu finden Sie im nachfolgenden Abschnitt **Escape Sequenzen und Pfadnamen unter Windows**.

Auf Macintosh ergibt sich folgender Pfadname, wenn alles auf der Festplatte (Volume) **Interner Treiber** liegt:

Interner Treiber:Aktuelle Projekte:Dokumente:Notizen:Wichtige Notiz.txt

Unter Windows hat der Dokumentname die Endung .txt; die Erläuterungen dazu folgen im nächsten Absatz.

Für den kompletten Pfadnamen gilt, unabhängig von der Plattform folgendes Schema:

VolName Trenner { OrdnerName Trenner { OrdnerName Trenner { ... } } } DokName

Die Dokumente (Dateien) auf den Volumes haben mehrere Merkmale, das sind die **Attribute** oder **Eigenschaften**, wie der **Dokumentname** selbst und seine **Endung**.

DocRef: Referenznummer des Dokuments

Ein Dokument ist **offen im Lese-/Schreibmodus, im Modus Nur-Lesen** oder **geschlossen**. Mit den in 4D integrierten Befehlen kann nur ein Prozess ein Dokument gleichzeitig im Lese-/Schreibmodus öffnen. Ein Prozess kann zwar mehrere Dokumente öffnen und mehrere Prozesse können wiederum mehrere Dokumente öffnen. Sie können dasselbe Dokument beliebig oft im Modus Nur-Lesen öffnen, jedoch nur einmal zur gleichen Zeit im Lese-/Schreibmodus.

Mit den Funktionen **Open document**, **Create document**, **Append document** öffnen Sie ein Dokument. **Create document** und **Append document** öffnen die Dokumente automatisch im Lese-/Schreibmodus. Nur mit **Open document** können Sie den Modus beim Öffnen wählen. Im geöffneten Dokument können Sie Zeichen schreiben sowie aus- und einlesen. Weitere Informationen dazu finden Sie unter den Befehlen **RECEIVE PACKET** und **SEND PACKET**. Haben Sie die Bearbeitung abgeschlossen, schließen Sie das Dokument mit dem Befehl **CLOSE DOCUMENT**.

Der Bezug auf alle offenen Dokumente erfolgt mit dem Ausdruck **DocRef**, den die Funktionen **Open document**, **Create document** und **Append document** zurückgeben. *DocRef* erkennt nur ein offenes Dokument. Es ist formell ein Ausdruck vom Typ Zeit. Alle Befehle und Funktionen, die mit offenen Dokumenten arbeiten, erwarten als Parameter *DocRef*. Übergeben Sie für diese Befehle eine inkorrekte *DocRef,* tritt ein Fehler im Dateimanager auf.

Hinweis: Wird eine Referenz *DocRef* von einem preemptive Prozess aufgerufen, ist sie nur von diesem preemptive Prozess verwendbar. Bei Aufruf von einem kooperativen Prozess ist *DocRef* von jedem anderen kooperativen Prozess verwendbar.

Ein- /Ausgabefehler verwalten

Ein-/Ausgabefehler können auftreten, wenn Sie auf Dokumente zugreifen, also Dokumente öffnen, schließen, löschen, umbenennen bzw. kopieren; wenn Sie die Eigenschaften eines Dokuments ändern; wenn Sie Zeichen in ein Dokument schreiben oder einlesen. Das Dokument wird z.B. nicht gefunden, es ist gesperrt oder bereits im Lese-/Schreibmodus geöffnet. Solche Fehler können Sie über eine Fehlermethode **ON ERR CALL** abfangen. Die häufigsten Fehler, die beim Verwenden von Systemdokumenten auftreten, werden im Abschnitt **OS Systemfehler (-124 -> -33)** beschrieben.

Die Systemvariable Document

Mit den Befehlen **Open document**, **Create document**, **Append document** und **Select document** können Sie auf die Dokumente über die Standarddialogfenster zum Öffnen oder Sichern zugreifen. 4D gibt dann in der Systemvariable *Document* den kompletten Pfadnamen des Dokuments zurück. Verwechseln Sie diese Systemvariable nicht mit dem Parameter *Document*, der in der Parameterliste der Befehle erscheint!

Weitere Informationen dazu finden Sie im Abschnitt **Systemvariablen**.

Escape Sequenzen und Pfadnamen unter Windows

Der Methodeneditor von 4D lässt Escape Sequenzen zu. Das ist eine Zeichenfolge, die bestimmte Zeichen ersetzt. Die Sequenz enthält einen linksgerichteten Schrägstrich (\) und ein Zeichen. \t ersetzt z.B. die Eingabe *Zeichen Tab* für den Tabulator.
Das Zeichen \ wird unter Windows auch als Trenner in Pfadnamen verwendet. In der Regel ersetzt 4D einfache Schrägstriche in Pfadnamen unter Windows durch doppelte Schrägstriche \\, so wird **C:\Ordner** zu **C:\\Ordner**.
Schreiben Sie dagegen **C:\MeinDokument\Neu**, zeigt 4D **C:\\MeinDokument\Neu** an. In diesem Fall wird der zweite Schrägstrich fälschlicherweise als die Escape Sequenz \N interpretiert. Aus diesem Grund müssen Sie bei Zeichen, die 4D als Escape Sequenz erkennt, stets zwei Schrägstriche eingeben \\.

4D arbeitet mit folgenden Sequenzen:

Escape Sequenz	Ersetzte Zeichen
\n	LF (Zeilenvorschub)
\t	HT (Tabulator)
\r	CR (Zeilenschaltung)
\\	\ (linksgerichteter Schrägstrich)
\"	" (Anführungszeichen)

Absolute oder relative Pfadnamen

Die meisten Befehle dieses Abschnitts erlauben **Dokumentnamen**, **relative Pfadnamen** oder **absolute Pfadnamen**:

- **Relative Pfadnamen** definieren einen Speicherort in Bezug auf den Ordner, wo er auf der Festplatte liegt. Geben Sie nur einen **Dokumentnamen** an, gilt das als Verwendung eines relativen Pfadnamens. In 4D wird ein relativer Pfadname in der Regel in Bezug auf den Datenbankordner definiert, z.B. dem Ordner mit der Strukturdatei. Relative Pfadnamen sind besonders hilfreich, wenn Anwendungen in heterogenen Umgebungen eingesetzt werden.
- **Absolute Pfadnamen** definieren einen Speicherort in Bezug auf den Root des Volumes, so dass sie nicht von aktuellen Speicherort des Datenbankordners abhängen.

Zum Bestimmen, ob ein Pfadname, der in einem Befehl übergeben ist, als absolut oder relativ interpretiert wird, wendet 4D für jede Plattform einen spezifischen Algorithmus an.

Unter Windows

Enthält der Parameter nur zwei Zeichen **und** ist das zweite ein ':',
oder enthält der Text ':' und '\' als zweites und drittes Zeichen,
oder ´beginnt der Text mit "\\",
ist der Pfadname **absolute**.

In allen anderen Fällen ist der Pfadname **relativ**.

Beispiele mit dem Befehl **CREATE FOLDER**:

```
CREATE FOLDER("Monday") // relativer Pfad
CREATE FOLDER("\Monday") // relativer Pfad
CREATE FOLDER("\Monday\Tuesday") // relativer Pfad
CREATE FOLDER("c:") // absoluter Pfad
CREATE FOLDER("d:\Monday") // absoluter Pfad
CREATE FOLDER("\\srv-Internal\temp") // absoluter Pfad
```

Auf Mac OS

Startet der Text mit einem Ordnertrenner ':' **oder** enthält er keinen Trenner, ist der Pfad **relativ**.

In allen anderen Fällen ist er **absolut**.

Beispiele mit dem Befehl **CREATE FOLDER**:

```
CREATE FOLDER("Monday") // relativer Pfad
CREATE FOLDER("macintosh hd:") // absoluter Pfad
CREATE FOLDER("Monday:Tuesday") // absoluter Pfad (das Volume heißt Monday)
CREATE FOLDER(":Monday:Tuesday") // relativer Pfad
```

Inhalt von Pfadnamen extrahieren

Sie können Inhalt von Pfadnamen mit den Funktionen **Path to object** und **Object to path** verwalten. Sie können folgendes aus den Pfadnamen entnehmen:

- Dateinamen
- Pfad des Ordners
- Endung der Datei oder des Ordners

Append document

Append document (Dokumentname {; Dateityp}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Dokumentname	String	➡	Dokumentname oder kompletter Pfadname oder leerer String für Standarddialog zum Öffnen von Dokumenten
Dateityp	String	➡	Liste der anzuzeigenden Dateitypen oder *, um Dokumente nicht anzuzeigen
Funktionsergebnis	DokRef	⮌	Referenznummer des Dokuments

Beschreibung

Die Funktion **Append document** arbeitet ähnlich wie **Open document**, mit dem Unterschied, dass die hinzugefügten Daten an das Ende des Dokumentes angefügt werden.

Eine ausführliche Beschreibung finden Sie unter der Funktion **Open document**.

Beispiel

Folgendes Beispiel öffnet ein vorhandenes Dokumen mit Namen Notiz, hängt den String "usw. " sowie eine Zeilenschaltung an das Dokumentende an und schließt dann das Dokument. Enthielt das Dokument bereits den String "Adieu", werden "usw." und die Zeilenschaltung daran angefügt:

```
C_TIME(vhDocRef)
vhDocRef:=Append document("Notiz.txt") ` Öffne Dokument mit Namen Notiz
SEND PACKET(vhDocRef;" usw."+Char(13)) ` Hänge String an
CLOSE DOCUMENT(vhDocRef) ` Schließe Dokument
```

CLOSE DOCUMENT

CLOSE DOCUMENT (DokRef)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenznummer des Dokuments

Beschreibung

CLOSE DOCUMENT schließt das mit *DokRef* angegebene Dokument. Ist das Dokument nicht geöffnet, wird der Befehl nicht ausgeführt.

Achtung: Alle Dokumente, die Sie geöffnet oder erzeugt haben, müssen geschlossen werden. Andernfalls können die Dokumente nur nach Beenden des Programms wieder geöffnet werden. Außerdem können Sie nur dann davon ausgehen, dass die auf eine Festplatte geschriebenen Daten gesichert werden.

Wurde das Dokument mit den Funktionen **Open document**, **Create document** oder **Append document** geöffnet, müssen Sie *DokRef*, die beim Öffnen erhaltene Dokumentnummer, übergeben.

Beispiel

Folgendes Beispiel lässt den Benutzer ein neues Dokument anlegen, schreibt den String “Hallo” hinein und schließt das Dokument:

```
C_TIME(vhDocRef)
vhDocRef:=Create document("")
If(OK=1)
   SEND PACKET(vhDocRef;"Hallo")  ` Schreibe ein Wort in Dokument
   CLOSE DOCUMENT(vhDocRef)  ` Schließe Dokument
End if
```

Convert path POSIX to system

Convert path POSIX to system (posixPfad {; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
posixPfad	Text	→	POSIX Pfadname
*	Operator	→	Codierungsoptionen
Funktionsergebnis	Text	←	Pfadname in Syntax des Systems

Beschreibung

Die Funktion **Convert path POSIX to system** konvertiert einen Pfadnamen mit der Syntax POSIX (Unix) in einen Pfadnamen mit der Syntax des Systems.
Im Parameter *posixPfad* übergeben Sie den kompletten Pfadnamen einer Datei oder eines Ordners, ausgedrückt in der POSIX Syntax. Dieser Pfad muss absolut sein, d.h. mit dem Zeichen "/" beginnen. Sie müssen den Pfad einer Festplatte übergeben. Der Pfad eines Netzwerks ist nicht möglich, er beginnt z.B. mit ftp://ftp.mysite.de.

Diese Funktion gibt den kompletten Pfadnamen der Datei oder des Ordners in der Syntax des aktuellen Systems zurück.
Über den optionalen Parameter * können Sie angeben, ob der Parameter *posixPfad* codiert ist. In diesem Fall müssen Sie den Parameter übergeben, andernfalls ist die Konvertierung nicht gültig. Die Funktion gibt den Pfadnamen ohne Codierung zurück.

Beispiel 1

Beispiele auf Mac OS:

```
$path:=Convert path POSIX to system("/Volumes/machd/file 2.txt")
 //gibt "machd:file 2.txt" zurück
$path:=Convert path POSIX to system("/Volumes/machd/file%202.txt";*)
 //gibt "machd:file 2.txt" zurück
$path:=Convert path POSIX to system("/file 2.txt")
 //gibt "machd:file 2.txt" zurück, wenn machd die startup Festplatte ist
```

Beispiel 2

Beispiele unter Windows:

```
$path:=Convert path POSIX to system("c:/docs/file 2.txt")
 //gibt "c:\\docs\\file 2.txt" zurück
$path:=Convert path POSIX to system("c:/docs/file%202.txt";*)
 //gibt "c:\\docs\\file 2.txt" zurück
```

Convert path system to POSIX

Convert path system to POSIX (SystemPfad {; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
SystemPfad	Text	➡	Relativer oder absoluter Pfadname in System Syntax
*	Operator	➡	Codierungsoptionen
Funktionsergebnis	Text	⮌	Absolute pathname expressed in POSIX syntax

Beschreibung

Die Funktion **Convert path system to POSIX** konvertiert einen Pfadnamen in der Syntax des Systems in einen Pfadnamen in POSIX (Unix) Syntax.

Im Parameter *SystemPfad* übergeben Sie den Pfadnamen für eine Datei oder einen Ordner in der Syntax des Systems (Mac OS oder Windows). Dieser Pfad kann in Bezug auf den Anwendungsordner absolut oder relativ sein. Das ist der Ordner mit der Struktur der Anwendung. Es ist nicht zwingend, dass die Elemente des Pfads auf der Festplatte vorhanden sind, wenn die Funktion ausgeführt wird. Sie testet nicht die Gültigkeit des Pfadnamens.

Die Funktion gibt den kompletten Pfadnamen der Datei oder des Ordners in der POSIX Syntax zurück. Sie gibt immer einen absoluten Pfadnamen zurück, unabhängig, welcher Pfadtyp im Parameter *SystemPfad* übergeben wird. Haben Sie in *SystemPfad* einen relativen Pfadnamen übergeben, vervollständigt 4D den zurückgegebenen Wert durch Hinzufügen des Pfadnamens des Anwendungsordners.

Über den optionalen Parameter * können Sie die Codierung des POSIX Pfads angeben. Die Funktion **Convert path system to POSIX** konvertiert standardmäßig nicht die spezifischen Zeichen des POSIX Pfads. Übergeben Sie den Parameter *, werden die spezifischen Zeichen umgewandelt (z.B. "My folder" wird "My%20folder").

Beispiel 1

Beispiel auf OS X

```
$path:=Convert path system to POSIX("machd:file 2.txt")
 //machd ist die startup Festplatte
 //gibt "/file 2.txt" zurück
$path:=Convert path system to POSIX("disk2:file 2.txt")
 //disk2 ist eine zusätzliche Festplatte (nicht startup)
 //gibt "/Volumes/disk2/file 2.txt" zurück
$path:=Convert path system to POSIX("machd:file 2.txt";*)
 //gibt "/file%202.txt" zurück
$path:=Convert path system to POSIX(":resources:images")  //relativer Pfad
 //gibt "/User/mark/Documents/videodatabase/resources/images" zurück
$path:=Convert path system to POSIX("resources:images")  //absoluter Pfad
 //gibt "/resources/images" zurück
```

Beispiel 2

Beispiel unter Windows:

```
$path:=Convert path system to POSIX("c:\docs\file 2.txt")
 `gibt zurück "c:/docs/file 2.txt"
$path:=Convert path system to POSIX("\\srv\tempo\file.txt")
 `gibt zurück "//srv/tempo/file.txt"
```

COPY DOCUMENT

COPY DOCUMENT (QuellName ; ZielName {; Neuer Name} {; *})

Parameter	Typ		Beschreibung
QuellName	String	➡	zu kopierender Pfadname der Datei oder des Ordners
ZielName	String	➡	Name oder Pfadname der kopierten Datei bzw. des Ordners
Neuer Name	String	➡	Neuer Name der kopierten Datei bzw. des Ordners
*	Operator	➡	Überschreibe Dokument, wenn vorhanden

Beschreibung

Der Befehl **COPY DOCUMENT** kopiert die in *QuellName* angegebene Datei bzw. den Ordner an den in *ZielName* angegebenen Ort und benennt es um (optional).

- **Dateien kopieren**
  In diesem Fall muss der Parameter *QuellName* den kompletten Pfadnamen enthalten gemäß dem Ursprung des Volumes. Der Parameter *ZielName* kann verschiedene Typen enthalten:
  - Den kompletten Pfadnamen der Datei gemäß dem Ursprung des Volumes: Die Datei wird an diese Stelle kopiert.
  - Einen Dateinamen oder relativen Pfadnamen der Datei: Die Datei wird in den Ordner der Datenbank kopiert (der Unterordner muss bereits vorhanden sein)
  - Einen kompletten Pfadnamen des Ordners oder einen Pfadnamen in Bezug auf den Ordner der Datenbank (*ZielName* muss mit dem Trenner für Ordner der jeweiligen Plattform enden): Die Datei wird in den angegebenen Ordner kopiert. Diese Ordner müssen bereits auf der Festplatte vorhanden sein, sie werden nicht angelegt.

  Gibt es bereits ein Dokument mit Namen *ZielName*, wird ein Fehler erzeugt, außer Sie geben den optionalen Parameter * an, der **COPY DOCUMENT** anweist, das am Zielort vorhandene Dokument zu löschen und zu ersetzen.

- **Ordner kopieren**
  Für einen Ordner müssen die Zeichenketten, übergeben in *QuellName* und *ZielName*, mit dem Trenner der jeweiligen Plattform enden. Unter Windows bezeichnet z.B. "C:\\Element\\" einen Ordner und "C:\\Element" eine Datei.
  Um einen Ordner zu kopieren, übergeben Sie in *QuellName* den kompletten Pfadnamen. Dieser Ordner muss bereits auf der Festplatte vorhanden sein. Geben Sie im Parameter *QuellName* einen Ordner an, müssen Sie auch im Parameter *ZielName* einen Ordner angeben. Sie müssen den kompletten Pfadnamen übergeben, wobei jedes Element bereits auf der Festplatte vorhanden sein muss
  Gibt es an der im Parameter *ZielName* angegebenen Stelle bereits einen Ordner mit demselben Namen wie im Parameter *QuellName* angegeben, und ist er nicht leer, prüft 4D vor dem Kopieren der Einträge seinen Inhalt. Existiert bereits eine Datei mit demselben Namen, wird ein Fehler erzeugt, außer der optionale Parameter * wurde übergeben. In diesem Fall wird die Datei an der in *ZielName* angegebenen Stelle gelöscht und ersetzt.

Wollen Sie eine Datei in einen Ordner kopieren, können Sie in *QuellName* eine Datei und in *ZielName* einen Ordner übergeben. Ist der optionale Parameter *NeuerName* übergeben, benennt er das kopierte Dokument an seinem Zielort um (Datei oder Ordner). Wird er beim Kopieren einer Datei übergeben, ersetzt er den Namen (sofern vorhanden) im Parameter *ZielName*.

Beispiel 1

Folgendes Beispiel dupliziert ein Dokument in seinen eigenen Ordner:

```
COPY DOCUMENT("C:\\FOLDER\\DocName";"C:\\FOLDER\\DocName2")
```

Beispiel 2

Folgendes Beispiel kopiert ein Dokument in den Datenbankordner (vorausgesetzt *C:\\FOLDER* ist nicht der Datenbankordner):

```
COPY DOCUMENT("C:\\FOLDER\\DocName";"DocName")
```

Beispiel 3

Folgendes Beispiel kopiert ein Dokument von einem Volume zu einem anderen:

```
COPY DOCUMENT("C:\\FOLDER\\DocName";"F:\\Archives\\DocName.OLD")
```

Beispiel 4

Folgendes Beispiel dupliziert ein Dokument in seinen eigenen Ordner und überschreibt die bereits vorhandene Kopie:

```
COPY DOCUMENT("C:\\FOLDER\\DocName";"C:\\FOLDER\\DocName2";*)
```

Beispiel 5

Eine Datei in einen bestimmten Ordner kopieren, unter Beibehaltung des Namens:

```
COPY DOCUMENT("C:\\Projects\\DocName";"C:\\Projects\\")
```

Beispiel 6

Eine Datei in einen bestimmten Ordner kopieren, unter Beibehaltung des Namens und mit Überschreiben des vorhandenen Dokuments:

```
COPY DOCUMENT("C:\\Projects\\DocName";"C:\\Projects\\"; *)
```

Beispiel 7

Einen Ordner in einen anderen Ordner kopieren (beide Ordner müssen bereits auf der Festplatte vorhanden sein):

```
COPY DOCUMENT("C:\\Projects\\";"C\\Archives\\2011\\")
```

Beispiel 8

Die folgenden Beispiele erstellen unterschiedliche Dateien und Ordner im Datenbankordner (Beispiele unter Windows). In allen Fällen muss der Ordner "folder2" vorhanden sein:

```
COPY DOCUMENT("folder1\\name1";"folder2\\")
 //erstellt die Datei "folder2/name1"

COPY DOCUMENT("folder1\\name1";"folder2\\" ; "new")
 //erstellt die Datei "folder2/new"

COPY DOCUMENT("folder1\\name1";"folder2\\name2")
 //erstellt die Datei "folder2/name2"

COPY DOCUMENT("folder1\\name1";"folder2\\name2";"new")
 //erstellt die Datei "folder2/new" (name2 wird ignoriert)

COPY DOCUMENT("folder1\\" ; "folder2\\")
 //erstellt den Ordner "folder2/folder1/"

COPY DOCUMENT("folder1\\" ; "folder2\\" ; "new")
 //erstellt den Ordner "folder2/new/"
```

CREATE ALIAS

CREATE ALIAS (OriginalPfad ; AliasPfad)

Parameter	Typ		Beschreibung
OriginalPfad	String	➡	Name oder Pfad zum Originaldokument bzw. -ordner
AliasPfad	String	➡	Name oder voller Pfadname für Verknüpfung/Alias

Beschreibung

Der Befehl **CREATE ALIAS** erstellt unter Windows eine Verknüpfung, auf Mac OS ein Alias für die Zieldatei bzw. den Zielordner, übergeben in *OriginalPfad*. Dieser Parameter definiert Name und Ort.
Sie können von jeder Art Dokument bzw. Ordner ein Alias machen. Sein Icon ist gleich mit dem Ziel-Icon. Unter Windows enthält es links unten einen Pfeil, auf Mac OS erscheint sein Name in Kursivschrift.

Dieser Befehl weist standardmäßig keinen Namen zu. Sie müssen im Parameter *AliasPfad* einen Namen übergeben. Enthält der Parameter nur einen Namen, wird das Alias in den aktuellen Arbeitsordner gelegt. Das ist normalerweise der Ordner mit der Strukturdatei.

Hinweis: Unter Windows hat die Verknüpfung die nicht sichtbare Endung ".LNK". Wird diese Endung nicht übergeben, fügt sie der Befehl automatisch hinzu.

Wird in *OriginalPfad* ein leerer String übergeben, hat der Befehl keine Auswirkung.

Beispiel

Ihre Datenbank erstellt Textdateien mit Namen "BerichtNr", die im Datenbankordner sortiert werden. Der Benutzer möchte auf diese Dateien Verknüpfungen bzw. Aliase einrichten und diese an einem passenden Ort speichern:

```
  `Methode CREATE_REPORT
C_TEXT($vtRport)
C_STRING(250;$vtpath)
C_STRING(80;$vaname)
C_TIME(vDoc)
C_INTEGER($ReportNber)

$vTReport:=... `Erstelle Bericht
$ReportNber:=... `Berechne die Berichtnummer
$vaName:="Report"+String($ReportNber)+".txt" `Dateiname
vDoc:=Create document($vaName)
If(OK=1)
   SEND PACKET(vDoc;$vTReport)
   CLOSE DOCUMENT(vDoc)
 `Alias hinzufügen
   CONFIRM("Für diesen Bericht ein Alias erstellen?")
   If(OK=1)
      $vtPath:=Select folder("Wo soll das Alias erstellt werden?")
      If(OK=1)
         CREATE ALIAS($vaName;$vtPath+$vaName)
         If(OK=1)
            SHOW ON DISK($vtPath+$vaName)
 `Zeige Ort des Alias
         End if
      End if
   End if
End if
```

Systemvariablen und Mengen

Die Systemvariable OK wird auf 1 gesetzt, wenn der Befehl erfolgreich ausgeführt wurde; sonst hat sie den Wert 0 (Null).

Create document

Create document (Dokumentname {; Dateityp}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Dokumentname	String	➡	Dokumentname oder kompletter Pfadname oder leerer String für Standarddialog zum Öffnen von Dokumenten
Dateityp	String	➡	Liste der anzuzeigenden Dateitypen oder *, um Dokumente nicht anzuzeigen
Funktionsergebnis	DokRef	⬅	Referenznummer des Dokuments

Beschreibung

Die Funktion **Create document** erzeugt ein neues Dokument und gibt dessen Referenznummer zurück.

Sie übergeben den Dokumentnamen oder den kompletten Pfadnamen in *Dokumentname*. Gibt es *Dokumentname* bereits auf der Festplatte, wird es überschrieben. Ist *Dokumentname* gesperrt oder bereits geöffnet, erhalten Sie eine Fehlermeldung.

Ist *Dokumentname* ein leerer Text, zeigt **Create document** den Standarddialog zum Erstellen von Dokumenten an. Brechen Sie den Dialog ab, wird kein Dokument erstellt. **Create document** gibt in RefNr Null zurück und setzt die OK Variable auf 0.

Wurde das Dokument korrekt erstellt und geöffnet, gibt **Create document** die Referenznummer des Dokuments zurück und setzt die Systemvariable OK auf 1. Die Systemvariable Document wird aktualisiert und gibt den kompletten Zugriffspfad des erstellten Dokuments zurück.

Create document erstellt standardmäßig unter Windows ein Dokument .TXT, auf Macintosh ein Dokument TEXT. Wollen Sie einen anderen Typ anlegen, übergeben Sie den Parameter *Dateityp*.
Im Parameter *Dateityp* übergeben Sie den/die Dateitypen, die im Öffnen-Dialog auswählbar sind. Sie können eine Liste der Dokumenttypen übergeben, getrennt durch Strichpunkt (;). Für jeden gesetzten Typ wird im Menü zum Auswählen des Typs im Dialogfenster ein Eintrag hinzugefügt.
Auf Mac OS können Sie einen Macintosh Standardtyp (TEXT, APPL, etc.) oder einen UTI Typ(Uniform Type Identifier) übergeben. Apple hat UTIs zur Standardisierung von Dateitypen definiert. So ist z.B. "public.text" der UTI Typ für Dateien vom Typ Text. Weitere Informationen dazu finden Sie im Internet unter:
https://developer.apple.com/library/mac/#documentation/Miscellaneous/Reference/UTIRef/Articles/System-DeclaredUniformTypeIdentifiers.html.

Unter Windows können Sie auch einen Mac OS Standardtyp übergeben - 4D sorgt intern für die Entsprechung - oder eine Dateierweiterung (.txt, .exe, etc.) Beachten Sie, dass der Benutzer unter Windows durch Eingabe von ** die Anzeige aller Dateitypen erzwingen kann. In diesem Fall führt 4D jedoch eine zusätzliche Überprüfung der gewählten Dateitypen aus: Wählt der Benutzer einen nicht zugelassenen Dateityp, gibt die Funktion einen Fehler zurück.
Wollen Sie die angezeigten Dateien nicht auf bestimmte Typen beschränken, übergeben Sie in Dateityp den String * (Stern) oder .*.

Unter Windows übergeben Sie eine Endung oder einen Macintosh-Dateityp, der mit dem Befehl **_o_MAP FILE TYPES** automatisch umgesetzt wird. Wollen Sie ein Dokument ohne Endung, mit mehreren Endungen bzw. mit einer längeren Endung, verwenden Sie nicht den Parameter *Dateityp*, sondern übergeben den kompletten Namen in *Dokumentname* (siehe Beispiel 2).

Mit den Befehlen **RECEIVE PACKET** und **SEND PACKET**, kombiniert mit **Get document position** und **SET DOCUMENT POSITION** können Sie direkt auf jeden beliebigen Teil des Dokuments zugreifen.

Vergessen Sie nicht, bei Bedarf **CLOSE DOCUMENT** für das Dokument aufzurufen.

Beispiel 1

Folgendes Beispiel erstellt und öffnet ein neues Dokument mit Namen Notiz, trägt den String “Hallo” ein und schließt das Dokument wieder:

```
C_TIME(vhDocRef)
vhDocRef:=Create document("Notiz.txt") ` Erstelle neues Dokument mit Namen Notiz
If(OK=1)
   SEND PACKET(vhDocRef;"Hallo") ` Schreibe ein Wort in das Dokument
   CLOSE DOCUMENT(vhDocRef) ` Schließe Dokument
End if
```

Beispiel 2

Folgendes Beispiel erstellt Dokumente mit nicht-standardmäßigen Endungen unter Windows:

```
$vtMyDoc:=Create document("Doc.ext1.ext2") `Mehrere Endungen
$vtMyDoc:=Create document("Doc.shtml") `Lange Endung
$vtMyDoc:=Create document("Doc.") `Keine Endung (das Zeichen "." ist zwingend)
```

Systemvariablen und Mengen

Wurde das Dokument korrekt erstellt, wird die Systemvariable OK auf 1 gesetzt, die Systemvariable *Document* enthält den kompletten Pfadnamen und den Namen von *Dokumentname*.

CREATE FOLDER

CREATE FOLDER (Ordnerpfad {; *})

Parameter	Typ		Beschreibung
Ordnerpfad	String	➡	Pfadname für neu anzulegenden Ordner
*	Operator	➡	Ordnerhierarchie erstellen

Beschreibung

Der Befehl **CREATE FOLDER** erstellt einen Ordner mit dem in *Ordnerpfad* übergebenen Pfadnamen.

Übergeben Sie einen Namen, wird der Ordner im Datenbankordner angelegt.

Sie können auch eine Ordnerhierarchie übergeben, die am Ursprung eines Volumes oder auf Ebene eines Datenbankordners startet. In diesem Fall muss der String mit dem Ordnertrenner enden.
Lassen Sie den Parameter * weg, wird ein Fehler erzeugt und es wird kein Ordner angelegt, wenn einer der dazwischenliegenden Ordner nicht vorhanden ist.
Übergeben Sie den Parameter *, erstellt **CREATE FOLDER** die Ordnerhierarchie bei Bedarf und es wird kein Fehler erzeugt. In diesem Fall können Sie in *Ordnerpfad* auch den Pfadnamen eines Dokuments übergeben. Dann wird der Dokumentname ignoriert, aber die Hierarchie der in *Ordnerpfad* angegebenen Ordner wird rekursiv erstellt.

Beispiel 1

Folgendes Beispiel erstellt den Ordner "Archiv" im Datenbankordner :

```
CREATE FOLDER("Archiv")
```

Beispiel 2

Folgendes Beispiel erstellt den Ordner "Archiv" im Datenbankordner und dann die Unterordner "Januar" und "Februar":

```
CREATE FOLDER("Archiv")
CREATE FOLDER("Archiv\\Januar")
CREATE FOLDER("Archiv\\Februar")
```

Beispiel 3

Folgendes Beispiel erstellt den Ordner "Archiv" auf der obersten Ebene des Volumes C:

```
CREATE FOLDER("C:\\Archiv")
```

Beispiel 4

Folgendes Beispiel liefert kein Ergebnis, wenn auf der obersten Ebene des Volumes C kein Ordner "NeuesMat" existiert:

```
CREATE FOLDER("C:\\NeuesMat\\Bilder")
  ` FALSCH, in einem Aufruf können nicht zwei Ordnerebenen angelegt werden
```

Beispiel 5

Im vorhandenen Ordner "C:\Archiv\" den Unterordner "\Februar\" anlegen:

```
CREATE FOLDER("C:\\Archiv\\2011\\Februar\\Doc.txt";*)
  // die Datei "Doc.txt" wird ignoriert
```

DELETE DOCUMENT

DELETE DOCUMENT (Dokumentname)

Parameter	Typ		Beschreibung
Dokumentname	String	➡	Dokumentname oder kompletter Pfadname

Beschreibung

Der Befehl **DELETE DOCUMENT** löscht das unter dem Namen *Dokumentname* abgespeicherte Dokument. Sie erhalten eine Fehlermeldung, wenn der Dokumentname bzw. der Pfadname nicht korrekt eingegeben wurden oder Sie versuchen, ein geöffnetes Dokument zu löschen.

Um Zugriff auf ein Dokument zu erhalten, das sich außerhalb Ihres Datenbankordners befindet, müssen Sie den Pfad zu dem Dokument in *Dokumentname* genau angeben.

DELETE DOCUMENT akzeptiert keinen leeren Text. Ist *Dokumentname* ein leerer Text, erscheint nicht der Standarddialog zum Öffnen von Dokumenten. Sie erhalten eine Fehlermeldung.

Warnung: DELETE DOCUMENT kann jede Datei auf einer Festplatte löschen, wie zum Beispiel eine Ihrer Datenbanken oder die Systemdatei. Verwenden Sie deshalb diesen Befehl mit äußerster Vorsicht, denn der Löschvorgang lässt sich nicht mehr rückgängig machen.

Beispiel 1

Folgendes Beispiel löscht das Dokument mit Namen Notiz:

```
DELETE DOCUMENT("Notiz") `Lösche Dokument
```

Beispiel 2

Siehe Beispiel zum Befehl **APPEND DATA TO PASTEBOARD**.

Systemvariablen und Mengen

Wird ein Dokument gelöscht, hat die Systemvariable OK den Wert 1. Konnte **DELETE DOCUMENT** das Dokument nicht löschen, hat die Systemvariable OK den Wert 0.

DELETE FOLDER

DELETE FOLDER (Ordner {; LöschOption})

Parameter	Typ		Beschreibung
Ordner	String	➡	Name oder ganzer Pfad des Ordners zum Löschen
LöschOption	Lange Ganzzahl	➡	Option Ordner Löschen

Beschreibung

Der Befehl **DELETE FOLDER** löscht den Ordner, dessen Name bzw. voller Pfad in *Ordner* übergeben wurde.

Aus Sicherheitsgründen lässt **DELETE FOLDER** standardmäßig nur das Löschen leerer Ordner zu. Damit der Befehl auch nicht-leere Ordner löschen kann, müssen Sie den Parameter *LöschOption* übergeben. Sie können eine der folgenden Konstanten unter dem Thema "**Systemdokumente**" übergeben:

Konstante	Typ	Wert	Kommentar
Delete only if empty	Lange Ganzzahl	0	Löscht den Ordner nur, wenn er leer ist
Delete with contents	Lange Ganzzahl	1	Löscht den Ordner mitsamt seinem Inhalt

- Ist Delete only if empty (0) übergeben oder der Parameter *LöschOption* wird weggelassen:
  - Der im Parameter *Ordner* angegebene Ordner wird nur gelöscht, wenn er leer ist; andernfalls führt der Befehl nichts aus und es wird ein Fehler -47 generiert (Die Datei ist schon offen oder der Ordner ist nicht leer.)
  - Existiert der angegebene Ordner nicht, wird der Fehler -120 generiert (Dateizugriff über Pfadname zu einem nicht existierenden Ordner)

- Ist Delete with contents (1) übergeben:
  - Der im Parameter *Ordner* angegebene Ordner wird mit dem gesamten Inhalt gelöscht.
    Warnung: Selbst wenn dieser Ordner bzw. sein Inhalt gesperrt oder auf Nur-Lesen gesetzt sind, und der aktuelle Benutzer passende Zugriffsrechte hat, werden Ordner und Inhalt trotzdem gelöscht.
  - Kann dieser Ordner oder eine darin enthaltene Datei nicht gelöscht werden, wird der Löschvorgang beim ersten nicht löschbaren Element abgebrochen und ein Fehler (*) zurückgegeben. Der Ordnerinhalt kann also teilweise gelöscht sein. Bei Abbruch können Sie über den Befehl **GET LAST ERROR STACK** Name und Pfad der dafür verantwortlichen Datei ermitteln.
  - Existiert der im Parameter *Ordner* angegebene Ordner nicht, führt der Befehl nichts aus und es wird kein Fehler zurückgegeben.

    (*) unter Windows: -54 (Es wurde versucht, ein gesperrtes Volume im Schreibmodus zu öffnen)
    auf OS X: -45 (Datei ist gesperrt oder der Pfadname ist nicht korrekt)

Sie können diese Fehler mit einer Methode abfangen, die der Befehl **ON ERR CALL** einrichtet.

DOCUMENT LIST

DOCUMENT LIST (Pfadname ; Dokumente {; Optionen})

Parameter	Typ		Beschreibung
Pfadname	String	➡	Pfadname zu Volume, Festplatte oder Ordner
Dokumente	Array Text	⬅	Namen der an dieser Stelle vorhandenen Dokumente
Optionen	Lange Ganzzahl	➡	Optionen zum Erstellen von Listen

Beschreibung

Der Befehl **DOCUMENT LIST** füllt das Array *Dokumente* mit den Namen der Dokumente, die unter dem in *Pfadname* übergebenen Pfadnamen liegen.

Hinweis: *Pfadname* akzeptiert nur absolute Pfadnamen.

Standardmäßig, d.h. ohne den Parameter *Optionen* werden im Array *Dokumente* nur die Namen der Dokumente zurückgegeben. Das können Sie ändern, wenn Sie im Parameter *Optionen* eine bzw. mehrere der folgenden Konstanten unter dem Thema **Systemdokumente** übergeben:

Konstante	Typ	Wert	Kommentar
Absolute path	Lange Ganzzahl	2	Das Array Dokumente enthält absolute Pfadnamen
Ignore invisible	Lange Ganzzahl	8	Unsichtbare Dokumente werden nicht aufgelistet
Posix path	Lange Ganzzahl	4	Das Array Dokumente enthält Pfadnamen im Posix Format
Recursive parsing	Lange Ganzzahl	1	Das Array Dokumente enthält alle Dateien und Unterordner des angegebenen Ordners

Hinweise:

- Mit der Option Recursive parsing im relativen Modus (nur Option 1), beginnen die Pfade von Dokumenten in Unterordnern je nach Plattform mit dem Zeichen ":" oder "\" .
- Mit der Option Posix path im relativen Modus (nur Option 4) beginnen Pfade nicht mit "/".
- Mit der Option Posix path im absoluten Modus (Optionen 4 + 2) beginnen Pfade immer mit "/".

Gibt es keine Dokumente an der angegebenen Stelle, gibt der Befehl ein leeres Array zurück. Ist der in *Pfadname* übergebene Pfadname ungültig, erzeugt **DOCUMENT LIST** einen OS Systemfehler, den Sie mit einer Methode **ON ERR CALL** abfangen können.

Beispiel 1

Liste aller Dokumente in einem Ordner (Standardsyntax):

```
DOCUMENT LIST("C:\\";arrFiles)
```

-> arrFiles:
Text1.txt
Text2.txt

Beispiel 2

Liste aller Dokumente in einem Ordner im absoluten Modus:

```
DOCUMENT LIST("C:\\";arrFiles; Absolute path)
```

-> arrFiles:
C:\Text1.txt
C:\Text2.txt

Beispiel 3

Liste aller Dokumente im rekursiven (relativen) Modus:

```
DOCUMENT LIST("C:\\";arrFiles;Recursive parsing)
```

-> arrFiles:
Text1.txt
Text2.txt
\Folder1\Text3.txt
\Folder1\Text4.txt

\Folder2\Text5.txt
\Folder2\Folder3\Picture1.png

Beispiel 4

Liste aller Dokumente im rekursiven absoluten Modus:

```
DOCUMENT LIST("C:\\MyFolder\\";arrFiles;Recursive parsing+Absolute path)
```

-> 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

Beispiel 5

Liste aller Dokumente im rekursiven Posix (relativen) Modus:

```
DOCUMENT LIST("C:\\MyFolder\\";arrFiles;Recursive parsing+Posix path)
```

-> arrFiles:
MyText1.txt
MyText2.txt
Folder1/MyText3.txt
Folder1/MyText4.txt
Folder2/MyText5.txt
Folder2/Folder3/MyPicture1.png

Document to text

Document to text (DateiName {; Zeichensatz {; UmbruchModus}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
DateiName	String	➡	Dokumentname oder Pfadname zum Dokument
Zeichensatz	Text, Lange Ganzzahl	➡	Name oder Nummer des Zeichensatzes
UmbruchModus	Lange Ganzzahl	➡	Bearbeitungsmodus für Zeilenumbrüche
Funktionsergebnis	Text	↩	Text aus dem Dokument

Beschreibung

Die Funktion **Document to text** ermöglicht, den Inhalt einer Datei direkt auf der Festplatte in einer 4D Textvariablen oder einem Textfeld wiederzufinden.

In *DateiName* übergeben Sie Name oder Pfadname der Datei zum Lesen. Die Datei muss auf der Festplatte vorhanden sein, sonst wird ein Fehler erzeugt. Sie können folgendes übergeben:

- Nur den Dateinamen, z.B. "meineDatei.txt": In diesem Fall muss die Datei neben der Strukturdatei der Anwendung liegen.
- Einen Pfadnamen in Bezug auf die Strukturdatei der Anwendung, z.B. "\\docs\\meineDatei.txt" unter Windows oder ":docs:meineDatei.txt" auf OS X.
- Einen absoluten Pfadnamen, z.B. "c:\\app\\docs\\meineDatei.txt" unter Windows oder "MacHD:docs:meineDatei.txt" auf OS X.

In *Zeichensatz* übergeben Sie den Zeichensatz zum Lesen des Inhalts. Sie können einen String mit dem standardmäßigen Namen übergeben, z.B. “ISO-8859-1” oder “UTF-8” oder seine MIBEnum ID (Lange Ganzzahl). Weitere Informationen zur Liste der Zeichensätze, die 4D unterstützt, finden Sie in der Beschreibung zum Befehl **CONVERT FROM TEXT**.

Bei Dokumenten mit Byte Order Mark (BOM) verwendet 4D den hier gesetzten Zeichensatz anstelle der Vorgabe in *Zeichensatz* (dieser Parameter wird dann ignoriert).

Bei Dokumenten ohne BOM und ohne Angabe des Parameters *Zeichensatz* verwendet 4D standardmäßig folgende Zeichensätze:

- unter Windows: ANSI
- auf OS X: MacRoman

In *UmbruchModus* können Sie eine Lange Ganzzahl übergeben, um die Handhabung der Zeichen für Zeilenende im Dokument anzugeben. Sie können eine der nachfolgenden Konstanten unter dem Thema **Systemdokumente** übergeben:

Konstante	Typ	Wert	Kommentar
Document unchanged	Lange Ganzzahl	0	Keine Bearbeitung
Document with CR	Lange Ganzzahl	3	Zeilenumbrüche werden in das Mac OS Format konvertiert: CR (carriage return)
Document with CRLF	Lange Ganzzahl	2	Zeilenumbrüche werden in das Windows Format konvertiert: CRLF (carriage return + line feed)
Document with LF	Lange Ganzzahl	4	Zeilenumbrüche werden in das Unix Format konvertiert: LF (line feed)
Document with native format	Lange Ganzzahl	1	(Standard) Zeilenumbrüche werden in das native Format des Betriebssystems konvertiert: CR (carriage return auf Mac OS), CRLF (carriage return + line feed unter Windows)

Lassen Sie den Parameter *UmbruchModus* weg, werden Zeilenumbrüche im native Modus (1) gehandhabt.

Hinweis: Diese Funktion verändert nicht die Variable OK. Schlägt die Operation fehl, wird ein Fehler generiert, den Sie mit einer Methode abfangen können, die der Befehl **ON ERR CALL** installiert.

Beispiel

Nehmen wir folgendes Textdokument (Felder sind durch Tabs getrennt):

```
id    name    price    vat
3    4D Tags    99    19.6
```

Führen Sie diesen Code aus:

```
$Text:=Document to text("products.txt")
```

... erhalten Sie:

```
// $Text = "id\tname\tprice\tvat\r\n3\t4D Tags\t99 \t19.6"
// \t = tab
// \r = CR
```

FOLDER LIST

FOLDER LIST (Pfadname ; Verzeichnisse)

Parameter	Typ		Beschreibung
Pfadname	String	➡	Pfadname zu Volume, Verzeichnis oder Ordner
Verzeichnisse	Array String	⬅	Namen der an dieser Stelle vorhandenen Verzeichnisse

Beschreibung

Der Befehl **FOLDER LIST** füllt *Verzeichnisse*, das Array vom Typ Text oder Alpha mit den Namen der Ordner, die unter dem in *Pfadname* übergebenen Pfadnamen liegen.

Hinweis: *Pfadname* akzeptiert nur absolute Pfadnamen.

Sind am angegebenen Ort keine Ordner vorhanden, gibt der Befehl ein leeres Array zurück. Ist *Pfadname* ungültig, erzeugt **FOLDER LIST** einen Systemfehler, den Sie mit einer Fehlermethode **ON ERR CALL** abfangen können.

Warnung: Der Parameter *Pfadname* darf max. 255 Zeichen lang sein. Ein längerer Pfadname wird abgeschnitten. Dann erscheint ein Systemfehler.

GET DOCUMENT ICON

GET DOCUMENT ICON (DokPfad ; Icon {; Größe})

Parameter	Typ		Beschreibung
DokPfad	String	➡	Name oder Pfad des Dokuments des zu ladenden Icon oder leerer String für Standard-Öffnen Dialog
Icon	Bildfeld, Bildvariable	⬅	Variable oder Feld vom Typ Bild, in welche das Icon geladen werden soll
Größe	Lange Ganzzahl	⬅	Größe des zurückgegebenen Bilds (in Pixel)

Beschreibung

Der Befehl **GET DOCUMENT ICON** gibt in einer 4D Bildvariablen oder in einem 4D Feld das Icon des Dokuments zurück, dessen Name in *DokPfad* übergeben wurde. Das Dokument kann von einem beliebigen Typ sein (Dokument, exe-Datei, Alias...). Der Befehl gibt jedoch keine Icons von Ordnern zurück.

DokPfad sollte den vollständigen Pfadnamen zum Dokument enthalten. Übergeben Sie nur den Dokumentnamen, muss das Dokument im Ordner mit der Strukturdatei der Datenbank liegen.

Übergeben Sie in *DokPfad* einen leeren String, erscheint der Standard-Öffnen-Dialog. Hier kann der Benutzer das entsprechende Dokument auswählen. Wird das Dialogfenster bestätigt, enthält die Systemvariable *Document* den vollständigen Pfadnamen zum ausgewählten Dokument.

Übergeben Sie in *Icon* eine 4D Variable bzw. ein 4D Feld vom Typ Bild. Wurde der Befehl ausgeführt, enthält dieser Parameter das Icon des Dokuments im PICT Format.

Mit dem optionalen Parameter *Größe* können Sie die Größe des zurückgegebenen Icon in Pixel setzen. Dieser Wert gibt die Seitenlänge des Quadrats für das Icon an. Icons haben normalerweise die Größe 32x32 Pixel (“große Icons”) oder 16x16 Pixel (“kleine Icons”).
Übergeben Sie 0 (Null) oder geben diesen Parameter nicht an, wird der größtmöglichste Icon zurückgegeben.

Get document position

Get document position (DokRef) -> Funktionsergebnis

Parameter	Typ		Beschreibung
DokRef	DokRef	⇒	Referenznummer des Dokuments
Funktionsergebnis	Zahl	⇐	Position der Datei (in Bytes) ab Dateibeginn

Beschreibung

Die Funktion **Get document position** arbeitet nur in einem geöffneten Dokument mit der Referenznummer, übergeben in *DokRef*.

Get document position gibt die Position ab Beginn des Dokuments zurück, wo das nächste Lesen (**RECEIVE PACKET**) oder Schreiben (**SEND PACKET**) auftritt.

GET DOCUMENT PROPERTIES

GET DOCUMENT PROPERTIES (Dokumentname ; Gesperrt ; Ausgeblendet ; Erstellt am ; Erstellt um ; Geändert am ; Geändert um)

Parameter	Typ		Beschreibung
Dokumentname	String	➡	Name des Dokuments
Gesperrt	Boolean	⬅	Gesperrt (True) oder freigegeben (False)
Ausgeblendet	Boolean	⬅	Unsichtbar (True) oder sichtbar (False)
Erstellt am	Datum	⬅	Datum der Erstellung
Erstellt um	Zeit	⬅	Uhrzeit der Erstellung
Geändert am	Datum	⬅	Datum der letzten Änderung
Geändert um	Zeit	⬅	Uhrzeit der letzten Änderung

Beschreibung

Der Befehl **GET DOCUMENT PROPERTIES** gibt Information über das Dokument zurück, dessen Name oder Pfadname in *Dokumentname* übergeben wurde.

Nach dem Aufruf gibt:

- *Gesperrt* den Wert wahr zurück, wenn das Dokument gesperrt ist. Ein gesperrtes Dokument lässt sich weder öffnen noch löschen.
- *Ausgeblendet* den Wert wahr zurück, wenn das Dokument unsichtbar ist.
- *Erstellt am* und *Erstellt um* Datum und Uhrzeit des erstellten Dokuments zurück.
- *Geändert am* und *Geändert um* Datum und Uhrzeit der letzten Änderung des Dokuments zurück.

Beispiel

Sie haben eine Dokumentationsdatenbank erstellt und wollen alle erstellten Datensätze der Datenbank in Dokumente auf der Festplatte exportieren. Da die Datenbank regelmäßig aktualisiert wird, schreiben Sie einen Algorithmus für den Export, der jedes Dokument auf der Festplatte erstellt bzw. wieder erstellt, wenn dieses noch nicht existiert bzw. der dazugehörige Datensatz nach dem letzten Export geändert wurde. Von daher müssen Sie Änderungsdatum und Uhrzeit eines Dokuments mit dem entsprechenden Datensatz vergleichen.

Wir gehen von folgender Tabelle aus:

Sie speichern Datum und Uhrzeit nicht in jedem Datensatz, sondern in einem "Zeitstempel". Dieser misst die Anzahl Sekunden, die von einem bestimmten Stichdatum und einer bestimmten Stichzeit bis zu Datum und Zeit der Sicherung des Datensatzes verstrichen sind. In diesem Beispiel verwenden wir den 1. Jan 1995 um 00:00:00.

Das Datenfeld *[Documents]Creation Stamp* enthält den Zeitstempel der Erstellung des Datensatzes, das Datenfeld *[Documents]Modification Stamp* den Zeitstempel der letzten Änderung.

Die Projektmethode ***Time stamp*** berechnet den Zeitstempel für ein bestimmtes Datum und eine bestimmte Uhrzeit bzw. das aktuelle Datum und die aktuelle Uhrzeit, wenn keine Parameter übergeben wurden:

```
  ` Projektmethode Time stamp
  ` Zeitstempel { ( Datum ; Uhrzeit ) } -> Lange Ganzzahl
  ` Zeitstempel { ( Datum ; Uhrzeit ) } -> Anzahl Sekunden seit 1.Jan 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
```

Hinweis: Mit dieser Methode können Sie Datum und Uhrzeit ab dem *01.01.95* um *00:00:00* bis zum *19.01.2063* um *03:14:07* codieren. Das ist der Bereich der langen Ganzzahl von *0* bis *2^31* minus Eins.

Mit den Projektmethoden ***Time stamp to date*** und ***Time stamp to time*** können Sie umgekehrt das in einem Zeitstempel gespeicherte Datum und die Uhrzeit entnehmen:

```
  ` Projektmethode Time stamp to date
  ` Zeitstempel zu Datum (Lange Ganzzahl) -> Datum
  ` Zeitstempel zu Datum (Zeitstempel) -> Entnommenes Datum

C_DATE($0)
C_LONGINT($1)

$0:=!01/01/95!+($1\86400)

  ` Projektmethode Time stamp to time
  ` Zeitstempel zu Zeit (Lange Ganzzahl) -> Datum
  ` Time stamp to time (Zeitstempel) -> Entnommene Zeit

C_TIME($0)
C_LONGINT($1)

$0:=Time(Time string(†00:00:00†+($1%86400)))
```

Um sicherzugehen, dass die Zeitstempel der Datensätze richtig aktualisiert werden, egal wie sie erstellt oder geändert wurden, setzen wir den Trigger für die Tabelle *[Documents]* ein:

```
  ` Trigger für die Tabelle [Documents]

Case of
   :(Database event=On Saving New Record Event)
      [Documents]Creation Stamp:=Time stamp
      [Documents]Modification Stamp:=Time stamp
   :(Database event=On Saving Existing Record Event)
      [Documents]Modification Stamp:=Time stamp
End case
```

Die Datenbank enthält nun alle Elemente für die Projektmethode ***CREATE DOCUMENTATION***. Mit den Befehlen **GET DOCUMENT PROPERTIES** und **SET DOCUMENT PROPERTIES** verwalten Sie Datum und Zeit der Erstellung bzw. Änderung der Dokumente.

```
  ` Projektmethode CREATE DOCUMENTATION

C_STRING(255;$vsPath;$vsDocPathName;$vsDocName)
C_LONGINT($vlDoc)
C_BOOLEAN($vbOnWindows;$vbDoIt;$vbLocked;$vbInvisible)
C_TIME($vhDocRef;$vhCreatedAt;$vhModifiedAt)
C_DATE($vdCreatedOn;$vdModifiedOn)

If(Application type=4D Client)
  ` Wenn 4D Client läuft, sichere die Dokumente
  ` lokal auf dem entsprechenden Client-Rechner
   $vsPath:=Long name to path name(Application file)
Else
  ` Sichere die Dokumente sonst dort, wo auch die Datendatei liegt
   $vsPath:=Long name to path name(Data file)
End if
  ` Sichere die Dokumente in ein Verzeichnis mit dem Namen "Dokumentation"
$vsPath:=$vsPath+"Dokumentation"+Char(Directory symbol)
  ` Gibt es dieses Verzeichnis nicht, lege es an
If(Test path name($vsPath)#Is a folder)
   CREATE FOLDER($vsPath)
End if
  ` Erstelle die Liste der bereits vorhandenen Dokumente, da wir die überflüssigen löschen müssen, d.h. die Dokumente, deren dazugehörige Datensätze gelöscht wurden.
ARRAY STRING(255;$asDocument;0)
DOCUMENT LIST($vsPath;$asDocument)
  ` wähle alle Datensätze aus der Tabelle [Documents]
ALL RECORDS([Documents])
  ` Für jeden Datensatz
$vlNbRecords:=Records in selection([Documents])
$vlNbDocs:=0
$vbOnWindows:=On Windows
For($vlDoc;1;$vlNbRecords)
  ` Wir nehmen an, dass das Dokument auf der Festplatte wieder erstellt werden muss
```

```
    $vbDoIt:=True
  ` Berechne Name und Pfadname des Dokuments
    $vsDocName:="DOC"+String([Documents]Number;"00000")
    $vsDocPathName:=$vsPath+$vsDocName
  ` Gibt es dieses Dokument bereits?
    If(Test path name($vsDocPathName+".HTM")=Is a document)
  ` Wenn ja,entferne das Dokument aus der Liste der Dokumente, die schließlich gelöscht werden
      $vlElem:=Find in array($asDocument;$vsDocName+".HTM")
      If($vlElem>0)
        DELETE FROM ARRAY($asDocument;$vlElem)
      End if
  ` Wurde das Dokument nach der letzten Änderung des Datensatzes gesichert?
      GET DOCUMENT PROPERTIES($vsDocPathName+
      ".HTM";$vbLocked;$vbInvisible;$vdCreatedOn;
      $vhCreatedAt;$vdModifiedOn;$vhModifiedAt)
      If(Time stamp($vdModifiedOn;$vhModifiedAt)>=[Documents]Modification Stamp)
  ` Wenn ja, müssen wir das Dokument nicht wieder erstellen
        $vbDoIt:=False
      End if
    Else
  ` Das Dokument gibt es nicht, setze wieder diese beiden Variablen, damit wir wissen, dass sie vor Setzen der endgültigen Eigenschaften
des Dokuments berechnet werden müssen
      $vdModifiedOn:=!00.00.00!
      $vhModifiedAt:=†00:00:00†
    End if
  ` Muss das Dokument (wieder) erstellt werden?
    If($vbDoIt)
  ` Wenn ja, erhöhe die Nummer der aktualisierten Dokumente
      $vlNbDocs:=$vlNbDocs+1
  ` Lösche das Dokument, wenn es bereits vorhanden ist
      DELETE DOCUMENT($vsDocPathName+".HTM")
  ` Und erstelle es wieder
      If($vbOnWindows)
        $vhDocRef:=Create document($vsDocPathName;"HTM")
      Else
        $vhDocRef:=Create document($vsDocPathName+".HTM")
      End if
      If(OK=1)
  ` Schreibe hier den Inhalt des Dokuments
        CLOSE DOCUMENT($vhDocRef)
        If($vdModifiedOn=!00.00.00!)
  ` Dokument war nicht vorhanden, setze Änderungsdatum und -zeit auf die richtigen Werte
          $vdModifiedOn:=Current date
          $vhModifiedAt:=Current time
        End if
  ` Ändere die Eigenschaften des Dokuments, so dass Erstellungsdatum und -zeit gleich sind zum entsprechenden Datensatz
        SET DOCUMENT PROPERTIES($vsDocPathName+".HTM";
        $vbLocked;$vbInvisible;Time stamp to date([Documents]Creation Stamp);
        Time stamp to time([Documents]Creation Stamp);$vdModifiedOn;$vhModifiedAt)
      End if
    End if

  ` Um zu wissen, wie es weitergeht
    SET WINDOW TITLE(" Dokumente bearbeiten "+String($vlDoc)+" of "+String($vlNbRecords))
    NEXT RECORD([Documents])
End for
  ` Lösche die überflüssigen Dokumente, d.h. jene, die noch im Array $asDocument sind
For($vlDoc;1;Size of array($asDocument))
    DELETE DOCUMENT($vsPath+$asDocument{$vlDoc})
    SET WINDOW TITLE("Überflüssige Dokumente löschen: "+Char(34)+$asDocument{$vlDoc}+Char(34))
End for
  ` Wir sind fertig
ALERT("Anzahl der bearbeiteten Dokumente: "+String($vlNbRecords)+Char(13)+"Anzahl der aktualisierten Dokumente:
"+String($vlNbDocs)+Char(13)+ "Anzahl der gelöschten Dokumente"+String(Size of array($asDocument)))
```

Get document size

Get document size (Dokumentname {; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Dokumentname	String, DokRef	➡	Referenznummer oder Name des Dokuments
*	Operator	➡	Nur auf Mac OS: ohne Stern: Größe des Data fork mit Stern: Größe des Resource fork
Funktionsergebnis	Zahl	⮌	Größe (in Bytes) des Dokuments

Beschreibung

Die Funktion **Get document size** gibt die Größe eines Dokuments in Bytes zurück.

Ist das Dokument geöffnet, übergeben Sie in *Dokumentname* seine Referenznummer.
Ist das Dokument geschlossen, übergeben Sie in *Dokumentname* seinen Namen bzw. Pfadnamen.

Übergeben Sie auf Macintosh nicht den optionalen Parameter *, wird die Größe des Data fork zurückgegeben, sonst die Größe des Resource fork.

Get localized document path

Get localized document path (relativerPfad) -> Funktionsergebnis

Parameter	Typ		Beschreibung
relativerPfad	Text	➡	Relativer Pfadname des Dokuments, dessen lokalisierte Version wir erhalten wollen
Funktionsergebnis	Text	⮌	Absoluter Pfadname des lokalisierten Dokuments

Beschreibung

Die Funktion **Get localized document path** gibt den kompletten (absoluten) Pfadnamen eines Dokuments zurück, definiert durch *relativerPfad* und abgelegt in einem xxx.lproj Ordner.
Diese Funktion muss in einer mehrsprachig ausgerichteten Anwendung verwendet werden, die einen Ordner **Resources** mit Unterordnern xxx.lproj enthält, wobei xxx eine Sprache darstellt. Mit dieser Architektur unterstützt 4D automatisch lokalisierte Dateien vom Typ .xliff sowie Bilder. Sie benötigen denselben Mechanismus evtl. auch für andere Dateitypen.

In *relativerPfad* übergeben Sie den relativen Pfadnamen des zu suchenden Dokuments. Der eingegebene Pfad muss relativ zur ersten Ebene des Ordners "xxx.lproj" der Anwendung sein. Die Funktion gibt einen kompletten Pfadnamen unter Verwendung des Ordners "xxx.lproj" zurück, der der aktuellen Sprache der Anwendung entspricht.

Hinweis: Die aktuelle Sprache wird entweder automatisch von 4D gemäß dem Inhalt des Ordners **Resources** gesetzt (siehe **Get database localization**) oder über den Befehl **SET DATABASE LOCALIZATION**).

Sie können den Inhalt des Parameters *RelativerPfad* über eine Syntax vom System oder POSIX darstellen. Zum Beispiel:

- xsl/log.xsl (POSIX syntax: auf Mac OS oder Windows verwendbar)
- xsllog.xsl (nur Windows)
- xsl:log.xsl (nur Mac OS)

Der von der Funktion zurückgegebene absolute Pfadname wird immer in der Syntax des Systems dargestellt.

4D Server: Im remote Modus gibt die Funktion den Pfad des Ordners **Resources** auf dem Client Rechner zurück, wenn sie von einem Client Prozess aufgerufen wird.

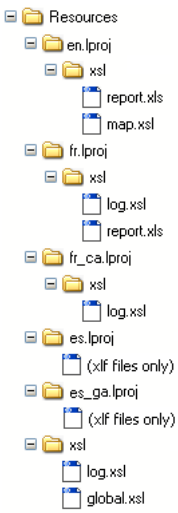
4D sucht nach der Datei in einer bestimmten Reihenfolge, die ermöglicht, alle Fälle mehrsprachiger Anwendungen abzuarbeiten. Bei jedem Schritt prüft 4D, ob der Parameter *relativerPfad* im Ordner für die Sprache vorhanden ist und gibt, wenn er gefunden wird, den kompletten Pfad zurück. Wird *relativerPfad* nicht gefunden oder existiert der Ordner nicht, geht 4D zum nächsten Schritt. Nachfolgend sind die Ordner für die verschiedenen Suchschritte aufgelistet:

Aktuelle Sprache (z.B.: de-at)
Aktuelle Sprache ohne Region (z.B.: de)
Sprache, die beim Start standardmäßig geladen wird (z.B.: es-CU)
Sprache, die beim Start ohne Region standardmäßig geladen wird (z.B.: es)
Erster gefundener Ordner .lproj (z.B.: en.lproj)
Erste Ebene des Ordners Resources

Wird *relativerPfad* an keiner dieser Stellen gefunden, gibt die Funktion einen leeren String zurück.

Beispiel

Zur Umwandlung einer XML oder HTML Datei wollen Sie eine Umwandlungsdatei "log.xsl" verwenden. Da diese Datei je nach der aktuellen Sprache unterschiedlich ist, wollen Sie wissen, welchen "log.xsl" Dateipfad Sie verwenden sollen.
Nachfolgend sehen Sie den Inhalt des Ordners **Resources**:



Um eine an die aktuelle Sprache angepasste .xsl Datei zu verwenden, übergeben Sie einfach:

```
$myxsl:=Get localized document path("xsl/log.xsl")
```

Ist die aktuelle Sprache z.B. kanadisches Französisch (fr-ca), gibt die Funktion folgendes zurück:

- unter Windows: C:\users\..... \resources\fr_ca.lproj\xsl\log.xsl

- auf Mac OS: "HardDisk:users:…:…:…:resources:fr_ca.lproj:xsl:log.xsl"

MOVE DOCUMENT

MOVE DOCUMENT (QuellPfadname ; ZielPfadname)

Parameter	Typ		Beschreibung
QuellPfadname	String	➡	Kompletter Pfadname für vorhandenes Dokument
ZielPfadname	String	➡	ZielPfadname

Beschreibung

Der Befehl **MOVE DOCUMENT** verschiebt oder benennt ein Dokument um.

In *QuellPfadName* geben Sie den kompletten Pfadnamen zum Dokument an, in *ZielPfadName* den neuen Namen und/oder die neue Position für das Dokument.

Warnung: Mit **MOVE DOCUMENT** können Sie ein Dokument innerhalb desselben Volumes verschieben. Wollen Sie dagegen ein Dokument zwischen zwei verschiedenen Volumes bewegen, kopieren Sie das Dokument mit **COPY DOCUMENT**, und löschen Sie dann das Original mit **DELETE DOCUMENT**.

Beispiel 1

Folgendes Beispiel benennt das Dokument **DocName** um:

```
MOVE DOCUMENT("C:\\FOLDER\\DocName";"C:\\FOLDER\\NewDocName")
```

Beispiel 2

Folgendes Beispiel verschiebt und benennt das Dokument **DocName** um:

```
MOVE DOCUMENT("C:\\FOLDER1\\DocName";"C:\\FOLDER2\\NewDocName")
```

Beispiel 3

Folgendes Beispiel verschiebt das Dokument **DocName**:

```
MOVE DOCUMENT("C:\\FOLDER1\\DocName";"C:\\FOLDER2\\DocName")
```

Hinweis: In den beiden letzten Beispielen muss der Zielordner *"C:\\FOLDER2"* vorhanden sein. Der Befehl **MOVE DOCUMENT** verschiebt nur das Dokument, legt jedoch keine Ordner an.

Object to path

Object to path (PfadObjekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
PfadObjekt	Objekt	➡	Objekt mit Pfadinhalt
Funktionsergebnis	Text	⮌	Pfadname

Beschreibung

Die Funktion **Object to path** gibt einen Pfadnamen (String) gemäß der Angaben im Parameter *PfadObjekt* zurück. Folgende Pfade werden unterstützt:

- Systempfad (Windows oder macOS) oder Posix Pfad. Der Pfadtyp wird durch das letzte Zeichen der Eigenschaft *parentFolder* definiert (siehe unten).
- Relativer Pfad oder absoluter Pfad (weitere Informationen dazu siehe **Absolute oder relative Pfadnamen**).

In *PfadObjekt* übergeben Sie ein Objekt, das den zu erstellenden Pfad definiert. Es muss folgende Eigenschaften enthalten:

Eigenschaft	Typ	Beschreibung
parentFolder	Text	Verzeichnisinformation für den Pfad. Das letzte Zeichen muss ein Ordnertrenner sein, da es zur Bestimmung des Dateipfads dient. Bei einem Posix Trenner ("/") wird der Pfad mit Posix Trennern erstellt; sonst mit den Systemtrennern.
name	Text	Datei- oder Ordnername des angegebenen Pfads ohne Endung.
extension	Text	Endung zum Datei- oder Ordnername. Startet mit "." (kann entfallen). Leerstring "", wenn es keine Endung gibt.
isFolder	Boolean	Wahr, wenn Name ein Ordnername ist, sonst falsch (standardmäßig falsch)

In der Regel wird *PfadObjekt* über die Funktion **Path to object** erstellt, es lässt sich aber auch auf andere Weise anlegen. Beachten Sie, dass **Object to path** nur Strings verwaltet und weder prüft, ob der Pfadtyp gültig ist oder eine Datei bzw. Ordner tatsächlich vorhanden sind.

Beispiel

Eine Datei in ihrem eigenen Ordner duplizieren und umbenennen:

```
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 (Dokumentname {; Dateityp}{; Modus}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Dokumentname	String	➡	Dokumentname oder kompletter Pfadname oder leerer String für Standarddialog für Datei
Dateityp	String	➡	Liste der anzuzeigenden Dokumenttypen oder *, um alle Typen anzuzeigen
Modus	Lange Ganzzahl	➡	Modus für Öffnen des Dokuments
Funktionsergebnis	DokRef	⬅	Referenznummer des Dokuments

Beschreibung

Die Funktion **Open document** öffnet das Dokument mit dem in *Dokumentname* übergebenen Namen bzw. Pfadnamen.
Ist *Dokumentname* ein leerer Text, wird durch **Open document** der Standarddialog zum Öffnen von Dokumenten angezeigt. Brechen Sie diesen Dialog ab, wird kein Dokument geöffnet. **Open document** gibt als DokRef Null zurück und setzt die OK Variable auf 0.

- Wird das Dokument korrekt geöffnet, gibt **Open document** die Referenznummer des Dokuments zurück, die Systemvariable OK nimmt den Wert 1 an.
- Ist das Dokument bereits im Lesemodus geöffnet und der Parameter *Modus* nicht gewählt, öffnet **Open document** das Dokument im Lese-/Schreibmodus und setzt die Systemvariable OK auf 1.
- Ist das Dokument bereits im Lese-/Schreibmodus geöffnet und Sie versuchen, es im Schreibmodus zu öffnen, erhalten Sie eine Fehlermeldung (-43). Sie können es jedoch im Nur-Lesen Modus öffnen. Die Systemvariable OK wird dann auf 1 gesetzt.
- Gibt es das Dokument nicht oder ist es bereits geöffnet, erhalten Sie eine Fehlermeldung.

Im Parameter *Dateityp* übergeben Sie den/die Dateitypen, die im Öffnen-Dialog auswählbar sind. Sie können eine Liste der Dokumenttypen übergeben, getrennt durch Strichpunkt (;). Für jeden gesetzten Typ wird im Menü zum Auswählen des Typs im Dialogfenster ein Eintrag hinzugefügt.
Auf Mac OS können Sie einen Macintosh Standardtyp (TEXT, APPL, etc.) oder einen UTI Typ(Uniform Type Identifier) übergeben. Apple hat UTIs zur Standardisierung von Dateitypen definiert. So ist z.B. "public.text" der UTI Typ für Dateien vom Typ Text. Weitere Informationen dazu finden Sie auf der Web Site *developer.apple.com* unter Uniform Type Identifier Concepts.

Unter Windows können Sie auch einen Mac OS Standardtyp übergeben - 4D sorgt intern für die Entsprechung - oder eine Dateierweiterung (.txt, .exe, etc.) Beachten Sie, dass der Benutzer unter Windows durch Eingabe von *.* die Anzeige aller Dateitypen erzwingen kann. In diesem Fall führt 4D jedoch eine zusätzliche Überprüfung der gewählten Dateitypen aus: Wählt der Benutzer einen nicht zugelassenen Dateityp, gibt die Funktion einen Fehler zurück.
Wollen Sie die angezeigten Dateien nicht auf bestimmte Typen beschränken, übergeben Sie in *Dateityp* den String * (Stern) oder ".*".

Mit dem optionalen Parameter *Modus* definieren Sie, wie *Dokumentname* geöffnet werden soll. Es gibt vier Möglichkeiten. 4D bietet unter dem Thema **Systemdokumente** folgende vordefinierten Konstanten:

Konstante	Typ	Wert
Get Pathname	Lange Ganzzahl	3
Read and Write	Lange Ganzzahl	0
Read Mode	Lange Ganzzahl	2
Write Mode	Lange Ganzzahl	1

Ist ein Dokument geöffnet, setzt **Open document** die Dateiposition an den Anfang des Dokuments. Die Funktion **Append document** setzt sie ans Ende der Datei.

Haben Sie ein Dokument geöffnet, können Sie mit den Befehlen **RECEIVE PACKET** und **SEND PACKET** im Dokument lesen und schreiben. Sie können diese auch mit **Get document position** und **SET DOCUMENT POSITION** kombinieren, um direkt auf jeden beliebigen Teil des Dokuments zuzugreifen.

Vergessen Sie nicht, bei Bedarf **CLOSE DOCUMENT** für das Dokument aufzurufen.

Beispiel 1

Folgendes Beispiel öffnet ein vorhandenes Dokument mit Namen Notiz, trägt den String “Adieu” ein und schließt das Dokument. Bereits im Dokument vorhandener Inhalt wird überschrieben:

```
C_TIME(vhDoc)
vhDoc:=Open document("Notiz.txt";Read and Write) //Öffne ein Dokument mit Namen Notiz
If(OK=1)
   SEND PACKET(vhDoc;"Adieu") //Schreibe ein Wort in das Dokument
   CLOSE DOCUMENT(vhDoc) //Schließe das Dokument
End if
```

Beispiel 2

Sie können nun ein Dokument lesen, das bereits im Schreibmodus geöffnet ist:

```
vDoc:=Open document("ÜbergebeDatei";"TEXT")  ` Datei ist geöffnet
  ` Bevor Datei geschlossen wird, kann sie im Nur Lesen Modus eingesehen werden:
vRef:=Open document("ÜbergebeDatei";"TEXT";Read Mode)
```

Systemvariablen und Mengen

Wurde das Dokument korrekt geöffnet, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null). Die Systemvariable *Document* enthält nach dem Aufruf den vollen Namen des Dokuments.
Übergeben Sie in Modus 3, gibt die Funktion ?00:00:00? zurück (keine Referenz auf Dokument). Das Dokument wird nicht geöffnet, die Systemvariablen Document und OK werden jedoch aktualisiert:

- OK ist gleich 1.
- Document enthält den kompletten Pfadnamen und den Namen von *Dokument* .

Hinweis: Übergeben Sie einen leeren String, erscheint das Dialogfenster zum Öffnen einer Datei. Wählt der Benutzer ein Dokument und klickt auf die Schaltfläche **OK**, wird *Dokument* auf den Pfad des Dokuments gesetzt, das der Benutzer gewählt hat und OK wird auf 1 gesetzt. Hat der Benutzer auf die Schaltfläche **Abbrechen** geklickt, ist OK gleich 0 (Null).

Path to object

Path to object (Pfad {; Pfadtyp}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Pfad	Text	➡	Pfadname
Pfadtyp	Lange Ganzzahl	➡	Typ der Pfadsyntax (Standard) oder Posix
Funktionsergebnis	Objekt	⬅	Objekt mit Pfadinhalt

Beschreibung

Die Funktion **Path to object** gibt ein Objekt mit den spezifischen Eigenschaften des Parameters *Pfad* zurück.

Ohne Angabe des Parameters *PfadTyp* nimmt 4D standardmäßig an, dass Sie einen System *Pfad* mit den entsprechenden Separatoren übergeben haben ("\" unter Windows, ":" auf macOS). Übergeben Sie einen Posix *Pfad* mit Posix Separatoren ("/") oder wollen Sie den Pfadtyp angeben, übergeben Sie eine der beiden Konstanten im Parameter *PfadTyp*:

Konstante	Typ	Wert	Kommentar
Path is POSIX	Lange Ganzzahl	1	Der Pfad wird in Posix Syntax angegeben.
Path is system	Lange Ganzzahl	0	(Standard) Der Pfad wird in der Syntax des aktuellen Systems angegeben (Windows oder macOS).

Die Funktion gibt ein Objekt zurück, das sich nach der Analyse von *Pfad* ergibt. Es enthält folgende Eigenschaften:

Eigenschaft	Typ	Beschreibung
parentFolder	Text	Verzeichnisinformation für den Pfad. Das letzte Zeichen ist immer ein Ordnertrenner.
name	Text	Datei- oder Ordnername des angegebenen Pfads, ohne Endung.
extension	Text	Endung des Datei- oder Ordnernamens. Startet immer mit "." und ist Leerstring "", wenn es keine Endung gibt.
isFolder	Boolean	Wahr, wenn Name ein Ordnername ist, sonst falsch (standardmäßig falsch)

Ist das letzte Zeichen von *Pfad* ein Trenner gemäß dem Pfadtyp (zum Beispiel "\" unter Windows), nimmt 4D an, dass ein Ordnerpfad übergeben wurde. Sonst nimmt 4D an, dass es ein Dateiname ist.
Die Endung (wenn übergeben) wird zurückgegeben, unabhängig ob der Pfad eine Datei oder einen Ordner angibt. In jedem Fall müssen Sie Name und Endung zusammenfügen, um den kompletten Namen zu erhalten.

Beachten Sie, dass **Path to object** nur Strings verwaltet und weder prüft, ob der Pfadtyp gültig ist oder eine Datei bzw. Ordner tatsächlich vorhanden sind.

Beispiel 1

Nachfolgende Beispiele zeigen verschiedene Ergebnisse mit Dateipfaden:

```
C_OBJECT($o)
$o:=Path to object("C:\\first\\second\\fileZ") //unter Windows
 //$o.parentFolder="C:\\first\\second\\"
 //$o.name="fileZ"
 //$o.extension=""
 //$o.isFolder=false
```

```
C_OBJECT($o)
$o:=Path to object("osx:Users:john:Documents:Comments.text") //auf macOS
 //$o.parentFolder="osx:Users:john:Documents:"
 //$o.name="Comments"
 //$o.extension=".text"
 //$o.isFolder=false
```

```
C_OBJECT($o)
$o:=Path to object("\\images\\jan\\pict1.png";Path is system) //unter Windows
 //$o.parentFolder="\\images\\jan\\"
 //$o.name="pict1"
 //$o.extension=".png"
 //$o.isFolder=false
```

Pfad zu einem Ordner definieren:

```
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
```

Ist der Pfad nicht in einem Root-Verzeichnis, ist *parentFolder* leer:

```
C_OBJECT($o)
$o:=Path to object("C:\\")  //unter Windows
 //$o.parentFolder=""
 //$o.name="c:"
 //$o.extension=""
 //$o.isFolder=true
```

```
C_OBJECT($o)
$o:=Path to object("osx:")  //auf macOS
 //$o.parentFolder=""
 //$o.name="osx"
 //$o.extension=""
 //$o.isFolder=true
```

Ist der letzte Teil des Pfads ein Text *(hier .invisible*), wird er als Dateiname gewertet:

```
C_OBJECT($o)
$o:=Path to object("/folder/.invisible";Path is POSIX)
 //$o.parentFolder="/folder/"
 //$o.name=".invisible"
 //$o.extension=""
 //$o.isFolder=false
```

Beispiel 2

Sie können diese Funktion mit **Object to path** kombinieren, um eine Datei in einem Pfad umzubenennen:

```
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"
```

Beispiel 3

Sie wollen die Anzahl Unterordner innerhalb eines Pfads wissen:

```
C_OBJECT($o)
C_TEXT($path)
C_LONGINT($vCount)
```

```
$path:=Select folder  //den Benutzer einen Ordner auswählen lassen
$o:=Path to object($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 (AliasPfad ; OriginalPfad)

Parameter	Typ		Beschreibung
AliasPfad	String	⇒	Name oder Pfad für Verknüpfung bzw. Alias
OriginalPfad	String	⇐	Name oder Pfad für Originalverknüpfung bzw. -alias

Beschreibung

Der Befehl **RESOLVE ALIAS** gibt den vollen Pfadnamen zur Originaldatei bzw. zum Originalordner der Verknüpfung/des Alias zurück.

Im Parameter *AliasPfad* wird der vollständige Pfad zur Verknüpfung/zum Alias übergeben.
Sobald der Befehl ausgeführt ist, enthält die Variable *OriginalPfad* den vollständigen Pfad zur Ausgangsdatei bzw. zum Ausgangsordner der Verknüpfung/des Alias und die Variable OK wird auf 1 gesetzt.

Systemvariablen und Mengen

Gibt *AliasPfad* ein Alias bzw. eine Verknüpfung an, wird die Systemvariable OK auf 1 gesetzt. Gibt *AliasPfad* eine Standarddatei an, wird die Systemvariable OK auf 0 gesetzt.

Select document

Select document (Verzeichnis ; Dateitypen ; Titel ; Optionen {; Gewählt}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Verzeichnis	Text, Lange Ganzzahl	➡	Zugriffspfad für Verzeichnis, das standardmäßig im Dialogfenster für Dokumentauswahl angezeigt werden soll oder leerer String für Anzeige des Standard Benutzerordners (“My documents” unter Windows, “Documents” auf Mac OS) oder Nummer des gespeicherten Zugriffspfads
Dateitypen	Text	➡	Liste der zu filternden Dokumenttypen oder "*", um Dokumente nicht zu filtern
Titel	Text	➡	Titel des Dialogfensters für Dokumentauswahl
Optionen	Lange Ganzzahl	➡	Auswahloption(en)
Gewählt	Array Text	⬅	Array mit der Liste der Zugriffspfade + Namen der gewählten Dateien
Funktionsergebnis	String	↩	Name der gewählten Datei (bei mehrfacher Auswahl erste Datei in der Liste)

Beschreibung

Die Funktion **Select document** zeigt einen Standard Öffnen-Dialog, in dem der Benutzer eine oder mehrere Dateien setzen kann und Name bzw. vollen Zugriffspfad der gewählten Datei(en) zurückgeben kann.

Der Parameter *Verzeichnis* gibt den Ordner an, dessen Inhalt zu Beginn im Öffnen-Dialog angezeigt wird. Sie können drei Arten übergeben:

- Einen Text mit dem kompletten Zugriffspfad des anzuzeigenden Ordners.
- Einen leeren String (“”), um den Standard Benutzerordner für das aktuelle Betriebssystem anzuzeigen (“My documents” unter Windows, “Documents” auf Mac OS).
- Die Nummer des gespeicherten Zugriffspfads (von 1 bis 32000) zum Anzeigen der zugeordneten Felder.
  So können Sie den Zugriffspfad des geöffneten Ordners im Speicher sichern, wenn der Benutzer auf die Schaltfläche **Auswahl** klickt, mit anderen Worten, den vom Benutzer gewählten Ordner. Beim ersten Aufrufen einer wilkürlichen Nummer, z.B. 5, zeigt die Funktion den Standard Benutzerordner des Betriebssystems (entspricht der Übergabe eines leeren Strings). Der Benutzer kann auch die Ordner auf der Festplatte durchlaufen. Klickt er auf die Schaltfläche **Benutzer**, wird der Zugriffspfad gespeichert und Nummer 5 zugewiesen. Wird diese Nummer später aufgerufen, wird der gespeicherte Zugriffspfad standardmäßig verwendet. Wird ein neuer Speicherort gewählt, wird die Pfadnummer aktualisiert.
  Mit dieser Arbeitsweise können Sie bis zu 32.000 Zugriffspfade speichern. Jeder Pfad wird für die gesamte Sitzung beibehalten.

Hinweis: Die Funktionsweise ist dieselbe wie für die Funktion **Select folder**. Die Anzahl der gespeicherten Pfadnamen wird von beiden Funktionen gemeinsam genutzt.

Im Parameter *Dateitypen* übergeben Sie den/die Datentypen, die im Öffnen-Dialog auswählbar sind. Sie können mehrere Typen übergeben, getrennt durch ; (Strichpunkt). Für jeden definierten Typ wird im Auswahlmenü des Dialogfensters eine Zeile hinzugefügt.

- Auf Mac OS können Sie entweder ein Mac OS Standardformat (TEXT, APPL, etc.) oder ein UTI Format (Uniform Type Identifier) übergeben. UTI-Typen wurden von Apple definiert, um den Anforderungen zur Standarisierung von Dateitypen gerecht zu werden. "public.text" ist z.B. der UTI-Typ für Dateien vom Typ Text. Weitere Informationen dazu finden Sie auf der Web Site *developer.apple.com* unter Uniform Type Identifier Concepts.
- Unter Windows können Sie auch einen standardmäßigen Mac OS Dateityp oder die Dateiendung (.txt, .exe, etc.) übergeben - 4D führt die Konvertierung intern aus. Beachten Sie, dass der Benutzer unter Windows die Anzeige aller Dokumenttypen durch Eingabe der Symbole *.* erzwingen kann. In diesem Fall führt 4D eine zusätzliche Überprüfung der ausgewählten Dateitypen durch: Wählt der Benutzer einen nicht zugelassenen Typ, gibt die Funktion einen Fehler zurück.

Wollen Sie die angezeigten Dateien nicht auf bestimmte Typen beschränken, übergeben Sie in *Dateitypen* den String * (Stern) oder ".*".

Im Parameter *Titel* übergeben Sie die Bezeichnung für das Dialogfenster. Standardmäßig, also wenn Sie einen leeren String übergeben, erscheint die Bezeichnung “Öffnen”.

Im Parameter *Optionen* geben Sie erweitere Funktionen für den Öffnen-Dialog an. 4D bietet dafür unter dem Thema **Systemdokumente** folgende vordefinierten Konstanten:

Konstante	Typ	Wert	Kommentar
Alias selection	Lange Ganzzahl	8	Ermöglicht unter Windows eine Verknüpfung, auf Mac OS ein Alias als Dokument aufzurufen. Wird diese Konstante beim Auswählen einer Verknüpfung bzw. eines Alias nicht verwendet, gibt die Funktion den Pfad des Zielelements zurück (Standardeinstellung). Übergeben Sie die Konstante, gibt die Funktion den Pfad der Verknüpfung bzw. des Alias direkt zurück.
File name entry	Lange Ganzzahl	32	Erlaubt dem Benutzer, einen Dateinamen im Dialogfenster "Speichern unter" einzugeben. Es wird keine Datei gesichert, der Entwickler muss als Reaktion auf diese Aktion selbst eine Datei erstellen (die Systemvariable *Document* wird aktualisiert). In diesem Kontext kann der Parameter *Verzeichnis* den Pfad zu einer Datei anstatt zu einem Verzeichnis enthalten. Der Dateiname wird im Textbereich "Speichern unter" als Name vorgeschlagen. Das dazugehörige Verzeichnis wird als Standardpfad verwendet.
Multiple files	Lange Ganzzahl	1	Ermöglicht, mehrere Dateien auszuwählen; für eine zusammenhängende Auswahl über die Kombination **Umschalttaste + Klick**, für eine unterbrochene Auswahl unter Windows über die **Strg-Taste + Klick**, auf Mac OS die **Befehlstaste + Klick**. In diesem Fall enthält der übergebene Parameter die Liste mit allen gewählten Dateien. Wird diese Konstante nicht verwendet, ermöglicht die Funktion standardmäßig nicht, mehrerer Dateien auszuwählen.
Package open	Lange Ganzzahl	2	(nur Mac OS) ermöglicht, „Packages" als Ordner zu öffnen und so ihren Inhalt anzusehen bzw. auszuwählen. Wird diese Konstante nicht verwendet, ermöglicht die Funktion standardmäßig kein Öffnen von Packages.
Package selection	Lange Ganzzahl	4	(nur Mac OS) ermöglicht, „Packages" als Einheit auszuwählen. Wird diese Konstante nicht verwendet, ermöglicht die Funktion standardmäßig keine Auswahl von Software-Paketen.
Use sheet window	Lange Ganzzahl	16	(nur Mac OS): Zeigt das Auswahlfenster in Form eines Sheet Fensters. Diese Option wird unter Windows ignoriert. Sheet Fenster sind spezifisch für die Mac OS X Oberfläche, da diese grafische Animation zulässt. Weitere Informationen dazu finden Sie im Abschnitt **Fenstertypen (Kompatibilität)**. Wird diese Konstante nicht verwendet, zeigt die Funktion das Standard Dialogfenster.

Wollen Sie keine der Optionen verwenden, übergeben Sie 0 (Null) im Parameter *Optionen*.

Über den optionalen Parameter *Gewählt* erhalten Sie den kompletten Zugriffspfad (Zugriffspfad + Name) jeder vom Benutzer gewählten Datei. Die Funktion erstellt und füllt das Array gemäß der Auswahl des Benutzers. Dieser Parameter ist hilfreich in Verbindung mit der Option *Multiple files* oder wenn Sie den Zugriffspfad der gewählten Datei herausfinden wollen. Nehmen Sie einfach den Dateinamen, den die Funktion als Wert des Array zurückgibt.
Wurde keine Datei gewählt, wird das Array leer zurückgegeben.

Hinweis: Auf Mac OS wird ein ausgewähltes Package als Ordner gewertet. Der im Array *Gewählt* zurückgegebene Pfadname enthält am Ende immer das Zeichen ":",
z.B. **Festplatte:Applications:4D:4D v11.4:US:4D Volume Desktop.app:**

Select document gibt unter Windows den Namen + Endung, auf Mac OS nur den Namen der gewählten Datei zurück. Wurden mehrere Dateien gewählt, gibt die Funktion den Namen der ersten Datei in der Liste der gewählten Dateien zurück. Diese Liste erhalten Sie über den optionalen Parameter *Gewählt*. Wurde keine Datei gewählt, wird ein leerer String zurückgegeben.

Beispiel 1

Dieses Beispiel dient zum Spezifizieren einer 4D Datendatei:

```
C_LONGINT($platform)
PLATFORM PROPERTIES($platform)
If($platform=Windows)
   $DocType:=".4DD"
Else
   $DocType:="com.4d.4d.data-file"  `UTI Typ
End if
$Options:=Alias selection+Package open+Use sheet window
$Doc:=Select document("";$DocType;"Wähle die Datendatei";$Options)
```

Beispiel 2

Erstellen eines eigenen Dokuments durch den Benutzer:

```
$doc:=Select document(System folder(Documents folder)+"Report.pdf";"pdf";"Report name:";File name entry)
If(OK=1)
   BLOB TO DOCUMENT(Document;$blob)  // $blob enthält Dokument zum Speichern
End if
```

Systemvariablen und Mengen

Läuft die Funktion korrekt und wurde ein gültiges Dokument gewählt, wird die Systemvariable OK auf 1 gesetzt, die Systemvariable *Document* enthält den kompletten Zugriffspfad der gewählten Datei. Wurde keine Datei gewählt, z.B. weil der Benutzer auf die Schaltfläche **Abbrechen** geklickt hat, wird die Systemvariable OK auf 0 (Null) gesetzt, die Systemvariable *Document* ist leer.

Select folder

Select folder ({Meldung }{;}{ Standardpfad {; Optionen}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Meldung	String	➡	Fenstertitel
Standardpfad	String, Lange Ganzzahl	➡	Standard Pfadname oder Leerer String zum Anzeigen des Standard Benutzerordners (“Meine Documente” unter Windows, “Dokumente” auf Mac OS) oder Nummer des gespeicherten Pfadnamens
Optionen	Lange Ganzzahl	➡	Auswahloption(en) unter Mac OS
Funktionsergebnis	String	⬅	Zugriffspfad auf ausgewählten Ordner

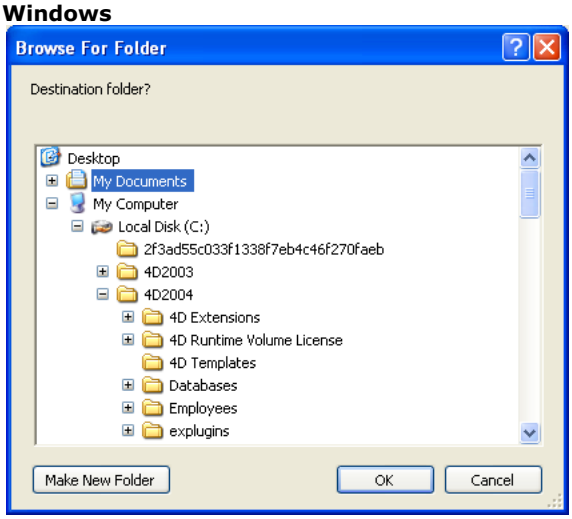
Beschreibung

Die Funktion **Select folder** zeigt ein Dialogfenster, in dem Sie einen Ordner manuell auswählen und dann den kompletten Zugriffspfad dafür wiederfinden können. Mit dem optionalen Parameter *Standardpfad* können Sie die Position eines Ordners bestimmen, der zu Beginn im Auswahlfenster für Ordner angezeigt wird.

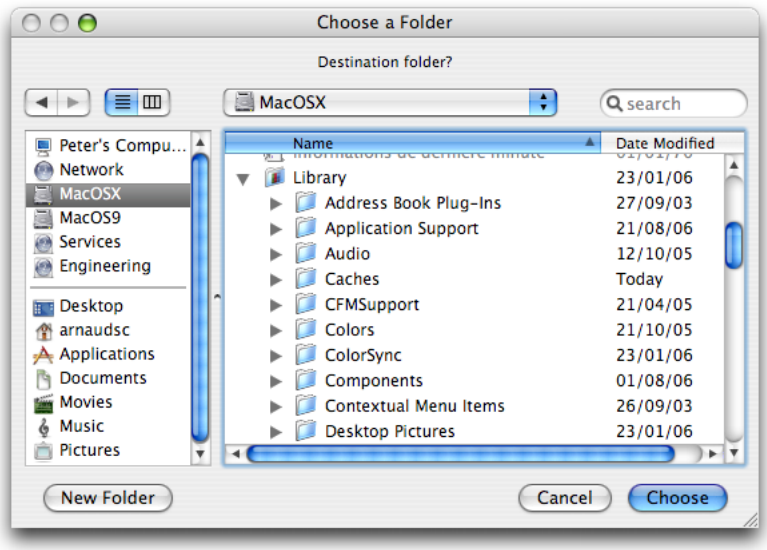
Hinweis: Select folder ändert nicht den aktuellen Ordner von 4D.

Select folder zeigt das Standarddialogfenster zum Navigieren in den Volumes und Ordnern der Workstations.
Mit dem optionalen Parameter *Meldung* zeigen Sie im Dialogfenster eine Meldung an:

Windows



Mac OS



Mit *Standardpfad* können Sie die Position des Standardordners im Auswahldialog für Ordner liefern. Sie können in diesem Parameter drei Arten von Werten übergeben:

- Den Pfadnamen eines gültigen Ordners in der Schreibweise der aktuellen Plattform.
- Einen leeren String (“”), um den Standard Benutzerordner des Systems anzuzeigen (“Meine Dokumente” unter Windows, “Dokumente” auf Mac OS).
- Die Nummer eines gespeicherten Pfadnamens (von 1 bis 32.000), um den zugewiesenen Ordner anzuzeigen. Damit können Sie den Pfadnamen des Ordners im Speicher ablegen, der sich öffnet, wenn der Benutzer auf die Schaltfläche **Auswählen** klickt; das ist der vom Benutzer geöffnete Ordner. Bei Aufruf einer will-kürlichen Nummer, z.B. 5, zeigt die Funktion den Standard Benutzerordner des Systems (entspricht Übergabe eines leeren Strings). Der Benutzer kann dann die Ordner auf der Festplatte durchlaufen. Klickt er auf die Schaltfläche **Auswählen**, wird der Pfadname gespeichert und der Nummer 5 zugeordnet. Der gespeicherte Pfad wird dann standardmäßig verwendet. Bei Auswahl einer neuen Position wird die

Pfadnummer 5 aktualisiert, usw.
Mit dieser Vorgehensweise können Sie bis zu 32.000 Pfadnamen speichern. Unter Windows wird jeder Pfad nur während der Arbeitssitzung beibehalten. Auf Mac OS bleiben die Pfade von einer zur nächsten Sitzung erhalten. Ist der Pfadname inkorrekt, wird der Parameter *Standardpfad* ignoriert.

Hinweis: Diese Arbeitsweise ist identisch mit der von **Select document**. Die Nummern der gespeicherten Pfadnamen werden von beiden Funktionen gemeinsam genutzt.

Über den Parameter *Optionen* können Sie zusätzliche Funktionalitäten auf Mac OS nutzen. Sie können eine der folgenden Konstanten unter dem Thema **Systemdokumente** übergeben:

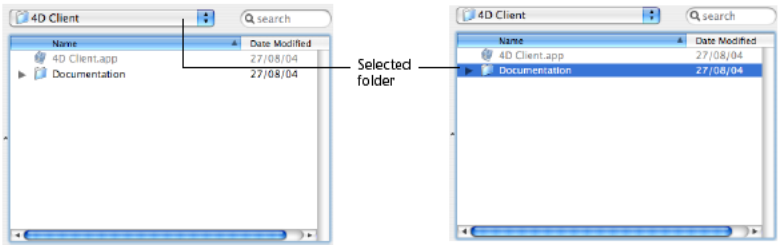
Konstante	Typ	Wert	Kommentar
Package open	Lange Ganzzahl	2	(nur Mac OS) ermöglicht, „Packages" als Ordner zu öffnen und so ihren Inhalt anzusehen bzw. auszuwählen. Wird diese Konstante nicht verwendet, ermöglicht die Funktion standardmäßig kein Öffnen von Packages.
Use sheet window	Lange Ganzzahl	16	(nur Mac OS): Zeigt das Auswahlfenster in Form eines Sheet Fensters. Diese Option wird unter Windows ignoriert. Sheet Fenster sind spezifisch für die Mac OS X Oberfläche, da diese grafische Animation zulässt. Weitere Informationen dazu finden Sie im Abschnitt **DISPLAY SELECTION**. Wird diese Konstante nicht verwendet, zeigt die Funktion das Standard Dialogfenster.

Sie können eine der Konstanten übergeben oder beide miteinander kombinieren. Diese Optionen werden nur auf Mac OS berücksichtigt. Unter Windows wird der Parameter *Optionen* ignoriert.

Der Benutzer wählt einen Ordner und klickt dann unter Windows auf die Schaltfläche **OK**, auf Macintosh auf die Schaltfläche **Auswählen**. Die Funktion gibt den Zugriffspfad für diesen Ordner zurück.

- Unter Windows erscheint er im Format: "C:\Ordner1\Ordner2\AusgewählterOrdner\"
- Auf Macintosh erscheint er im Format: "Festplatte:Ordner1:Ordner2:AusgewählterOrdner:"

Hinweis: Auf Macintosh können Sie, je nachdem, ob der Ordnername im Dialogfenster ausgewählt ist oder nicht, u.U. einen anderen Pfadnamen erhalten.



4D Server: Mit dieser Funktion sehen Sie die Volumes, die mit der 4D Arbeitsstation verbunden sind. Sie können **Select folder** nicht in einer Serverprozedur aufrufen.

Bestätigt der Benutzer das Dialogfenster, hat die Systemvariable OK den Wert 1. Klickt der Benutzer auf die Schaltfläche **Abbrechen**, hat die Systemvariable **OK** den Wert 0 (Null), die Funktion gibt einen leeren String zurück.

Hinweis: Wählt der Benutzer unter Windows inkorrekte Elemente, wie "Workstation", "Papierkorb", etc., hat die Systemvariable **OK** den Wert 0 (Null), selbst wenn der Benutzer das Dialogfenster bestätigt.

Beispiel

In folgendem Beispiel wählen Sie den Ordner aus, in dem die Bilder in der Bildbibliothek gespeichert werden:

```
$PictFolder:=Select folder("Wähle Zielordner für Ihre Bilder.")
PICTURE LIBRARY LIST(pictRefs;pictNames)
For($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 (DokRef ; Offset {; Ziffer})

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenznummer des Dokuments
Offset	Zahl	➡	Position der Datei (in Bytes)
Ziffer	Lange Ganzzahl	➡	1 = Bezogen auf Dateibeginn, 2 = Bezogen auf Dateiende, 3 = Bezogen auf aktuelle Position

Beschreibung

Der Befehl **SET DOCUMENT POSITION** arbeitet nur in einem geöffneten Dokument mit der in *DokRef* übergebenen Referenznummer.

SET DOCUMENT POSITION setzt die in *Offset* übergebene Position, wo das nächste Lesen (**RECEIVE PACKET**) oder Schreiben (**SEND PACKET**) auftritt.

Geben Sie den optionalen Parameter *Ziffer* nicht an, wird die Position bezogen auf den Dateibeginn ausgerichtet, sonst je nach der oben angegebenen Ziffer.

Sie können in *Offset* je nach *Ziffer* positive oder negative Werte übergeben.

SET DOCUMENT PROPERTIES

SET DOCUMENT PROPERTIES (Dokumentname ; Gesperrt ; Ausgeblendet ; Erstellt am ; Erstellt um ; Geändert am ; Geändert um)

Parameter	Typ		Beschreibung
Dokumentname	String	➡	Name des Dokuments oder kompletter Pfadname
Gesperrt	Boolean	➡	Gesperrt (True) oder Freigegeben (False)
Ausgeblendet	Boolean	➡	Ausgeblendet (True) oder Eingeblendet (False)
Erstellt am	Datum	➡	Datum der Erstellung
Erstellt um	Zeit	➡	Zeit der Erstellung
Geändert am	Datum	➡	Datum der letzten Änderung
Geändert um	Zeit	➡	Zeit der letzten Änderung

Beschreibung

Der Befehl **SET DOCUMENT PROPERTIES** ändert die Information über das Dokument mit dem in *Dokumentname* übergebenen Namen oder Pfadnamen.

Übergeben Sie vor dem Aufruf:

- Den Wert True in *Gesperrt*, um das Dokument zu sperren. Ein gesperrtes Dokument kann weder geöffnet noch gelöscht werden. Übergeben Sie den Wert False in *Gesperrt*, um das Dokument freizugeben.
- Den Wert True in *Ausgeblendet*, um das Dokument auszublenden. Übergeben Sie den Wert False in *Ausgeblendet*, um das Dokument in den Fenstern auf der Schreibtischoberfläche einzublenden.
- In *Erstellt am* und *Erstellt um* Erstellungsdatum und -uhrzeit des Dokuments.
- In *Geändert am* und *Geändert um* Änderungsdatum und -uhrzeit des Dokuments.

Der Dateimanager Ihres Systems verwaltet bei jeder Erstellung bzw. jedem Zugriff Datum und Uhrzeit der Erstellung sowie der letzten Änderung. Mit diesem Befehl können Sie diese Eigenschaften für besondere Zwecke ändern. Siehe Beispiel zum Befehl **GET DOCUMENT PROPERTIES**.

SET DOCUMENT SIZE

SET DOCUMENT SIZE (DokRef ; Größe)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenznummer des Dokuments
Größe	Zahl	➡	Neue Größe in Bytes

Beschreibung

Der Befehl **SET DOCUMENT SIZE** setzt die Größe eines Dokuments auf die in *Größe* übergebene Anzahl in Bytes.
Ist das Dokument geöffnet, übergeben Sie in *DokRef* seine Referenznummer.
Auf Macintosh wird die Größe des Data fork des Dokuments geändert.

SHOW ON DISK

SHOW ON DISK (Pfadname {; *})

Parameter	Typ		Beschreibung
Pfadname	String	➡	Pfadname auf anzuzeigenden Eintrag
*	Operator	➡	Ist Eintrag ein Ordner, zeige seinen Inhalt

Beschreibung

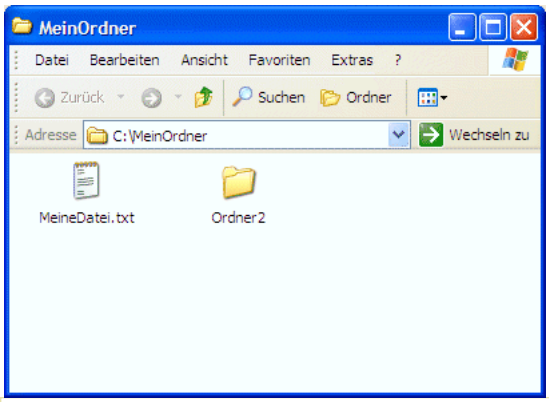
Der Befehl **SHOW ON DISK** zeigt die Datei oder den Ordner an, dessen Pfadname im Parameter *Pfadname* in einem Standardfenster des Betriebssystems übergeben wurde.

In einer Benutzeroberfläche können Sie über diesen Befehl den Ort einer spezifischen Datei bzw. eines Ordners angeben.
Gibt *Pfadname* einen Ordner an, zeigt der Befehl standardmäßig die Ebene des Ordners an. Mit dem optionalen Parameter * öffnet der Befehl den Ordner und zeigt seinen Inhalt im Fenster an.
Gibt *Pfadname* eine Datei an, wird der Parameter * ignoriert.

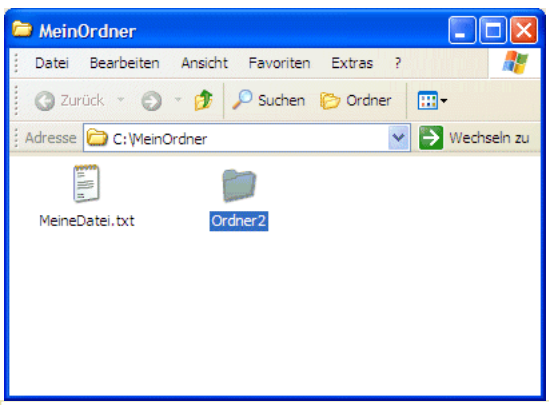
Beispiel

Folgende Beispiele zeigen die Arbeitsweise dieses Befehls:

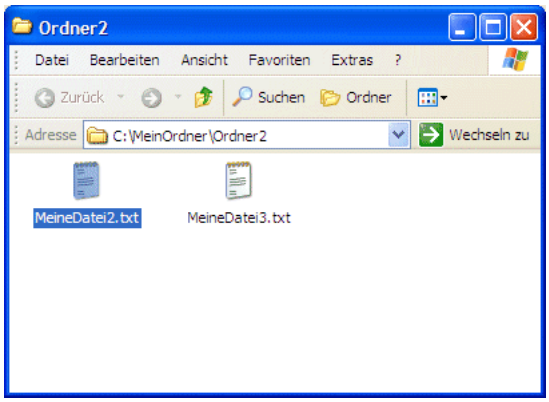
```
SHOW ON DISK("c:\\MeinOrdner\\MeineDatei.txt") ` Zeigt die angegebene Datei
```




```
SHOW ON DISK("c:\\MeinOrdner\\Ordner2") ` Zeigt den angegebenen Ordner
```




```
SHOW ON DISK("c:\\MeinOrdner\\Ordner2";*) ` Zeigt den Inhalt des angegebenen Ordners
```



Systemvariablen und Mengen

Die Systemvariable OK wird auf 1 gesetzt, wenn der Befehl korrekt ausgeführt wird.

Test path name

Test path name (Pfadname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Pfadname	String	➡	Pfadname zu Verzeichnis, Ordner oder Dokument
Funktionsergebnis	Lange Ganzzahl	⮌	1: Pfadname bezieht sich auf vorhandenes Dokument 0: Pfadname bezieht sich auf vorhandenes Verzeichnis oder Ordner <0, ungültiger Pfadname, OS Systemfehler

Beschreibung

Die Funktion **Test path name** prüft, ob ein Dokument oder Ordner mit dem Namen *Pfadname* auf der Festplatte vorhanden ist. Sie können entweder einen relativen oder einen absoluten Pfad in der Syntax des jeweiligen Systems übergeben.
Wurde ein Dokument gefunden, gibt **Test path name** den Wert 1 zurück. Wurde ein Ordner gefunden, gibt die Funktion den Wert 0 zurück.

4D bietet folgende vordefinierten Konstanten:

Konstante	Typ	Wert
Is a document	Lange Ganzzahl	1
Is a folder	Lange Ganzzahl	0

Wird weder ein Dokument noch ein Ordner gefunden, gibt **Test path name** einen negativen Wert zurück, z.B. -43 für "Datei wurde nicht gefunden".

Beispiel

Der Code prüft, ob das Dokument "Journal" im Datenbankordner vorhanden ist und erstellt es, falls keines gefunden wurde:

```
If(Test path name("Journal")#Is a document)
   $vhDocRef:=Create document("Journal")
   If(OK=1)
      CLOSE DOCUMENT($vhDocRef)
   End if
End if
```

TEXT TO DOCUMENT

TEXT TO DOCUMENT (Dateiname ; Text {; Zeichensatz {; Umbruchmodus}})

Parameter	Typ		Beschreibung
Dateiname	String	➡	Dokumentname oder Pfadname zum Dokument
Text	Text	➡	Text zum Speichern im Dokument
Zeichensatz	Text, Lange Ganzzahl	➡	Name oder Nummer des Zeichensatzes
Umbruchmodus	Lange Ganzzahl	➡	Bearbeitungsmodus für Zeilenumbrüche

Beschreibung

Der Befehl **TEXT TO DOCUMENT** schreibt den *Text* direkt in eine Datei auf der Festplatte.

In *Dateiname* übergeben Sie Name oder Pfadname der zu schreibenden Datei. Sie wird angelegt, wenn sie noch nicht existiert. Ist sie bereits auf der Festplatte vorhanden, wird der bisherige Inhalt entfernt. Bei noch geöffneter Datei wird der Inhalt gesperrt und ein Fehler generiert. In *DateiName* können Sie folgendes übergeben:

- Nur den Dateinamen, z.B. "meineDatei.txt": In diesem Fall wird die Datei neben der Strukturdatei der Anwendung gesetzt.
- Einen Pfadnamen in Bezug auf die Strukturdatei der Anwendung, z.B. "\\docs\\meineDatei.txt" unter Windows oder ":docs:meineDatei.txt" auf OS X.
- Einen absoluten Pfadnamen, z.B. "c:\\app\\docs\\meineDatei.txt" unter Windows oder "MacHD:docs:meineDatei.txt" auf OS X.

Wollen Sie, dass der Benutzer Name oder Speicherort des Dokuments angeben kann, verwenden Sie die Funktionen **Open document** oder **Create document** sowie die Systemvariable **Document**.

Hinweis: Von diesem Befehl erzeugte Dokumente haben standardmäßig keine Endung. Dazu müssen Sie eine Endung in *Dateiname* übergeben oder den Befehl **_o_SET DOCUMENT TYPE** verwenden.

In *Text* übergeben Sie den Text zum Schreiben auf die Festplatte. Das kann eine Konstante ("mein Text") oder ein 4D Textfeld bzw. Variable sein.

In *Zeichensatz* übergeben Sie den Zeichensatz zum Schreiben des Dokuments. Sie können einen String mit dem standardmäßigen Namen übergeben, z.B. "ISO-8859-1" oder "UTF-8" oder seine MIBEnum ID (Lange Ganzzahl). Weitere Informationen zu den von 4D unterstützten Zeichensätzen finden Sie in der Beschreibung zum Befehl **CONVERT FROM TEXT**. Gibt es ein Byte Order Mark (BOM)für den Zeichensatz, fügt 4D diesen in das Dokument ein. Geben Sie keinen Zeichensatz an, verwendet 4D standardmäßig den Zeichensatz "UTF_8" und ein BOM.

In *Umbruchmodus* önnen Sie eine Lange Ganzzahl übergeben, um die Handhabung der Zeichen für Zeilenende vor dem Sichern in eine Datei anzugeben. Sie können eine der nachfolgenden Konstanten unter dem Thema "**Systemdokumente**" übergeben:

Konstante	Typ	Wert	Kommentar
Document unchanged	Lange Ganzzahl	0	Keine Bearbeitung
Document with CR	Lange Ganzzahl	3	Zeilenumbrüche werden in das Mac OS Format konvertiert: CR (carriage return)
Document with CRLF	Lange Ganzzahl	2	Zeilenumbrüche werden in das Windows Format konvertiert: CRLF (carriage return + line feed)
Document with LF	Lange Ganzzahl	4	Zeilenumbrüche werden in das Unix Format konvertiert: LF (line feed)
Document with native format	Lange Ganzzahl	1	(Standard) Zeilenumbrüche werden in das native Format des Betriebssystems konvertiert: CR (carriage return auf Mac OS), CRLF (carriage return + line feed unter Windows)

Lassen Sie den Parameter *Umbruchmodus* weg, werden Zeilenumbrüche im native Modus (1) gehandhabt.

Hinweis: Dieser Befehl verändert nicht die Variable OK. Schlägt die Operation fehl, wird ein Fehler generiert, den Sie mit einer Methode abfangen können, die der Befehl **ON ERR CALL** installiert.

Beispiel 1

Hier sehen Sie typische Beispiele zu diesem Befehl:

```
TEXT TO DOCUMENT("myTest.txt";"Dies ist ein Test")
TEXT TO DOCUMENT("myTest.xml";"Dies ist ein Test")
```

Beispiel 2

Hier kann der Benutzer den Speicherort für die zu erstellende Datei angeben:

```
$MyTextVar:="Dies ist ein Test"
ON ERR CALL("IO ERROR HANDLER")
$vhDocRef :=Create document("")
 // Dokument mit der Endung ".txt" speichern
 // In diesem Fall wird an den Namen immer die Endung .txt angefügt; das lässt sich nicht verändern
If(OK=1) // Wurde das Dokument erfolgreich erstellt
```

```
    CLOSE DOCUMENT($vhDocRef)  // Dokument schließen
    TEXT TO DOCUMENT(Document;$MyTextVar )
  // Wir schreiben das Dokument
Else
  // Fehlerverwaltung
End if
```

VOLUME ATTRIBUTES

VOLUME ATTRIBUTES (Volume ; Größe ; belegt ; frei)

Parameter	Typ		Beschreibung
Volume	String	➡	Volume-Name
Größe	Zahl	⬅	Volume-Größe in Bytes
belegt	Zahl	⬅	Belegter Platz in Bytes
frei	Zahl	⬅	Freier Platz in Bytes

Beschreibung

Der Befehl **VOLUME ATTRIBUTES** gibt die Größe, den belegten sowie den freien Platz in Bytes für das Volume mit dem in *Volume* übergebenen Namen zurück.

Hinweis: Gibt *Volume* ein nicht-hochgefahrenes Remote Volume an, wird die Variable OK auf 0 (Null) gesetzt, die drei Parameter geben dann -1 zurück.

Beispiel

Ihre Anwendung enthält einige Stapeloperationen, die über Nacht oder am Wochenende laufen, da sie umfangreiche temporäre Dateien auf der Festplatte speichern. Damit dieser Vorgang möglichst flexibel und automatisch läuft, schreiben Sie eine Methode, die automatisch das erste Volume mit genügend freiem Speicherplatz für Ihre temporären Dateien findet:

```
  ` Projektmethode Find volume for space
  ` Find volume for space ( Long ) -> String
  ` Find volume for space ( benötigter Platz in Bytes ) -> Volume-Name oder leerer String

C_STRING(31;$0)
C_STRING(255;$vsDocName)
C_LONGINT($1;$vlNbVolumes;$vlVolume)
C_REAL($vlSize;$vlUsed;$vlFree)
C_TIME($vhDocRef)

  ` Initialisiere Funktionsergebnis
$0:=""
  ` Schütze alle E/A Operationen mit einer Fehlermethode bei Unterbrechung
ON ERR CALL("ERROR METHOD")
] ` Erhalte die Liste der Volumes
ARRAY STRING(31;$asVolumes;0)
gError:=0
VOLUME LIST($asVolumes)
If(gError=0)
  ` Überspringe unter Windows die beiden gängigen Floppy Treiber
   If(On Windows)
      $vlVolume:=Find in array($asVolumes;"A:")
      If($vlVolume>0)
         DELETE FROM ARRAY($asVolumes;$vlVolume)
      End if
      $vlVolume:=Find in array($asVolumes;"B:")
      If($vlVolume>0)
         DELETE FROM ARRAY($asVolumes;$vlVolume)
      End if
   End if
   $vlNbVolumes:=Size of array($asVolumes)
  ` Für jedes Volume
   For($vlVolume;1;$vlNbVolumes)
  ` Erhalte Größe, belegten sowie freien Platz
      gError:=0
      VOLUME ATTRIBUTES($asVolumes{$vlVolume};$vlSize;$vlUsed;$vlFree)
      If(gError=0)
  ` Reicht der freie Platz aus (plus 32K extra) ?
         If($vlFree>=($1+32768))
  ` Wenn ja, prüfe, ob das Volume nicht gesperrt ist...
            $vsDocName:=$asVolumes{$vlVolume}+Char(Directory symbol)+"XYZ"+String(Random)+".TXT"
            $vhDocRef:=Create document($vsDocName)
            If(OK=1)
               CLOSE DOCUMENT($vhDocRef)
               DELETE DOCUMENT($vsDocName)
  ` Ist alles in Ordnung, gib Namen des Volume zurück
```

```
                $0:=$asVolumes{$vlVolume}
                $vlVolume:=$vlNbVolumes+1
            End if
        End if
    End if
  End for
End if
ON ERR CALL("")
```

Haben Sie diese Projektmethode in Ihre Anwendung eingefügt, können Sie z.B. schreiben:

```
$vsVolume:=Find volume for space(100*1024*1024)
If($vsVolume#"")
  ` Fortfahren
Else
  ALERT("Ein Volume mit mindestens 100 MB freiem Platz wird benötigt!")
End if
```

VOLUME LIST

VOLUME LIST (Volumes)

Parameter	Typ		Beschreibung
Volumes	Array String	⇐	Namen der derzeit angemeldeten Volumes

Beschreibung

Der Befehl **VOLUME LIST** füllt *Volumes*, das Array vom Typ Text mit den Namen der aktuell definierten Volumes (Windows) oder hochgefahrenen Volumes (Mac OS) auf Ihrem Rechner.

- Auf Macintosh gibt er die Liste der Volumes an, die auf der Finder-Ebene sichtbar sind. Nur die Namen der Volumes werden zurückgegeben, z.B. "MacHD", "BootCamp".
- Unter Windows gibt er die Liste der derzeit definierten Volumes an, unabhängig davon, ob jedes Volume auch wirklich präsent ist. So wird beispielsweise Volume E:\ angezeigt, egal ob im Laufwerk derzeit eine CD oder DVD vorhanden ist. Das Trennungszeichen für Ordner wird an den Namen angehängt, z.B. C:\

Beispiel

Mit folgendem Code können Sie mit dem rollbaren Bereich *atVolumes* die Liste der definierten bzw. auf Ihrem Rechner angemeldeten Volumes anzeigen:

```
Case of
   :(Form event=On Load)
      ARRAY TEXT(atVolumes;0)
      VOLUME LIST(atVolumes)

 //...
End case
```

_o_Document creator

_o_Document creator (Dokumentname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Dokumentname	String	➡	Dokumentname oder kompletter Pfadname
Funktionsergebnis	String	↩	Leerer String (Windows) oder Datei Creator (Mac OS)

Hinweis zur Kompatibilität

Diese Funktion ist überholt. Sie führt in 4D v16 R6 und höher nichts aus.

_o_Document type

_o_Document type (Dokumentname) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Dokumentname	String	→	Name des Dokuments
Funktionsergebnis	String	⮌	Endung der Windows-Datei (1 bis 3 Zeichen) oder Mac OS-Dateityp (4 Zeichen)

Hinweis zur Kompatibilität

Diese Funktion ist überholt. Verwenden Sie jetzt zum Extrahieren der Dateiendung die Funktion **Path to object**.

_o_MAP FILE TYPES

_o_MAP FILE TYPES (Mac OS ; Windows ; Meldung)

Parameter	Typ		Beschreibung
Mac OS	String	➡	Mac OS Dateityp (4 Zeichen)
Windows	String	➡	Windows Dateierweiterung (1 bis 3 Zeichen)
Meldung	String	➡	Text zur Anzeige im Windows-Öffnen Dialog

Hinweis zur Kompatibilität

Dieser Befehl ist ab 4D v16 R6 überholt und sollte nicht mehr verwendet werden.

Die Verwendung von Mac OS Dateitypen ist seit vielen Versionen von macOS überholt. Wie unter Windows basiert die Identifikation der Datei auf der Dateiendung. Dateiendungen können Sie mit den Funktionen **Object to path** und **Path to object** verwalten. Dateitypen können Sie mit UTIs zusammen mit Info.plist Deklarationen verwalten.

_o_SET DOCUMENT CREATOR

_o_SET DOCUMENT CREATOR (Dokumentname ; DateiCreator)

Parameter	Typ		Beschreibung
Dokumentname	String	➡	Name des Dokuments oder kompletter Pfadname
DateiCreator	String	➡	Mac OS Datei Creator (4 Zeichen) oder leerer String (Windows)

Hinweis zur Kompatibilität

Dieser Befehl ist überholt. Er führt in 4D v16 R6 und höher nichts aus.

_o_SET DOCUMENT TYPE

_o_SET DOCUMENT TYPE (Dokumentname ; Dateityp)

Parameter	Typ		Beschreibung
Dokumentname	String	➡	Name des Dokuments oder kompletter Pfadname
Dateityp	String	➡	Erweiterung der Windows Datei (1 bis 3 Zeichen) oder Mac OS Dateityp (4 Zeichen)

Hinweis zur Kompatibilität

Diese Funktion ist ab 4D v16 R6 überholt. Verwenden Sie jetzt zum Ändern der Dateiendung die Funktion **Object to path**.

Systemumgebung

- Count screens
- Current client authentication
- Current machine
- Current system user
- FONT LIST
- FONT STYLE LIST
- GET SYSTEM FORMAT
- Get system info Neu 17.0
- Is macOS Neu 17.0
- Is Windows Neu 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 -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Lange Ganzzahl	Anzahl der Monitore

Beschreibung

Die Funktion **Count screens** gibt die Anzahl der an Ihren Rechner angeschlossenen Monitore zurück.

Current client authentication

Current client authentication {(Domain ; Protokoll)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Domain	Text	⇐	Domain Name
Protokoll	Text	⇐	"Kerberos", "NTML" oder leerer String
Funktionsergebnis	Text	⇒	Von Windows zurückgegebenes Benutzer-Login der Sitzung

Beschreibung

Die Funktion **Current client authentication** fordert Active Directory des Windows Server auf, den aktuellen Client zu authentifizieren und gibt bei erfolgreicher Authentifizierung den Login Namen für diesen Client zurück (Session Identifier). Schlägt die Authentifizierung fehl, wird ein leerer String zurückgegeben.

Diese Funktion lässt sich nur mit 4D Server unter Windows verwenden, wenn SSO aktiviert ist. Weitere Informationen dazu finden Sie im Abschnitt **Single Sign On (SSO) unter Windows**.

In der Regel müssen Client und Server vom gleichen Active Directory verwaltet werden. Es werden jedoch verschiedene Konfigurationen unterstützt. Weitere Informationen dazu finden Sie im Abschnitt **Anforderungen für SSO**.

Der zurückgegebene String des Login wird an das 4D Modul zur Identifizierung übergeben, um dem Client gemäß dem Login für die Windows-Sitzung den Zugriff zu gewähren; um den 4D Server Login Dialog zu entfernen, setzen Sie einen Standardbenutzer, so dass der Benutzer seine Kennung nicht erneut eingeben muss (siehe Beispiel).

Die Funktion kann optional zwei Textparameter zurückgeben:

- *Domain*: Name des Domain, zu dem der Client gehört
- *Protokoll*: Name des Protokolls, das Windows zum Authentifizieren des Benutzers verwendet. Das ist je nach verfügbaren Ressourcen "Kerberos" oder "NTLM".
  Schlägt die Authentifizierung fehl, wird ein leerer String ("") zurückgegeben

Über diese Parameter können Sie den Zugriff zusätzlich über Domain oder Protokoll filtern und Verbindungen über diese Kriterien zulassen oder abweisen.

Sicherheitsstufen der Authentifizierung

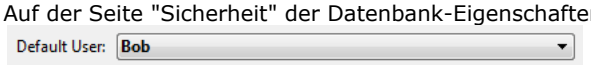
Die Sicherheitsstufe der Authentifizierung (z.B. wie vertrauenswürdig ist das Login des Benutzers) hängt davon ab, wie der Benutzer aktuell identifiziert wurde. Gibt die Funktion **Current client authentication** in ihren Parametern Werte (optional) zurück, können Sie erkennen, worauf das Login basiert und so Aufschluss über die Sicherheitsstufe erhalten:

Login	Domain	Protokoll	Kommentar
Leer	Leer	Leer	Die Funktion konnte keine Angaben zur Authentifizierung des eingeloggten Benutzers erhalten
Gefüllt	Leer	NTLM	Die zurückgegebene ID ist lokal, sie wurde auf dem lokalen Rechner definiert
Gefüllt	Gefüllt	NTLM	Die zurückgegebene ID wurde über das NTLM Protokoll im Domain authentifiziert, das im Parameter *Domain* zurückgegeben wird. In diesem Fall können Sie das Domain überprüfen, um die Sicherheitsstufe zu erhöhen. Da einige Architekturen ein "Domain Forest" haben, müssen Sie sicherstellen, dass das Domain, in welchem der Benutzer authentifiziert wurde, das erwartete ist.
Gefüllt	Gefüllt	Kerberos	Die zurückgegebene ID wurde über das Kerberos Protokoll im erwarteten Domain authentifiziert. Diese Konfiguration bietet die höchste Sicherheitsstufe.

Weitere Informationen dazu finden Sie im Abschnitt .

Beispiel

Sie haben in Ihrer 4D Server Anwendung ein Kennwortsystem mit 4D Benutzern und Gruppen eingerichtet. Sie wollen Ihre Anwendung so einstellen, dass 4D remote Benutzer unter Windows sich direkt an 4D Server anmelden können (kein Kennwortdialog wird angezeigt), wenn sie mit ihren aktuellen Rechten eingeloggt sind. Dazu müssen Sie folgendes ausführen:

1. Auf der Seite "Sicherheit" der Datenbank-Eigenschaften definieren Sie einen Benutzer als "Standardbenutzer":
   

   Mitt dieser Einstellung erscheint kein Kennwortdialog, wenn sich ein remote 4D an den Server anmeldet. Alle Clients werden als "Bob" eingeloggt.
2. In der Datenbankmethode *On Server Open Connection* fügen Sie folgenden Code hinzu, um die Benutzerauthentifizierung von Active Directory zu prüfen:

```
 //Datenbankmethode On Server Open Connection
C_LONGINT($0;$1;$2;$3)
$login:=Current client authentication($domain;$protocol)
If($login #"")  //ein Login wurde zurückgegeben
 //eigene Authentifizierungsmethode aufrufen
   $0:=CheckCredentials
 //sollte, wenn erfolgreich 0 zurückgeben, bei Fehler -1
Else
   $0:=-1  //Verbindung abweisen
End if
```

Hinweis: Dieses Beispiel zeigt einen einfachen Ablauf, den Sie an Ihre eigenen Lösungen anpassen müssen. Die eigene Authentifizierungsmethode für 4D Benutzer (hier ***CheckCredentials*** genannt) könnte folgende Schritte enthalten:

- für einen automatischen Ablauf in den Namen für 4D Benutzer und Gruppen die Namen von Active Directory abbilden
- die zurückgegebene Information an eine eigene Tabelle [Benutzer] weiterleiten
- über LDAP Funktionen die Benutzer-Zugangsdaten erhalten

Current machine

Current machine -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	String	Netzwerkname des Rechners

Beschreibung

Die Funktion **Current machine** gibt den Namen des Rechners zurück, wie er in den Netzwerk-Parametern des Betriebssystems eingerichtet wurde.

Beispiel

Auch wenn Sie nicht im Client/Server-Betrieb arbeiten, enthält Ihre Anwendung einige Netzwerkdienste, für die Ihr Rechner korrekt konfiguriert sein muss. In der **Datenbankmethode On Startup** Ihrer Anwendung schreiben Sie:

```
If((Current machine="")|(Current machine owner=""))
   ` Zeige Dialogfenster, das den Benutzer auffordert, die Netzwerkkennung auf seinem Rechner einzurichten
End if
```

Current system user

Current system user -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	String	Netzwerkname des Eigentümers des Rechners

Beschreibung

Die Funktion **Current system user** gibt den Eigentümernamen Ihres Rechners zurück, der im aktuellen Benutzerkonto auf dem Rechner angegeben ist.

Beispiel

Siehe Beispiel zum Befehl **Current machine**.

FONT LIST

FONT LIST (Schriften {; ListeTyp | *})

Parameter	Typ		Beschreibung
Schriften	Array Text	⇐	Array der Schriftnamen
ListeTyp | *	Lange Ganzzahl, Operator	⇒	Zurückzugebende Liste mit Schrifttypen oder * für Schriftnamen auf OS X

Beschreibung

Der Befehl **FONT LIST** füllt das Textarray *Schriften* mit den Namen der auf Ihrem System verfügbaren, skalierbaren Schriften.

Im Parameter *ListeTyp* geben Sie die gewünschte Liste der Schrifttypen an. Dazu übergeben Sie eine der Konstanten unter dem Thema **Schriften Typliste**:

Konstante	Typ	Wert	Kommentar
Favorite fonts	Lange Ganzzahl	1	*Schriften* enthält die Liste der bevorzugten Schriften. - Unter Windows: Liste mit den Namen der aktiven Schriftfamilien - Unter OS X: Liste mit Namen der Schriftfamilien aus der Sammlung, in Englisch genannt "Favorites", in Französisch "Favoris", in Deutsch "Favoriten", etc. Hat der Benutzer keine bevorzugten Schriften hinzugefügt, ist die Sammlung leer.
Recent fonts	Lange Ganzzahl	2	Schriften enthält die Liste der zuletzt benutzten Schriften, d.h. die während der 4D Sitzung verwendet werden. Diese Liste wird insbesondere für Textbereiche mit Mehrfachstil eingesetzt.
System fonts	Lange Ganzzahl	0	Schriften enthält die Liste aller Systemschriften. Standardoption, wenn *ListeTyp* weggelassen wird.

Übergeben Sie auf OS X den optionalen Parameter *, füllt der Befehl das Array *Schriften* mit den Namen der Schriften selbst, und nicht mit den Namen der Schriftfamilien. Die Standardoperation vereinfacht die programmierte Verwaltung von Bereichen mit Rich Text, die Schriftfamilien verwenden. Mit dem Parameter * werden Schriftnamen wie z.B. "Arial bold", "Arial italic", "Arial narrow italic," zurückgegeben, anstelle der Schriftfamilie "Arial".
Unter Windows hat der Parameter * keine Auswirkung. Der Befehl gibt wie in früheren 4D Versionen Schriftfamilien zurück.

Hinweis: Verwenden Sie auf OS X das Ergebnis dieses Befehls mit **ST SET ATTRIBUTES** in einem Textbereich mit Mehrfachstil, dürfen Sie den Parameter * nicht übergeben (nur Schriftfamilien werden als Attribute font name unterstützt). Diese Einschränkung gilt nicht für 4D Write Pro Bereiche, die Name einer Schrift aber auch einer Schriftfamilie akzeptieren.

Über skalierbare Schriften

Dieser Befehl gibt nur Vektor-Schriften zurück. Die Verwendung nicht-vektorientierter Schriften (z.B. Bitmap-Schriften) zum Gestalten von Oberflächen wird nicht empfohlen, da sie auf einer veralteten Technologie basieren und bei Variationen der Größe eingeschränkt sein können. Neue und innovative 4D Features, wie 4D Write Pro Areas, unterstützen sie nicht mehr.

Für OS X gilt dieses Prinzip bereits seit OS X 10.4, da Bitmap-Schriften (QuickDraw) ab dieser Version veraltet sind.

Unter Windows gilt dieses Prinzip ab 4D v15 R4. Um Entwickler zum Auswählen moderner Schriften für ihre Oberflächen zu ermuntern, gibt der Befehl **FONT LIST** keine Bitmap-Schriften zurück, sondern nur Vektor-Schriften vom Typ "trueType" oder "openType". Schriften wie z.B. "ASI_Mono", "MS Sans Serif", "System" werden ignoriert. Außerdem wurden GDI Namen entfernt; nur DirectWrite Namen von Schriftfamilien sind verfügbar. Beispiel: die Schriftfamilien "Arial Black" oder "Segoe UI Black" werden ignoriert, nur "Arial" und "Segoe" werden zurückgegeben.

Hinweis zur Kompatibilität unter Windows:

- Sie können in Ihren 4D Formularen weiterhin Bitmap-Schriften verwenden, außer in 4D Write Pro Bereichen. Sie wurden nur aus der Liste entfernt, die dieser Befehl zurückgibt. Um die Kompatibität mit zukünftigen 4D Versionen sicherzustellen, empfehlen wir jedoch, nur DirectWrite Namen von Schriftfamilien zu verwenden.
- Da Bitmap-Schriften im Parameter *Schriften* unter Windows aussortiert werden, unterscheidet sich die Ergebnisliste in 4D v15 R4 Applikationen und höher von den Listen in bisherigen Releases. Sie müssen Ihren Code entsprechend anpassen, wenn Sie diesen Befehl zum Auswählen nicht-vektororientierter Schriften verwenden.

Beispiel 1

Sie wollen in einem Formular eine DropDown Liste, die alle in Ihrem System verfügbaren Schriften anzeigt. Dazu schreiben Sie folgende Methode:

```
Case of
   :(Form event=On Load)
      ARRAY TEXT(asFont;0)
      FONT LIST(asFont)
   ` ...

End case
```

Beispiel 2

Die Liste der zuletzt benutzten Schriften erhalten:

```
FONT LIST($arrFonts;Recent fonts)
```

FONT STYLE LIST

FONT STYLE LIST (SchriftFamilie ; SchriftStilListe ; SchriftNameListe)

Parameter	Typ		Beschreibung
SchriftFamilie	Text	➡	Name der Schriftfamilie
SchriftStilListe	Array Text	⬅	Liste der Schriftstile, die von SchriftFamilie unterstützt wird
SchriftNameListe	Array Text	⬅	Liste der kompletten Schriftnamen, die von SchriftFamilie unterstützt wird

Beschreibung

Der Befehl **FONT STYLE LIST** gibt die Liste der Schriftstile und der kompletten Schriftnamen zurück, die von der im Parameter *SchriftFamilie* definierten Schriftfamilie unterstützt werden. Mit diesem Befehl lassen sich Oberflächen gestalten, die mit Schriften und Schriftstilen arbeiten, insbesondere 4D Write Pro Areas (siehe **4D Write Pro Handbuch**).

In *SchriftFamilie* übergeben Sie den Namen der Schriftfamilie, deren unterstützte Schriftstile und Namen Sie wissen wollen.

In *SchriftStilListe* übergeben Sie ein Text Array, das mit den Schriftstilen gefüllt wird, die von *SchriftFamilie* unterstützt werden. Stile werden mit lokalisierten Namen zurückgegeben. Beispiel: das Element "Italic" wird auf einem deutschen System mit "kursiv" zurückgegeben. Auf diese Weise können Sie z.B. ein lokalisiertes PopUp-Menü "Stil" einrichten.

In *SchriftNameListe* übergeben Sie ein Text Array, das mit den kompletten Schriftnamen gefüllt wird, die von *SchriftFamilie* unterstützt werden. Dieses Array gibt jedoch, im Gegensatz zum Array *SchriftStilListe* keine lokalisierten Werte zurück, wie z.B. Schriftnamen basierend auf der Systemidentifikation. Somit sind Schriftnamen unabhängig von der Sprache des Systems. Elemente dieses Arrays sind Strings für das 4D Write Pro Attribut wk font im Befehl **WP SET ATTRIBUTES**. Über dieses Feature können 4D Write Pro Dokumente Schriftnamen speichern, die sich ohne Schriftenprobleme auf Rechnern in jeder Systemsprache öffnen lassen.
Wird *SchriftFamilie* nicht auf dem Rechner gefunden, werden die Arrays leer zurückgegeben. Über den Befehl **FONT LIST** können Sie die Liste der Schriftfamilien erhalten, die auf dem Rechner vorhanden sind

Beispiel

Die für die Schrift "Verdana" vorhandenen Stilarten auswählen (wenn vorhanden):

```
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

 //Die Ergebnisarrays könnten zum Beispiel lauten:
 //$aTStyles{1}="Normal"
 //$aTStyles{1}="Kursiv"
 //$aTStyles{1}="Fett"
 //$aTStyles{1}="Fett Kursiv"

 // $aTnames{1}="Verdana"
 // $aTnames{1}="Verdana Italic"
 // $aTnames{1}="Verdana Bold"
 // $aTnames{1}="Verdana Bold Italic"
```

GET SYSTEM FORMAT

GET SYSTEM FORMAT (Format ; Wert)

Parameter	Typ		Beschreibung
Format	Lange Ganzzahl	➡	Zu findendes Systemformat
Wert	String	⬅	Wert des im System definierten Formats

Beschreibung

Der Befehl **GET SYSTEM FORMAT** gibt den aktuellen Wert verschiedener landesspezifischer Parameter zurück, die im Betriebssystem definiert sind. Mit diesem Befehl können Sie "automatische" angepasste Formate erstellen, die auf Systemeinstellungen basieren.

Im Parameter *Format* übergeben Sie den Parametertyp, dessen Wert Sie wissen möchten. Das Ergebnis wird direkt vom System im Parameter *Wert* als Zeichenkette zurückgegeben.

Sie müssen in *Format* eine der folgenden Konstanten unter dem Thema **Systemformat** übergeben:

Konstante	Typ	Wert	Kommentar
Currency symbol	Lange Ganzzahl	2	Währungszeichen, z.B."€"
Date separator	Lange Ganzzahl	13	Trenner für Datumsformate, z.B. "."
Decimal separator	Lange Ganzzahl	0	Dezimaltrenner, z.B. ","
Short date day position	Lange Ganzzahl	15	Position des Tages für kurzes Datumsformat: "1" = links, "2" = mittig, "3" = rechts
Short date month position	Lange Ganzzahl	16	Position des Monats für kurzes Datumsformat: "1" = links, "2" = mittig, "3" = rechts
Short date year position	Lange Ganzzahl	17	Position des Jahres für kurzes Datumsformat: "1" = links, "2" = mittig, "3" = rechts
System date long pattern	Lange Ganzzahl	8	Anzeigeformat für langes Datumsformat in Form von "dddd MMMM yyyy"
System date medium pattern	Lange Ganzzahl	7	Anzeige für mittleres Datumsformat in Form von "dddd MMMM yyyy"
System date short pattern	Lange Ganzzahl	6	Anzeige für kurzes Datumsformat in Form von "dddd MMMM yyyy"
System time AM label	Lange Ganzzahl	18	Zusatzbezeichnung für Zeit vor Mittag in 12-Stunden Formaten, z.B. "Morgen"
System time long pattern	Lange Ganzzahl	5	Anzeige für langes Zeitformat in Form von "HH:MM:SS"
System time medium pattern	Lange Ganzzahl	4	Anzeige für mittleres Zeitformat in Form von "HH:MM:SS"
System time PM label	Lange Ganzzahl	19	Zusatzbezeichnung für Zeit nach Mittag in 12-Stunden Formaten, z.B. "Abend"
System time short pattern	Lange Ganzzahl	3	Anzeige für kurzes Zeitformat in Form von "HH:MM:SS"
Thousand separator	Lange Ganzzahl	1	Trenner für Tausend, z.B "."
Time separator	Lange Ganzzahl	14	Trenner für Zeitformate, z.B ":"

Beispiel

Bei einem per Hand ausgefüllten Scheck wird vor den Betrag in der Regel das Zeichen "*" gesetzt, um Betrug zu unterbinden. Ist das standardmäßige Anzeigeformat des Systems für Währung "$ 5,422.33", sollte das Format für Schecks vom Typ "$***5432.33" sein, d.h. kein Komma nach der Tausenderstelle, kein Leerzeichen zwischen dem Symbol $ und der 1. Ziffer.
In der Funktion **String** muss das Format lauten "$*******.**".
Um das per Programmierung einzurichten, müssen Währungssymbol und Dezimaltrenner bekannt sein. Sie schreiben folgende Anweisung:

```
GET SYSTEM FORMAT(Currency symbol;$vCurrSymb)
GET SYSTEM FORMAT(Decimal separator;$vDecSep)
$MyFormat:="###"+$vCurrSymb+"*******"+$vDecSep"**"
$Result:=String(amount;$MyFormat)
```

Get system info

Get system info -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Objekt	Systeminformation

Beschreibung

Der Befehl **Get system info** gibt ein Objekt zurück, das Informationen über das Betriebssystem und die Eigenschaften der Systemhardware und -software von der Maschine, auf der es ausgeführt wird, enthält.

Der Befehl gibt die folgenden Informationen zurück:

Eigenschaft	Untereigenschaft		Typ	Beschreibung	Beispiel
accountName			String	Der Name des Kontos für den aktuellen Benutzer. Wird üblicherweise verwendet, um ein Konto im Verzeichnis zu identifizieren.	"msmith"
cores			Zahl	Gesamtzahl der Kerne. Bei virtuellen Maschinen die Gesamtzahl der Cores, die ihr zugewiesen wurden.	4
cpuThreads			Zahl	Gesamtzahl der Threads	8
machineName			String	Der Name der Maschine, wie er in den Netzwerkparametern des Betriebssystems festgelegt ist.	"LAPTOP-M3BLHGSG"
model			String	Name des Computermodells	"iMac12,2", "Dell", "Acer", "VMware", etc.
networkInterfaces			Collection	Nur physikalische und aktive Netzwerkadressen	
	ipAddresses		Collection		
		ip	String	Adresse der Netzwerkschnittstelle	"129.186.81.80"
		type	String	Typ der Netzwerkschnittstelle	"ipv4", "ipv6"
	name		String	Name der Schnittstelle	"Intel(R) 82574L Gigabit Network Connection"
	type		String	Typ der Schnittstelle (beachten Sie, dass für bluetooth der Typ "ethernet" geliefert wird)	"wifi", "ethernet"
osVersion			String	Die Betriebssystem-Version und Build-Nummer (*).	"Microsoft Windows 10 Professionnel 10.0.14393"
osLanguage			String	Sprache, die vom aktuellen Benutzer des Systems eingestellt wird. Ausgedrückt in dem Standard, der durch den RFC 3066 definiert ist. Eine vollständige Liste finden Sie unter **Programmiersprache Codes** im Handbuch *Designmodus*.	"fr", "en", "ja", "de", etc.
physicalMemory			Zahl	Größe des auf der Maschine verfügbaren Hauptspeichers (in Kilobyte)	16777216
processor			String	Name, Typ und Geschwindigkeit des Prozessors	"Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz"
uptime			Zahl	Die Gesamtzeit (in Sekunden) seit dem Start der Maschine.	3600
userName			String	Der aktuelle Benutzer auf der Maschine. Wird üblicherweise als Anzeigename verwendet (z.B. beim Einloggen in Ihren Computer).	"Mary Smith"
volumes			Collection		
	available		Zahl	Der verbleibende Platz, der genutzt werden kann.	524288
	capacity		Zahl	Das Gesamtvolumen (in Kilobyte), das möglich ist.	1048576
	disk		Objekt | Collection (nur Mac)		
		description	String	Kurze Zusammenfassung, die die Festplatte beschreibt.	"HP LOGICAL VOLUME SCSI Disk Device"
		identifier	String	ID der Festplatte(n) (UUID auf Mac und physisches Laufwerk auf Windows)	Mac - "87547BDD-EA75-4F48-8BFA-9A7E393EEAB0", Windows - "\\\\.\\PHYSICALDRIVE0"
		size	Zahl	Die Gesamtkapazität (in Kilobyte) der Festplatte	104857600
		interface	String	Die Art der Schnittstelle auf der Maschine	"USB" "Netzwerk" "SATA", "SCSI", "cd/dvd", "PCI"
	fileSystem		String	Das Dateisystem, das vom Betriebssystem zum Speichern und Abrufen von Dateien auf der Festplatte verwendet wird.	"NTFS", "Journaled HFS+", "GPFS", etc.
	mountPoint		String	Das Verzeichnis im aktuell	Mac - "/Volumes/Free

		zugänglichen Dateisystem, auf dem ein zusätzliches Dateisystem gemountet (d.h. logisch angehängt) ist. Beachten Sie, dass dies im POSIX-Format für Macs ist.	HD", Windows - "C:"
name	String	nur auf Mac - Name des Volumes	"iMac-27-Program6"

(*) Um nur die verwendete Plattform zu bestimmen, stehen zwei Befehle zur Verfügung: **Is macOS** und **Is Windows**.
Hinweis: Bei virtuellen Maschinen sind die zurückgegebenen Informationen die der virtuellen Maschine.

Beispiel

Folgender Code auf einem Windows-Rechner:

```
C_OBJECT($systemInfo)
$systemInfo:=Get system info
```

gibt ein Objekt mit den folgenden Informationen zurück:

{ "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 -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	True wenn Betriebssystem = macOS, sonst False

Beschreibung

Der Befehl **Is macOS** gibt True zurück, wenn das aktuelle Betriebssystem macOS ist.

Beispiel

Sie wollen feststellen, ob das aktuelle Betriebssystem macOS ist:

```
If(Is macOS)
    ALERT("Es ist macOS")
Else
    ALERT("Es ist nicht macOS")
End if
```

Is Windows

Is Windows -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		True wenn Betriebssystem = Windows, sonst False

Beschreibung

Der Befehl **Is Windows** gibt True zurück, wenn das aktuelle Betriebssystem Windows ist.

Beispiel

Sie möchten feststellen, ob das aktuelle Betriebssystem Windows ist:

```
If(Is Windows)
    ALERT("Es ist Windows")
Else
    ALERT("Es ist nicht Windows")
End if
```

LOG EVENT

LOG EVENT ({AusgabeTyp ;} Meldung {; Wichtigkeit})

Parameter	Typ		Beschreibung
AusgabeTyp	Lange Ganzzahl	➡	Ausgabetyp für Meldung
Meldung	String	➡	Inhalt der Meldung
Wichtigkeit	Lange Ganzzahl	➡	Wichtigkeitsebene der Meldung

Beschreibung

Der Befehl **LOG EVENT** ermöglicht, ein eigenes System zum Protokollieren interner Ereignisse einzurichten, die während dem Einsatz der Anwendung auftreten.

In *Meldung* übergeben Sie die eigene Information, die je nach Ereignis protokolliert wird.

Im optionalen Parameter *AusgabeTyp* können Sie für *Meldung* den Ausgabekanal angeben. Sie können eine der folgenden Konstanten unter dem Thema **Log Ereignisse** übergeben:

Konstante	Typ	Wert	Kommentar
Into 4D commands log	Lange Ganzzahl	3	Dieser Wert weist 4D an, *Meldung* im Logbuch der 4D Befehle zu speichern, wenn diese Datei aktiviert wurde. Das Logbuch der 4D Befehle lässt sich über den Befehl **SET DATABASE PARAMETER** (Selector 34) aktivieren. **Hinweis:** 4D Log Dateien werden im Ordner **Logs** gesammelt, der neben der Strukturdatei der Datenbank liegt.
Into 4D debug message	Lange Ganzzahl	1	Dieser Wert weist 4D an, *Meldung* an die Debugging Umgebung des Systems zu senden. Das Ergebnis richtet sich nach der jeweiligen Plattform: - Mac OS: Der Befehl sendet *Meldung* an die Konsole. - Windows: Der Befehl sendet *Meldung* als eine Debug Meldung. Zum Lesen dieser Meldung benötigen Sie Microsoft Visual Studio oder das Hilfsprogramm DebugView für Windows (http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx).
Into 4D diagnostic log	Lange Ganzzahl	5	Dieser Wert weist 4D an, *Meldung* in der Protokolldatei von 4D einzutragen, wenn diese Datei aktiviert wurde. Diese Datei lässt sich über den Befehl **SET DATABASE PARAMETER** (Selector 79) aktivieren.
Into 4D request log	Lange Ganzzahl	2	Dieser Wert weist 4D an, *Meldung* im Logbuch 4D Anfragen zu speichern, wenn diese Datei aktiviert wurde.
Into Windows log events	Lange Ganzzahl	0	Dieser Wert weist 4D an, *Meldung* die "Log Ereignisse" des Fensters zu senden. Dieses Logbuch empfängt und speichert Meldungen von den laufenden Anwendungen. Dann können Sie *Meldung* über den optionalen Parameter *Wichtigkeit* eine Wichtigkeitsstufe zuordnen (siehe oben). **Hinweise:** • Diese Funktionalität ist nur verfügbar, wenn der Service Windows Log Events läuft. • Auf Mac OS führt der Befehl mit diesem Wert in *AusgabeTyp* nichts aus.

Übergeben Sie nicht den Parameter *AusgabeTyp*, wird standardmäßig der Wert 0 (Into Windows Log Events) verwendet.

Haben Sie den Parameter *AusgabeTyp* als Into Windows Log Events definiert, können Sie in *Meldung* eine Wichtigkeitsebene zuordnen, das vereinfacht die Interpretation von Log Ereignissen. Es gibt drei Ebenen: Information, Warnung und Fehler. 4D bietet dafür drei vordefinierte Konstanten unter dem Thema **Log Ereignisse**:

Konstante	Typ	Wert
Error message	Lange Ganzzahl	2
Information message	Lange Ganzzahl	0
Warning message	Lange Ganzzahl	1

Geben Sie im Parameter *Wichtigkeit* keinen oder einen inkorrekten Wert an, wird der Standardwert 0 verwendet.

Beispiel

Wollen Sie eine Meldung erhalten, wenn Ihre Datenbank unter Windows geöffnet ist, könnten Sie in der **Datenbankmethode On Startup** z.B. folgenden Code schreiben:

```
LOG EVENT(Into Windows log events;"Die Datenbank Rechnungen wurde geöffnet.")
```

Immer wenn die Datenbank geöffnet ist, wird diese Information in die **Windows Log Events** geschrieben; die Wichtigkeitsebene ist 0.

Menu bar height

Menu bar height -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Höhe (in Pixel) der Menüleiste (gibt bei ausgeblendeter Menüleiste Null zurück)

Beschreibung

Die Funktion **Menu bar height** gibt die Höhe der Menüleiste in Pixel zurück.
Die Funktion gibt 0 zurück:

- Wenn die Menüleiste ausgeblendet ist
- Im SDI Modus unter Windows, wenn sie in einem Prozess ohne Formularfenster aufgerufen wird. Weitere Informationen dazu finden Sie im Abschnitt **SDI Modus unter Windows**.

Hinweis: Läuft die Anwendung im SDI Modus unter Windows, gibt **Menu bar height** immer die Höhe einer Zeile der Menüleiste zurück, auch wenn Fenster verkleinert wurde und die Menüleiste in zwei oder mehr Zeilen umgebrochen wird.

Menu bar screen

Menu bar screen -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl	⮌	Nummer des Bildschirms, der die Menüleiste enthält

Beschreibung

Menu bar screen gibt die Nummer der Bildschirms an, der die Menüleiste enthält.
Windows Hinweis: Unter Windows gibt **Menu bar screen** normalerweise 1 zurück.

OPEN COLOR PICKER

OPEN COLOR PICKER {(TextOderHintergrund)}

Parameter	Typ		Beschreibung
TextOderHintergrund	Lange Ganzzahl	➡	0 oder weggelassen = Farbe Text 1 = Farbe Texthintergrund

Beschreibung

Der Befehl **OPEN COLOR PICKER** zeigt die Farbpalette des Systems.

Hinweis: Dies ist unter Windows ein modales Dialogfenster, aber nicht auf OS X.

Wählt der Benutzer eine Farbe und bestätigt das Dialogfenster, wird diese Farbe auf die aktuelle Textauswahl im Objekt mit Fokus angewandt, wenn für dieses Objekt die Eigenschaft **Schrift/Farbpalette erlauben** markiert ist. Weitere Informationen dazu finden Sie im Abschnitt **Textattribute** des Handbuchs *4D Designmodus*.

Übergeben Sie 0 im Parameter *TextOderHintergrund* oder lassen diesen Parameter weg, wird die gewählte Farbe auf den Text angewendet. Übergeben Sie 1 in *TextOderHintergrund*, wird die Farbe auf den Texthintergrund angewandt.

Bei Ändern der Farbe wird für das Objekt das Formularereignis On After Edit erzeugt.

OPEN FONT PICKER

OPEN FONT PICKER
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **OPEN FONT PICKER** zeigt die Schriftenpalette des Systems.

Hinweis: Dies ist unter Windows ein modales Dialogfenster, aber nicht auf OS X.

Wählt der Benutzer eine Schrift bzw. einen Stil und bestätigt das Dialogfenster, werden die Änderungen für die aktuelle Textauswahl im Objekt mit Fokus angewandt, wenn für dieses Objekt die Eigenschaft **Schrift/Farbpalette erlauben** markiert ist. Andernfalls führt der Befehl nichts aus.
Weitere Informationen dazu finden Sie im Abschnitt **Textattribute** des Handbuchs *4D Designmodus*.

Bei Ändern der Schrift wird für das Objekt das Formularereignis On After Edit erzeugt.

Beispiel

In einen Formular eine Schaltfläche zum Anzeigen der Schriftenpalette hinzufügen, damit Benutzer Schriftart oder -stil eines Bereichs mit Textvariable ändern können. Stellen Sie sicher, dass:

- Für die Textvariable die Eigenschaft "Schrift/Farbpalette erlauben" markiert ist.
- Für die Schaltfläche die Eigenschaft "Fokusfähig" deaktiviert ist.

Die Anweisung für die Schaltfläche lautet:

```
Case of
   :(Form event=On Clicked)
      GOTO OBJECT(textVar)  //gibt der Variablen den Fokus
      OPEN FONT PICKER
End case
```

SCREEN COORDINATES

SCREEN COORDINATES (Links ; Oben ; Rechts ; Unten {; Bildschirm})

Parameter	Typ		Beschreibung
Links	Lange Ganzzahl	⇐	Globale linke Koordinate des Bildschirms
Oben	Lange Ganzzahl	⇐	Globale obere Koordinate des Bildschirms
Rechts	Lange Ganzzahl	⇐	Globale rechte Koordinate des Bildschirms
Unten	Lange Ganzzahl	⇐	Globale untere Koordinate des Bildschirms
Bildschirm	Lange Ganzzahl	⇒	Nummer des Bildschirms, ohne Angabe Hauptbildschirm

Beschreibung

Der Befehl **SCREEN COORDINATES** gibt in *Links*, *Oben*, *Rechts* und *Unten* die globalen Koordinaten des Bildschirms zurück, der in *Bildschirm* angegeben wurde.

Geben Sie den Parameter *Bildschirm* nicht an, gibt der Befehl die Koordinaten des Hauptbildschirms zurück, d.h. des Bildschirms, auf dem die Menüleiste angezeigt wird.

SCREEN DEPTH

SCREEN DEPTH (Tiefe ; Farbe {; Bildschirm})

Parameter	Typ		Beschreibung
Tiefe	Lange Ganzzahl	⇐	Bildschirmtiefe (Anzahl der Farben = 2 ^ Tiefe)
Farbe	Lange Ganzzahl	⇐	1 = Farbbildschirm, 0 = schwarz/weiß oder Graustufen
Bildschirm	Lange Ganzzahl	⇒	Nummer des Bildschirms, ohne Angabe Hauptbildschirm

Beschreibung

Die Funktion **SCREEN DEPTH** gibt in *Tiefe* und *Farbe* Informationen über den Monitor zurück.

Nach Aufrufen der Funktion wird die Bildschirmtiefe in *Tiefe* zurückgegeben. Die Tiefe wird aus der Anzahl der auf dem Bildschirm dargestellten Farben berechnet, die immer die Potenz von zwei ist. Hat Ihr Monitor z.B. 256 Farben (2^8), ist die Bildschirmtiefe gleich 8.

4D bietet folgende vordefinierten Konstanten:

Konstante	Typ	Wert
Black and white	Lange Ganzzahl	0
Four colors	Lange Ganzzahl	2
Millions of colors 24 bit	Lange Ganzzahl	24
Millions of colors 32 bit	Lange Ganzzahl	32
Sixteen colors	Lange Ganzzahl	4
Thousands of colors	Lange Ganzzahl	16
Two fifty six colors	Lange Ganzzahl	8

Ist der Monitor auf Farben eingestellt, wird in *Farbe* 1 zurückgegeben. Ist der Monitor auf Graustufen eingestellt, wird in *Farbe* 0 zurückgegeben. Beachten Sie, dass dieser Wert auf Macintosh signifikant ist.

4D bietet folgende vordefinierten Konstanten:

Konstante	Typ	Wert
Is color	Lange Ganzzahl	1
Is gray scale	Lange Ganzzahl	0

Der optionale Parameter *Bildschirm* gibt den Monitor an, über den Sie Informationen haben wollen. Geben Sie *Bildschirm* nicht an, gibt der Befehl die Tiefe des Hauptbildschirms zurück, d.h. des Bildschirms, der die Menüleiste anzeigt.

Beispiel

Ihre Anwendung zeigt viele farbige Grafiken an. Sie können an beliebiger Stelle in Ihrer Datenbank schreiben:

```
SCREEN DEPTH($vlDepth;$vlColor)
If($vlDepth<8)
   ALERT("Die Formulare sehen besser aus, wenn der Monitor"+" auf 256 oder mehr Farben eingestellt wird.")
End if
```

Screen height

Screen height {(*)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	→	Windows: Höhe des Anwendungsfensters, mit *: Höhe des Bildschirms Macintosh: Höhe des Hauptbildschirms
Funktionsergebnis	Lange Ganzzahl	←	Höhe in Pixel

Beschreibung

Unter Windows gibt die Funktion **Screen height** die Höhe des 4D Anwendungsfensters zurück (MDI Fenster). Der optionale Parameter * übergibt die Höhe des benutzten Bildschirms.

Auf Macintosh gibt die Funktion **Screen height** die Höhe des Hauptbildschirms, d.h. des Bildschirms mit der Menüleiste, zurück.

Screen width

Screen width {(*)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	→	Windows: Breite des Anwendungsfensters, wenn * angegeben, Breite des Bildschirms Macintosh: Breite des Hauptbildschirms
Funktionsergebnis	Lange Ganzzahl	↩	Breite in Pixel

Beschreibung

Unter Windows gibt die Funktion **Screen width** die Breite des 4D Anwendungsfensters zurück (MDI Fenster). Der optionale Parameter * übergibt die Breite des benutzten Bildschirms.

Auf Macintosh gibt die Funktion **Screen width** die Breite des Hauptbildschirms zurück, das ist der Bildschirm mit der Menüleiste.

Select RGB Color

Select RGB Color {(Standardfarbe {; Meldung})} -> Funktionsergebnis

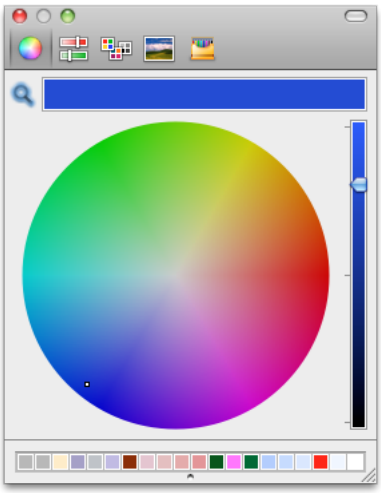
Parameter	Typ		Beschreibung
Standardfarbe	Lange Ganzzahl	➡	Vorgegebene RGB Farbe
Meldung	Alpha	➡	Titel des Auswahlfensters
Funktionsergebnis	Lange Ganzzahl	⮌	RGB Farbe

Beschreibung

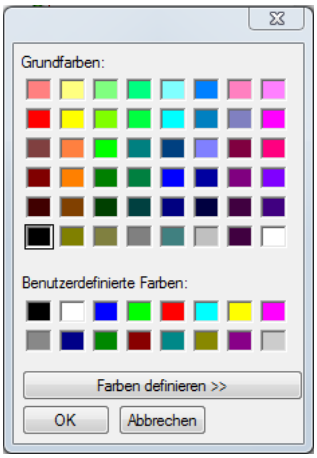
Die Funktion **Select RGB Color** zeigt das Auswahlfenster für die Systemfarbe und gibt den RGB Wert der vom Benutzer gewählten Farbe zurück.

Das Auswahlfenster sieht folgendermaßen aus:

Macintosh



Windows



Mit dem optionalen Parameter *Standardfarbe* können Sie vorab eine Farbe im Fenster auswählen. Auf diese Weise können Sie z.B. die zuletzt vom Benutzer gesetzte Farbe abspeichern. In diesem Parameter übergeben Sie den Farbwert eines RGB-Formats. Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **OBJECT SET RGB COLORS**. Sie können auch eine Konstante unter dem Thema **SET RGB COLORS** verwenden.

Übergeben Sie keinen Parameter *Standardfarbe* oder den Wert 0 (Null), wird beim Öffnen des Dialogfensters die Farbe schwarz ausgewählt.

Mit dem optionalen Parameter *Meldung* können Sie den Titel des Systemfensters anpassen. Standardmäßig, also ohne diesen Parameter, lautet der Titel “Farben”.

Bestätigen dieses Dialogfensters hat je nach Plattform eine andere Auswirkung:

- Klickt der Benutzer unter Windows auf die Schaltfläche **OK**, gibt die Funktion den Wert der gewählten Farbe im RGB-Format zurück und die Systemvariable wird auf 1 gesetzt. Klickt der Benutzer auf die Schaltfläche **Abbrechen**, gibt die Funktion -1 zurück und die Systemvariable OK wird auf 0 (Null) gesetzt.
- Auf Mac OS lässt sich dieses Fenster nur über die Schließbox oder die Esc-Taste schließen. In beiden Fällen wird die Systemvariable OK auf 1 gesetzt, unabhängig, welche Aktion der Benutzer im Fenster ausgeführt hat. Die Funktion gibt den Wert der gewählten Farbe im RGB-Format zurück. Hat der Benutzer keine Farbe gewählt, wird der im Parameter *Standardfarbe* angegebene Wert zurückgegeben oder 0 (Null), wenn *Standardfarbe* nicht übergeben wurde.

Hinweis: Dieser Befehl darf weder auf dem Server Rechner, noch innerhalb eines Web Prozesses ausgeführt werden.

SET RECENT FONTS

SET RECENT FONTS (ArraySchriften)

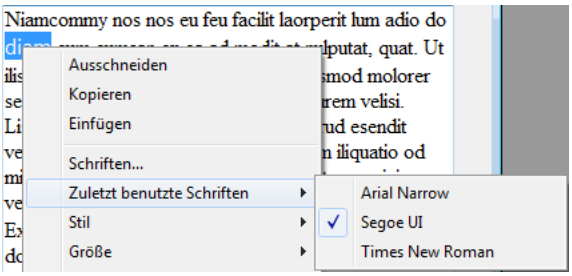
Parameter	Typ		Beschreibung
ArraySchriften	Array Text	➡	Array der Schriftnamen

Beschreibung

Der Befehl **SET RECENT FONTS** ändert die Liste der Schriften, die im Kontextmenü "zuletzt benutzte Schriften" angezeigt wird. Dieses Menü enthält die Namen der Schriften, die zuletzt in der Sitzung ausgewählt wurden. Dies wird insbesondere von **Programmierhinweise** verwendet.

Beispiel

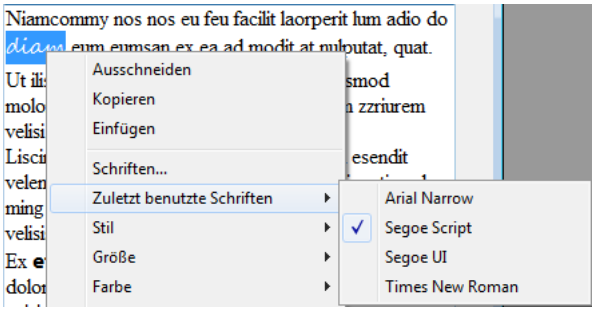
Über das Kontextmenü *Zuletzt benutzte Schriften* eine Schrift hinzufügen:



Sie führen folgenden Code aus:

```
ARRAY TEXT($arrRecent;0)
FONT LIST($arrRecent;2)
APPEND TO ARRAY($arrRecent;"Segoe Script")
APPEND TO ARRAY($arrRecent)
```

Das Menü sieht dann folgendermaßen aus:



SET SCREEN DEPTH

SET SCREEN DEPTH (Tiefe {; Farbe {; Bildschirm}})

Parameter	Typ		Beschreibung
Tiefe	Lange Ganzzahl	➡	Bildschirmtiefe (Anzahl der Farben = 2 ^ Bildschirmtiefe)
Farbe	Lange Ganzzahl	➡	1 = Farbe, 0 = Graustufen
Bildschirm	Lange Ganzzahl	➡	Nummer des Bildschirms, ohne Angabe Hauptbildschirm

Beschreibung

Der Befehl **SET SCREEN DEPTH** ändert die Tiefe und Einstellungen für Farben/Graustufen des Bildschirms mit der in *Bildschirm* übergebenen Nummer. Geben Sie diesen Parameter nicht an, gilt der Befehl für den Hauptbildschirm.

Weitere Informationen zu den Werten für *Farbe* und *Tiefe* finden Sie in der Beschreibung zum Befehl **SCREEN DEPTH**.

System folder

System folder {(Typ)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
Typ	Lange Ganzzahl	➡	Art des Systemordners
Funktionsergebnis	String	⮌	Pfadname zum Systemordner

Beschreibung

Die Funktion **System folder** gibt den Pfadnamen zu einem spezifischen Ordner des Betriebssystems oder direkt den aktiven Systemordner unter Windows bzw. auf Macintosh zurück.

Im optionalen Parameter *Typ* übergeben Sie einen Wert für die Art des Systemordners. 4D bietet dafür unter dem Thema **Systemordner** folgende vordefinierte Konstanten:

Konstante	Typ	Wert	Kommentar
Applications or program files	Lange Ganzzahl	16	
Desktop	Lange Ganzzahl	15	
Documents folder	Lange Ganzzahl	17	Benutzerordner "Documents"
Favorites Win	Lange Ganzzahl	14	
Fonts	Lange Ganzzahl	1	
Start menu Win_all	Lange Ganzzahl	8	
Start menu Win_user	Lange Ganzzahl	9	
Startup Win_all	Lange Ganzzahl	4	
Startup Win_user	Lange Ganzzahl	5	
System	Lange Ganzzahl	0	
System Win	Lange Ganzzahl	12	
System32 Win	Lange Ganzzahl	13	
User preferences_all	Lange Ganzzahl	2	
User preferences_user	Lange Ganzzahl	3	

- Konstanten mit der Endung **Win** sind nur unter Windows verwendbar. Bei Einsatz auf Mac OS gibt **System folder** einen leeren String zurück.
- Die Pfadnamen für einige Systemordner können spezifisch für den aktuellen Benutzer sein. Bei den Konstanten 2 bis 9 können Sie wählen zwischen dem Pfadnamen für einen Ordner, den alle Benutzer nutzen oder der für den aktuellen Benutzer angepasst wurde.

Lassen Sie den Parameter *Typ* weg, gibt die Funktion den Pfadnamen zum aktiven Systemordner zurück (= Konstante System).

Temporary folder

Temporary folder -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	String	Pfadname zu temporärem Ordner

Beschreibung

Die Funktion **Temporary folder** gibt den Pfadnamen zum aktuellen temporären Ordner zurück, den Ihr System festlegt.

Beispiel

Siehe Beispiel zum Befehl **APPEND DATA TO PASTEBOARD**.

_o_Font name

_o_Font name (Schriftnummer) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Schriftnummer	Lange Ganzzahl	➡	Schriftnummer für den dazugehörigen Schriftnamen
Funktionsergebnis	String	⮌	Schriftname

Beschreibung

Diese Funktion ist überholt und darf in 4D nicht mehr verwendet werden. Sie wird zur Wahrung der Kompatibilität in 4D v14 noch beibehalten, in nachfolgenden Versionen jedoch nicht mehr unterstützt.

_o_Font number

_o_Font number (SchriftName) -> Funktionsergebnis

Parameter	Typ		Beschreibung
SchriftName	String	➡	Schriftname, für den die Schriftnummer zurückgeben werden soll
Funktionsergebnis	Lange Ganzzahl	⮌	Schriftnummer

Beschreibung

Diese Funktion ist überholt und darf in 4D nicht mehr verwendet werden. Sie wird zur Wahrung der Kompatibilität in 4D v14 noch beibehalten, in nachfolgenden Versionen jedoch nicht mehr unterstützt.

_o_Gestalt

_o_Gestalt (Selector ; Wert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Selector	String	➡	4 stelliger Gestalt Selector
Wert	Lange Ganzzahl	⬅	Gestalt Ergebnis
Funktionsergebnis	Lange Ganzzahl	↩	Ergebnis der Fehlermeldung

Hinweis zur Kompatibilität

Dieser Befehl ist ab 4D v17 überholt. Sie können jetzt den Befehl **Get system info** für komplette Systeminformation verwenden.

_o_PLATFORM PROPERTIES

_o_PLATFORM PROPERTIES (Plattform {; System {; Prozessor {; Sprache}}})

Parameter	Typ		Beschreibung
Plattform	Lange Ganzzahl	⇐	2 = Mac OS, 3 = Windows
System	Lange Ganzzahl	⇐	Je nach eingesetzter Version
Prozessor	Lange Ganzzahl	⇐	Prozessorfamilie
Sprache	Lange Ganzzahl	⇐	Je nach eingesetztem System

Hinweis zur Kompatibilität

Dieser Befehl ist ab 4D v17 überholt. Sie können den Befehl **Get system info** für komplette Systeminformation verwenden oder **Is macOS** bzw. **Is Windows** für Informationen zur jeweiligen Plattform.

Tabelle

- Current default table
- Current form table
- DEFAULT TABLE
- NO DEFAULT TABLE

Current default table

Current default table -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Zeiger	Zeiger auf Tabelle

Beschreibung

Die Funktion **Current default table** gibt einen Zeiger auf die Tabelle zurück, die durch den letzten Aufruf von **DEFAULT TABLE** für den aktuellen Prozess definiert wurde.
Wurde die Funktion noch nie aufgerufen, gibt sie die Funktion **Is nil pointer** zurück.

Beispiel

Wurde eine Standardtabelle festgelegt, setzt folgende Anweisung den Fenstertitel in den Namen der aktuellen Standardtabelle:

```
SET WINDOW TITLE(Table name(Current default table))
```

Current form table

Current form table -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Zeiger	Zeiger auf die Tabelle des aktuell angezeigten Formulars

Beschreibung

Die Funktion **Current form table** gibt den Zeiger auf die Tabelle des Formulars zurück, das im aktuellen Prozess angezeigt oder gedruckt wird.
Sie gibt **Is nil pointer** zurück, wenn:

- Im aktuellen Prozess kein Formular angezeigt oder gedruckt wird
- Das aktuelle Formular ein Projektformular ist.

Sind für den aktuellen Prozess mehrere Fenster geöffnet, ist das zuletzt geöffnete Fenster das aktuelle aktive Fenster. Die Funktion gibt den Zeiger auf die Tabelle des Formulars zurück, das im aktiven Fenster angezeigt wird.
Ist das aktuell angezeigte Formular das Eingabeformular für ein Unterformular, befinden Sie sich in der Dateneingabe. Sie haben auf einen Datensatz oder Unterdatensatz doppelgeklickt. Die Funktion gibt dann folgendes zurück:

- Den Zeiger auf die Tabelle im Unterformular, wenn dieses eine Tabelle anzeigt.
- Einen nicht-signifikanten Zeiger, wenn das Unterformular eine Untertabelle anzeigt.

Beispiel

In Ihrer Anwendung verwenden Sie beim Anzeigen eines Datensatzes folgende Konvention:
Gibt es in einem Formular die Variable *vsCurrentRecord*, zeigt sie "Neuer Datensatz" an, wenn Sie mit einem neuen Datensatz arbeiten. Bearbeiten Sie den 56. Datensatz in einer Auswahl von 5200 Datensätzen, zeigt sie "56 von 5200" an.
Erstellen Sie dazu die Variable *vsCurrentRecord* in der Objektmethode, und setzen Sie diese dann per copy /paste in all Ihre Formulare ein:

```
` Objektmethode für die nicht eingebbare Variable vsCurrentRecord
Case of
   :(Form event=On Load)
      C_STRING(31;vsCurrentRecord)
      C_POINTER($vpParentTable)
      C_LONGINT($vlRecordNum)
      $vpParentTable:=Current form table
      $vlRecordNum:=Record number($vpParentTable->)
      Case of
         :($vlRecordNum=-3)
            vsCurrentRecord:="Neuer Datensatz"
         :($vlRecordNum=-1)
            vsCurrentRecord:="Kein Datensatz"
         :($vlRecordNum>=0)
            vsCurrentRecord:=String(Selected record number($vpParentTable->))+" of "+String(Records in selection($vpParentTable->))
      End case
End case
```

DEFAULT TABLE

DEFAULT TABLE (Tabellenname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Gibt die Haupttabelle an

Beschreibung

Empfehlung: Die Verwendung von **DEFAULT TABLE** und Weglassen des Tabellennamens machen den Code zwar leichter lesbar. Viele Programmierer finden jedoch, dass dieser Befehl eher für mehr Probleme und Verwirrung sorgt. Verwenden Sie **DEFAULT TABLE** z.B. zusammen mit dem Befehl **DIALOG**, hat die Standardtabelle Vorrang vor dem Projektformular, wenn die Namen gleich sind. Wir empfehlen zur Vermeidung von Problemen, **DEFAULT TABLE** nicht mehr zu verwenden.

Der Befehl **DEFAULT TABLE** setzt *Tabellenname* für den aktuellen Prozess als Haupttabelle.

Erwartet ein Befehl den Tabellennamen als ersten Parameter, setzt dieser Befehl den Namen der Haupttabelle. Dadurch entfällt in den Befehlen das ständige Wiederholen des Tabellennamens. Ohne Definition der Haupttabelle müssen Sie den Tabellennamen stets angeben.

Nehmen wir als Beispiel folgenden Befehl:

```
FORM SET INPUT([Table];"Formular")
```

Wird für die Haupttabelle zuvor [Table] festgelegt, schreiben Sie für denselben Befehl:

```
FORM SET INPUT("form")
```

Mit **DEFAULT TABLE** erstellen Sie generischen Code. So kann derselbe Code in verschiedenen Tabellen operieren. Mit Zeigern auf Tabellen erhalten Sie ebenfalls generischen Code. Weitere Informationen dazu finden Sie in der Beschreibung zur Funktion **Table name**.

Bei Datenfeldern müssen Sie den Tabellennamen angeben, auch wenn Sie vorher eine Tabelle als Haupttabelle deklariert haben. Sie können also die Anweisung:

```
[My Table]My Field:="Ein String" ` Korrekt
```

nicht schreiben als:

```
DEFAULT TABLE([My Table])
My Field:="Ein String" ` FALSCH
```

In Tabellenmethoden, Formularen und Objekten, die zu einer Tabelle gehören, können Sie dagegen bei Feldnamen den Tabellennamen weglassen, wenn Sie vorher eine Haupttabelle deklariert haben.

In 4D sind alle Tabellen "offen" und einsatzbereit. Mit **DEFAULT TABLE** können Sie weder eine Tabelle öffnen, für die Ein- oder Ausgabe vorbereiten bzw. eine aktuelle Tabelle festlegen. **DEFAULT TABLE** dient lediglich dazu, das Eintippen zu verringern und den Code leichter lesbar zu machen.

Beispiel

Folgendes Beispiel zeigt den Code zuerst ohne **DEFAULT TABLE** und dann mit **DEFAULT TABLE**. Diese Schleife wird oft eingesetzt, um neue Datensätze in einer Datenbank hinzuzufügen. Die Befehle **FORM SET INPUT** und **ADD RECORD** benötigen beide eine Tabelle als ersten Parameter:

```
FORM SET INPUT([Customers];"Datensätze hinzufügen")
Repeat
   ADD RECORD([Customers])
Until(OK=0)
```

Mit Definieren der Haupttabelle schreiben Sie folgendermaßen:

```
DEFAULT TABLE([Customers])
FORM SET INPUT("Datensätze hinzufügen")
Repeat
   ADD RECORD
Until(OK=0)
```

NO DEFAULT TABLE

NO DEFAULT TABLE
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **NO DEFAULT TABLE** hebt die Auswirkung des Befehls **DEFAULT TABLE** auf. Nach seiner Ausführung ist für den Prozess keine Standardtabelle mehr definiert.
Dieser Befehl ist nur wirksam, wenn zuvor der Befehl **DEFAULT TABLE** aufgerufen wurde.

Dieser Befehl betrifft die Verwendung von Projektformularen, also Formularen, die nicht mit einer Tabelle verknüpft sind: Die meisten Befehle für Formulare (außer Benutzerformulare) erlauben als 1. Parameter einen optionalen Parameter *Tabelle*. Das gilt zum Beispiel für die Routinen **FORM GET PARAMETER**, **Open form window** oder **DIALOG**. Da ein Projektformular und ein Tabellenformular denselben Namen haben können, kann dieser Parameter zum Bestimmen des zu verwendenden Formulars dienen: Sie übergeben den Parameter *Tabelle*, wenn das Ziel ein Tabellenformular ist und lassen ihn weg, wenn das Ziel ein Projektformular ist.

In einer Datenbank mit einem Projektformular mit Namen "Formular" und einem Tabellenformular mit demselben Namen für die Tabelle [Table1] schreiben Sie:

```
DIALOG([Table1];"Formular")  `4D verwendet das Tabellenformular
DIALOG("Formular")  `4D verwendet das Projektformular
```

Dieses Prinzip greift jedoch nicht bei Ausführen des Befehls **DEFAULT TABLE**, wenn die Datenbank ein Projektformular und ein Tabellenformular mit demselben Namen enthält. In diesem Fall verwendet 4D standardmäßig das Tabellenformular, selbst wenn der Parameter *Tabelle* nicht übergeben wurde. Mit dem Befehl **NO DEFAULT TABLE** können Sie sicherstellen, dass ein Projektformular verwendet wird.

Beispiel

In einer Datenbank mit einem Projektformular mit Namen "Formular" und einem Tabellenformular mit demselben Namen für die Tabelle [Table1] schreiben Sie:

```
DEFAULT TABLE([Table1])
DIALOG("Formular")  `4D verwendet das Tabellenformular
NO DEFAULT TABLE
DIALOG("Formular")  `4D verwendet das Projektformular
```

Temporäre Auswahl

Einführung in temporäre Auswahl

Temporäre Auswahlen sind eine einfache Möglichkeit, mehrere Auswahlen gleichzeitig zu verwalten. Eine temporäre Auswahl ist eine sortierte Liste von Datensätzen für eine Tabelle in einem Prozess. Sie können dieser Liste einen Namen geben und sie speichern. So können Sie die aktuelle Auswahl mit der richtigen Sortierfolge und dem aktuellen Datensatz erhalten und später wiederverwenden.

Sie können temporäre Auswahlen mit folgenden Befehlen verwenden:

- **COPY NAMED SELECTION**
- **CUT NAMED SELECTION**
- **USE NAMED SELECTION**
- **CLEAR NAMED SELECTION**
- **CREATE SELECTION FROM ARRAY**

Sie erzeugen eine temporäre Auswahl mit den Befehlen **COPY NAMED SELECTION**, **CUT NAMED SELECTION** und **CREATE SELECTION FROM ARRAY**. Temporäre Auswahlen dienen in der Regel dazu, um in einer oder mehreren Auswahlen zu arbeiten, sie zu sichern und später eine sortierte Auswahl wiederherzustellen. Jede Tabelle in einem Prozess kann mehrere temporäre Auswahlen haben.

Eine temporäre Auswahl wird im Hauptspeicher erzeugt und kann jederzeit über den Befehl **USE NAMED SELECTION** wieder geladen werden. Mit dem Befehl **CLEAR NAMED SELECTION** geben Sie den belegten Speicherplatz wieder frei. Im Gegensatz zur Menge ist die temporäre Liste sortiert.

Hinweis: Die Kombination der Anweisung **SET QUERY DESTINATION**(Into named selection;namedselection) mit einem Suchbefehl, z.B. **QUERY**, lässt sich zum Erstellen einer temporären Auswahl verwenden. Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **SET QUERY DESTINATION**.

Temporäre Auswahlen sind auf lokaler, Prozess- oder Interprozessebene möglich.

Eine temporäre Auswahl ist lokal, wenn dem Namen das Dollarzeichen ($) vorangestellt ist. Ohne vorangestelltes Zeichen ist es eine temporäre Auswahl auf Prozessebene, mit den Zeichen größer als und kleiner als (<>) eine temporäre Auswahl auf Interprozessebene.

Eine temporäre Auswahl auf Interprozess-Ebene hat dieselbe Reichweite wie eine Interprozess-Variable. Sie können von jedem Prozess aus darauf zugreifen.
Mit 4D im Remote Modus und 4D Server ist eine temporäre Auswahl auf Interprozess-Ebene nur für Prozesse des Clients verfügbar, die sie erstellt hat. Sie ist für andere Client-Rechner nicht verfügbar.

Hinweis: Diese Schreibweise gilt für Windows und Mac OS. Auf Mac OS können Sie auch das Diamond-Zeichen (**Wahl- + Umschalttaste + v**) verwenden.

Eine temporäre Auswahl auf Prozessebene ist in dem Prozess, der sie erstellt hat, und auf dem Server verfügbar.

Eine lokale temporäre Auswahl ist nur für den Prozess verfügbar, der sie erstellt hat, und auf dem Server nicht sichtbar.

Warnung: Zum Erstellen einer temporären Auswahl ist der Zugriff auf die Auswahl der Tabelle erforderlich. Da Auswahlen auf dem Server liegen und ein lokaler Prozess nicht auf Server-Daten zugreifen kann, sollten Sie in in lokalen Prozessen keine temporären Auswahlen verwenden.

Sichtbarkeit von Mengen

Nachfolgende Tabelle zeigt, wie temporäre Auswahlen sichtbar sind, je nachdem, wo sie erstellt wurden:

Sichtbarkeit temporärer Auswahlen und Mengen					
	Client Prozess	Andere Client Prozesse	Andere Clients	Server Prozess	Andere Server Prozesse
Erstellung in einem Client Prozess					
$test	x				
test	x			x (Trigger)	
<>test	x	x			
Erstellung in einem Server Prozess					
$test				x	
test				x	
<>test				x	x

Temporäre Auswahl und Mengen

Es gibt folgende Unterschiede:

- Eine temporäre Auswahl enthält die Sortierung, eine Menge nicht.
- Eine Menge lässt sich auf der Festplatte speichern, eine temporäre Auswahl nicht.
- Mengen können Vereinigungs-, Schnitt- oder Differenzmengen bilden, temporäre Auswahlen nicht.
- Eine temporäre Auswahl lässt sich nicht mit anderen temporären Auswahlen kombinieren.
- Mengen benötigen weniger Speicherplatz, da sie nur 1 Bit Speicherplatz pro Datensatz in der Tabelle belegen, temporäre Auswahlen dagegen belegen 4 Bytes pro Datensatz in der Auswahl.

Es gibt folgende Übereinstimmungen:

- Mengen und temporäre Auswahlen werden im Arbeitsspeicher gehalten.
- Die Lebensdauer einer Menge und einer temporären Auswahl ist beschränkt. Wird ein Datensatz geändert oder gelöscht, sind sie nicht mehr aktuell.

- Der aktuelle Datensatz des laufenden Prozesses bleibt erhalten.

CLEAR NAMED SELECTION

CLEAR NAMED SELECTION (Auswahlname)

Parameter	Typ		Beschreibung
Auswahlname	String	➡	Name für temporäre Auswahl

Beschreibung

CLEAR NAMED SELECTION löscht *Auswahlname* aus dem Speicher und gibt den von *Auswahlname* belegten Speicherplatz wieder frei. Der Befehl hat keine Auswirkung auf Tabellen, Auswahlen oder Datensätze. Da temporäre Auswahlen Speicher beanspruchen, empfiehlt es sich, temporäre Auswahlen, die nicht länger benötigt werden, zu löschen.

Wurde *Auswahlname* mit **CUT NAMED SELECTION** erstellt und dann mit **USE NAMED SELECTION** bearbeitet, existiert *Auswahlname* nicht länger im Speicher. In diesem Fall muss **CLEAR NAMED SELECTION** nicht verwendet werden.

COPY NAMED SELECTION

COPY NAMED SELECTION ({Tabellenname ;} Auswahlname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Auswahl kopiert wird, ohne Angabe Haupttabelle
Auswahlname	String	➡	Name für die temporäre Auswahl

Beschreibung

Der Befehl **COPY NAMED SELECTION** kopiert die aktuelle Auswahl von *Tabellenname* in die temporäre Auswahl *Auswahlname*. *Tabellenname* ist optional. Wird der Parameter nicht angegeben, bezieht sich **COPY NAMED SELECTION** auf die Haupttabelle. Der Parameter *Auswahlname* enthält eine Kopie der Auswahl. Die aktuelle Auswahl und der aktuelle Datensatz von *Tabellenname* bleiben bei dieser Operation unverändert.

Eine temporäre Auswahl enthält nicht die Datensätze ansich, sondern nur eine sortierte Liste der Referenzen auf die Datensätze. Jede dieser Referenzen beansprucht 4 Bytes im Speicher. Beim Kopieren einer Auswahl mit dem Befehl **COPY NAMED SELECTION** werden demzufolge als Speicherplatz 4 Bytes, multipliziert mit der Anzahl der Datensätze in der Auswahl benötigt. Da die temporäre Auswahl im Cache-Speicher gehalten wird, sollte genügend Speicher für die temporäre und für die aktuelle Auswahl der Tabelle vorhanden sein. Bei zu vollem Speicher führt 4D diesen Befehl nicht aus, sondern zeigt den Fehler -108 an.

Verwenden Sie den Befehl **CLEAR NAMED SELECTION**, um den *von Auswahlname* verwendeten Speicher wieder freizugeben.

Beispiel

Sie wollen prüfen, ob in der Tabelle [People] fällige Rechnungen vorhanden sind. Die Auswahl wird sortiert und dann gesichert. Wir suchen nach allen Datensätzen mit fälligen Rechnungen. Dann verwenden wir die Auswahl erneut und löschen die temporäre Auswahl im Speicher. Das Löschen ist optional, falls der Designer der Datenbank die gewählte Auswahl zur späteren Verwendung im Speicher lassen will:

```
ALL RECORDS([People])
  `Erlaube dem Benutzer, die Auswahl zu sortieren
ORDER BY([People])
  ` Sichere die sortierte Auswahl als temporäre Auswahl
COPY NAMED SELECTION([People];"UserSort")
  ` Suche nach Datensätzen mit fälligen Rechnungen
QUERY([People];[People]InvoiceDue=True)
  ` Werden Datensätze gefunden
If(Records in selection([People])>0)
  ` Den Benutzer informieren
   ALERT("Ja, es gibt fällige Rechnungen in der Tabelle.")
End if
  ` Sortierte temporäre Auswahl wiederverwenden
USE NAMED SELECTION("UserSort")
  ` Temporäre Auswahl aus dem Speicher entfernen
CLEAR NAMED SELECTION("UserSort")
```

CUT NAMED SELECTION

CUT NAMED SELECTION ({Tabellenname ;} Auswahlname)

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Tabelle, deren Auswahl ausgeschnitten wird Ohne Angabe Haupttabelle
Auswahlname	String	➡	Name für die temporäre Auswahl

Beschreibung

Dieser Befehl bezieht sich auf die aktuelle Auswahl des laufenden Prozesses.
CUT NAMED SELECTION übernimmt die aktuelle Auswahl. Danach ist die aktuelle Auswahl leer.

Tabellenname ist optional. Wird der Parameter nicht angegeben, bezieht sich **CUT NAMED SELECTION** auf die Haupttabelle. Der Name für die temporäre Auswahl wird im zweiten Parameter übergeben.

Im Gegensatz zum Befehl **COPY NAMED SELECTION** wird der Speicherplatz durch diesen Befehl nicht verändert, da die bisherige aktuelle Auswahl übernommen, keine zusätzliche Auswahl erzeugt und der belegte Speicherplatz freigegeben wird. Dadurch ist der Befehl sehr schnell und wird in der Praxis häufiger als **COPY NAMED SELECTION** benutzt.
Benutzen Sie **CUT NAMED SELECTION** nicht beim Ändern eines Datensatzes, da nach seiner Ausführung die Auswahl leer ist, außer Sie erzeugen unmittelbar danach wieder eine Auswahl.

Beispiel

Nachfolgende Methode leert die aktuelle Auswahl der Tabelle [Customers]:

```
CUT NAMED SELECTION([Customers];"ToBeCleared")
CLEAR NAMED SELECTION("ToBeCleared")
```

USE NAMED SELECTION

USE NAMED SELECTION (Auswahlname)

Parameter	Typ		Beschreibung
Auswahlname	String	➡	Name für temporäre Auswahl

Beschreibung

Der Befehl **USE NAMED SELECTION** verwendet die in *Auswahlname* übergebene temporäre Auswahl als die aktuelle Auswahl für die dazugehörige Tabelle.

Beim Erstellen einer temporären Auswahl merkt sich 4D den aktuellen Datensatz. **USE NAMED SELECTION** findet die Position dieses Datensatzes und macht den Datensatz zum neuen aktuellen Datensatz; dieser Befehl lädt also den aktuellen Datensatz. Wurde der aktuelle Datensatz nach Erstellen von *Auswahlname* geändert, sollten Sie ihn sichern, bevor **USE NAMED SELECTION** ausgeführt wird, damit auch die Änderungen berücksichtigt werden.

Die Ausführung hängt davon ab, wie die temporäre Auswahl erzeugt wurde:

- Durch den Befehl **CUT NAMED SELECTION**
  Die temporäre Auswahl ersetzt die aktuelle Auswahl der angegebenen Tabelle. Nach der Ausführung ist *Auswahlname* gelöscht.
- Durch den Befehl **COPY NAMED SELECTION**
  Die temporäre Auswahl ersetzt die aktuelle Auswahl der angegebenen Tabelle. Nach der Ausführung bleibt *Auswahlname* bestehen. Es enthält immer noch die gleiche Anzahl von Elementen. Mit dem Befehl **CLEAR NAMED SELECTION** geben Sie den belegten Speicherplatz wieder frei.

Gibt es kein *Auswahlname*, erhalten Sie den Fehler -9977.
Auswahlname ist die exakte Darstellung einer aktuellen Auswahl zu einem bestimmten Zeitpunkt in Form einer Adresstabelle. Haben Sie zwischen ihrer Erzeugung und dem Augenblick ihrer Benutzung Datensätze gelöscht oder neu angelegt, ist *Auswahlname* veraltet. Arbeiten Sie trotzdem mit *Auswahlname*, kann die Datenintegrität beeinträchtigt werden.

Transaktionen

- Transaktionen verwenden
- Transaktionen anhalten
- Active transaction
- CANCEL TRANSACTION
- In transaction
- RESUME TRANSACTION
- START TRANSACTION
- SUSPEND TRANSACTION
- Transaction level
- VALIDATE TRANSACTION

Transaktionen verwenden

Eine Transaktion führt hintereinander Operationen aus, die Sie jederzeit komplett wieder rückgängig machen können. Die logische Datenintegrität wird dabei gewahrt. Eine Transaktion wird erst in der Datenbank gesichert, wenn die Transaktion bestätigt wird. Änderungen werden nicht gesichert, wenn eine Transaktion nicht abgeschlossen ist, weil sie entweder abgebrochen oder durch ein von außen kommendes Ereignis unterbrochen wird.

Während einer Transaktion werden alle Änderungen in der Datenbank innerhalb eines Prozesses lokal in einem temporären Speicher gesichert. Wird die Transaktion mit **VALIDATE TRANSACTION** bestätigt, werden die Änderungen dauerhaft gesichert. Wird die Transaktion mit **CANCEL TRANSACTION** abgebrochen, werden die Änderungen nicht gesichert. Befehle zum Steuern von Transaktionen verändern weder die aktuelle Auswahl noch den aktuellen Datensatz.

4D unterstützt verschachtelte Transaktionen, d.h. Transaktionen auf verschiedenen hierarchischen Ebenen. Die Anzahl der erlaubten Untertransaktionen ist unbegrenzt. Über die Funktion **Transaction level** können Sie die aktuelle Transaktionsebene finden, auf welcher der Code ausgeführt wird.

Bei Verwendung verschachtelter Transaktionen hängt das Ergebnis der Untertransaktion davon ab, ob die höher gelegene Transaktion annulliert oder bestätigt wurde. Bei Bestätigung werden die Ergebnisse der Untertransaktionen bestätigt. Bei Annullierung werden alle dazugehörigen Untertransaktionen annulliert, unabhängig von ihrem Ergebnis.

Hinweis: Zur Wahrung der Kompatibilität werden verschachtelte Transaktionen in Anwendungen, die aus Versionen älter als v11 konvertiert wurden, standardmäßig deaktiviert. Weitere Informationen dazu finden Sie im Abschnitt **Seite Kompatibilität**.

4D bietet ein Feature, um Transaktionen in Ihrem 4D Code anzuhalten und zu reaktivieren. Ist eine Transaktion *angehalten*, können Sie Operationen unabhängig von der Transaktion selbst ausführen und dann die Transaktion *reaktivieren*, um sie wie gewohnt zu bestätigen oder abzubrechen. Ist eine Transaktion angehalten, können Sie folgendes ausführen:

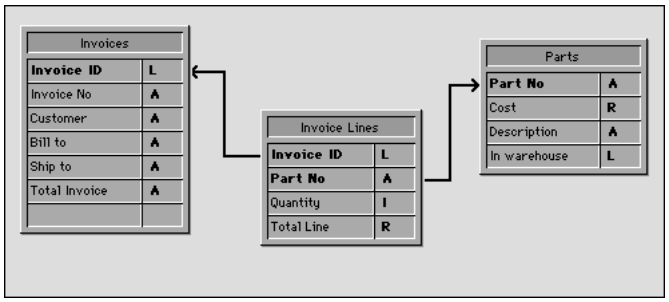
- Datensätze außerhalb der Transaktion anlegen oder ändern, z.B. um den Zähler der Rechnungsnummer zu erhöhen
- Andere Transaktionen starten, anhalten bzw. schließen

Weitere Informationen dazu finden Sie im Abschnitt **Transaktionen anhalten**.

Beispiele für Transaktionen

In diesem Beispiel ist die Datenbank ein einfaches Rechnungssystem. Die Rechnungszeilen werden in einer Tabelle [Invoice Lines] gespeichert, die über die Felder [Invoices]Invoice ID und [Invoice Lines]Invoice ID mit der Tabelle [Invoices] verknüpft ist. Beim Hinzufügen einer Rechnung wird mit der Funktion **Sequence number** eine einmalige Kennung berechnet. Die Verknüpfung zwischen [Invoices] und [Invoice Lines] ist eine automatische Eine-zu-Viele Verknüpfung. Im Dialogfenster Verknüpfungseigenschaften ist das Ankreuzfeld Automatische Zuweisung des verknüpften Wertes aktiviert.

Die Verknüpfung zwischen [Invoice Lines] und [Parts] ist manuell.



Gibt ein Benutzer eine Rechnung ein, werden folgende Schritte ausgeführt:

- In der Tabelle [Invoices] wird ein Datensatz hinzugefügt.
- In der Tabelle [Invoice Lines] werden mehrere Datensätze hinzugefügt.
- Das Datenfeld [Parts]In Warehouse wird für jeden in der Rechnung aufgeführten Artikel aktualisiert.

Dieses Beispiel ist eine typische Situation für eine Transaktion. Sie müssen sicher sein, dass Sie all diese Datensätze während der Operation sichern bzw. die Transaktion abbrechen können, wenn sich ein Datensatz nicht hinzufügen oder aktualisieren lässt. Mit anderen Worten, Sie müssen verknüpfte Daten sichern. Ohne Transaktion ist die logische Datenintegrität Ihrer Datenbank nicht gewährleistet. Ist zum Beispiel ein Datensatz in der Tabelle [Parts] gesperrt, kann die Anzahl im Datenfeld [Parts]In Warehouse nicht aktualisiert werden. Dieses Feld ist also logisch nicht mehr korrekt. Die Summe der verkauften Artikel und der Lagerbestand sind nicht identisch mit der im Datensatz eingegebenen Anzahl. Mit einer Transaktion verhindern Sie solch eine Situation.

Es gibt verschiedene Arten, bei der Dateneingabe mit Transaktionen zu arbeiten:

1. Sie verwalten die Transaktion selbst. Dazu setzen Sie die Befehle **START TRANSACTION**, **VALIDATE TRANSACTION** und **CANCEL TRANSACTION** ein. Wir schreiben beispielsweise:

```
READ WRITE([Invoice Lines])
READ WRITE([Parts])
FORM SET INPUT([Invoices];"Input")
Repeat
   START TRANSACTION
   ADD RECORD([Invoices])
   If(OK=1)
```

```
        VALIDATE TRANSACTION
    Else
        CANCEL TRANSACTION
    End if
Until(OK=0)
READ ONLY(*)
```

2. Um das Datensatzsperren während der Dateneingabe zu reduzieren, können Sie auch Transaktionen von der Formularmethode aus steuern und auf die Tabellen nur wenn notwendig, im **LESE/SCHREIBMODUS** zugreifen.

Sie führen die Dateneingabe im Eingabeformular für [Invoices] aus, die die verknüpfte Tabelle [Invoice Lines] in einem Unterformular einhält. Das Formular hat zwei Schaltflächen: *bCancel* und *bOK*, beide führen keine Aktion aus. Die Schleife zum Hinzufügen lautet:

```
READ WRITE([Invoice Lines])
READ ONLY([Parts])
FORM SET INPUT([Invoices];"Input")
Repeat
    ADD RECORD([Invoices])
Until(bOK=0)
READ ONLY([Invoice Lines])
```

Beachten Sie, dass die Tabelle [Parts] nun während der Dateneingabe im Modus **Nur Lesen** ist. Der Zugriff im Lese-/Schreibmodus ist nur verfügbar, wenn die Dateneingabe bestätigt wird.

Die Transaktion wird in folgender [Invoices] Eingabeformularmethode gestartet:

```
Case of
    :(Form event=On Load)
        START TRANSACTION
        [Invoices]Invoice ID:=Sequence number([Invoices]Invoice ID)
    Else
        [Invoices]Total Invoice:=Sum([Invoice Lines]Total line)
End case
```

Klicken Sie auf die Schaltfläche *bCancel*, muss sowohl die Dateneingabe als auch die Transaktion annulliert werden. Die Objektmethode für die Schaltfläche *bCancel* lautet:

```
Case of
    :(Form event=On Clicked)
        CANCEL TRANSACTION
        CANCEL
End case
```

Klicken Sie auf die Schaltfläche *bValidate*, muss die Dateneingabe angenommen und die Transaktion bestätigt werden. Die Objektmethode für die Schaltfläche *bOK* lautet:

```
Case of
    :(Form event=On Clicked)
        $NbLines:=Records in selection([Invoice Lines])
        READ WRITE([Parts])  ` Wechsle in den Lese/Schreibzugriff für die Tabelle [Parts]
        FIRST RECORD([Invoice Lines])  ` Starte mit der ersten Zeile
        $ValidTrans:=True  ` Nimm an, alles ist OK
        For($Line;1;$NbLines)  ` Für jede Zeile
            RELATE ONE([Invoice Lines]Part No)
            OK:=1  ` Nimm an, fortzufahren
            While(Locked([Parts]) & (OK=1))
 ` Versuche, den Datensatz im Lese/Schreibzugriff zu erhalten
                CONFIRM("Der Artikel "+[Invoice Lines]Part No+" wird benutzt. Warten?")
                If(OK=1)
                    DELAY PROCESS(Current process;60)
                    LOAD RECORD([Parts])
                End if
            End while
            If(OK=1)
 ` Aktualisiere Anzahl im Lager
                [Parts]In Warehouse:=[Parts]In Warehouse-[Invoice Lines]Quantity
                SAVE RECORD([Parts])  ` Sichere Datensatz
            Else
                $Line:=$NbLines+1  `Verlasse die Schleife
                $ValidTrans:=False
            End if
            NEXT RECORD([Invoice Lines])  ` Gehe in nächste Zeile
        End for
```

```
        READ ONLY([Parts]) ` Setze Tabelle in Status Nur Lesen
        If($ValidTrans)
            SAVE RECORD([Invoices]) ` Sichere Datensatz in Rechnungen
            VALIDATE TRANSACTION ` Bestätige alle Änderungen in der Datenbank
        Else
            CANCEL TRANSACTION ` Brich Transaktion ab
        End if
        CANCEL ` Verlasse Formular
    End case
```

In diesem Code rufen wir den Befehl **CANCEL** unabhängig von der Schaltfläche auf. Der neue Datensatz wird nicht durch Aufrufen von **ACCEPT** bestätigt, sondern durch den Befehl **SAVE RECORD**. Beachten Sie, dass zusätzlich **SAVE RECORD** unmittelbar vor dem Befehl **VALIDATE TRANSACTION** aufgerufen wird. So ist das Sichern des [Invoices] Datensatzes aktueller Teil der Transaktion. Das Aufrufen des Befehls **ACCEPT** bestätigt auch den Datensatz, in diesem Fall würde die Transaktion ebenfalls den Datensatz bestätigen, die Transaktion würde jedoch vor Sichern des [Invoices] Datensatzes bestätigt. Demnach würde der Datensatz außerhalb der Transaktion bestätigt.

Bei der Dateneingabe können Sie die Transaktionen entweder von 4D verwalten lassen oder - wie in den obigen Beispielen gezeigt - in Ihrer Datenbank individuell einstellen. Das Steuern von gesperrten Datensätzen im letzten Beispiel kann noch weiter entwickelt werden.

Transaktionen anhalten

Überblick

Eine Transaktion anhalten ist nützlich zum Durchführen von Operationen innerhalb einer Transaktion, für die keine Steuerung durch die Transaktion erforderlich ist.
Hierzu ein Beispiel: Innerhalb einer Transaktion macht ein Kunde eine Bestellung und aktualisiert auch seine Adresse. Dann ändert er seine Meinung und annulliert die Bestellung. Die Transaktion wird abgebrochen, aber die Adressänderung soll nicht rückgängig gemacht werden. Hier ist es hilfreich, die Transaktion anzuhalten. Es gibt drei Befehle zum Anhalten und Reaktivieren von Transaktionen:

- **SUSPEND TRANSACTION** hält die aktuelle Transaktion an. Alle aktualisierten und hinzugefügten Datensätze bleiben gesperrt.
- **RESUME TRANSACTION** reaktiviert die angehaltene Transaktion.
- **Active transaction** gibt Falsch zurück, wenn die Transaktion angehalten oder keine aktuelle Transaktion vorhanden ist. Wahr, wenn sie gestartet oder reaktiviert ist.

Beispiel

Hier ein Anwendungsbeispiel für eine angehaltene Transaktion. In einer Anwendung mit Rechnungen wollen wir während einer Transaktion eine neue Rechnungsnummer erhalten. Diese Nummer wird berechnet und in der Tabelle [Settings] gespeichert. Bei mehreren Benutzern müssen miteinander konkurrierende Zugriffe geschützt werden. Wegen der Transaktion kann jedoch die Tabelle [Settings] durch einen anderen Benutzer gesperrt sein, obwohl diese Daten von der Haupttransaktion unabhängig sind. Für diesen Fall können Sie die Transaktion beim Zugreifen auf die Tabelle anhalten:

```
 //Standardmethode zum Erstellen einer Rechnung
START TRANSACTION
...
CREATE RECORD([Invoices])
[Invoices]InvoiceID:=GetInvoiceNum //Die Methode aufrufen, um eine verfügbare Nummer zu erhalten.
...
SAVE RECORD([Invoices])
VALIDATE TRANSACTION
```

Die Methode ***GetInvoiceNum*** hält die Transaktion vor der Ausführung an. Beachten Sie, dass dieser Code auch funktioniert, wenn die Methode außerhalb einer Transaktion aufgerufen wird:

```
 //Projektmethode GetInvoiceNum
 //GetInvoiceNum -> Nächste verfügbare Rechnungsnummer
C_LONGINT($0)
SUSPEND TRANSACTION
ALL RECORDS([Settings])
If(Locked([Settings])) //Zugriff mehrerer Benutzer
   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
```

Operationen im Detail

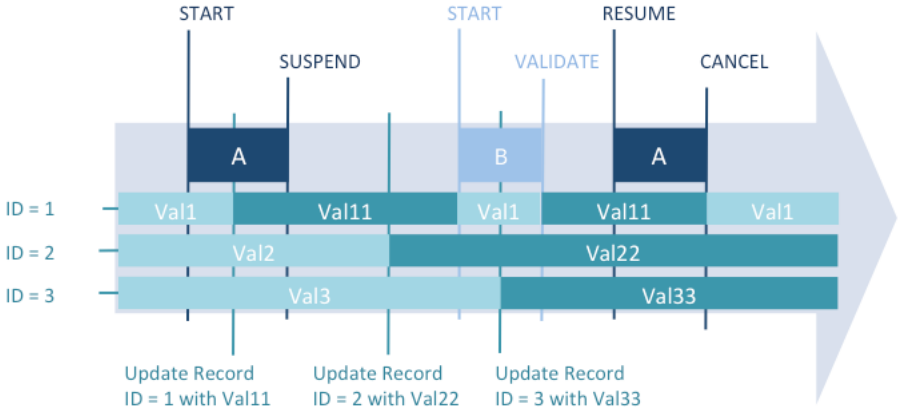
Wie funktioniert eine angehaltene Transaktion?

Ist eine Transaktion angehalten, gilt folgendes:

- In der Transaktion angelegt, geänderte oder gelöschte Datensätze behalten diesen Status bei.
- Sie können Datensätze außerhalb der Transaktion anlegen, ändern oder löschen.
- Sie können eine neue Transaktion starten, aber innerhalb dieser Transaktion keine Datensätze oder Datensatzwerte sehen, die in der angehaltenen Transaktion hinzugefügt oder geändert wurden. Diese neue Transaktion ist in der Tat vollkommen unabhängig von der angehaltenen, ähnlich wie eine Transaktion in einem anderen Prozess. Da die angehaltene Transaktion später reaktiviert und abgebrochen werden kann, sind die hinzugefügten, geänderten oder gelöschten Datensätze für die neue Transaktion automatisch ausgeblendet. Sobald Sie die neue Transaktion beenden oder abbrechen, können Sie diese wieder sehen.

- Alle Datensätze, die innerhalb der angehaltenen Transaktion geändert, gelöscht oder hinzugefügt werden, bleiben für die anderen Prozesse gesperrt. Versuchen Sie, diese außerhalb der Transaktion zu ändern oder zu löschen, wird ein Fehler erzeugt.

Folgende Grafik gibt einen Überblick:



Während Transaktion A bearbeitete Werte (Datensatz ID1 erhält Val11) sind nicht in der neuen Transaktion B verfügbar, die während der "angehaltenen" Zeit angelegt wird. Während der "angehaltenen" Zeit bearbeitete Werte (Datensatz ID2 erhält Val22 und Datensatz ID3 erhält Val33) werden gesichert, selbst nachdem Transaktion A abgebrochen ist.

Es gibt spezifische Features zur Fehlerverwaltung:

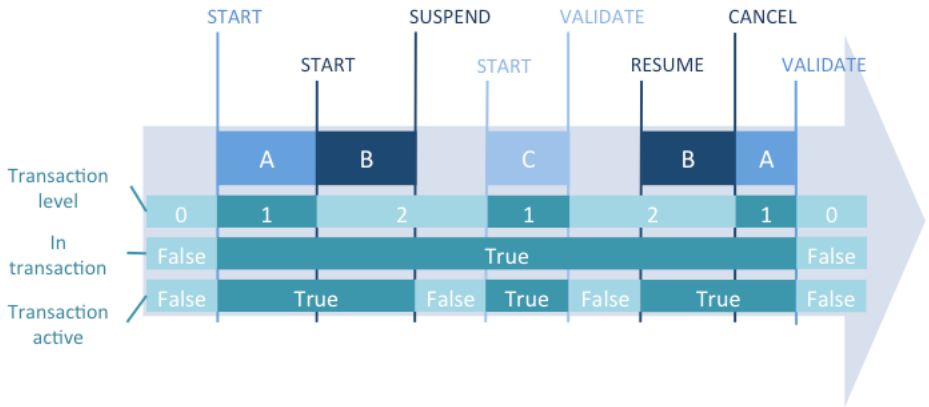
- Der aktuelle Datensatz jeder Tabelle wird temporär gesperrt, wenn er während der Transaktion geändert wird, und beim Reaktivieren der Transaktion automatisch entsperrt. Dieser Mechanismus ist wichtig, um ungewolltes Sichern auf Teile der Transaktion zu verhindern.
- Führen Sie eine ungültige Sequenz aus, wie *start transaction / suspend transaction / start transaction / resume transaction*, wird ein Fehler generiert. Auf diese Weise werden Entwickler darauf hingewiesen, dass sie eine eingefügte Transaktion beenden oder abbrechen müssen, ehe sie reaktiviert wird.

Angehaltene Transaktionen und Prozesstatus

Die vorhandene Funktion **In transaction** gibt **Wahr** zurück, wenn eine Transaktion gestartet wurde, auch wenn sie angehalten ist. Über die neue Funktion **Transaction active** können Sie herausfinden, ob die aktuelle Transaktion angehalten ist. Sie gibt in diesem Fall **Falsch** zurück.

Dagegen geben beide Befehle **Falsch** zurück, wenn keine Transaktion gestartet wurde. Sie müssen dann u.U. die vorhandene Funktion **Transaction level** verwenden, die in diesem Kontext 0 zurückgibt.

Nachfolgende Grafik zeigt die verschiedenen Kontexte für Transaktionen und die jeweils von den Befehlen zurückgegebenen Werte:



Active transaction

Active transaction -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Gibt Falsch zurück, wenn die aktuelle Transaktion angehalten ist

Beschreibung

Die Funktion **Active transaction** gibt **Wahr** zurück, wenn der aktuelle Prozess in Transaktion ist und die Transaktion nicht angehalten ist. Sie gibt **Falsch** zurück, wenn es keine aktuelle Transaktion gibt oder diese angehalten ist. Eine Transaktion lässt sich mit dem Befehl **SUSPEND TRANSACTION** anhalten.

Da die Funktion auch **Falsch** zurückgibt, wenn der aktuelle Prozess nicht in Transaktion ist, müssen Sie u.U. über die Funktion **In transaction** prüfen, ob der Prozess in Transaktion ist.

Weitere Informationen dazu finden Sie im Abschnitt **Transaktionen anhalten**.

Beschreibung

Den Status der aktuellen Transaktion abfragen:

```
If(In transaction)
    If(Not(Active transaction))
        ALERT("Die aktuelle Transaktion ist angehalten")
    Else
        ALERT("Die aktuelle Transaktion ist aktiv")
    End if
Else
    ALERT("Wir sind nicht in Transaktion")
End if
```

CANCEL TRANSACTION

CANCEL TRANSACTION
Dieser Befehl benötigt keine Parameter

Beschreibung

CANCEL TRANSACTION bricht die Transaktion auf der entsprechenden Ebene im aktuellen Prozess ab, die mit **START TRANSACTION** gestartet wurde. **CANCEL TRANSACTION** annulliert alle in den Daten ausgeführten Operationen, die während der Transaktion gespeichert wurden.

Hinweis: CANCEL TRANSACTION hat keine Auswirkung auf Änderungen in aktuellen Datensätzen, die nicht gesichert wurden - sie bleiben nach Ausführung des Befehls unverändert erhalten.

In transaction

In transaction -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		Gibt TRUE zurück, wenn der aktuelle Prozess in Transaktion ist.

Beschreibung

Die Funktion **In transaction** gibt den Wert *Wahr* zurück, wenn der aktuelle Prozess in einer Transaktion ist, sonst den Wert *Falsch*.

Beispiel

Führen Sie Operationen mit mehreren Datensätzen durch, wie Datensätze hinzufügen, ändern oder löschen, stoßen Sie unter Umständen auf gesperrte Datensätze. Müssen Sie in diesem Fall die Datenintegrität wahren, sollten Sie innerhalb der Transaktion sein, damit Sie die ganze Operation zurückverfolgen und die Datenbank unverändert verlassen können.

Führen Sie eine Operation von einem Trigger oder einer Unterroutine (die in Transaktion oder nicht in Transaktion aufgerufen werden kann) aus durch, können Sie mit der Funktion **In transaction** prüfen, ob die aktuelle Prozessmethode oder die aufrufende Methode eine Transaktion gestartet hat. Wurde keine Transaktion gestartet, brauchen Sie die Operation erst gar nicht zu starten, da Sie bereits wissen, dass Sie diese bei Fehlern nicht zurückverfolgen können.

RESUME TRANSACTION

RESUME TRANSACTION
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **RESUME TRANSACTION** reaktiviert die Transaktion der entsprechenden Ebene im aktuellen Prozess, die zuvor über **SUSPEND TRANSACTION** angehalten wurde. Nach diesem Befehl ausgeführte Operationen erfolgen unter Transaktionskontrolle (außer wenn mehrere suspendierte Transaktionen eingebunden sind).

Weitere Informationen dazu finden Sie im Abschnitt **Transaktionen anhalten**.

START TRANSACTION

START TRANSACTION
Dieser Befehl benötigt keine Parameter

Beschreibung

START TRANSACTION startet eine Transaktion im aktuellen Prozess. Alle während der Transaktion gemachten Änderungen an den Datensätzen werden temporär gespeichert, bis die Transaktion bestätigt oder abgebrochen wird.

Ab 4D Version 11 sind auch verschachtelte Transaktionen möglich. Jede Haupt- oder Untertransaktion muss u.U. einzeln bestätigt oder abgebrochen werden. Durch Annullieren der Haupttransaktion werden auch alle Untertransaktionen abgebrochen, unabhängig von ihrem Ergebnis.

SUSPEND TRANSACTION

SUSPEND TRANSACTION
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **SUSPEND TRANSACTION** hält die aktuelle Transaktion im aktuellen Prozess an. Sie können dann z.B. Daten in anderen Teilen der Anwendung außerhalb der Transaktion bearbeiten, wobei jedoch der Kontext der Transaktion beibehalten wird. Alle Datensätze, die in der Transaktion aktualisiert oder hinzugefügt wurden, sind gesperrt, bis die Transaktion über den Befehl **Versioning / Client** reaktiviert wird.

Weitere Informationen dazu finden Sie im Abschnitt **Transaktionen anhalten**.

Transaction level

Transaction level -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl	➲	Aktuelle Transaktionsebene (0 wenn keine Transaktion gestartet wurde)

Beschreibung

Die Funktion **Transaction level** gibt die aktuelle Transaktionsebene für den Prozess zurück. Sie berücksichtigt alle Transaktionen des aktuellen Prozesses, unabhängig ob sie über die 4D Programmiersprache oder über SQL gestartet wurden.

VALIDATE TRANSACTION

VALIDATE TRANSACTION
Dieser Befehl benötigt keine Parameter

Beschreibung

VALIDATE TRANSACTION bestätigt die mit **START TRANSACTION** begonnene Transaktion auf der entsprechenden Ebene im aktuellen Prozess. 4D aktualisiert mit den Änderungen, die im Hilfsbereich abgelegt wurden, die Datensätze der Datenbank und sichert sie auf der Festplatte.

Ab 4D Version 11 sind auch verschachtelte Transaktionen möglich. Durch Annullieren der Haupttransaktion werden auch alle Untertransaktionen abgebrochen. Das gilt auch, wenn diese einzeln über **VALIDATE TRANSACTION** bestätigt wurden.

Systemvariablen und Mengen

Die Systemvariable OK wird auf 1 gesetzt, wenn die Transaktion korrekt bestätigt wurde; sonst wird sie auf 0 (Null) gesetzt.

Beachten Sie, dass die Transaktion automatisch intern abgebrochen wird, wenn OK auf 0 gesetzt wird (entspricht **CANCEL TRANSACTION**). Folglich dürfen Sie nicht explizit **CANCEL TRANSACTION** aufrufen, wenn OK=0, insbesondere bei eingebundenen Transaktionen, da Abbrechen dann auf dieTransaktion auf höherer Ebene angewandt wird.

Trigger

Einführung in Trigger

Ein Trigger ist eine Methode, die einer Tabelle zugeordnet ist. Er ist eine Tabelleneigenschaft. Sie rufen Trigger nicht auf; die Engine der 4D Datenbank ruft sie automatisch auf, wenn Sie Datensätze der Tabelle bearbeiten, also Datensätze hinzufügen, löschen, ändern und laden. Sie können ganz einfache Trigger schreiben und sie dann komplexer machen.

Trigger sind ein leistungsstarkes Werkzeug, denn sie unterbinden "illegale" Operationen auf Datensätze Ihrer Datenbank. Sie lassen nur bestimmte Operationen für eine Tabelle zu und verhindern, dass Daten versehentlich verloren gehen oder beschädigt werden. Ein Trigger sorgt zum Beispiel in einem Rechnungssystem dafür, dass der Benutzer eine Rechnung nur hinzufügen kann, wenn er auch den Kunden einträgt, an den die Rechnung gestellt wird.

Hinweis: 4D bietet ein breites Spektrum an Sicherheitsvorkehrungen. Weitere Informationen dazu finden Sie im 4D Security guide.

Trigger aktivieren und einstellen

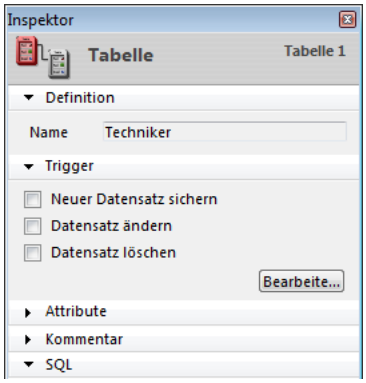
Erstellen Sie in der Designumgebung eine Tabelle, hat sie standardmäßig keinen Trigger.
Wollen Sie für eine Tabelle einen Trigger verwenden, müssen Sie:

- Den Trigger aktivieren und 4D anweisen, dass er aufgerufen werden soll.
- Den Code für den Trigger schreiben.

Aktivieren Sie einen Trigger, der noch nicht geschrieben wurde bzw. schreiben Sie einen Trigger, ohne ihn zu aktivieren, hat er keine Auswirkung auf die Operationen der Tabelle.

1. Trigger aktivieren

Wählen Sie dazu im Inspektorfenster der Struktur unter **Trigger** eine Option (Datenbankereignisse) für die Tabelle:



Datensatz ändern

Mit dieser Option wird der Trigger immer aufgerufen, wenn in der Tabelle ein Datensatz geändert wird.
Dieser Fall tritt ein, wenn Sie:

- Einen Datensatz in Dateneingabe ändern (Designumgebung, über den Befehl **MODIFY RECORD** oder den SQL Befehl **UPDATE**).
- Einen bereits vorhandenen Datensatz mit **SAVE RECORD** sichern.
- Andere Befehle aufrufen, die vorhandene Datensätze sichern, z.B. **ARRAY TO SELECTION**, **APPLY TO SELECTION**, etc.).
- Ein Plug-In einsetzen, das den Befehl **SAVE RECORD** aufruft.

Hinweis: Zur Optimierung wird der Trigger nicht aufgerufen, wenn der Datensatz vom Benutzer oder über den Befehl **SAVE RECORD** gesichert wird und in der Tabelle kein Feld im Datensatz geändert wurde. Wollen Sie den Aufruf des Triggers in diesem Fall erzwingen, können Sie einfach ein Feld auf sich selbst zuweisen:

```
[thetable]thefield:=[thetable]thefield
```

Datensatz löschen

Mit dieser Option wird der Trigger immer aufgerufen, wenn in der Tabelle ein Datensatz gelöscht wird.
Dieser Fall tritt ein, wenn Sie:

- Einen Datensatz löschen (Designumgebung, über **DELETE RECORD**, **DELETE SELECTION** oder den SQL Befehl **DELETE**).
- Eine Operation ausführen, die verknüpfte Datensätze über die Löschkontrollen einer Verknüpfung löscht.
- Ein Plug-In einsetzen, das den Befehl **DELETE RECORD** aufruft.

Hinweis: Der Befehl **TRUNCATE TABLE** ruft keinen Trigger auf!

Wichtig: Führen Sie eine Operation bzw. einen Befehl für mehrere Datensätze aus, wird der Trigger für jeden Datensatz aufgerufen. Rufen Sie zum Beispiel den Befehl **APPLY TO SELECTION** für eine Tabelle auf mit einer aktuellen Auswahl von 100 Datensätzen, wird der Trigger 100 Mal aufgerufen.

Neuer Datensatz sichern

Mit dieser Option wird der Trigger immer aufgerufen, wenn in der Tabelle ein Datensatz hinzugefügt wird.
Dieser Fall tritt ein, wenn Sie:

- Einen Datensatz in Dateneingabe hinzufügen (Designumgebung, über den Befehl **ADD RECORD** oder den SQL Befehl **INSERT**).
- Einen Datensatz mit **CREATE RECORD** und **SAVE RECORD** erstellen und sichern. Beachten Sie, dass der Trigger beim Aufrufen von **SAVE RECORD** und nicht beim Erstellen aktiv wird.
- Datensätze importieren (Designumgebung oder über einen Importbefehl).
- Andere Befehle aufrufen, die neue Datensätze erstellen und/oder sichern, z.B. **ARRAY TO SELECTION**, **SAVE RELATED ONE**, etc..
- Ein Plug-In einsetzen, das die Befehle **CREATE RECORD** und **SAVE RECORD** aufruft.

2. Trigger erstellen

Einen Trigger für eine Tabelle erstellen Sie entweder über das Fenster **Explorer**, oder Sie klicken im Inspektorfenster auf die Schaltfläche **Bearbeiten** oder doppelklicken bei gedrückter **alt-Taste** unter Windows bzw. **Wahltaste** auf Macintosh auf den Tabellentitel im Strukturfenster. Weitere Informationen dazu finden Sie im Abschnitt **Trigger** des Handbuchs *4D Designmodus*.

Datenbankereignisse

Ein Trigger kann für eines der oben beschriebenen **Datenbankereignisse** ausgelöst werden. Mit der Funktion **Trigger event** prüfen Sie das Ereignis innerhalb des Triggers. Diese Funktion gibt für das jeweilige Datenbankereignis eine Zahl zurück.

Sie schreiben einen Trigger mit der Struktur **Case of** auf das von **Trigger event** zurückgegebene Ergebnis. Sie können die Konstanten unter dem Thema **_o_LAST SUBRECORD** verwenden:

```
// Trigger für [beliebigeTabelle]
C_LONGINT($0)
$0:=0 // Eine Datenbankanfrage wird ausgelöst
Case of
   :(Trigger event=On Saving New Record Event)
// Geeignete Aktionen zum Sichern eines neu angelegten Datensatzes ausführen
   :(Trigger event=On Saving Existing Record Event)
// Geeignete Aktionen zum Sichern eines bereits vorhandenen Datensatzes ausführen
   :(Trigger event=On Deleting Record Event)
// Geeignete Aktionen zum Löschen eines Datensatzes ausführen
End case
```

Trigger sind Funktionen

Ein Trigger hat zwei grundlegende Ziele:

- Aktionen an einem Datensatz ausführen, bevor dieser gesichert oder gelöscht wird.
- Eine Operation der Datenbank bestätigen oder zurückweisen.

1. Aktionen ausführen

Sie verwenden einen Trigger, um damit einen Datensatz zu prägen, sobald er gesichert oder geändert wird bzw. Informationen an eine andere Tabelle weitergibt. Der folgende Programmcode prägt z.B. einen Datensatz mit der Objektmethode Zeitstempel:

```
// Trigger für Tabelle [Dokumente]
Case of
   :(Trigger event=On Saving New Record Event)
      [Dokumente]Erstellt:=Time stamp
      [Dokumente]Geändert:=Time stamp
   :(Trigger event=On Saving Existing Record Event)
      [Dokumente]Geändert:=Time stamp
End case
```

Hinweis: *Time stamp* gibt die Anzahl Sekunden zurück, die seit einem willkürlich festgelegten Datum vergangen sind.

Haben Sie diesen Trigger geschrieben und aktiviert, spielt es keine Rolle, ob Sie einen Datensatz über Dateneingabe, Import, eine Projektmethode oder ein 4D Plug-In hinzufügen bzw. ändern. Der Trigger prägt automatisch die Felder *[Dokumente]Erstellt* und *[Dokumente]Geändert*, bevor der Datensatz auf die Festplatte geschrieben wird.

Hinweis: Dieses Beispiel wird unter dem Befehl **GET DOCUMENT PROPERTIES** ausführlich beschrieben.

2. Operation der Datenbank bestätigen oder zurückweisen

Um eine Datenbankoperation zu bestätigen oder zurückzuweisen, muss der Trigger im Funktionsergebnis *$0* einen **Trigger Fehlercode** zurückgeben.

Beispiel

Wir gehen aus von einer Tabelle *[Angestellte]*. Dafür gilt die Regel, dass sämtliche Datensätze eine gültige Sozialversicherungsnummer enthalten müssen. Dafür weisen Sie dem Feld*[Angestellte]SV Nummer* die Objektmethode Gute SVNr zu. Diese überprüft die Eingaben der Sozialversicherungsnummer, wenn Sie auf die Schaltfläche **Bestätigen** klicken:

```
// Objektmethode für Schaltfläche bAccept
If(Good SS number([Employees]SS number))
   ACCEPT
```

```
Else
   BEEP
   ALERT("Sozialversicherungsnummer eingeben und dann erneut auf OK klicken.")
End if
```

Ist die Sozialversicherungsnummer korrekt, bestätigen Sie die Dateneingabe; ist sie falsch, erscheint eine Meldung. Sie bleiben in der Dateneingabe.

Sie können die Datensätze in *[Employees]* auch per Programmierung erstellen. Der folgende Teil eines Code wäre zwar von der Programmierung her gültig, würde aber gegen die Regel der vorigen Objektmethode verstoßen:

```
// Auszug aus einer Projektmethode
// ...
CREATE RECORD([Employees])
[Employees]Name:="DOE"
SAVE RECORD([Employees])  // <-- DB Verstoß gegen die Regel! Die SV Nummer wurde nicht zugewiesen!
// ...
```

Mit einem Trigger für die Tabelle *[Employees]* können Sie die Regel *[Employees]SS number* auf allen Ebenen der Datenbank vorschreiben. Der Trigger sieht folgendermaßen aus:

```
// Trigger für [Employees]
$0:=0
$dbEvent:=Trigger event
Case of
   :(($dbEvent=On Saving New Record Event)|($dbEvent=On Saving Existing Record Event))
      If(Not(Good SS number([Employees]SS number)))
         $0:=-15050
      Else
// ...
      End if
// ...
End case
```

Sobald dieser Trigger geschrieben und aktiviert ist, erzeugt die Zeile **SAVE RECORD** ([Employees]) den Fehler -15050 der Datenbank-Engine und der Datensatz wird NICHT gesichert.

Dasselbe gilt, wenn ein 4D Plug-In versucht, einen Datensatz *[Employees]* mit einer ungültigen Sozialversicherungsnummer zu sichern. Der Trigger erzeugt denselben Fehler und der Datensatz wird NICHT gesichert.

Der Trigger garantiert, dass niemand bewusst oder versehentlich eine ungültige Sozialversicherungsnummer eingeben und sichern kann.

Beachten Sie, dass Sie auch ohne Trigger auf eine Tabelle beim Sichern bzw. Löschen eines Datensatzes Fehler der Datenbank-Engine erhalten können. Versuchen Sie zum Beispiel, einen Datensatz mit doppelten Wert in einem einmaligen indizierten Feld zu sichern, erhalten Sie den Fehler -9998.

Trigger, die Fehler zurückgeben, fügen demnach in Ihrer Anwendung neue Fehler der Datenbank-Engine hinzu:

- 4D verwaltet die "regulären" Fehler: Einmaliger Index, relationale Datenkontrolle, usw.
- Über Trigger verwalten Sie eigene Fehler speziell in Ihrer Anwendung.

Wichtig: Sie können einen beliebigen Wert für den Fehlercode zurückgeben. Verwenden Sie jedoch KEINE Fehlercodes, die bereits für die 4D Datenbank-Engine reserviert sind. Sie sollten Nummern zwischen -32000 und -15000 verwenden.

Trigger-Fehler auf Prozessebene behandeln Sie genauso wie Fehler der Datenbank-Engine:

- Sie können 4D das Standardfenster für Fehlermeldung anzeigen lassen, die Methode wird dann angehalten.
- Sie können eine mit **ON ERR CALL** eingebaute Fehlerverwaltungsmethode verwenden und den Fehler in geeigneter Weise ausfindig machen.

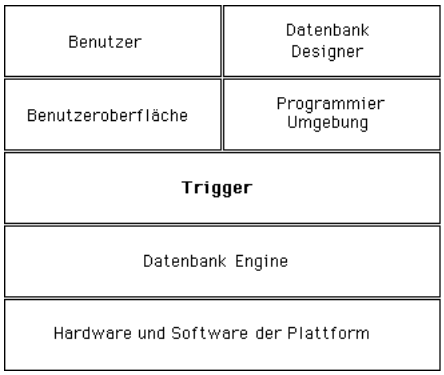
Hinweise:

- Wird während der Dateneingabe ein Trigger-Fehler beim Bestätigen bzw. Löschen eines Datensatzes zurückgegeben, wird der Fehler wie ein einmaliger indizierter Fehler behandelt. Der Fehlerdialog wird angezeigt und Sie bleiben weiter in der Dateneingabe. Auch wenn Sie eine Datenbank nur in der Designumgebung einsetzen und nicht in der Anwendungsumgebung, können Sie die Vorteile von Triggern nutzen.
- Erzeugt ein Trigger einen Fehler, während gerade ein Befehl auf eine Datensatzauswahl ausgeführt wird, z.B. **DELETE SELECTION**, stoppt diese Operation sofort, d.h. die Auswahl ist u.U. nicht vollständig bearbeitet worden. Solch ein Fall erfordert vom Entwickler eine geeignete Abwicklung, wie z.B. die temporäre Erhaltung der Auswahl, Bearbeiten und Beheben des Fehlers vor Ausführen des Triggers, o. ä.

Auch wenn ein Trigger keinen Fehler zurückgibt ($0:=0), heißt das nicht zwingend, dass die Datenbankoperation erfolgreich war – es kann ein Verstoß gegen den einmaligen Index eintreten. Handelt es sich bei der Operation um die Aktualisierung eines Datensatzes, kann dieser gesperrt sein, ein E/A Fehler kann auftreten, u.v.m.. Das Prüfen erfolgt nach Ausführung des Triggers. Auf höherer Ebene des ausführenden Prozesses dagegen sind die von der Datenbank-Engine oder einem Trigger zurückgegebenen Fehler dieselben – ein Trigger-Fehler ist ein Fehler der Datenbank-Engine.

Trigger und 4D Architektur

Trigger arbeiten auf der Ebene der Datenbank-Engine. Das zeigt folgendes Schema:



Trigger werden auf dem Rechner ausgeführt, auf dem die Datenbank-Engine liegt. Das ist im 4D Einzelplatzbetrieb offensichtlich. Im Mehrplatzbetrieb dagegen werden Trigger im ausführenden Prozess auf dem Server-Rechner ausgeführt (im Zwillingsprozess des Prozesses, der den Trigger ausgelöst hat), und nicht auf dem Client-Rechner .

Wird ein Trigger ausgelöst, läuft er im Kontext des Prozesses, der die Datenbankoperation versucht. Der Prozess, der die Trigger-Ausführung auslöst, heißt **auslösender Prozess**.
Die in diesem Kontext enthaltenen Elemente sind unterschiedlich, je nachdem ob die Datenbank mit 4D im lokalen Modus oder mit 4D Server ausgeführt wird:

- Mit 4D im lokalen Modus arbeiten Trigger mit aktuellen Auswahlen, aktuellen Datensätzen, Tabellen im Lese-/Schreibmodus und Operationen zum Sperren des Datensatzes des auslösenden Prozesses.
- Mit 4D Server wird nur der Kontext der Datenbank des auslösenden Client Prozesses beibehalten (gesperrte Datensätze und Stadien der Transaktion) 4D Server und nur dieser, garantiert auch, dass der aktuelle Datensatz der Trigger-Tabelle korrekt gesetzt wird. Die anderen Elemente dieses Kontexts, z.B. aktuelle Auswahlen, gehören zum Trigger Prozess.

Verwenden Sie andere Objekte der Datenbank oder der Programmiersprache der 4D Umgebung mit Bedacht, da ein Trigger auf einem anderen Rechner als dem, der den Prozess auslöst, ausgeführt werden kann. Das ist mit 4D Server der Fall!

- **Interprozessvariablen**: Ein Trigger hat Zugriff auf die Interprozessvariablen des Rechners, auf dem er ausgeführt wird. Mit 4D Server kann er auch auf einen anderen Rechner, als dem mit dem auslösenden Prozess zugreifen.
- **Prozessvariablen**: Jeder Trigger hat seine eigene Tabelle mit den Prozessvariablen. Ein Trigger hat keinen Zugriff auf die Prozessvariablen des auslösenden Prozesses.
- **Lokale Variablen**: Sie können in einem Trigger lokale Variablen verwenden. Sie gelten während der Ausführung des Triggers; sie werden für jede Ausführung erstellt und anschließend wieder gelöscht.
- **Semaphoren**: Ein Trigger kann globale Semaphoren testen und setzen, und zwar auf dem Rechner, wo er ausgeführt wird. Ein Trigger muss jedoch schnell ablaufen. Deshalb ist Vorsicht geboten, wenn Sie Semaphoren von Triggern aus testen oder setzen wollen.
- **Mengen und temporäre Auswahlen**: Verwenden oder setzen Sie eine Menge bzw. eine temporäre Auswahl von Triggern heraus, arbeiten Sie auf dem Rechner, von welchem die Trigger ausgeführt werden. Im Client/Server Modus sind auf dem Client-Rechner erstellte Prozessmengen und temporäre Auswahlen (den Namen ist weder $ noch <> vorangestellt) in einem Trigger sichtbar.
- **Benutzeroberfläche**: Verwenden Sie in einem Trigger KEINE Elemente der Benutzeroberfläche, wie z.B. Warnungen, Meldungen und Dialogboxen. Analog dazu sollten Sie auch den Schrittmodus für Trigger im **Debugging** Fenster ausgrenzen. Denn Trigger im Client/Server-Betrieb laufen auf dem Server-Rechner. Eine Meldung auf dem Server-Rechner hilft dem Benutzer auf dem Client-Rechner nicht. Deshalb sollte der aufrufende Prozess die Benutzeroberfläche verwalten.

Beachten Sie, dass Sie bei Verwendung des 4D Kennwortsystems den Befehl **Current user** im Trigger ausführen können, um z.B. den Benutzernamen am Ursprung des Trigger-Aufrufs in einer Journal-Tabelle zu sichern, inkl. im Client/Server-Modus.

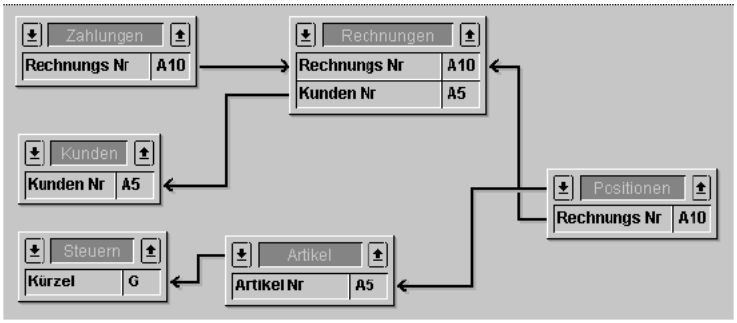
Trigger und Transaktionen

Sie müssen Transaktionen auf der Ebene des auslösenden Prozesses verwalten und nicht auf der Trigger-Ebene. Wollen Sie während Ausführung eines Triggers mehrere Datensätze hinzufügen, ändern oder löschen, müssen Sie zuerst mit der Funktion **In transaction** innerhalb des Triggers prüfen, ob der auslösende Prozess derzeit in Transaktion ist. Ist das nicht der Fall, findet der Trigger höchstwahrscheinlich einen gesperrten Datensatz vor. Es macht also keinen Sinn, eine Operation auf Datensätze zu starten, wenn der auslösende Prozess nicht in Transaktion ist. Geben Sie in $0 einen Fehler zurück, der dem auslösenden Prozess signalisiert, dass die angestrebte Datenbankoperation in Transaktion ausgeführt werden muss. Sonst kann der auslösende Prozess, wenn er auf gesperrte Datensätze stößt, die Aktionen des Triggers nicht zurückfahren.

Hinweis: 4D ruft nach der Ausführung von **VALIDATE TRANSACTION** keine Trigger auf. Das optimiert das Zusammenspiel von Triggern und Transaktionen, denn so werden Trigger nicht zweimal aufgerufen.

Trigger verschachteln

Wir gehen von folgender Beispielstruktur aus:



Hinweis: Die Tabellen sind verkürzt; sie haben natürlich mehr Felder als hier gezeigt.

Wir nehmen an, dass die Datenbank das Löschen einer Rechnung zulässt. So können wir nachvollziehen, wie solch eine Operation auf Trigger-Ebene verwaltet wird (Löschen ist auf Prozessebene möglich).

Zur Wahrung der relationalen Integrität der Daten müssen beim Löschen einer Rechnung im Trigger für [Rechnungen] folgende Aktionen ablaufen:

- Im Datensatz [Kunden] den Wert im Feld Gesamtumsatz um den Rechnungsbetrag kürzen.
- Alle mit der Rechnung verknüpften Datensätze [Positionen] löschen.
- Daraus ergibt sich, dass der Trigger zu [Positionen] den Wert im Feld Anzahl verkauft des Datensatzes [Artikel] verringert, der zum zu löschenden Artikel gehört.
- Alle mit der gelöschten Rechnung verknüpften Datensätze [Zahlungen] löschen.

1. Der Trigger für [Rechnungen] muss diese Aktionen nur ausführen, wenn der auslösende Prozess in Transaktion ist, so dass bei gesperrten Datensätzen ein Zurücksetzen möglich ist.

2. Der Trigger für [Positionen] ist mit dem Trigger für [Rechnungen] verschachtelt. Der Trigger [Positionen] wird innerhalb des Triggers [Rechnungen] ausgeführt. Denn durch Aufrufen von **DELETE SELECTION** im Trigger [Rechnungen] erfolgt das Löschen der Positionen.

Wir gehen davon aus, dass für alle Tabellen dieses Beispiels Trigger für alle Datenbankereignisse aktiviert sind. Die Trigger sind dann folgendermaßen verschachtelt:

- Trigger [Rechnungen] wird ausgelöst, da der auslösende Prozess eine Rechnung löscht
  - Trigger [Kunden] wird ausgelöst, da der Trigger [Rechnungen] das Feld Gesamtumsatz aktualisiert
  - Trigger [Positionen] wird ausgelöst, da der Trigger [Rechnungen] einen Artikel löscht (wiederholt)
    - Trigger [Artikel] wird ausgelöst, da der Trigger [Positionen] das Feld Anzahl verkauft aktualisiert
  - Trigger [Zahlungen] wird ausgelöst, da der Trigger [Rechnungen] eine Zahlung löscht (wiederholt)

In dieser Verschachtelung läuft der Trigger [Rechnungen] auf Ebene 1 ab, die Trigger [Kunden], [Positionen] und [Zahlungen] auf Ebene 2 und der Trigger [Artikel] auf Ebene 3.

Innerhalb der Trigger können Sie mit der Funktion **Trigger level** prüfen, auf welcher Ebene der Trigger abläuft. Mit dem Befehl **TRIGGER PROPERTIES** erhalten Sie Informationen über die anderen Ebenen.

Wird zum Beispiel ein Datensatz [Artikel] auf Prozessebene gelöscht, wird der Trigger [Artikel] auf Ebene 1, nicht auf Ebene 3 ausgeführt.

Mit **Trigger level** und **TRIGGER PROPERTIES** können Sie den Grund für eine Aktion prüfen. In unserem Beispiel wird eine Rechnung auf Prozessebene gelöscht. Löschen wir einen Datensatz [Kunden] auf Prozessebene, sollte der Trigger [Kunden] versuchen, alle mit diesem Kunden verknüpften Rechnungen zu löschen. Das heißt, dass der Trigger [Rechnungen] – wie oben beschrieben – ausgelöst wird, jedoch aus einem anderen Grund. Innerhalb des Triggers [Rechnungen] können Sie prüfen, ob er auf Ebene 1 oder 2 abgelaufen ist. Lief er auf Ebene 2 ab, können Sie prüfen, ob er aktiv wurde, weil der Datensatz [Kunden] gelöscht wurde. In diesem Fall erübrigt sich die Aktualisierung des Feldes Gesamtumsatz.

Trigger event

Trigger event -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Lange Ganzzahl	0=Außerhalb jeder Ausführungsphase eines Triggers, 1=Neuen Datensatz sichern, 2=Vorhandenen Datensatz sichern, 3=Datensatz löschen

Beschreibung

Rufen Sie in einem Trigger die Funktion **Trigger event** auf, gibt sie eine Zahl zurück, die den Typ des Datenbankereignisses spezifiziert. d.h. Sie erfahren, weshalb der Trigger ausgelöst wurde.

Folgende vordefinierten Konstanten unter dem Thema **Trigger Ereignisse** stehen zur Verfügung:

Konstante	Typ	Wert
On Deleting Record Event	Lange Ganzzahl	3
On Saving Existing Record Event	Lange Ganzzahl	2
On Saving New Record Event	Lange Ganzzahl	1

Führen Sie in einem Trigger Datenbankoperationen auf mehrere Datensätze aus, können Sie Umstände vorfinden – normalerweise gesperrte Datensätze – die die korrekte Ausführung des Triggers verhindern.
Beispiel: In einer Tabelle [Produkte] werden mehrere Datensätze aktualisiert, während in der Tabelle [Rechnungen] ein Datensatz hinzugefügt wird. An dieser Stelle müssen Sie die Datenbankoperation abbrechen und einen Fehler der Datenbank zurückgeben, so dass der auslösende Prozess weiß, dass diese Operation nicht durchführbar ist. Der auslösende Prozess muss in der Lage sein, die Datenbankoperation, die vom Trigger während der Transaktion unvollständig durchgeführt wird, abzubrechen. In solch einem Fall müssen Sie – bevor Sie versuchen, irgendetwas auszuführen – im Trigger wissen, ob Sie in Transaktion sind. Das testen Sie mit der Funktion **In transaction**.

Beim Verschachteln von Triggeraufrufen, wird 4D nur durch den verfügbaren Speicherplatz beschränkt. Zum Optimieren der Triggerausführung schreiben Sie den Code Ihrer Trigger nicht nur in Bezug auf die Datenbankereignisse, sondern auch in Bezug auf die jeweilige Ebene bei verschachtelten Triggern.
Beispiel: Sie wollen bei einem Datenbankereignis **On delete** für die Tabelle [Rechnungen] die Aktualisierung des Feldes [Kunden] Gesamtumsatz ausschalten, wenn das Löschen des Datensatzes [Rechnungen] bedeutet, dass **alle** Rechnungen gelöscht werden, die mit einem Datensatz [Kunden] verknüpft sind. Dafür verwenden Sie **Trigger level** und **TRIGGER PROPERTIES**.

Beispiel

Mit der Funktion **Trigger event** strukturieren Sie Ihre Trigger wie folgt:

```
// Trigger für [beliebigeTabelle]
C_LONGINT($0)
$0:=0 ` Die Datenbankanfrage wird angenommen
Case of
   :(Trigger event=On Saving New Record Event)
// Führt die Aktionen aus, um einen neuen Datensatz zu sichern
   :(Trigger event=On Saving Existing Record Event)
// Führt die Aktionen aus, um einen bestehenden Datensatz zu sichern
   :(Trigger event=On Deleting Record Event)
// Führt die Aktionen aus, um einen Datensatz zu löschen
End case
```

Trigger level

Trigger level -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Ausführungsebene der Trigger (0 wenn außerhalb der Ausführungsphase eines Triggers)

Beschreibung

Die Funktion **Trigger level** gibt die Ausführungsebene des Triggers zurück.
Weitere Informationen zu den Ausführungsebenen finden Sie im Abschnitt **Trigger verschachteln**.

TRIGGER PROPERTIES

TRIGGER PROPERTIES (TriggerEbene ; DatenbankEreignis ; TabelleNum ; DatensatzNum)

Parameter	Typ		Beschreibung
TriggerEbene	Lange Ganzzahl	➡	Ausführungsebene des Triggers
DatenbankEreignis	Lange Ganzzahl	⬅	Datenbankereignis
TabelleNum	Lange Ganzzahl	⬅	Betroffene Tabellennummer
DatensatzNum	Lange Ganzzahl	⬅	Betroffene Datensatznummer

Beschreibung

Der Befehl **TRIGGER PROPERTIES** gibt Information über die Ausführungsebene des Triggers in *TriggerEbene* zurück. Sie verwenden diesen Befehl in Verbindung mit **Trigger level**, um je nach Verschachtelung der Ausführungsebenen des Triggers verschiedene Aktionen auszuführen. Weitere Informationen dazu finden Sie im Abschnitt **Einführung in Trigger**.

Übergeben Sie eine nicht vorhandene Ausführungsebene des Triggers, gibt der Befehl in allen Parametern 0 (Null) zurück.

Die Art des Datenbankereignisses für die Ausführungsebene des Triggers wird in *DatenbankEreignis* zurückgegeben.

Folgende vordefinierten Konstanten unter dem Thema **Trigger Ereignisse** stehen zur Verfügung:

Konstante	Typ	Wert
On Deleting Record Event	Lange Ganzzahl	3
On Saving Existing Record Event	Lange Ganzzahl	2
On Saving New Record Event	Lange Ganzzahl	1

Tabellen- und Datensatznummer für den Datensatz, der vom Datenbankereignis für die Ausführungsebene des Triggers betroffen ist, werden in *TabelleNum* und *DatensatzNum* zurückgegeben.

Unterbrechungen

- 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
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **ABORT** wird innerhalb einer Projektmethode zur Fehlerverwaltung verwendet, die mit dem Befehl **ON ERR CALL** installiert wurde.

Gibt es keine Projektmethode zur Fehlerverwaltung, wenn ein Fehler auftritt (zum Beispiel, ein Fehler der Datenbank-Engine), zeigt 4D den Standarddialog für Fehler und unterbricht dann die Ausführung Ihres Code. Ist der ausführende Code:

- Eine Objektmethode, Formularmethode (bzw. eine Projektmethode, die von einem Formular oder einer Objektmethode aufgerufen wird), kehrt die Steuerung zum gerade angezeigten Formular zurück.
- Eine Methode, die von einem Menü aufgerufen wird, kehrt die Steuerung zur gerade angezeigten Menüleiste bzw. dem gerade angezeigten Formular zurück.
- Die Hauptmethode eines Prozesses, endet der Prozess.
- Eine Methode, die direkt oder indirekt über eine Import- oder Export-Operation ausgeführt wird, bricht die Operation ab. Dasselbe gilt für sequentielle Suchen oder Anweisungen über Operationen.
- usw...

Verwenden Sie zur Fehlersuche eine Projektmethode zur Fehlerverwaltung, zeigt 4D weder den Standarddialog für Fehler an noch unterbricht es die Ausführung Ihres Code. Stattdessen ruft 4D Ihre Fehlermethode auf (eine Ausnahme-Behandlungsroutine) und setzt die Ausführung auf die nächste Codezeile der Methode, die den Fehler auslöst.

Diese Fehler können Sie per Programmierung bearbeiten; stellen Sie z.B. während einer Importoperation einen Fehler wie doppelter Datensatz der Datenbank-Engine fest, können Sie den Fehler vor dem Anwender verstecken und mit dem Import fortfahren. Es gibt jedoch auch Fehler, die Sie nicht per Programmierung bearbeiten bzw. "verdecken" können. In diesen Fällen müssen Sie die Ausführung abbrechen, indem Sie in der Projektmethode zur Fehlerverwaltung den Befehl **ABORT** aufrufen.

Historischer Hinweis

Der Befehl **ABORT** soll nur in Projektmethoden zur Fehlerverwaltung verwendet werden. Einige 4D Entwickler unterbrechen jedoch damit auch die Ausführung anderer Projektmethoden. Dass dies funktioniert, ist nur ein Nebeneffekt. Wir raten davon ab, diesen Befehl außerhalb von Fehlerverwaltungsmethoden zu verwenden.

ASSERT

ASSERT (boolAusdruck {; MeldungText})

Parameter	Typ		Beschreibung
boolAusdruck	Boolean	➡	Boolean Ausdruck
MeldungText	Text	➡	Text der Fehlermeldung

Beschreibung

Der Befehl **ASSERT** bewertet die im Parameter *boolAusdruck* übergebene Assertion und stoppt die Ausführung des Code mit einer Fehlermeldung, wenn *boolAusdruck* falsch zurückgibt. Der Befehl arbeitet im interpretierten und kompilierten Modus.

Ist *boolAusdruck* wahr, passiert nichts. Ist er falsch, löst der Befehl den Fehler -10518 aus und zeigt standardmäßig den Text der Assertion mit dem Anfang "Assertion fehlgeschlagen:" Sie können diesen Fehler mit einer Methode abfangen, die über den Befehl **ON ERR CALL** installiert wird, um z.B. Informationen zu einem Logbuch zu liefern.

Optional können Sie den Parameter *MeldungText* übergeben, um anstelle des Textes der Assertion eine eigene Fehlermeldung anzuzeigen.

Eine Assertion ist eine in den Code eingefügte Anweisung, die zum Herausfinden von Unstimmigkeiten während der Code-Ausführung dient. Sie überprüft, ob eine Anweisung zu einem gegebenen Moment wahr ist und verursacht, wenn das Gegenteil eintritt, eine Ausnahme. Assertionen dienen vorallem dazu, Fälle herauszufinden, die in der Regel gar nicht auftreten sollten. Sie werden hauptsächlich zum Herausfinden von Programmierfehlern verwendet. Über den Befehl **SET ASSERT ENABLED** können Sie alle Assertionen einer Anwendung (z.B. gemäß dem Versionstyp) global aktivieren oder deaktivieren.

Weitere Informationen zu Assertionen in der Programmierung finden Sie in Wikipedia unter http://en.wikipedia.org/wiki/Assertion_(computing).

Beispiel 1

Der Entwickler möchte vor Ausführen von Operationen auf einen Datensatz sicherstellen, dass er aktuell im Lese-/Schreibmodus geladen ist:

```
READ WRITE([Table 1])
LOAD RECORD([Table 1])
ASSERT(Not(Locked([Table 1])))
  // Löst Fehler -10518 aus, wenn der Datensatz gesperrt ist
```

Beispiel 2

Über eine Assertion können in einer Projektmethode übergebene Parameter auf ungültige Werte getestet werden. In diesem Beispiel wird eine eigene Meldung verwendet:

```
  // Methode, welche die Kundennummer gemäß dem in $1 übergebenen Kundennamen zurückgibt.
C_TEXT($1)  // Kundenname
ASSERT($1#"";"Suche nach einem leeren Kundennamen")
  // Ein leerer Name ist in diesem Fall ein abwegiger Wert
  // Bei falscher Assertion erscheint im Fehlerdialogfenster folgendes:
  // "Assertion fehlgeschlagen: Suche nach leerem Kundennamen"
```

Asserted

Asserted (boolExpression {; MeldungText}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
boolExpression	Boolean	➡	Boolean Ausdruck
MeldungText	Text	➡	Text der Fehlermeldung
Funktionsergebnis	Boolean	⮌	Ergebnis der Bewertung von BoolExpression

Beschreibung

Die Funktion **Asserted** arbeitet ähnlich wie der Befehl **ASSERT** mit dem Unterschied, dass sie einen Wert zurückgibt, der sich aus der Bewertung des Parameters *boolExpression* ergibt. Sie ermöglicht so, während der Bewertung einer Bedingung eine Assertion zu verwenden (siehe Beispiel). Weitere Informationen zur Funktionsweise von Assertionen und den Parametern dieses Befehls finden Sie in der Beschreibung von **ASSERT**.

Asserted akzeptiert einen Boolean Ausdruck als Parameter und gibt das Ergebnis der Bewertung dieses Ausdrucks zurück. Ist der Ausdruck falsch und sind die Assertionen aktivert (siehe Befehl **SET ASSERT ENABLED**), wird der Fehler -10518 generiert, wie beim Befehl **ASSERT**. Sind die Assertionen deaktiviert, gibt **Asserted** das Ergebnis des Ausdrucks zurück, das ohne Auslösen eines Fehlers übergeben wurde.

Hinweis: Wie **ASSERT** arbeitet auch **Asserted** im interpretierten und kompilierten Modus.

Beispiel

Eine Assertion in die Bewertung eines Ausdrucks einfügen:

```
READ WRITE([Table 1])
LOAD RECORD([Table 1])
If(Asserted(Not(Locked([Table 1]))))
  // Dieser Code löst den Fehler -10518 aus, wenn der Datensatz gesperrt ist.
    ...
End if
```

FILTER EVENT

FILTER EVENT
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **FILTER EVENT** wird innerhalb einer Projektmethode zur Ereignisverwaltung aufgerufen, die mit **ON EVENT CALL** installiert wurde.

Ruft eine Ereignisverwaltungsmethode **FILTER EVENT** auf, wird das aktuelle Ereignis nicht an 4D übergeben.

Dieser Befehl entfernt das aktuelle Ereignis aus der Schleife, z.B. Mausklick, Anschlag auf der Tastatur, so dass 4D nur das Ereignis in der Projektmethode zur Ereignisverwaltung bearbeitet.

Warnung: Erstellen Sie keine Ereignisverwaltungsmethode, die nur den Befehl **FILTER EVENT** aufruft, da dann alle laufenden Ereignisse ignoriert werden. Tippen Sie in solch einem Fall die Tastenkombination **strg-/Umschalt-/Rückschrittaste** unter Windows, **Befehls-/Umschalt-/Rückschrittaste** auf Macintosh. Dies wandelt die Methode ***On Event Call*** um in einen normalen Prozess ohne Ereignisse.

Sonderfall: Sie können **FILTER EVENT** auch innerhalb einer Formularmethode für die Standardausgabe verwenden, wenn das Formular über die Befehle **DISPLAY SELECTION** oder **MODIFY SELECTION** angezeigt wird. In diesem Fall ermöglicht **FILTER EVENT** Doppelklicks auf die Datensätze zu filtern. So können Sie auch andere Aktionen als Öffnen der Datensätze im Seitenmodus ausführen.

Dazu fügen Sie Sie folgende Zeilen in die Formularmethode für die Ausgabe ein:

```
If(Form event=On Double Clicked)
   FILTER EVENT
   ... `Doppelklick abarbeiten
End if
```

Beispiel

Siehe Beispiel zum Befehl **ON EVENT CALL**.

Get assert enabled

Get assert enabled -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	Wahr = Assertionen sind aktiviert, Falsch = Assertionen sind deaktiviert

Beschreibung

Die Funktion **Get assert enabled** gibt Wahr oder Falsch zurück, je nachdem, ob Assertionen im aktuellen Prozess aktiviert sind oder nicht. Weitere Informationen dazu finden Sie unter dem Befehl **ASSERT**.

Assertionen sind standardmäßig aktiviert, außer sie wurden über den Befehl **SET ASSERT ENABLED** deaktiviert.

GET LAST ERROR STACK

GET LAST ERROR STACK (CodesArray ; IntCompArray ; TextArray)

Parameter	Typ		Beschreibung
CodesArray	Array Lange Ganzzahl	⇐	Fehlernummern
IntCompArray	Array String	⇐	Interner Code für Komponente
TextArray	Array String	⇐	Fehlermeldung

Beschreibung

Der Befehl **GET LAST ERROR STACK** gibt Informationen über den aktuellen Fehler "Stapel" der 4D Anwendung zurück. Verursacht eine 4D Anweisung einen Fehler, enthält der aktuelle Fehlerstapel eine Beschreibung des Fehlers, sowie weitere dadurch ausgelöste Fehler.
Beispiel: Der Fehler "Festplatte voll" verursacht einen Schreibfehler in der Datei und einen Fehler im Befehl zum Datensatz-Sichern: Der Stapel enthält deshalb drei Fehler. Hat die letzte 4D Anweisung keinen Fehler erzeugt, ist der aktuelle Fehlerstapel leer.

Dieser generische Befehl kann jeden auftretenden Fehlertyp bearbeiten.

Hinweis: Bei Fehlern, die von einer ODBC Quelle erzeugt werden, muss der Befehl **SQL GET LAST ERROR** verwendet werden.

GET LAST ERROR STACK muss in einer Fehlerverwaltungsmethode aufgerufen werden, die mit dem Befehl **ON ERR CALL** installiert wurde.

Die Information wird in drei aufeinander abgestimmten Arrays zurückgegeben:

- *CodesArray*: Dieses Array empfängt die Liste der erzeugten Fehlernummern.
- *IntCompArray*: Dieses Array enthält den Code der internen Komponenten, die jedem Fehler zugewiesen sind.
- *TextArray*: Dieses Array enthält den Text jedes Fehlers.

Die Liste der Fehlermeldungen finden Sie im Kapitel **Fehlermeldungen**.

Method called on error

Method called on error -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	String	Name der bei Ereignis aufgerufenen Methode

Beschreibung

Die Funktion **Method called on error** gibt den Namen der Methode zurück, die über den Befehl **ON ERR CALL** für den aktuellen Prozess installiert wurde.
Wurde keine Methode installiert, wird ein leerer String ("") zurückgegeben.

Beispiel

Diese Funktion ist besonders hilfreich im Zusammenhang mit Komponenten, da Sie damit die bei Fehler aufgerufenen Methoden zeitweise ändern und dann wiederherstellen können:

```
$methCurrent:=Method called on error
ON ERR CALL("NeueMethode")
  ` Kann das Dokument nicht geöffnet werden, wird ein Fehler generiert
$ref:=Open document("MeinDokument")
  ` Erneute Installation der vorigen Methode
ON ERR CALL($methCurrent)
```

Method called on event

Method called on event -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	String	Name der bei Ereignis aufgerufenen Methode

Beschreibung

Die Funktion **Method called on event** gibt den Namen der Methode zurück, die über den Befehl **ON EVENT CALL** installiert wurde.
Wurde keine Methode installiert, wird ein leerer String ("") zurückgegeben.

ON ERR CALL

ON ERR CALL (Methodenname)

Parameter	Typ		Beschreibung
Methodenname	String	➡	Aufzurufende Fehlermethode, oder Leerer String, um Fehlersuche zu stoppen

Beschreibung

Der Befehl **ON ERR CALL** legt eine Unterbrechungsmethode mit dem Namen *FehlerMethode* an. Sie wird jedes Mal ausgeführt, wenn ein Fehler bei der Ausführung von Befehlen aufgetreten ist. Diese Projektmethode heißt **Fehlerverwaltungsmethode** oder **Fehlerauffindmethode**.

Ist eine Fehlerverwaltungsmethode installiert, ruft 4D die Methode immer auf, wenn während der Ausführung eines 4D Befehls der Programmiersprache ein Fehler auftritt.

ON ERR CALL gilt für den aktuellen Prozess. Sie können pro Prozess immer nur eine Fehlerverwaltungsmethode haben.

Hinweise:

- Bei Verwendung von Komponenten gilt dieser Befehl nur für die aktuelle Datenbank. Erzeugt eine Komponente einen Fehler, wird *FehlerMethode* nicht in der Host-Datenbank aufgerufen.
- Wird **ON ERR CALL** von einem Prozess aufgerufen, der preemptive ausgeführt werden soll (in 64-bit kompilierter Modus), prüft der Compiler, ob *FehlerMethode* thread-safe ist und gibt Fehler zurück, wenn sie nicht mit dem preemptive Modus kompatibel ist. Weitere Informationen finden Sie im Abschnitt **Preemptive 4D Prozesse**.

Wollen Sie das Auffinden von Fehlern abbrechen, rufen Sie erneut **ON ERR CALL** auf und übergeben in *FehlerMethode* einen leeren String.

Den Fehlercode erhalten Sie mit Hilfe der Systemvariablen *Error*. Die Fehlerliste finden Sie im Kapitel **Fehlermeldungen**, siehe zum Beispiel Abschnitt **Syntaxfehler (1 -> 81)**. Der Wert der Systemvariablen *Error* ist nur in der Fehlerverwaltungsmethode signifikant; benötigen Sie den Fehlercode in der Methode, die den Fehler hervorgerufen hat, kopieren Sie *Error* in Ihre eigene Prozessvariable. Sie können auch auf die Systemvariablen *Error method, Error line* und *Error formula* zugreifen, die jeweils den Namen der Methode, die Zeilennummer und den Text der Formel enthalten, wo der Fehler auftritt (siehe **Error, Error method, Error line**).

Mit dem Befehl **GET LAST ERROR STACK** erhalten Sie die Fehlersequenz (z.B. den Fehlerstapel), die zur Unterbrechung geführt hat.

Die Fehlerverwaltungsmethode sollte Fehler in geeigneter Weise verwalten bzw. dem Benutzer eine Fehlermeldung anzeigen. Fehler können beim Bearbeiten in folgenden Bereichen erzeugt werden:

- 4D Datenbank-Engine; zum Beispiel, wenn beim Sichern eines Datensatzes eine Trigger Regel missachtet wird.
- 4D Umgebung; zum Beispiel, wenn nicht genügend Speicher vorhanden ist, um ein Array zuzuweisen.
- Betriebssystem, auf dem die Anwendung läuft; zum Beispiel, die Festplatte ist voll oder Eingabe/Ausgabefehler.

Ist die Unterbrechungsprozedur ausgeführt, kehrt 4D zur unterbrochenen Methode zurück, außer Sie haben in der Unterbrechungsmethode den Befehl **ABORT** aufgerufen. Verwenden Sie **ABORT**, wenn ein Fehler nicht behoben werden kann.

Tritt ein Fehler in der Fehlerverwaltungsmethode selbst auf, übernimmt 4D die Fehlerverwaltung. Von daher sollten Sie sicherstellen, dass die Fehlerverwaltungsmethode keinen Fehler erzeugen kann. Außerdem können Sie **ON ERR CALL** nicht innerhalb der Fehlerverwaltungsmethode verwenden.

Beispiel 1

Folgende Projektmethode versucht, ein Dokument zu erstellen, dessen Name als Parameter empfangen wird. Kann das Dokument nicht erstellt werden, gibt die Projektmethode den Wert 0 (Null) oder den Fehlercode zurück:

```
  ` Projektmethode Create doc
  ` Create doc ( String ; Zeiger ) -> Lange Ganzzahl
  ` Create doc ( DocName ; ->DocRef ) -> Ergebnis des Fehlercode

gError:=0
ON ERR CALL("IO ERROR HANDLER")
$2->:=Create document($1)
ON ERR CALL("")
$0:=gError
```

Die Projektmethode ***IO ERROR HANDLER*** lautet:

```
  ` Projektmethode IO ERROR HANDLER
gError:=Error  ` Kopiere den Fehlercode in die Prozessvariable gError
```

Beachten Sie den Einsatz der Prozessvariablen *gError*, um das Ergebnis innerhalb der aktuell ausführenden Methode zu erhalten. Mit diesen Methoden in Ihrer Datenbank können Sie schreiben:

```
  ` ...
C_TIME(vhDocRef)
$vlErrCode:=Create doc($vsDocumentName;->vhDocRef)
```

```
If($vlErrCode=0)
   ` ...
   CLOSE DOCUMENT($vlErrCode)
Else
   ALERT("Das Dokument konnte nicht erstellt werden, I/O Fehler "+String($vlErrCode))
End if
```

Beispiel 2

Siehe Beispiel im Abschnitt **Arrays und Speicher**.

Beispiel 3

Beim Integrieren einer komplexen Reihe an Operationen können Sie mehrere Unterroutinen haben, die unterschiedliche Fehlerverwaltungsmethoden benötigen. Da pro Prozess jeweils nur eine Fehlerverwaltungsmethode ausführbar ist, gibt es zwei Möglichkeiten:

- Verfolgen Sie die aktuelle Methode, immer wenn Sie **ON ERR CALL** aufrufen oder
- Setzen Sie eine Variable Prozess Array ein (in diesem Fall *asErrorMethod*), um die Fehlerverwaltungsmethoden "zu stapeln", sowie eine Projektmethode (in diesem Fall ***ON ERROR CALL***, um diese Methoden zu installieren bzw. zu entfernen.

Sie müssen das Array gleich zu Beginn der Prozessausführung initialisieren:

```
\\ Vergessen Sie nicht, das Array zu Beginn der Prozessmethode zu initialisieren(die Projektmethode, die den Prozess aktiviert)
ARRAY STRING(63;asErrorMethod;0)
```

Hier die Methode ***ON ERROR CALL***:

```
   ` Projektmethode ON ERROR CALL
   ` ON ERROR CALL { ( String ) }
   ` ON ERROR CALL { ( Methodenname ) }

C_STRING(63;$1;$ErrorMethod)
C_LONGINT($vlElem)

If(Count parameters>0)
   $ErrorMethod:=$1
Else
   $ErrorMethod:=""
End if

If($ErrorMethod#"")
   C_LONGINT(gError)
   gError:=0
   $vlElem:=1+Size of array(asErrorMethod)
   INSERT IN ARRAY(asErrorMethod;$vlElem)
   asErrorMethod{$vlElem}:=$1
   ON ERR CALL($1)
Else
   ON ERR CALL("")
   $vlElem:=Size of array(asErrorMethod)
   If($vlElem>0)
      DELETE FROM ARRAY(asErrorMethod;$vlElem)
      If($vlElem>1)
         ON ERR CALL(asErrorMethod{$vlElem-1})
      End if
   End if
End if
```

Nun können Sie sie folgendermaßen aufrufen:

```
gError:=0
ON ERROR CALL("I/O ERRORS")  ` Installiert die Fehlermethode IO ERRORS
   ` ...
ON ERROR CALL("ALL ERRORS")  ` Installiert die Fehlermethode ALL ERRORS
   ` ...
ON ERROR CALL  ` Entfernt die Fehlermethode ALL ERRORS und installiert erneut IO ERRORS
   ` ...
ON ERROR CALL  ` Entfernt die Fehlermethode IO ERRORS
   ` ...
```

Beispiel 4

Folgende Fehlerverwaltungsmethode ignoriert Unterbrechungen des Benutzers:

```
  //Projektmethode Show_errors_only
If(Error#1006)  //Dies ist keine Benutzerunterbrechung
    ALERT("Der Fehler "+String(Error)+" ist aufgetreten. Der betreffende Code ist: \""+Error formula+"\"")
End if
```

ON EVENT CALL

ON EVENT CALL (MethodenName {; ProzessName})

Parameter	Typ		Beschreibung
MethodenName	String	➡	Aufzurufende Ereignismethode, oder leerer String zum Stoppen der Ereignissuche
ProzessName	String	➡	Name des Prozesses

Beschreibung

Der Befehl **ON EVENT CALL** legt die Unterbrechungsmethode mit Namen *MethodenName* an, um Ereignisse ausfindig zu machen. Sie heißt auch **Ereignisverwaltungsmethode** oder **Ereignisabfangmethode**.

Tipp: Dieser Befehl erfordert fortgeschrittene Programmierkenntnisse. Beim Arbeiten mit Ereignissen benötigen Sie **ON EVENT CALL** normalerweise nicht. In Formularen verwaltet 4D die Ereignisse und sendet sie zu den entsprechenden Formularen und Objekten.

Tipp: Befehle wie z.B. **GET MOUSE**, **Shift down**, etc..., liefern Informationen über Ereignisse. Sie können diese Befehle innerhalb von Objektmethoden aufrufen, um die erforderliche Information über ein Ereignis mit einem Objekt zu erhalten. So müssen Sie nicht einen Algorithmus nach dem Schema **ON EVENT CALL** schreiben.

Der Befehl gilt für die aktuelle Arbeitssitzung. Die Methode läuft standardmäßig in einem separaten lokalen Prozess. Sie können zur gleichen Zeit immer nur eine Ereignisverwaltungsmethode haben. Wollen Sie das Auffinden von Ereignissen stoppen, rufen Sie erneut **ON EVENT CALL** auf und übergeben in *MethodenName* einen leeren String.

Da die Ereignisverwaltungsmethode in einem separaten Prozess läuft, ist sie immer aktiv, selbst wenn keine 4D Methode läuft. 4D ruft diese Methode immer auf, wenn ein Ereignis eintritt. Das kann ein Mausklick oder Drücken einer Taste sein.

Der optionale Parameter *ProzessName* gibt den Namen des Prozesses an, der von **ON EVENT CALL** erzeugt wurde. Ist *ProzessName* das Dollarzeichen (*$*) vorangestellt, wird ein lokaler Prozess gestartet. Geben Sie den Parameter *ProzessName* nicht an, erstellt 4D standardmäßig einen lokalen Prozess mit Namen *$Event Manager*.

Warnung: Seien Sie vorsichtig, was Sie in einer Ereignisverwaltungsmethode ausführen. Rufen Sie KEINE Befehle auf, die Ereignisse generieren, da es sonst u.U. nicht mehr gelingt, aus dieser Methode herauszukommen. Über die Tastenkombination **strg-/Umschalt-/Rückschrittaste** unter Windows bzw. **Befehls-/Umschalt-/Rückschrittaste** auf Macintosh können Sie den Event Manager Prozess stoppen. Auf diese Weise können Sie aus einer Ereignisverwaltungsmethode herauskommen, die unkontrollierbar geworden ist.

Mit den Systemvariablen—*MouseDown, KeyCode, Modifiers, MouseX, MouseY* und *MouseProc* können Sie zwischen nachfolgenden Ereignissen unterscheiden. Beachten Sie, dass diese Variablen Prozessvariablen sind. Von daher ist ihre Reichweite der Ereignisverwaltungsprozess. Sollen Ihre Werte auch in einem anderen Prozess verfügbar sein, kopieren Sie diese Variablen in Interprozessvariablen.

- *MouseDown* wird auf 1 gesetzt, wenn das Ereignis ein Mausklick ist, und 0, wenn nicht.
- *KeyCode* erhält den Code einer Taste. Diese Variable gibt den Code eines Zeichens oder einer Funktionstaste zurück. Eine Liste dieser Codes finden Sie in den Abschnitten **Unicode Codes** und **EXPORT TEXT** sowie **Funktionstasten**. 4D bietet für die meisten ASCII Codes und Funktionstasten vordefinierte Konstanten. Die Themen der Konstanten finden Sie im Explorer Fenster.
- *Modifiers* enthält den Wert der Zusatztaste. Diese Variable gibt an, ob bei Eintreten eines Ereignisses eine der folgenden Tasten gedrückt war:

Plattform	Modifiers
Windows	Umschalttaste, Feststelltaste, alt-, strg-Taste
Macintosh	Umschalttaste, Feststelltaste,Wahltaste, Befehlstaste, ctrl-Taste

Hinweise:

- Die **alt-Taste** unter Windows entspricht der **Wahltaste** auf Macintosh
- Die **strg-Taste** unter Windows entspricht der **Befehlstaste** auf Macintosh
- Die **ctrl-Taste** auf Macintosh entspricht keiner Taste unter Windows. Hier wird die **rechte Maustaste** verwendet.

Allein bewirken diese Tasten noch kein Ereignis. Sie können aber zusammen mit anderen Tasten oder einem Mausklick abgefragt werden. Die Variable *Modifiers* ist vom Typ Lange Ganzzahl mit einem Bit-Feld. 4D bietet vordefinierte Konstanten, die die Bit Position bzw. Bit Maske für die jeweilige Funktionstaste angibt. Wollen Sie z.B. prüfen, ob die **Umschalttaste** gedrückt wurde, schreiben Sie:

```
If(Modifiers?? Shift key bit)  //Wurde die Umschalttaste gedrückt
```

oder:

```
If((Modifiers  &  Shift key mask)#0)  //Wurde die Umschalttaste gedrückt
```

Sie können eine der nachfolgenden Konstanten unter dem Thema **Ereignisse (Zusatztasten)** verwenden:

Modifiers	Konstante
Shift	Shift key bit / Shift key mask
Caps Lock	Caps lock key bit / Caps lock key mask
Alt (Wahltaste auf OS X)	Option key bit / Option key mask
Strg unter Windows	Command key bit / Command key mask
Ctrl auf OS X	Control key bit / Control key mask
Befehl auf OS X	Command key bit / Command key mask
Rechter Klick	Control key bit / Control key mask

- *MouseX* und *MouseY* geben die horizontale und vertikale Position des Mausklicks zurück, gemäß dem lokalen Koordinatensystem des Fensters, wo der Klick aufgetreten ist. Die Koordinaten werden von der linken oberen Ecke des Fensters aus berechnet. Sie sind nur relevant, wenn es einen Mausklick gibt.
- *MouseProc* enthält die Referenznummer des Prozesses, in dem das Ereignis (Mausklick) aufgetreten ist.

Wichtig: Die Systemvariablen *MouseDown, KeyCode, Modifiers, MouseX, MouseY* und *MouseProc* enthalten signifikante Werte nur in einer Ereignisverwaltungsmethode, die mit **ON EVENT CALL** installiert wurde.

Beispiel

Dieses Beispiel bricht den Druckvorgang ab, wenn der Benutzer die Tastenkombination **ctrl+Punkt** drückt. Zuerst wird die Ereignisverwaltungsmethode installiert, dann erscheint eine Meldung, dass der Benutzer den Druckvorgang abbrechen kann. Wird die Interprozessvariable *◇vbWeStop* in der Ereignisverwaltungsmethode auf den Wert Wahr gesetzt, erhält der Benutzer eine Meldung über die Anzahl der bereits gedruckten Datensätze. Dann wird die Ereignisverwaltungsmethode entfernt:

```
PAGE SETUP
If(OK=1)
   ◇vbWeStop:=False
   ON EVENT CALL("EVENT HANDLER") ` Installiert die Ereignisverwaltungsmethode
   ALL RECORDS([People])
   MESSAGE("Unterbrich Drucken mit Tastenkombination ctrl/Punkt")
   $vlNbRecords:=Records in selection([People])
   For($vlRecord;1;$vlNbRecords)
      If(◇vbWeStop)
         ALERT("Drucken abgebrochen bei Datensatz "+String($vlRecord)+" von "+String($vlNbRecords))
         $vlRecord:=$vlNbRecords+1
      Else
         Print form([People];"Spezialbericht")
      End if
   End for
   PAGE BREAK
   ON EVENT CALL("") ` Entferne die Ereignisverwaltungsmethode
End if
```

Durch Drücken der Tastenkombination **ctrl/Punkt** erhält die Ereignisverwaltungsmethode den Wert Wahr:

```
` Projektmethode EVENT HANDLER
If((Modifiers?? Command key bit) & (KeyCode=Period))
   CONFIRM("Sind Sie sicher?")
   If(OK=1)
      ◇vbWeStop:=True
      FILTER EVENT ` Vergessen Sie NICHT diesen Aufruf, da 4D sonst auch dieses Ereignis erhält
   End if
End if
```

Beachten Sie, dass dieses Beispiel **ON EVENT CALL** einsetzt, da es mit den Befehlen **PAGE SETUP**, **Print form** und **PAGE BREAK** mit einer **For...End for** Schleife einen speziellen Druckbericht ausführt.

Drucken Sie einen Bericht mit **PRINT SELECTION**, benötigen Sie keine Ereignisverwaltung zum Unterbrechen des Druckvorgangs; das erledigt dieser Befehl bereits.

SET ASSERT ENABLED

SET ASSERT ENABLED (Assertionen {; *})

Parameter	Typ		Beschreibung
Assertionen	Boolean	➡	Wahr = Assertionen aktivieren, Falsch = Assertionen deaktivieren
*	Operator	➡	Ohne * = Befehl gilt für alle Prozesse, Mit * = Befehl gilt nur für den aktuellen Prozess

Beschreibung

Der Befehl **SET ASSERT ENABLED** ermöglicht, jede in den 4D Code der Anwendung eingefügte Assertion zu deaktivieren oder wieder zu aktivieren. Weitere Informationen dazu finden Sie unter dem Befehl **ASSERT**.

Im Programm hinzugefügte Assertionen sind standardmäßig im interpretierten und kompilierten Modus aktiviert. Dieser Befehl ist hilfreich, wenn Sie diese deaktivieren wollen, da die Bewertung u.U. eine gewisse Ausführungszeit beansprucht und Sie die Assertionen auch für den Endbenutzer der Anwendung ausblenden wollen. Eine typische Verwendung ist in der **Datenbankmethode On Startup**. Hier lassen sich die Assertionen aktivieren bzw. deaktivieren, wenn die Anwendung im Testmodus bzw. im Designmodus ist.

SET ASSERT ENABLED gilt standardmäßig für alle Prozesse der Anwendung. Soll er nur für den aktuellen Prozess gelten, übergeben Sie den Parameter *.

Beachten Sie, dass bei deaktivierten Assertionen die in **ASSERT** übergebenen Ausdrücke nicht mehr bewertet werden. Code-Zeilen, die diesen Befehl aufrufen, haben keine Auswirkung mehr auf die Arbeitsweise der Anwendung, weder bezüglich Verhalten noch Performance.

Beispiel

Assertionen deaktivieren:

```
SET ASSERT ENABLED(False)
ASSERT(TestMethod)  // TestMethod wird nicht aufgerufen, da Assertionen deaktiviert sind
```

Untertabellen

- 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 (IDFeld) -> Funktionsergebnis

Parameter	Typ		Beschreibung
IDFeld	Feld	→	Feld vom Typ "Verknüpfung Untertabelle" oder "Lange Ganzzahl" einer früheren Verknüpfung Untertabelle
Funktionsergebnis	Lange Ganzzahl	↩	Interner Schlüssel der Verknüpfung

Beschreibung

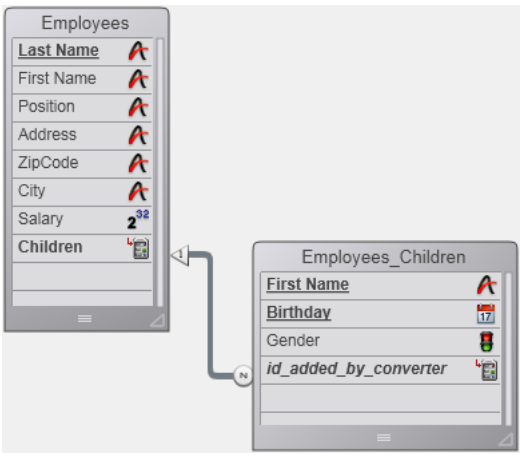
Die Funktion **Get subrecord key** vereinfacht die Migration von 4D Code für konvertierte Untertabellen in Standardcode zum Arbeiten mit Tabellen.

Zur Erinnerung: Ab 4D Version 11 werden Untertabellen nicht mehr unterstützt. Wird eine ältere Datenbank konvertiert, werden alle vorhandenen Untertabellen in Standardtabellen umgewandelt, die mit den Originaltabellen über eine automatische Verknüpfung verbunden sind. Die frühere Untertabelle wird die Viele-Tabelle, die Originaltabelle wird die Eine-Tabelle. In der Eine-Tabelle wird das frühere Untertabellenfeld in ein spezielles Feld vom Typ "Untertabelle Verknüpfung" umgewandelt. Im Viele-Feld wird ein spezielles Feld vom Typ "Untertabelle Verknüpfung" mit Namen "id_added_by_converter" hinzugefügt.

Auf diese Weise können konvertierte Datenbanken weiter funktionieren. Wir empfehlen jedoch dringend, in Ihren konvertierten Datenbanken alle Funktionalitäten mit Untertabellen durch Funktionalitäten für Standardtabellen zu ersetzen.

Im ersten Schritt löschen Sie die spezifischen automatischen Verknüpfungen, was die von Untertabellen geerbten Mechanismen dauerhaft deaktiviert. Danach müssen Sie den zugeordneten Code umschreiben. Hier unterstützt die Funktion **Get subrecord key** durch Zurückgeben der internen ID solcher Verknüpfungen. Diese interne ID macht die aktuelle Verknüpfung überflüssig und Sie können dann mit der Auswahl der bisherigen Untertabelle sogar arbeiten, wenn die Verknüpfung nicht mehr existiert.

Werfen wir z.B. einen Blick auf folgende konvertierte Struktur:



In 4D funktioniert der folgende Code noch, er muss aber aktualisiert werden:

```
ALL SUBRECORDS([Employees]Children)
$total:=Records in subselection([Employees]Children)
vFirstnames:=""
For($i;1;$total)
   vFirstnames:=vFirstnames+[Employees]Children'FirstName+" "
   NEXT SUBRECORD([Employees]Children)
End for
```

Sie können diesen Code ersetzen durch:

```
QUERY([Employees_Children];[Employees_Children]id_added_by_converter=Get subrecord key([Employees]Children))
$total:=Records in selection([Employees_Children])
vFirstnames:=""
For($i;1;$total)
   vFirstnames:=vFirstnames+[Employees_Children]FirstName+" "
   NEXT RECORD(Employees_Children)
End for
```

Hinweis: Get subrecord key gibt 0 zurück, wenn bei der Ausführung kein aktueller Datensatz geladen wird.

Der zweite Teil des Code hat den Vorteil, dass er standardmäßige 4D Befehle verwendet und auf dieselbe Weise funktioniert, egal ob die Verknüpfung besteht oder nicht. Entfernen Sie die Verknüpfung, gibt die Funktion nur den Schlüsselwert zurück, der im Feld Lange Ganzzahl gespeichert wird.

Im Parameter *IDFeld* akzeptiert die Funktion entweder ein Feld vom Typ Untertabelle Verknüpfung (wenn die Verknüpfung noch existiert) oder vom Typ Lange Ganzzahl (wenn die Verknüpfung entfernt wurde). In allen anderen Fällen wird ein Fehler erzeugt.

Auf diese Weise können Sie Übergangscode schreiben. Im letzten Stadium der Aktualisierung der Anwendung können Sie die Aufrufe dieser Funktion entfernen.

Dem Feld id_added_by_converter einen Wert zuweisen

Ab 4D v14 R3 können Sie dem speziellen Feld "id_added_by_converter" auch selbst einen Wert zuweisen. Dieser Wert wurde in bisherigen Releases nur von 4D verwaltet, und Sie mussten Befehle wie **_o_CREATE SUBRECORD** verwenden, um neue Datensätze in konvertierten Untertabellen hinzuzufügen.

Mit diesem neuen Feature können Sie Ihre alten Datenbanken mit Untertabellen nach und nach konvertieren: Sie können die spezielle Verknüpfung "Untertabelle Verknüpfung" vorerst beibehalten und verknüpfte Datensätze ändern oder hinzufügen, als ob sie normale Datensätze wären. Sind dann all Ihre Methoden aktualisiert, können Sie diese spezielle Verknüpfung durch eine reguläre Verknüpfung ersetzen, ohne Ihren Code zu verändern.

Sie können beispielsweise schreiben:

```
CREATE RECORD([Employees])
[Employees]LastName:="Jonas"
CREATE RECORD([Employees_Children])
[Employees_Children]FirstName:="Natascha"
[Employees_Children]BirthDate:=!24/12/2013!
[Employees_Children]id_added_by_converter:=Get subrecord key([Employees]Children)
SAVE RECORD([Employees_Children])
SAVE RECORD([Employees]
```

Dieser Code funktioniert für spezielle und reguläre Verknüpfungen gleichermaßen.

_o_ALL SUBRECORDS

_o_ALL SUBRECORDS (Untertabelle)

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle, deren Unterdatensätze geladen werden sollen

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_APPLY TO SUBSELECTION

_o_APPLY TO SUBSELECTION (Untertabelle ; Anweisung)

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Zu ändernde Untertabelle
Anweisung	Anweisung	➡	Zuweisung oder Methode

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_Before subselection

_o_Before subselection (Untertabelle) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle, auf die die Funktion angewandt wird
Funktionsergebnis	Boolean	⮌	Ja (TRUE) oder Nein (FALSE)

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_CREATE SUBRECORD

_o_CREATE SUBRECORD (Untertabelle)

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle, für die ein Unterdatensatz angelegt werden soll

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_DELETE SUBRECORD

_o_DELETE SUBRECORD (Untertabelle)

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle, in der ein Unterdatensatz gelöscht werden soll

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_End subselection

_o_End subselection (Untertabelle) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle, auf die die Funktion angewandt wird
Funktionsergebnis	Boolean	⮌	Ja (TRUE) oder Nein (FALSE)

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_FIRST SUBRECORD

_o_FIRST SUBRECORD (Untertabelle)

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle, deren erster Unterdatensatz ausgewählt werden soll

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_LAST SUBRECORD

_o_LAST SUBRECORD (Untertabelle)

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle, deren letzter Unterdatensatz ausgewählt werden soll

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_NEXT SUBRECORD

_o_NEXT SUBRECORD (Untertabelle)

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle, deren nächster Unterdatensatz ausgewählt werden soll

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_ORDER SUBRECORDS BY

_o_ORDER SUBRECORDS BY (Untertabelle ; Unterdatenfeld {; > oder <}{; Unterdatenfeld2 ; > oder <2 ; ... ; UnterdatenfeldN ; > oder <N})

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Zu sortierende Untertabelle
Unterdatenfeld	Unterfeld	➡	Unterdatenfeld, nach dem sortiert werden soll
> oder <	Operator	➡	Sortierordnung: > aufsteigend oder < absteigend

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_PREVIOUS SUBRECORD

_o_PREVIOUS SUBRECORD (Untertabelle)

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle, deren vorheriger Unterdatensatz ausgewählt werden soll

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_QUERY SUBRECORDS

_o_QUERY SUBRECORDS (Untertabelle ; Formel)

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle, in der gesucht werden soll
Formel	Boolean	➡	Formel

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

_o_Records in subselection

_o_Records in subselection (Untertabelle) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Untertabelle	Untertabelle	➡	Untertabelle, in der die Anzahl der Unterdatensätze gezählt werden soll
Funktionsergebnis	Lange Ganzzahl	⮌	Anzahl der Unterdatensätze in der aktuellen Auswahl

Hinweis zur Kompatibilität

Untertabellen werden ab 4D Version 11 nicht mehr unterstützt. Ein Kompatibilitätsmechanismus sorgt dafür, dass dieser Befehl in konvertierten Datenbanken weiter funktioniert. Wir raten jedoch dringend, alle Untertabellen durch verknüpfte Standardtabellen zu ersetzen.

Variablen

- CLEAR VARIABLE
- LOAD VARIABLES
- SAVE VARIABLES

CLEAR VARIABLE

CLEAR VARIABLE (Variablenname)

Parameter	Typ		Beschreibung
Variablenname	Variable	➡	Zu löschende Variable

Beschreibung

CLEAR VARIABLE setzt *Variablenname* auf den entsprechenden Standardwert des Typs (z.B. leerer String für String und Textvariablen, 0 für numerische Variablen, keine Elemente für Arrays, etc.). Die Variable existiert weiterhin im Speicher.

Die in *Variablenname* übergebene Variable kann eine lokale, eine Prozess- oder Interprozessvariable sein.

Hinweis: Prozessvariablen müssen Sie nicht löschen, wenn ein Prozess endet; 4D löscht sie automatisch. Lokale Variablen werden automatisch gelöscht, wenn die Methode, in der sie enthalten sind, ausgeführt wurde.

Beispiel

Sie verwenden in einem Formular während der Dateneingabe die DropDown Liste *asMyDropDown*. Sie ist nur für die Benutzeroberfläche notwendig. Sie benötigen dieses Array nicht mehr, wenn das Formular fertig ist. Deshalb entfernen Sie das Array während des Ereignisses On Unload:

```
   ` Objektmethode für DropDown Liste asMyDropDown
Case of
   :(Form event=On Load)
   ` Initialisiere Array auf die eine oder andere Art
      ARRAY STRING(63;asMyDropDown;...)
   ` ...
   :(Form event=On Unload)
   ` Array wird nicht mehr benötigt
      CLEAR VARIABLE(asMyDropDown)
   ` ...
End case
```

LOAD VARIABLES

LOAD VARIABLES (Dokumentname ; Variablenname {; Variablenname2 ; ... ; VariablennameN})

Parameter	Typ		Beschreibung
Dokumentname	String	➡	Dokument mit den 4D Variablen
Variablenname	Variable	⬅	Variablen zum Empfangen von Werten

Beschreibung

Der Befehl **LOAD VARIABLES** lädt die Variablen Var1, Var2 bis VarN aus dem Dokument *Dokumentname*. Dieses Dokument muss zuvor mit dem Befehl **SAVE VARIABLES** angelegt worden sein.

- Gibt es die Variablen nicht, werden sie angelegt.
- Gibt es sie, werden sie überschrieben.

Übergeben Sie für *Dokumentname* einen leeren String, zeigt 4D das Standardfenster zum Öffnen von Dokumenten an. Wurde das Dokument geladen, gibt die Variable OK den Wert 1 zurück, sonst 0. Der Name des Dokumentes steht in der Systemvariablen *Document*.

Der Compiler legt alle Variablen bei Programmstart an.
In kompilierten Datenbanken muss jede Variable vom selben Typ sein wie die von der Festplatte geladene, d.h. der Typ darf nicht geändert werden.

Warnung: Dieser Befehl unterstützt keine Variablen vom Typ Array. Verwenden Sie dafür die BLOB Befehle.

Beispiel

Folgendes Beispiel lädt drei Variablen aus dem Dokument mit Namen *UserPrefs*:

```
LOAD VARIABLES("User Prefs";vsName;vlCode;vgIconPicture)
```

Systemvariablen und Mengen

Wurden die Variablen korrekt geladen, gibt die Variable OK den Wert 1 zurück, sonst den Wert Null (0).

SAVE VARIABLES

SAVE VARIABLES (Dokumentname ; Variablenname {; Variablenname2 ; ... ; VariablennameN})

Parameter	Typ		Beschreibung
Dokumentname	String	➡	Dokument zum Sichern der Variablen
Variablenname	Variable	➡	Zu sichernde Variablen

Beschreibung

Der Befehl **SAVE VARIABLES** sichert die Variablen Var1, Var2 bis VarN im Dokument *Dokumentname*. Die Variablen müssen nicht vom selben Typ sein. Jedoch müssen sie von folgendem Typ sein: String, Text, Zahl, Ganzzahl, Lange Ganzzahl, Datum, Zeit, Boolean oder Bild.

Ist *Dokumentname* ein leerer String, zeigt 4D das Standardfenster zum Erstellen von Dokumenten an. Wurden die Variablen korrekt gesichert, gibt die Systemvariable *OK* den Wert 1 zurück, sonst 0. Der Name des Dokumentes steht in der Systemvariablen *Document*.

Das von **SAVE VARIABLES** angelegte Dokument verwendet ein internes Datenformat. Sie können die Variablen nur mit dem Befehl **LOAD VARIABLES** wiederfinden. Dieses Dokument lässt sich weder mit **RECEIVE PACKET** noch mit **RECEIVE VARIABLE** lesen.

SAVE VARIABLES ist der einzige Weg, Bilder zu sichern. Im Normalfall ist es sinnvoller, Variablen mit **SEND PACKET** zu sichern oder die Variablen in Datenfeldern abzulegen.

Warnung: Dieser Befehl unterstützt keine Variablen vom Typ Array. Verwenden Sie dazu die BLOB Befehle.

Beispiel

Folgendes Beispiel sichert drei Variablen in dem Dokument mit Namen *UserPrefs*:

```
SAVE VARIABLES("UserPrefs";vsName;vlCode;vgIconPicture)
```

Systemvariablen und Mengen

Wurden die Variablen korrekt gesichert, gibt die Systemvariable OK den Wert 1 zurück; sonst den Wert Null (0).

Verknüpfungen

- Verknüpfungen
- 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

Verknüpfungen

Die Befehle der folgenden Abschnitte, insbesondere **RELATE ONE** und **RELATE MANY** erstellen und verwalten automatische und manuelle Verknüpfungen zwischen Tabellen. Bevor Sie diese Befehle verwenden, lesen Sie im Handbuch *4D Designmodus* nach, wie Sie Verknüpfungen zwischen Tabellen anlegen.

Operationen, die Verknüpfung aufrufen

Nachstehend finden Sie die Liste der Operationen, die automatisch eine Verknüpfung aufrufen und, gemäß der gewählten Verknüpfungsart, die aktuelle Auswahl der verknüpften Tabelle ändern:

- Daten eingeben
- Datensätze auf dem Bildschirm in Ausgabeformularen anzeigen
- Berichte erstellen
- Operationen, wie Such- und Sortierläufe oder Formeln auf Auswahl anwenden.

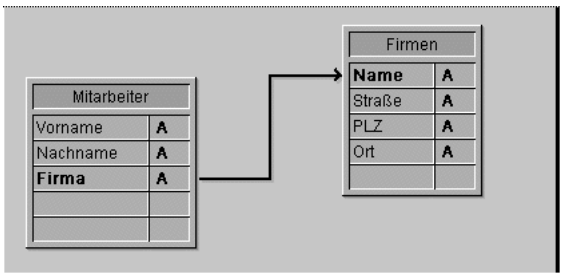
Zur Leistungsoptimierung macht 4D beim Einrichten von automatischen Verknüpfungen nur einen Datensatz zum aktuellen Datensatz für die Tabelle. Der Datensatz wird für jede der oben aufgelisteten Operationen folgendermaßen geladen:

- Wählt eine Verknüpfung nur einen Datensatz aus einer verknüpften Tabelle, wird dieser Datensatz von der Festplatte geladen.
- Wählt eine Verknüpfung mehrere Datensätze aus einer verknüpften Tabelle, wird für diese Tabelle eine neue Datensatzauswahl erstellt, der erste Datensatz dieser Auswahl wird von der Festplatte geladen.

Wir nehmen als Beispiel eine Datenbank mit zwei Tabellen, die miteinander verknüpft sind.

Laden Sie einen Datensatz der Tabelle [Mitarbeiter], wird der entsprechende Datensatz der Tabelle [Firmen] geladen. Der Inhalt des Datensatzes der Tabelle [Firmen] steht dem Datensatz der Tabelle [Mitarbeiter] zur Verfügung. Die aktuelle Auswahl der verknüpften Tabelle besteht also aus einem einzigen Datensatz, dem der Firma, zu der der Mitarbeiter gehört.

Wollen Sie einen Datensatz der Tabelle [Firmen] ändern, werden alle Mitarbeiter dieser Firma, die in der Mitarbeitertabelle gespeichert sind, in den Arbeitsspeicher geladen. Die aktuelle Auswahl der verknüpften Tabelle besteht also aus allen Mitarbeitern, die für diese Firma arbeiten.



Eine Verknüpfung kann eine **Viele-zu-Eine**-Relation oder eine **Eine-zu-Viele**-Relation sein. Viele Mitarbeiter können zu einer Firma gehören und eine Firma kann viele Mitarbeiter haben.

Es ist nicht immer möglich, dass die Tabelle ein einmaliges Datenfeld enthält, das für die Verknüpfung geeignet ist. So kann beispielsweise in der Tabelle [Firmen] das Datenfeld Name mehrere Firmendatensätze mit demselben Wert haben. Dieses Problem umgehen Sie, indem Sie ein einmaliges Feld einrichten, z.B. die Firmennummer und die Verknüpfung zu diesem Feld ziehen.

Folgende Tabelle zeigt Befehle, die beim Laden von Datensätzen mit automatischen Verknüpfungen arbeiten. Alle Befehle verwenden vorhandene automatische **Viele-zu-Eine**-Verknüpfungen. Nur die mit Ja gekennzeichneten Befehle verwenden automatische **Eine-zu-Viele**-Verknüpfungen.

Befehl	Eine-zu-Viele Relation
ADD RECORD	Ja
ADD SUBRECORD	Nein
APPLY TO SELECTION	Nein
DISPLAY SELECTION	Nein
EXPORT DIF	Nein
EXPORT SYLK	Nein
EXPORT TEXT	Nein
EXPORT DATA	Nein
MODIFY RECORD	Ja
MODIFY SUBRECORD	Nein
MODIFY SELECTION	Ja (bei der Dateneingabe)
ORDER BY	Nein
ORDER BY FORMULA	Nein
QUERY BY FORMULA	Ja
QUERY SELECTION	Ja
QUERY	Ja
PRINT LABEL	Nein
PRINT SELECTION	Ja
QR REPORT	Nein
SELECTION TO ARRAY	Nein
SELECTION RANGE TO ARRAY	Nein

Verknüpfungen über Befehle aufrufen

Automatische Verknüpfung heißt nicht, dass ein bzw. mehrere verknüpfte Datensätze für eine Tabelle bereits ausgewählt werden, weil ein Befehl einen Datensatz lädt. In manchem Fällen müssen Sie die verknüpften Datensätze explizit mit den Befehlen **RELATE ONE** oder **RELATE MANY** aufrufen, um darauf zugreifen zu können.

Einige der oben aufgeführten Befehle, z.B. die Suchbefehle laden den aktuellen Datensatz nach Beenden des Vorgangs. In diesem Fall wählt der geladene Datensatz nicht automatisch den verknüpften Datensatz. Sie müssen also, wenn Sie auf die verknüpften Daten zugreifen wollen, die Datensätze explizit mit **RELATE ONE** oder **RELATE MANY** aufrufen.

CREATE RELATED ONE

CREATE RELATED ONE (Feldname)

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Feld, von dem die Verknüpfung ausgeht

Beschreibung

Der Befehl **CREATE RELATED ONE** führt zwei Aktionen aus. Gibt es für *Feldname* noch keinen verknüpften Datensatz, d.h. für den aktuellen Wert von *Feldname* wird keine Entsprechung gefunden, legt **CREATE RELATED ONE** diesen Datensatz automatisch an. Der Befehl erstellt oder lädt den verknüpften Datensatz zu *Feldname* in den Speicher.

Um einen Wert im passenden Feld zu sichern, weisen Sie Werte zum Eine-Feld über das Viele-Feld zu. Rufen Sie **SAVE RELATED ONE** auf, um den neuen Datensatz zu sichern.

Existiert ein verknüpfter Datensatz, arbeitet CREATE RELATED ONE wie **RELATE ONE** und lädt den verknüpften Datensatz in den Speicher.

GET AUTOMATIC RELATIONS

GET AUTOMATIC RELATIONS (Eine ; Viele)

Parameter	Typ		Beschreibung
Eine	Boolean	⇐	Status aller Viele-zu-Eine Verknüpfungen
Viele	Boolean	⇐	Status aller Eine-zu-Viele Verknüpfungen

Beschreibung

Der Befehl **GET AUTOMATIC RELATIONS** zeigt an, ob der Status automatisch/manuell aller manuellen Viele-zu-Eine und Eine-zu-Viele Verknüpfungen der Datenbank im aktuellen Prozess geändert wurde.

- *Eine:* Dieser Parameter gibt *Wahr* zurück, wenn ein früherer Aufruf vom Befehl **SET AUTOMATIC RELATIONS** alle manuellen Eine-zu-Viele Verknüpfungen auf automatisch gesetzt hat – zum Beispiel **SET AUTOMATIC RELATIONS**(*Wahr;Falsch*).
  Dieser Parameter gibt *Falsch* zurück, wenn **SET AUTOMATIC RELATIONS** nicht aufgerufen wurde oder seine frühere Ausführung die manuelle Viele-zu-Eine Verknüpfungen nicht verändert hat – zum Beispiel **SET AUTOMATIC RELATIONS**(*Falsch;Falsch*).
- *Viele:* Dieser Parameter gibt *Wahr* zurück, wenn ein früherer Aufruf vom Befehl **SET AUTOMATIC RELATIONS** alle manuellen Eine-zu-Viele Verknüpfungen auf automatisch gesetzt hat – zum Beispiel **SET AUTOMATIC RELATIONS**(*Wahr;Wahr*).
  Dieser Parameter gibt *Falsch* zurück, wenn **SET AUTOMATIC RELATIONS** nicht aufgerufen wurde oder seine frühere Ausführung die manuellen Eine-zu-Viele Verknüpfungen nicht verändert hat – zum Beispiel **SET AUTOMATIC RELATIONS**(*Wahr;Falsch*).

Beispiel

Siehe Beispiel zum Befehl **GET FIELD RELATION**.

GET FIELD RELATION

GET FIELD RELATION (VieleFeld ; Eine ; Viele {; *})

Parameter	Typ		Beschreibung
VieleFeld	Feld	➡	Feld, wo die Verknüpfung startet
Eine	Lange Ganzzahl	⬅	Status der Viele-zu-Eine Verknüpfung
Viele	Lange Ganzzahl	⬅	Status der Eine-zu-Viele Verknüpfung
*	Operator	➡	Mit *: Eine und Viele geben den aktuellen Status der Verknüpfung zurück (nur Werte 2 oder 3) Ohne * (Standard): Eine und Viele geben den Wert 1 zurück, wenn die Verknüpfung nicht durch Programmierung verändert wurde

Beschreibung

Der Befehl **GET FIELD RELATION** prüft, ob der Status automatisch/manuell der Verknüpfung, die vom *VieleFeld* ausgeht, für den aktuellen Prozess geändert wurde.

Sie können jede Verknüpfung sehen, inkl. automatische Verknüpfungen im Strukturfenster.

- Im *VieleFeld* übergeben Sie den Namen des Feldes in der VieleTabelle, von dem die zu prüfende Verknüpfung ausgeht. Geht vom *VieleFeld* keine Verknüpfung aus, geben die Parameter *Eine* und *Viele* den Wert 0 (Null) zurück. Es wird ein Fehler generiert und die Systemvariable OK wird auf 0 (Null) gesetzt (siehe unten).
- Nach Ausführen des Befehls enthält der Parameter *Eine* einen Wert, der angibt, ob die Viele-zu-Eine Verknüpfung auf automatisch gesetzt ist:
  0 = Vom *VieleFeld* geht keine Verknüpfung aus. Syntaxfehler No. 16 wird erzeugt ("Das Feld hat keine Verknüpfung"), die Systemvariable OK hat den Wert 0 (Null).
  1 = Vom *VieleFeld* geht keine Verknüpfung aus. Syntaxfehler No. 16 wird erzeugt ("Das Feld hat keine Verknüpfung"), die Systemvariable OK hat den Wert 0 (Null).
  2 = Die Viele-zu-Eine Verknüpfung ist für den Prozess manuell.
  3 = Die Viele-zu-Eine Verknüpfung ist für den Prozess automatisch.
- Nach Ausführen des Befehls enthält der Parameter *Viele* einen Wert, der angibt, ob die Eine-zu-Viele Verknüpfung automatisch ist:
  0 = Es gibt keine Verknüpfung, ausgehend vom *VieleFeld*. Der Syntaxfehler Nr. 16 ("Dieses Datenfeld ist nicht verknüpft") wird erzeugt, die Systemvariable OK wird auf 0 (Null) gesetzt.
  1 = Der Status automatisch/manuell der angegebenen Eine-zu Viele Verknüpfung ist der, welcher in den Verknüpfungseigenschaften der Designumgebung festgelegt – und nicht per Programmierung verändert wurde.
  2 = Die Eine-zu-Viele Verknüpfung ist für den Prozess manuell.
  3 = Die Eine-zu-Viele Verknüpfung ist für den Prozess automatisch.

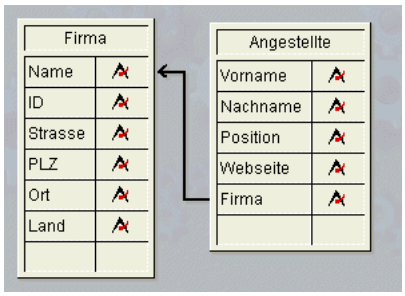
Sie können die Werte, welche in den Parametern Eine und Viele zurückgegeben werden, mit den Konstanten unter dem Thema **Verknüpfungen** vergleichen:

Konstante	Typ	Wert
Automatic	Lange Ganzzahl	3
Manual	Lange Ganzzahl	2
No relation	Lange Ganzzahl	0
Structure configuration	Lange Ganzzahl	1

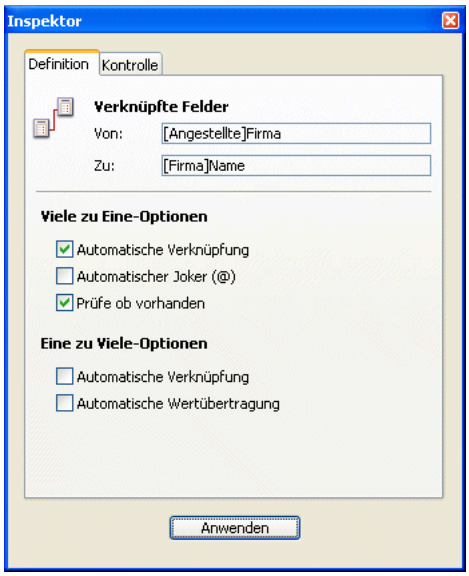
- Mit dem optionalen Parameter * können Sie das Lesen des aktuellen Status erzwingen, auch wenn er nicht per Programmierung geändert wurde. Das heißt, mit dem Parameter * kann in den Parametern *Eine* und *Viele* nur der Wert 2 oder 3 zurückgegeben werden.

Beispiel

Nehmen wir folgende Struktur:



Die Verknüpfung vom Feld [Angestellte]'Firma zum Feld [Firma]Name hat folgende Eigenschaften:



Nachfolgender Code zeigt die verschiedenen Möglichkeiten mit den Befehlen **GET FIELD RELATION**, **GET AUTOMATIC RELATIONS** und **SET FIELD RELATION**, **Bit Operatoren** und ihre Auswirkung:

```
GET AUTOMATIC RELATIONS(one;many)  `gibt Falsch, Falsch zurück
GET FIELD RELATION([Angestellte]Firma;one;many)  `gibt 1,1 zurück
GET FIELD RELATION([Angestellte]Firma;one;many;*)  `gibt 3,2 zurück

SET FIELD RELATION([Angestellte]Firma;2;0)  `Setzt die Viele-zu-Eine Verknüpfung auf manuell

GET FIELD RELATION([Angestellte]Firma;one;many)  `gibt 2,1 zurück
GET FIELD RELATION([Angestellte]Firma;one;many;*)  `gibt 2, 2 zurück

SET FIELD RELATION([Angestellte]Firma;1;0)  `Stellt für die Viele-zu-Eine Verknüpfung den in der Designumgebung definierten Parameter wieder her

GET FIELD RELATION([Angestellte]Firma;one;many)  `gibt 1,1 zurück
GET FIELD RELATION([Angestellte]Firma;one;many;*)  `gibt 3,2 zurück

SET AUTOMATIC RELATIONS(True;True)  `Setzt alle Verknüpfungen aller Tabellen auf automatisch

GET AUTOMATIC RELATIONS(one;many)  `gibt Wahr, Wahr zurück
GET FIELD RELATION([Angestellte]Firma;one;many)  `gibt 1,1 zurück
GET FIELD RELATION([Angestellte]Firma;one;many;*)  `gibt 3,3 zurück
```

OLD RELATED MANY

OLD RELATED MANY (Feldname)

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Eine-Feld

Beschreibung

Der Befehl **OLD RELATED MANY** ist identisch mit dem Befehl **RELATE MANY**, außer dass er zum Einrichten der Verknüpfung den alten Wert im Eine-Feld verwendet.

Hinweis: OLD RELATED MANY verwendet den alten Wert des Viele-Felds, der mit der Funktion **Old** zurückgegeben wird. Weitere Informationen dazu finden Sie in der Beschreibung zur Funktion **Old**.

OLD RELATED MANY ändert die Auswahl der verknüpften Tabelle und wählt den ersten Datensatz der Auswahl als den aktuellen Datensatz.

Beispiel: Ändert der Anwender das Verknüpfungsfeld, in der folgenden Methode das Feld Code, müssen natürlich auch alle verknüpften Datensätze geändert werden. Über den neuen Wert kann aber die Verknüpfung nicht aufgebaut werden, da in den Datensätzen noch der alte Wert steht. Rufen Sie dazu den Befehl **OLD RELATED MANY** auf. Er gibt die Auswahl der mit dem alten Wert verknüpften Datensätze an. Danach können die Verknüpfungsfelder aktualisiert werden:

```
If(During)  `Skript des Feldes Code
   If(Code#Alt(Code)  `Wenn das Feld Code geändert wurde
      OLD RELATED MANY(Code)  `Lade die alten Datensätze
      APPLY TO SELECTION([Kontakte];[Kontakte]CodeE:=Code)
  `Das Feld CodeE aktualisieren
   End if
End if
```

OLD RELATED ONE

OLD RELATED ONE (Feldname)

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Feld, von dem die Verknüpfung ausgeht

Beschreibung

Der Befehl **OLD RELATED ONE** arbeitet genauso wie der Befehl **RELATE ONE**, mit dem Unterschied, dass er den Datensatz, auf den die alte Verknüpfung gezeigt hat, d. h. den zuletzt gesicherten Wert von *Feldname* lädt. Dieser Befehl ist nur auf einen schon gesicherten Datensatz anwendbar.

Wollen Sie den alten verknüpften Datensatz ändern und sichern, rufen Sie **SAVE RELATED ONE** auf.

Hinweis: OLD RELATED ONE verwendet den alten Wert des Viele-Felds, der mit der Funktion **Old** zurückgegeben wird. Weitere Informationen dazu finden Sie in der Beschreibung zur Funktion **Old**.

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt und die verknüpften Datensätze geladen, wird die Systemvariable OK auf 1 gesetzt. Hat der Benutzer im Dialogfenster der Datenauswahl auf **Abbrechen** geklickt, wird die Variable OK auf 0 (Null) **gesetzt**. Der Dialog erscheint, wenn der verknüpfte Datensatz geändert wurde.

RELATE MANY

RELATE MANY (Eine-Tabelle | Feld)

Parameter	Typ		Beschreibung
Eine-Tabelle | Feld	Tabelle, Feld	➡	Tabelle bzw. Feld, die die Verknüpfungen erhält

Beschreibung

RELATE MANY hat zwei Syntaxmöglichkeiten:

Die erste Syntax, **RELATE MANY**(Eine-Tabelle) lädt alle Eine-zu-Viele Verknüpfungen für *Eine-Tabelle*. Sie ändert die aktuelle Auswahl aller Tabellen mit einer Eine-zu-Viele Verknüpfung auf *Eine-Tabelle*. Die aktuelle Auswahl in den Viele-Tabellen richtet sich nach dem aktuellen Wert jedes verknüpften Feldes in der Eine-Tabelle. Jedes Mal, wenn dieser Befehl ausgeführt wird, wird die aktuelle Auswahl der Viele-Tabellen neu erzeugt und der erste Datensatz der Auswahl wird als der aktuelle Datensatz geladen.

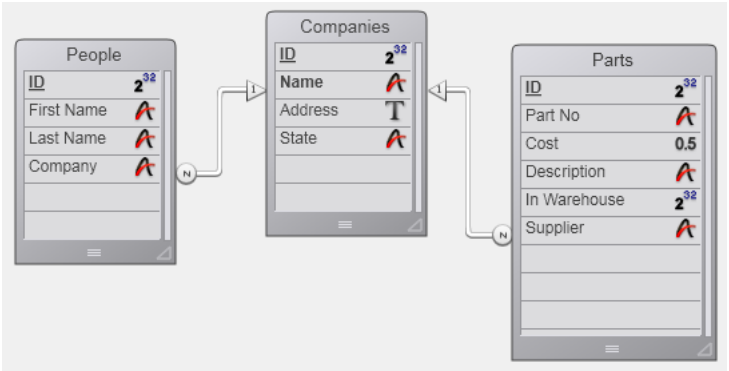
Die zweite Syntax, **RELATE MANY**(Eine-Feld) lädt die Eine-zu-Viele Verknüpfung für das *Eine-Feld*. Sie ändert nur die Auswahl und den aktuellen Datensatz der Tabellen, die mit *Eine-Feld* verbunden sind, d.h. die verknüpften Datensätze werden die aktuelle Auswahl für die Viele-Tabelle.

Hinweis: Ist die aktuelle Auswahl in *Eine-Tabelle* während der Ausführung des Befehls leer, hat er keine Auswirkung.

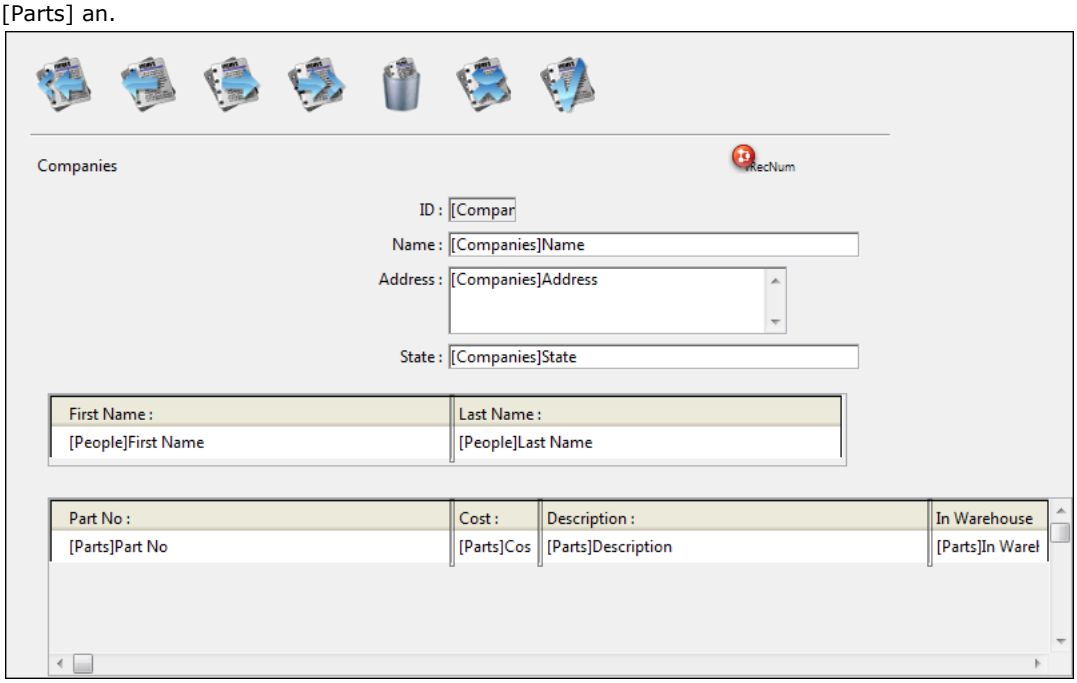
Hinweis: Dieser Befehl unterstützt keine Felder vom Typ Objekt.

Beispiel

Im folgenden Beispiel sind drei Tabellen über automatische Verknüpfung miteinander verbunden. Die beiden Tabellen [People] und [Parts] haben eine Viele-Zu-Eine Verknüpfung zur Tabelle [Companies].



Das Formular für die Tabelle [Companies] zeigt die verknüpften Datensätze aus den beiden anderen Tabellen [People] und [Parts] an.



Sobald die Formulare für Mitarbeiter und Artikel angezeigt werden, werden die verknüpften Datensätze für beide Tabellen geladen und darin zur aktuellen Auswahl.

Dagegen werden die verknüpften Datensätze nicht geladen, wenn ein Datensatz für die Tabelle [Companies] per Programmierung gewählt wird. In diesem Fall müssen Sie den Befehl **RELATE MANY** verwenden.

Hinweise:

- Wird **RELATE MANY** auf eine leere Auswahl angewandt, wird der Befehl nicht ausgeführt und die Auswahl für die Viele-Tabelle ändert sich nicht.
- Damit der Befehl funktioniert, muss das Fremdschlüsselfeld (Viele-Feld) indiziert sein.

Folgende Methode durchläuft jeden Datensatz der Tabelle [Companies] und zeigt für jede Firma eine Meldung an. Sie zeigt die Anzahl der Angestellten in der Firma (die Anzahl der verknüpften Datensätze aus [People]), sowie die Anzahl der Artikel (die Anzahl der verknüpften Datensätze aus [Parts]. Im Beispiel wird das Argument zum Befehl **ALERT** zur besseren Übersicht in mehreren Zeilen angezeigt.

Beachten Sie, dass der Befehl **RELATE MANY** auch bei automatischen Verknüpfungen benötigt wird.

```
ALL RECORDS([Companies]) ` Wähle alle Datensätze in der Tabelle
ORDER BY([Companies];[Companies]Name) ` Sortiere Datensätze alphabetisch
For($i;1;Records in table([Companies])) ` Durchlaufe einmal pro Datensatz
   RELATE MANY([Companies]Name) ` Wähle verknüpfte Datensätze
   ALERT("Firma: "+[Companies]Name+Char(13)+"Mitarbeiter in Firma: "
   +String(Records in selection([People]))+Char(13)+"Anzahl der Artikel: "+String(Records in selection([Parts])))
   NEXT RECORD([Companies]) ` Gehe zum nächsten Datensatz
End for
```

RELATE MANY SELECTION

RELATE MANY SELECTION (Feldname)

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Feld der Viele-Tabelle (Start der Verknüpfung)

Beschreibung

Der Befehl **RELATE MANY SELECTION** erstellt in der Viele-Tabelle eine Auswahl an Datensätzen, ausgehend von der Auswahl in der Eine-Tabelle und lädt den ersten Datensatz der Viele-Tabelle als den aktuellen Datensatz.

Hinweis: RELATE MANY SELECTION ändert den aktuellen Datensatz für die Eine-Tabelle.

Beispiel

Folgendes Beispiel wählt alle offenen Rechnungen an Kunden mit einem Betrag größer oder gleich 1.000 Euro. Das Datenfeld *[Invoices]Customer ID* ist mit dem Datenfeld *[Customers]ID Number* verknüpft.

```
` Wähle die Kunden aus
QUERY([Customers];[Customers]Credit>=1000)
` Finde alle mit diesen Kunden verknüpften Rechnungen
RELATE MANY SELECTION([Invoices]Customer ID)
```

RELATE ONE

RELATE ONE (Viele-Tabelle | Viele-Feld {; Auswahlfeld})

Parameter	Typ		Beschreibung
Viele-Tabelle | Viele-Feld	Tabelle, Feld	➡	Tabelle bzw. Feld, von der die Verknüpfung ausgeht
Auswahlfeld	Feld	➡	Feld der Tabelle, zu der die Verknüpfung geht

Beschreibung

Der Befehl **RELATE ONE** hat zwei Syntaxmöglichkeiten..

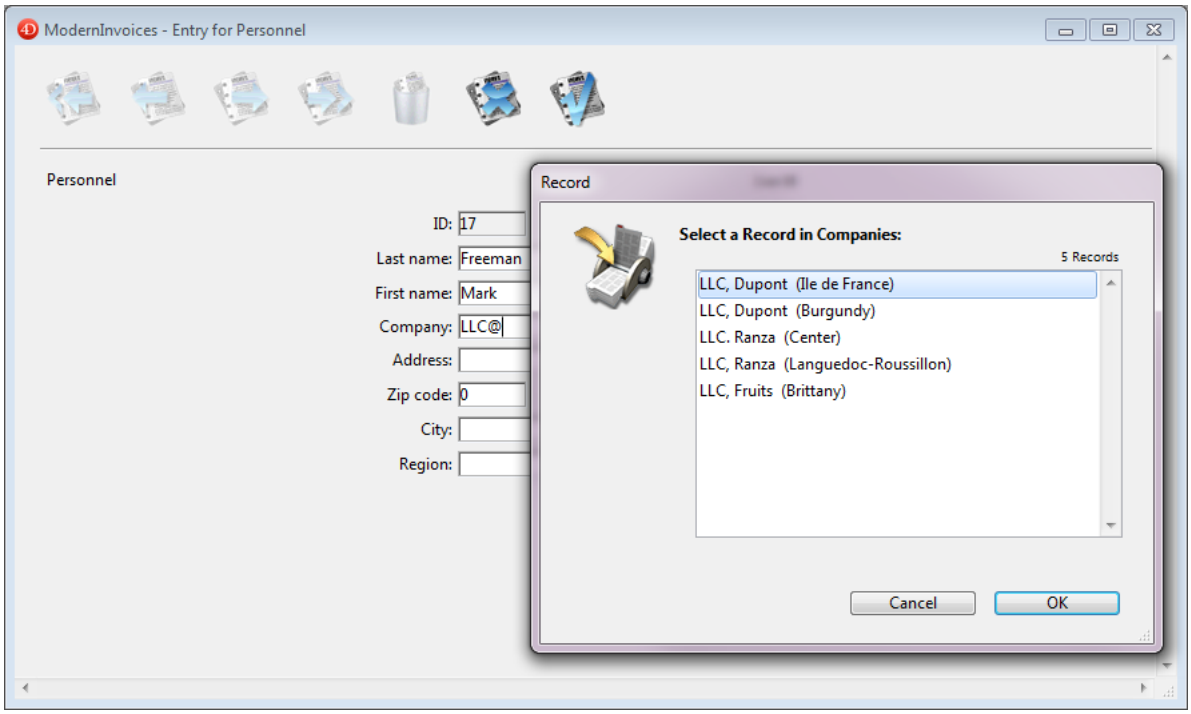
Die erste Syntax, **RELATE ONE**(Viele-Tabelle) lädt alle automatischen Verknüpfungen der Tabelle *Viele-Tabelle* im aktuellen Prozess. Der Befehl lädt für jedes Feld in *Viele-Tabelle* mit einer automatischen Viele-zu-Eine Verknüpfung den verknüpften Datensatz der dazugehörigen Tabelle. Das ändert für diesen Prozess den aktuellen Datensatz in den verknüpften Tabellen.

Die zweite Syntax, **RELATE ONE**(Viele-Feld{;Auswahlfeld}) lädt die Verknüpfung, die vom Datenfeld *Viele-Feld* ausgeht, d.h., sie sucht und lädt einen Datensatz der Tabelle, auf die sich die Verknüpfung bezieht. Sie muss nicht automatisch sein. Dieser Datensatz wird zum aktuellen Datensatz der verknüpften Tabelle. Enthält das aktuelle Formular Felder der aufgerufenen Tabelle, werden diese durch den Aufruf **RELATE ONE** aktualisiert.

Der optionale Parameter *Auswahlfeld* muss ein Feld in der verknüpften Tabelle sein. Er kann nur vom Typ Alpha, Text, Numerisch, Datum, Zeit oder Boolean sein; es kann weder vom Typ Bild noch BLOB sein.
Ist *Auswahlfeld* angegeben und findet 4D in der verknüpften Tabelle mehrere mögliche Datensätze, zeigt **RELATE ONE** eine Liste der Datensätze, die zum Wert in *Viele-Feld* passen, so dass der Benutzer einen Datensatz auswählen kann. Die linke Spalte der Liste zeigt die Werte des verknüpften Feldes, die rechte die Werte des Auswahlfeldes.

Endet *Viele-Feld* mit dem Joker (@), werden u.U. mehrere Datensätze gefunden. Gibt es nur eine Übereinstimmung, erscheint keine Liste.

Im folgenden Beispiel wird ein Datensatz eingegeben und eine Auswahlliste erscheint im Vordergrund:



Mit der folgenden Anweisung wird die Auswahlliste angezeigt:

```
RELATE ONE([Personnel]Company;[Companies]Region)
```

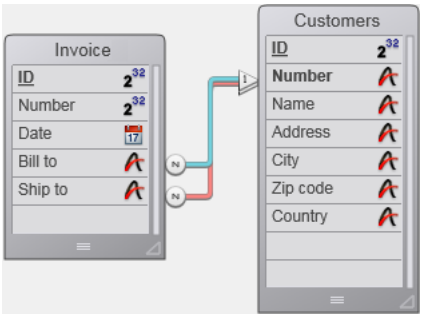
Ein Benutzer hat LLC@ eingegeben, um eine Liste aller Firmen zu erhalten, deren Namen mit LLC beginnen, sowie der dazugehörigen Region.

Das Angeben von *Auswahlfeld* hat denselben Effekt wie die Wahl automatischer Joker im Fenster Verknüpfungseigenschaften. Weitere Informationen dazu finden Sie im Abschnitt **Viele-zu-Eine Optionen** des Handbuchs *4D Designmodus*.

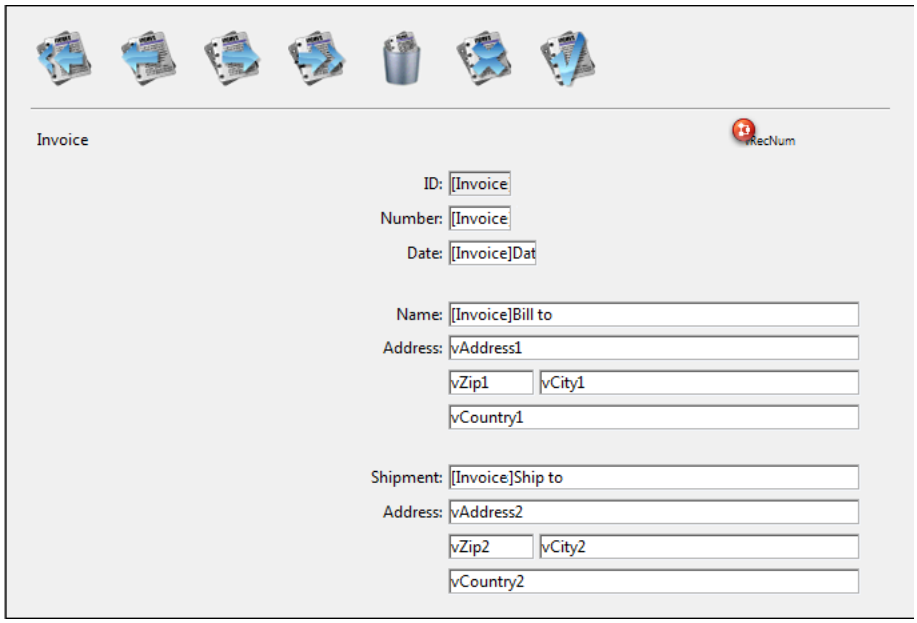
Hinweis: Dieser Befehl unterstützt keine Felder vom Typ Objekt.

Beispiel

Wir gehen aus von einer Tabelle *[Invoices]* verknüpft mit der Tabelle *[Customers]* mit zwei manuellen Verknüpfungen. Die erste geht von *[Invoices]Bill to* zu *[Customers]ID,* die zweite von *[Invoices]Ship to* zu *[Customers]ID*.



Hier das Formular für die Tabelle [Invoices] mit den Angaben "Bill to" und "Ship to":



Da beide Verknüpfungen zur selben Tabelle *[Customers]* gehen, können Sie nicht gleichzeitig die Adresse für die Rechnung und die Adresse für die Lieferung erhalten. Deshalb werden beide Adressen in einem Formular in Variablen gesetzt und jeweils **RELATE ONE** aufgerufen. Würden stattdessen die Datenfelder von *[Customers]* angezeigt, würden nur die Daten von einer der beiden Verknüpfungen angezeigt.

Die folgenden beiden Methoden sind den Feldern *[Invoices]Bill to* und *[Invoices]Ship to* zugeordnet. Sie werden bei Eingabe von Daten ausgeführt.

Die Objektmethode für das Feld *[Invoices]Bill to* lautet:

```
RELATE ONE([Invoices]Bill to)
vAddress1:=[Customers]Address
vCity1:=[Customers]City
vState1:=[Customers]State
vZIP1:=[Customers]ZIP
```

Die Objektmethode für das Feld *[Invoices]Ship to* lautet:

```
RELATE ONE([Invoices]Ship to)
vAddress2:=[Customers]Address
vCity2:=[Customers]City
vState2:=[Customers]State
vZIP2:=[Customers]ZIP
```

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt und die verknüpften Datensätze geladen, wird die Systemvariable OK auf 1 gesetzt. Hat der Benutzer im Dialogfenster der Datenauswahl auf **Abbrechen** geklickt, wird die Variable OK auf 0 (Null) gesetzt. Der Dialog erscheint, wenn der verknüpfte Datensatz geändert wurde.

RELATE ONE SELECTION

RELATE ONE SELECTION (Viele-Tabelle ; Eine-Tabelle)

Parameter	Typ		Beschreibung
Viele-Tabelle	Tabelle	➡	Name der Viele-Tabelle (Start der Verknüpfung)
Eine-Tabelle	Tabelle	➡	Name der Eine-Tabelle (Ende der Verknüpfung)

Beschreibung

Der Befehl **RELATE ONE SELECTION** erstellt eine neue Auswahl an Datensätzen für die Tabelle *Eine-Tabelle*, ausgehend von der Auswahl in der Tabelle *Viele-Tabelle* und lädt den ersten Datensatz der neuen Auswahl als den aktuellen Datensatz.

Dieser Befehl kann nur verwendet werden, wenn eine Verknüpfung von der *Viele-Tabelle* zur *Eine-Tabelle* existiert. **RELATE ONE SELECTION** funktioniert auch auf mehreren Verknüpfungsebenen. Zwischen der *Viele-Tabelle* und der *Eine-Tabelle* können mehrere verknüpfte Tabellen liegen. Die Verknüpfungen können manuell oder automatisch sein.

RELATE ONE SELECTION verwendet den "kürzesten" Pfad, um von der Start- zur Zieltabelle zu gehen. Sind mehrere gleich lange Pfade vorhanden und Sie wollen den verwendeten Pfad prüfen, müssen Sie den Befehl **SET FIELD RELATION** verwenden.

Beispiel

Folgendes Beispiel findet alle Kunden, deren Rechnungen heute fällig sind.

In der Tabelle *[Customers]* wird eine Auswahl erstellt, die von den in der Tabelle *[Invoices]* ausgewählten Datensätzen ausgeht:

```
CREATE EMPTY SET([Customers];"Zahlung ist fällig")
QUERY([Invoices];[Invoices]DueDate=Current date)
While(Not(End selection([Invoices])))
   RELATE ONE([Invoices]CustID)
   ADD TO SET([Customers];"Zahlung ist fällig")
   NEXT RECORD([Invoices])
End while
```

Folgende Technik kommt mit **RELATE ONE SELECTION** zum gleichen Ergebnis:

```
QUERY([Invoices];[Invoices]DueDate=Current date)
RELATE ONE SELECTION([Invoices];[Customers])
```

Hinweis: Seit Version 11 lässt sich dieser Code ohne Performance-Verlust auch folgendermaßen schreiben:

```
QUERY([Customers];[Invoices]DueDate=Current date)
```

SAVE RELATED ONE

SAVE RELATED ONE (Feldname)

Parameter	Typ		Beschreibung
Feldname	Feld	➡	Feld, von dem die Verknüpfung ausgeht

Beschreibung

Der Befehl **SAVE RELATED ONE** sichert den mit *Feldname* verknüpften Datensatz. Führen Sie diesen Befehl aus, um einen Datensatz zu aktualisieren, der mit **CREATE RELATED ONE** erstellt wurde oder um Änderungen an einem Datensatz zu sichern, der mit **RELATE ONE** geladen wurde.

SAVE RELATED ONE sichert keine gesperrten Datensätze. Sie erhalten keine Fehlermeldung. Stellen Sie deshalb sicher, dass die verknüpfte Tabelle im Lese-/Schreibmodus ist und der Datensatz nicht gesperrt ist.

SET AUTOMATIC RELATIONS

SET AUTOMATIC RELATIONS (Eine {; Viele})

Parameter	Typ		Beschreibung
Eine	Boolean	➡	Viele-zu-Eine Verknüpfung
Viele	Boolean	➡	Eine-zu-Viele Verknüpfung

Beschreibung

Der Befehl **SET AUTOMATIC RELATIONS** wandelt temporär alle manuellen Verknüpfungen für die gesamte Datenbank im aktuellen Prozess in automatische um. Die Verknüpfungen bleiben automatisch bis zum erneuten Aufruf von **SET AUTOMATIC RELATIONS**.

- Hat *Eine* den Wert TRUE, werden alle manuellen Viele-zu-Eine Verknüpfungen automatisch. Hat *Eine* den Wert FALSE, werden alle zuvor geänderten Viele-zu-Eine Verknüpfungen wieder zu manuellen Verknüpfungen.
- Dasselbe gilt für den Parameter *Viele*, außer manuelle Eine-zu-Viele Verknüpfungen sind betroffen.

Der Befehl hat keine Auswirkung auf Verknüpfungen, die bereits in der Designumgebung als automatisch definiert wurden.

Sind alle Verknüpfungen in der Designumgebung auf manuell gesetzt, können sie mit diesem Befehl direkt vor bestimmten Operationen, wie z.B. relationale Such- bzw. Sortierläufe, die automatische Verknüpfungen benötigen, auf automatisch gesetzt werden. Ist die Operation ausgeführt, können die Verknüpfungen durch erneuten Aufruf von **SET AUTOMATIC RELATIONS** wieder auf manuell gesetzt werden.

Hinweis: Übergeben Sie **True** im Befehl **SET AUTOMATIC RELATIONS**, wird der automatische Modus für alle manuellen Verknüpfungen während der Sitzung gesperrt. In diesem Fall werden Aufrufe von **SET FIELD RELATION** während derselben Sitzung ignoriert, unabhängig ob sie vor oder nach **SET AUTOMATIC RELATIONS** gesetzt werden. Um den automatischen Modus zu entsperren und die Aufrufe von **SET FIELD RELATION** zu berücksichtigen, übergeben Sie **False** in **SET AUTOMATIC RELATIONS**.

Beispiel

Folgendes Beispiel macht alle manuellen Viele-zu-Eine Verknüpfungen automatisch und macht jede zuvor geänderte Eine-zu-Viele Verknüpfung rückgängig:

```
SET AUTOMATIC RELATIONS(True;False)
```

SET FIELD RELATION

SET FIELD RELATION (Viele-Tabelle | Feld ; Eine ; Viele)

Parameter	Typ		Beschreibung
Viele-Tabelle | Feld	Tabelle, Feld	➡	Ausgangstabelle einer Verknüpfung oder Ausgangsfeld einer Verknüpfung
Eine	Lange Ganzzahl	➡	Status der Viele-zu-Eine Verknüpfung ausgehend von Feld oder Tabelle
Viele	Lange Ganzzahl	➡	Status der Eine-zu-Viele Verknüpfung ausgehend von Feld oder Tabelle

Beschreibung

Der Befehl **SET FIELD RELATION** setzt den Status automatisch/manuell jeder Verknüpfung der Datenbank für den aktuellen Prozess separat, unabhängig vom Anfangsstatus, der in den Verknüpfungseigenschaften der Designumgebung festgelegt wurde.

Im ersten Parameter übergeben Sie einen Tabellen- oder Feldnamen:

- Übergeben Sie einen Feldnamen (*VieleFeld*), gilt der Befehl nur für die Verknüpfung, die vom angegebenen VieleFeld ausgeht.
- Übergeben Sie einen Tabellenname (*VieleTabelle*), gilt der Befehl für alle Verknüpfungen, die von der angegebenen VieleTabelle ausgehen.
- Geht vom VieleFeld oder der VieleTabelle keine Verknüpfung aus, wird der Syntaxfehler No. 16 erzeugt ("Das Feld hat keine Verknüpfung"), die Systemvariable OK wird auf 0 (Null) gesetzt.

In den Parametern *Eine* und *Viele* übergeben Sie einen Wert für die Anderung des Status automatisch/manuell für die angegebene(n) Viele-zu-Eine bzw. Eine-u Viele Verknüpfung(en). Sie können eine Konstante unter dem Thema **Verknüpfungen** verwenden:

- Do not modify (0) = Den aktuellen Status der Viele-zu-Eine Verknüpfung(en) nicht verändern.
- Structure configuration (1) = Die Einstellung für die Viele-zu-Eine Verknüpfung(en) aus dem Strukturfenster der Anwendung verwenden.
- Manual (2) = Viele-zu-Eine Verknüpfung(en) für den aktuellen Prozess auf manuell setzen.
- Automatic (3) = Viele-zu-Eine Verknüpfung(en) für den aktuellen Prozess auf automatisch setzen.

Hinweis: Von diesem Befehl ausgeführte Änderungen gelten nur für den aktuellen Prozess. Sie beeinträchtigen nicht die Einstellung in den Verknüpfungseigenschaften im Designmodus.

Hinweis: Haben Sie in derselben Sitzung **True** im Befehl **SET AUTOMATIC RELATIONS** übergeben, werden Aufrufe von **SET FIELD RELATION** ignoriert, unabhängig, ob sie vor oder nach **SET AUTOMATIC RELATIONS** liegen. Um den automatischen Modus zu sperren und Aufrufe von **SET FIELD RELATION** zu berücksichtigen, übergeben Sie **False** in **SET AUTOMATIC RELATIONS**.

Beispiel

Dieser Befehl vereinfacht die Verwaltung von Verknüpfungen im Schnellberichteditor. In früheren 4D Versionen war es notwendig, alle Verknüpfungen auf automatisch zu setzen, um sie im Editor zu verwenden.
Jetzt ermöglicht der folgende Code, nur sinnvolle Verknüpfungen auf automatisch zu setzen:

```
SET AUTOMATIC RELATIONS(False;False)  `Verknüpfungen neu setzen
  `Nur folgende Verknüpfungen werden verwendet
SET FIELD RELATION([Invoices]Cust_IDt;Automatic;Automatic)
SET FIELD RELATION([Invoice_Row]Invoice_ID;Automatic;Automatic)
QR REPORT([Invoices];Char(1);True;True;True)
```

Verschlüsselung

GENERATE CERTIFICATE REQUEST
GENERATE ENCRYPTION KEYPAIR

GENERATE CERTIFICATE REQUEST

GENERATE CERTIFICATE REQUEST (PrivKey ; ZertAnfrage ; CodeArray ; NameArray)

Parameter	Typ		Beschreibung
PrivKey	BLOB	➡	BLOB mit dem privaten Schlüssel
ZertAnfrage	BLOB	⬅	BLOB zum Empfangen der Zertifikatsanfrage
CodeArray	Array Lange Ganzzahl	➡	Liste mit Informationscode
NameArray	Array String	➡	Namensliste

Beschreibung

Der Befehl **GENERATE CERTIFICATE REQUEST** erstellt eine Zertifikatsanfrage an das PEM (Privacy Enhanced Mail) Format, das direkt von Verisign® oder Thawte® verwendet werden kann. Das Zertifikat spielt eine wichtige Rolle im mit SSL gesicherten Protokoll. Es wird an jeden Browser gesendet, der sich im SSL Modus anmeldet. Es enthält die Kennkarte der Web Site (die im Befehl eingegebene Information), sowie dem öffentlichen Schlüssel, mit dem die Browser die empfangene Information entschlüsseln können. Das Zertifikat enthält außerdem verschiedene Informationen vom Herausgeber des Zertifikats, der für deren Integrität bürgt.

Hinweis: Mehr Informationen zum Einsetzen des SSL Protokolls mit 4D Web Server finden Sie im Abschnitt **TLS Protokoll verwenden (HTTPS)**.

Die Zertifikatsanfrage verwendet das Schlüsselpaar, das mit dem Befehl **GENERATE ENCRYPTION KEYPAIR** erstellt wurde. Sie enthält verschiedene Informationen. Der Herausgeber erstellt das Zertifikat durch Kombinieren dieser Anfrage mit anderen Parametern.

Übergeben Sie in *PrivKey* ein BLOB mit dem privaten Schlüssel, der mit dem Befehl **GENERATE ENCRYPTION KEYPAIR**.

Übergeben Sie in *ZertAnfrage* ein leeres BLOB. Sobald der Befehl ausgeführt wurde, enthält er die Zertifikatsanfrage im PKCS Format, codiert in base64. Sie können diese Anfrage in einer Textdatei speichern, z.B. über den Befehl **BLOB TO DOCUMENT**, um sie dem Herausgeber des Zertifikats vorzulegen.

Warnung: Mit dem privaten Schlüssel erstellen Sie die Anfrage. Dieser sollte jedoch **nicht** an den Herausgeber des Zertifikats gesendet werden.

Die Arrays *CodeArray* (Lange Ganzzahl) und *NameArray* (String) sollten analog dazu mit den Code-Nummern und dem Inhalt der Information gefüllt werden, die der Herausgeber des Zertifikats benötigt.

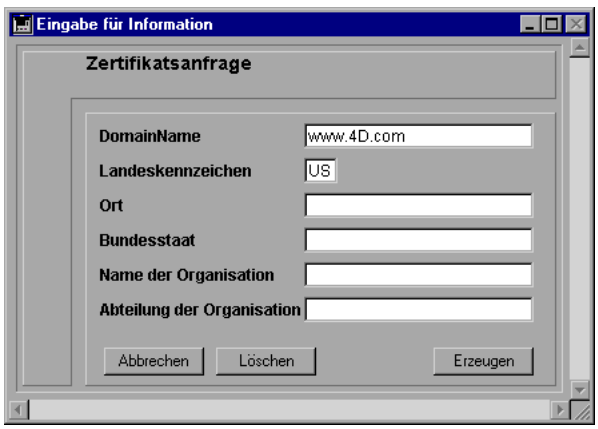
Die erforderlichen Codes und Namen können je nach Herausgeber und Verwendung des Zertifikats wechseln. Bei normaler Verwendung (Web Server Verbindung via SSL) sollten die Arrays folgende Elemente enthalten:

Erforderliche Information	CodeArray	NameArray (Beispiele)
DomainName	13	www.4D.com
Landeskennzeichen (zwei Buchstaben)	14	US
Ort	15	San Jose
Bundesstaat (innerhalb USA)	16	California
Name der Organisation	17	4D, Inc.
Abteilung der Organisation	18	Web Administrator

Die Eingabereihenfolge für Code und Inhalt der Information spielt keine Rolle, die beiden Arrays müssen jedoch zueinander passen: Enthält z.B. das dritte Element von CodeArray den Wert 15 (Ort), sollte das dritte Element von NameArray die dazugehörige Information enthalten, in unserem Beispiel San Jose.

Beispiel

Das Formular "Zertifikatsanfrage" enthält die sechs notwendigen Felder für eine standardmäßige Zertifikatsanfrage. Über die Schaltfläche **Erzeugen** wird auf der Festplatte ein Dokument mit der Zertifikatsanfrage erstellt. Das Dokument "PrivaterSchlüssel.txt", das mit dem Befehl **GENERATE ENCRYPTION KEYPAIR** erstellt wurde, sollte auf der Festplatte sein:



Die Methode für die Schaltfläche **Erzeugen** lautet:

```
` Objektmethode bGenerate

C_BLOB($vbprivateKey;$vbcertifRequest)
```

```
C_LONGINT($tableNber)
ARRAY LONGINT($tLCodes;6)
ARRAY STRING(80;$tSInfos;6)

$tableNber:=Table(Current form table)
For($i;1;6)
    $tSInfos{$i}:=Field($tableNber;$i)->
    $tLCodes{$i}:=$i+12
End for
If(Find in array($tSInfos;"")#-1)
    ALERT("Alle Felder müssen ausgefüllt sein.")
Else
    ALERT("Wähle eigenen Privaten Schlüssel.")
    $vhDocRef:=Open document("")
    If(OK=1)
        CLOSE DOCUMENT($vhDocRef)
        DOCUMENT TO BLOB(Document;$vbprivateKey)
    End if
    GENERATE CERTIFICATE REQUEST($vbPrivateKey;$vbcertifRequest;$tLCodes;$tSInfos)
    BLOB TO DOCUMENT("Request.txt";$vbcertifRequest)
End if
```

GENERATE ENCRYPTION KEYPAIR

GENERATE ENCRYPTION KEYPAIR (PrivKey ; PubKey {; Länge})

Parameter	Typ		Beschreibung
PrivKey	BLOB	⇐	BLOB für den privaten Schlüssel
PubKey	BLOB	⇐	BLOB für den öffentlichen Schlüssel
Länge	Lange Ganzzahl	⇒	Länge des Schlüssels (bits) [386...2048] Standardwert = 512

Beschreibung

Der Befehl **GENERATE ENCRYPTION KEYPAIR** erzeugt ein neues Paar RSA Schlüssel. Das Sicherheitssystem von 4D 6.7 basiert auf diesen Schlüsseln, um Information zu verschlüsseln bzw. entschlüsseln. Diese lassen sich im TLS/SSL Protokoll, mit 4D Web Server (Verschlüsselung und gesicherte Kommunikation) sowie in allen Datenbanken (zur Datenverschlüsselung) verwenden.

Sobald der Befehl ausgeführt ist, enthalten die BLOBs in den Parametern *PrivKey* und *PubKey* ein neues Paar Schlüssel zur Verschlüsselung.

Mit dem optionalen Parameter *Länge* lässt sich die Größe des Schlüssels in Bits festsetzen. Je länger der Schlüssel ist, desto schwieriger lässt sich der Verschlüsselungscode knacken.

Lange Schlüssel benötigen jedoch mehr Zeit zur Ausführung bzw. Beantwortung, insbesondere in einer gesicherten Verbindung.

Die Schlüsselgröße wird standardmäßig, d.h. wenn der Parameter *Länge* nicht angegeben ist, auf 512 Bits festgesetzt. Das ist ein guter Kompromiss für das Verhältnis Sicherheit/Effizienz. Für eine höhere Sicherheit können Sie den Schlüssel öfters wechseln, z.B. alle sechs Monate. Sie können auch Schlüssel mit 2048 Bits erstellen, das verlangsamt jedoch den Aufbau der Web Anwendung.

Dieser Befehl erstellt Schlüssel im PKCS Format, die in base64 codiert sind. So kann deren Inhalt ohne Änderung in ein E-Mail kopiert/eingefügt werden. Sobald das Schlüsselpaar erstellt ist, lässt sich ein Textdokument im PEM Format (Privacy Enhanced Mail) herstellen, z.B. mit dem Befehl **BLOB TO DOCUMENT**. Die Schlüssel können nun an einem sicheren Ort gespeichert werden.

Warnung: Der private Schlüssel sollte immer geheimgehalten werden.

RSA, privater und öffentlicher Schlüssel

Der Befehl **GENERATE ENCRYPTION KEYPAIR** verwendet das RSA Verschlüsselungsverfahren. Es beruht auf einem doppelten Verschlüsselungssystem: Einem privaten Schlüssel und einem öffentlichen Schlüssel. Ein öffentlicher Schlüssel kann - wie schon der Name sagt - an Dritte weitergegeben und zum Entschlüsseln von Informationen verwendet werden. Der öffentliche Schlüssel ist mit einem einmaligen privaten Schlüssel verbunden, mit dem die Information verschlüsselt wird. Demzufolge wird der private Schlüssel zum Verschlüsseln und der öffentliche Schlüssel zum Entschlüsseln verwendet, oder umgekehrt. Die mit einem Schlüssel verschlüsselte Information kann nur mit dem anderen Schlüssel entschlüsselt werden.

Die Verschlüsselungsfunktionalitäten des TLS/SSL Protokolls beruhen auf diesem Prinzip. Der öffentliche Schlüssel ist in dem Zertifikat enthalten, das an die Browser gesendet wird. Weitere Informationen dazu finden Sie im Abschnitt **TLS Protokoll verwenden (HTTPS)**.

Die erste Syntax der Befehle **ENCRYPT BLOB** und **DECRYPT BLOB** arbeitet mit demselben Verschlüsselungssystem. Der öffentliche Schlüssel sollte nur auf vertraulicher Basis zugänglich sein. Öffentliche und private Schlüssel von zwei Personen können für die Verschlüsselung auch vermischt werden. Dann kann allein der Empfänger die Information entschlüsseln und allein der Sender diese verschlüsselt haben. Dieses Prinzip ist in der zweiten Syntax von **ENCRYPT BLOB** und **DECRYPT BLOB** enthalten.

Beispiel

Siehe Beispiel zum Befehl **ENCRYPT BLOB**.

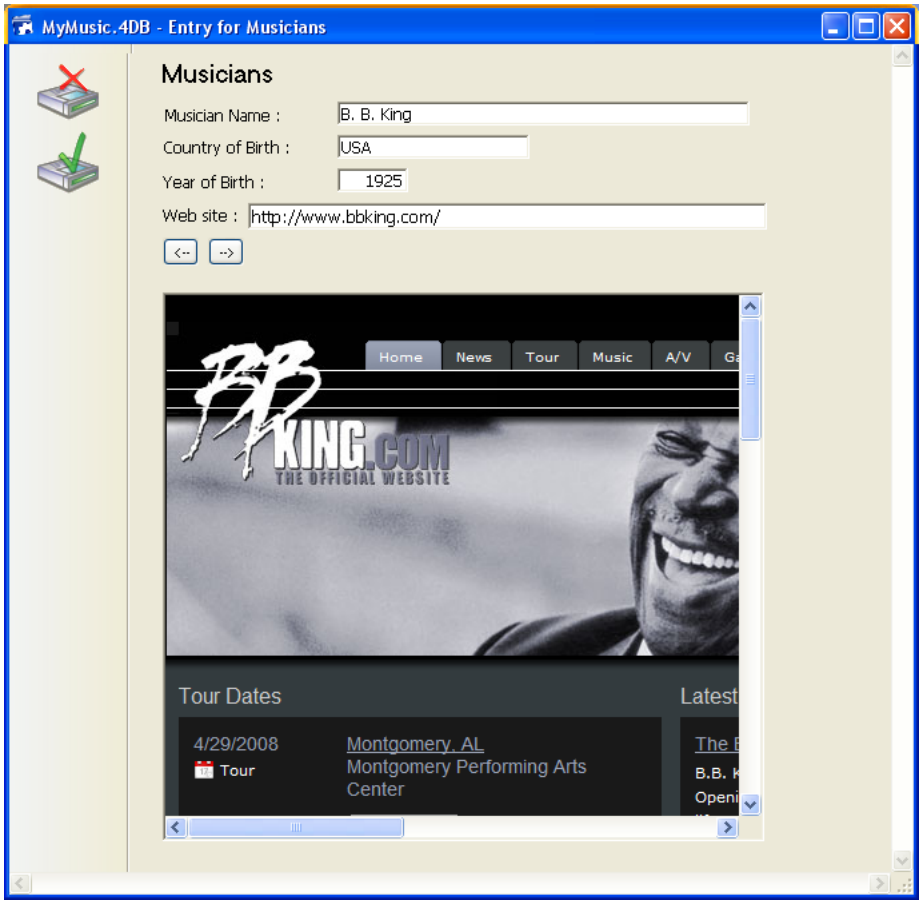
Web Area

- Web Areas per Programmierung verwalten
- 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

Web Areas per Programmierung verwalten

Die Befehle unter diesem Thema steuern Formularobjekte vom Typ Web Area per Programmierung.

Web Areas können verschiedene Arten von Web Inhalt **(*)** in Ihrer 4D Umgebung anzeigen: HTML Seiten mit statischem oder dynamischem Inhalt, Dateien, Bilder, Javascript, ... Das folgende Bild zeigt einen Web Bereich innerhalb eines Formulars mit einer HTML Seite:



(*) Dagegen wird die Verwendung von Web Plug-Ins und Java Applets nicht empfohlen (siehe **Hinweise zum Verwenden von Web Bereichen**).

Zusätzlich zu diesen Befehlen ermöglichen verschiedene Standardaktionen und Formularereignisse, die Funktionsweise der Web Bereiche zu steuern. Über spezifische Variablen lassen sich Informationen zwischen dem Web Bereich und der 4D Umgebung austauschen. Auf diese Weise können Sie in Ihren Formularen einen Web Browser mit Grundfunktionen nutzen.

Web Bereich erstellen und ansprechen

Sie erstellen einen Web Bereich über eine Variante der Schaltfläche Plug-in Bereich/Unterformular in der Objektleiste des 4D Formulareditors. Weitere Informationen dazu finden Sie im Abschnitt **Web Areas** im Handbuch *4D Designmodus*.

Hinweis: Bei Anzeige in einem neuen Prozess, insbesondere über die Funktion **New process**, muss im Parameter *Stapel* der Standardwert (0) verwendet werden, damit der Bereich korrekt angezeigt wird.

Ein Web Bereich hat, wie andere dynamische Formularobjekte, einen Objektnamen und einen Variablennamen, über die er sich per Programmierung steuern lässt (Befehle **OBJECT SET VISIBLE** und **OBJECT MOVE**). Standardmäßig ist eine Variable vom Typ Text zugeordnet.

Hinweis: Die dem Web Bereich zugeordnete Textvariable enthält keine Referenz. Sie kann deshalb nicht als Parameter für eine Methode übergeben werden. Für einen Web Bereich mit Namen *MyArea* ist z.B. der folgende Code nicht verwendbar:

```
Mymethod(MyArea)
```

Code für Mymethod:

```
WA REFRESH CURRENT URL($1)\\Das funktioniert nicht
```

Für diese Art der Programmierung müssen Sie Zeiger verwenden:

```
Mymethod(->MyArea)
```

Code für *Mymethod*:

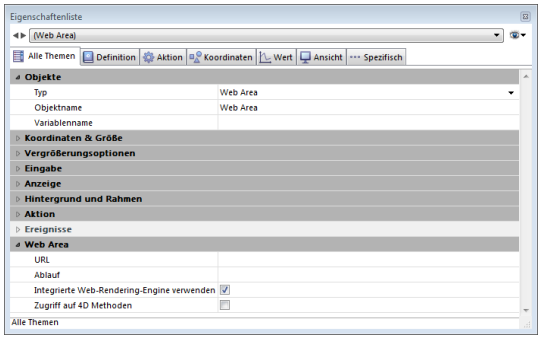
```
WA REFRESH CURRENT URL($1->)\\Das funktioniert
```

Zugewiesene Variablen verwalten

Zusätzlich zur standardmäßigen Objektvariablen (siehe voriger Abschnitt) werden jedem Web Bereich automatisch zwei spezifische Variablen zugewiesen:

- Die Variable "URL"
- Die Variable "Ablauf"

Sie können beliebige Namen vergeben. Über die Eigenschaftenliste können Sie auf die Variablen zugreifen:



Variable URL

"URL" ist eine Variable vom Typ String. Sie enthält die geladene URL oder die vom zugeordneten Web Bereich gerade geladene URL.
Die Zuweisung zwischen Variable und Web Bereich erfolgt in beiden Richtungen:

- Weist der Benutzer der Variablen eine neue URL zu, lädt der Web Bereich automatisch diese URL.
- Alles Navigieren innerhalb des Web Bereichs aktualisiert automatisch den Inhalt der Variablen.
  Vom Prinzip her funktioniert diese Variable wie der Adressbereich eines Web Browsers. Das können Sie in einem Textbereich über dem Web Bereich darstellen.

Variable URL und Befehl WA OPEN URL
Die Variable URL erzeugt dieselbe Wirkung wie der Befehl **WA OPEN URL**. Es gibt jedoch folgende Unterschiede:

- Um auf Dokumente zuzugreifen, akzeptiert diese Variable nur RFC kompatible URLs ("file://c:/My%20Doc") und keine System Pfadnamen ("c:\MyDoc"). **WA OPEN URL** akzeptiert beide Bezeichnungen.
- Enthält die Variable URL einen leeren String, versucht der Web Bereich nicht, die URL zu laden, **WA OPEN URL** erzeugt einen Fehler.
- Enthält die Variable URL kein Protokoll (http, mailto, file, etc.), fügt der Web Bereich "http://" hinzu, **WA OPEN URL** jedoch nicht.
- Wird der Web Bereich nicht im Formular angezeigt (wenn er im Formular auf einer anderen Seite liegt), hat der Befehl **WA OPEN URL** keine Auswirkung. Durch Zuweisen eines Wertes an die Variable URL lässt sich jedoch die aktuelle URL aktualisieren.

Variable Ablauf

Die Variable "Ablauf" ist vom Typ Lange Ganzzahl. Sie enthält einen Wert zwischen 0 und 100. Das ist der Prozentsatz beim Laden der Seite, die im Web Bereich angezeigt wird. 4D aktualisiert diese Variable automatisch. Sie lässt sich nicht manuell verändern

Zugriff auf 4D Methoden

Sie können 4D Methoden über JavaScript Code aufrufen, der in einem Web Bereich ausgeführt wird und Werte zurückerhalten.
Wichtig: Dieses Feature ist nur verfügbar, wenn der Web Bereich die eingebundene Web Rendering Engine verwendet.

Web Area konfigurieren

Damit Sie 4D Methoden aus einem Web Bereich aufrufen können, müssen Sie in der Eigenschaftenliste unter der Gruppe "Web Area" die Option **Zugriff auf 4D Methoden** aufrufen:

Hinweis: Diese Option erscheint nur, wenn die Option **Integrierte Web Rendering Engine verwenden** markiert ist.
Ist diese Eigenschaft markiert, wird im Web Bereich ein spezifisches JavaScript Objekt ($4d) eingerichtet, über das Sie Aufrufe von 4D Projektmethoden steuern können.

Objekt $4d verwenden

Ist die Option Zugriff auf 4D Methoden markiert, bietet das in 4D integrierte Web Kit im Web Bereich ein JavaScript Objekt mit Namen $4d, das Sie jeder 4D Projektmethode mit "." voranstellen können.
Um z.B. die 4D Methode ***HelloWorld*** aufzurufen, führen Sie einfach folgende Anweisung aus:

```
$4d.HelloWorld();
```

Warnung: JavaScript unterscheidet zwischen Groß- und Kleinschreibung. Achten Sie deshalb darauf, dass das Objekt $4d lautet (mit kleinem "d").
Die Syntax zum Aufrufen von 4D Methoden lautet:

```
$4d.4DMethodName(param1,paramN,function(result){})
```

- *param1...paramN*: Sie können in der 4D Methode soviel Parameter übergeben, wie benötigt werden und jeden Typ wählen, den JavaScript unterstützt (String, Zahl, Array, Objekt).
- *function(result)*: Funktion, die als letztes Argument übergeben wird. Diese "callback" Funktion wird synchron aufgerufen, wenn die Ausführung der 4D Methode endet. Sie empfängt den Parameter:
  - *result*: Das Ausführungsergebnis der 4D Methode, das in "$0" zurückgegeben wird.
    Dieses Ergebnis kann in jedem Typ sein, den JavaScript unterstützt, also String, Zahl, Array, Objekt. Für Objekte können Sie den Befehl **C_OBJECT** verwenden.
    Hinweis: 4D läuft standardmäßig in UTF-8. Geben Sie Text mit Sonderzeichen, wie z.B. Zeichen mit Akzenten zurück, müssen Sie sicherstellen, dass die Codierung der Seite, die im Web Bereich angezeigt wird, als UTF-8 deklariert ist, damit auch diese Zeichen korrekt gerendert werden. Fügen Sie dafür auf der HTML Seite folgende Zeile zur Codierung hinzu:
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

Beispiel 1

Wir nehmen eine 4D Projektmethode mit Namen ***today***. Sie empfängt keine Parameter, gibt aber das aktuelle Datum als Zeichenkette zurück.
4D Code der Methode ***today***:

```
C_TEXT($0)
$0:=String(Current date;System date long)
```

Die 4D Methode lässt sich im Web Bereich mit folgender Syntax aufrufen:

```
$4d.today()
```

Die 4D Methode empfängt keine Parameter, gibt aber den Wert von $0 in der Callback Funktion zurück, die 4D nach Ausführen der Methode aufruft.
Wir wollen das Datum in der HTML Seite anzeigen, die der Web Bereich lädt.
Hier der Code der HTML Seite:

```
<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>Heute ist: <div id="mydiv"></div> </body> </html>
```

Beispiel 2

Die 4D Projektmethode ***calcSum*** empfängt Parameter ($1...$n) und gibt deren Summe in $0 zurück:
4D Code der Methode ***calcSum***:

```
C_REAL($1) // empfängt n Parameter vom Typ Zahl
C_REAL($0) // gibt eine Zahl zurück
C_LONGINT($i;$n)
$n:=Count parameters
For($i;1;$n)
   $0:=$0+$1
End for
```

Der JavaScript Code im Web Bereich ist:

```
$4d.calcSum(33, 45, 75, 102.5, 7, function(dollarZero)      {           var result = dollarZero // Ergebnis ist 262.5      });
```

Formularereignisse

Es gibt spezifische Formularereignisse, um Web Bereiche per Programmierung zu steuern, insbesondere zur Aktivierung von Links:

- 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

Darüberhinaus unterstützen Web Bereiche folgende generische Formularereignisse:

- On Load
- On Unload

- On Getting Focus
- On Losing Focus

Weitere Informationen dazu finden Sie unter der Funktion **Form event**.

Zugriff auf Web Inspektor

Sie können in Web Bereichen Ihrer Formulare den Web Inspektor ansehen und verwenden. Das ist ein Debugger innerhalb der eingebundenen Web Engine, mit dem Sie Code und Informationsfluss der Web Seiten analysieren können.

Web Inspektor anzeigen

Sie müssen folgende Einstellungen machen, damit der Web Inspektor in einem Web Bereich erscheint:

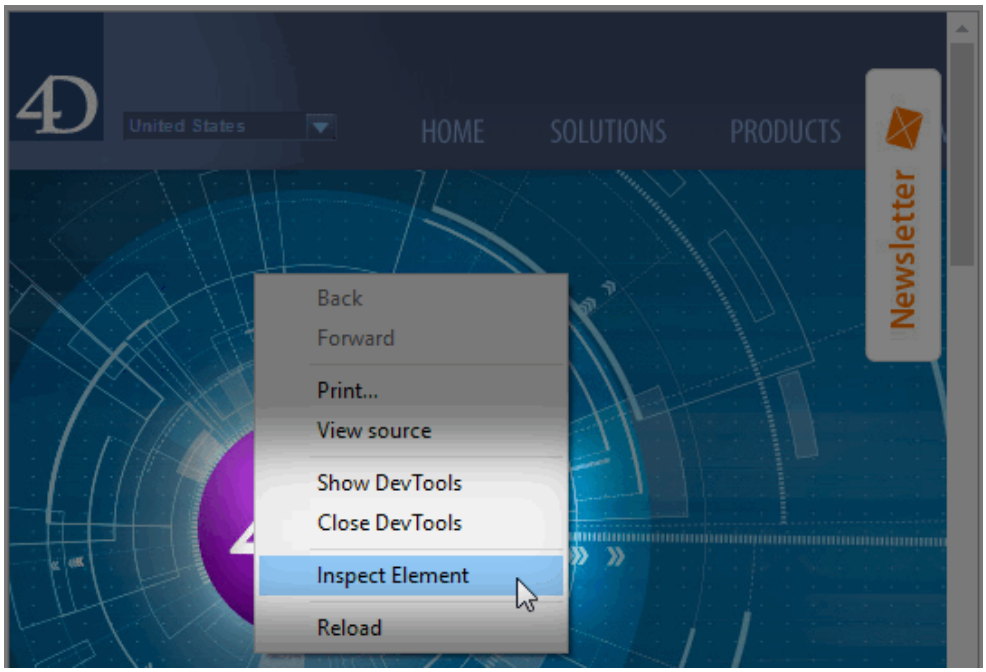
- Sie müssen die eingebundene Web Rendering Engine für den Bereich auswählen (der Web Inspektor ist nur mit dieser Konfiguration verfügbar - siehe Abschnitt **Integrierte Web Rendering-Engine verwenden** im Handbuch *4D Designmodus*)
- Sie müssen das Kontextmenü für den Bereich aktivieren (der Inspektor wird über dieses Menü aufgerufen - siehe **Kontextmenü** im Handbuch *4D Designmodus*)
- Sie müssen explizit die Verwendung des Inspektors im Bereich aktivieren. Dafür schreiben Sie folgende Anweisung:

```
WA SET PREFERENCE(*;"WA";WA enable Web inspector;True)
```

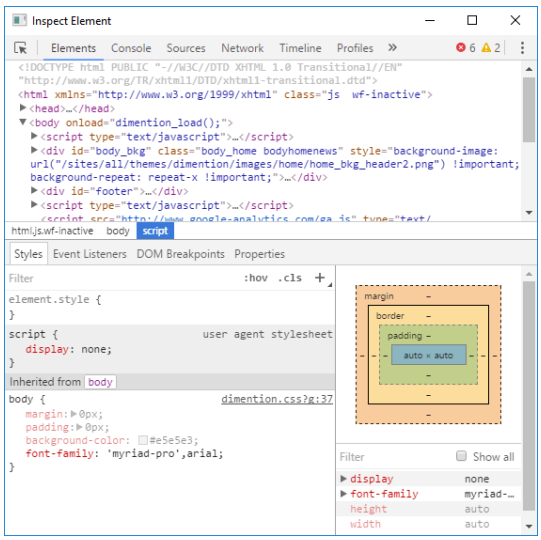
Weitere Informationen dazu finden Sie unter dem Befehl **WA SET PREFERENCE**.

Web Inspektor einsetzen

Haben Sie die Einstellungen wie oben beschrieben ausgeführt, erscheint im Kontextmenü des Bereichs der Eintrag **Inspect Element**:



Wählen Sie diesen Eintrag, erscheint das Fenster des Web Inspektors:



Der Web Inspektor ist in der eingebundenen Web Rendering Engine enthalten. Ausführliche Informationen zu den Features dieses Debugger finden Sie in der Dokumentation zur Web Rendering Engine.

Hinweise zum Verwenden von Web Bereichen

Benutzeroberfläche

Beim Ausführen des Formulars sind im Web Bereich die Standardfunktionen der Browser-Oberfläche verfügbar, die Interaktionen mit anderen Formularbereichen ermöglichen:

- **Befehle des Menüs Bearbeiten:** Hat der Web Bereich Fokus, lassen sich die Befehle des Menüs Bearbeiten zum Kopieren, Einsetzen, Alles auswählen, etc. verwenden.
- **Kontextmenü:** Das standardmäßige Kontextmenü des Systems lässt sich auch im Web Bereich verwenden (siehe **Kontextmenü**). Die Anzeige lässt sich über den Befehl **WA SET PREFERENCE** steuern.
- **Drag and Drop:** Der Benutzer kann Text, Bilder und Dokumente innerhalb eines Web Bereichs oder zwischen Web Bereichen und 4D Formularobjekten per Drag and Drop bewegen, je nach den zugewiesenen 4D Objekteigenschaften. Aus Sicherheitsgründen lässt sich ab 4D v14 R2 in einen Web Bereich standardmäßig keine Datei oder URL per Drag&Drop-Technik einfügen. In diesem Fall zeigt der Mauszeiger das Icon  . Sie müssen den Befehl **WA SET PREFERENCE** verwenden, um das Ziehen von URLs oder Dateien in den Bereich explizit zu erlauben.

Unterformulare

Bedingt durch den erneuten Fensteraufbau gibt es beim Einfügen eines Web Bereichs in ein Unterformular folgende Einschränkungen:

- Das Unterformular darf nicht scrollbar sein
- Der Web Bereich darf nicht größer als das Unterformular selbst sein

Konflikt Web Area und Web Server (Windows)

Unter Windows raten wir davon ab, über einen Web Bereich auf den Web Server der 4D Anwendung zuzugreifen, der diesen Bereich enthält. Diese Konfiguration kann zu einem Konflikt führen, der die Anwendung einfriert. Ein Remote 4D kann natürlich auf den Web Server von 4D Server zugreifen, jedoch nicht auf seinen eigenen Web Server.

Web Plug-Ins und Java Applets

Wir empfehlen, Web Plug-Ins und Java Applets in Web Areas **nicht zu verwenden**, da dies zu instabiler Arbeitsweise von 4D führen kann, insbesondere beim Verwalten von Ereignissen.

Protokoll in URL (Mac OS)

URLs, die auf Mac OS in Web Bereichen per Programmierung gesteuert werden, müssen mit dem Protokoll beginnen. Beispiel: Sie müssen den vollständigen String "http://www.mysite.com" und nicht nur "www.mysite.com" angeben.

WA Back URL available

WA Back URL available ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String) Ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	Boolean	⮌	Wahr, wenn in der Sequenz der geöffneten URL eine vorige URL existiert, sonst Falsch

Beschreibung

Die Funktion **WA Back URL available** findet heraus, ob in der Sequenz der geöffneten URLs im Web Bereich, definiert durch die Parameter * und *Objekt*, eine vorige URL verfügbar ist.

Die Funktion gibt **Wahr** zurück, wenn eine vorige URL existiert, sonst **Falsch**. Sie ist insbesondere in einer eigenen Oberfläche hilfreich, um die Schaltflächen zum Navigieren zu aktivieren bzw. deaktivieren.

WA Create URL history menu

WA Create URL history menu ({* ;} Objekt {; Richtung}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname, Ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Richtung	Lange Ganzzahl	➡	0 oder ohne Wert=Liste der vorigen URLs, 1=Liste der nächsten URLs
Funktionsergebnis	MenüRef	⮌	Referenz des Menüs

Beschreibung

Die Funktion **WA Create URL history menu** erstellt und füllt ein Menü, über das Sie direkt in den URLs navigieren können, die während einer Sitzung im Web Bereich, definiert durch die Parameter * und *Objekt*, aufgerufen werden. Damit können Sie eine eigene Oberfläche zum Navigieren einrichten.

Die gelieferte Information betrifft die Sitzung; das ist die Navigation, die im gleichen Web Bereich ausgeführt wird, solange das Formular nicht geschlossen wird.

Übergeben Sie in *Richtung* einen Wert für die Richtung der angezeigten URLs. Sie können eine der folgenden Konstanten unter dem Thema **Web Area** verwenden:

Konstante	Typ	Wert
WA next URLs	Lange Ganzzahl	1
WA previous URLs	Lange Ganzzahl	0

Geben Sie den Parameter *Richtung* nicht an, wird der Wert 0 verwendet.

Ist das Menü angelegt, können Sie es über die 4D Funktion **Dynamic pop up menu** anzeigen und mit den standardmäßigen 4D Befehlen zur Menüverwaltung verwenden. Die zurückgegebene Zeichenkette enthält die URL der aufgerufenen Seite (siehe Beispiel).

Rufen Sie den Befehl **RELEASE MENU** auf, um das Menü mit der URL Historie zu löschen, wenn es nicht länger benötigt wird.

Beispiel

Der folgende Code lässt sich einer 3D Schaltfläche mit dem PopUp-Menü "Previous" zuordnen:

```
Case of
  `Einfacher Klick
    :(Form event=On Clicked)
      WA OPEN BACK URL(WA_area)
  `Bei Klick auf Pfeil -> PopUp anzeigen
    :(Form event=On Alternative Click)
  `Create a previous history menu
      $Menu:=WA Create URL history menu(WA_area;WA previous URLs)
  `Dieses Menü als PopUp anzeigen
      $URL:=Dynamic pop up menu($Menu)
  `Wurde ein Eintrag ausgewählt
      If($URL#"")
  ` Web Seite öffnen
          WA OPEN URL(WA_area;$URL)
      End if
  `Menü löschen, um Speicher freizumachen
      RELEASE MENU($Menu)
End case
```

WA Evaluate JavaScript

WA Evaluate JavaScript ({* ;} Objekt ; jsCode {; Typ}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
jsCode	String	➡	JavaScript Code
Typ	Lange Ganzzahl	➡	Ergebnis in anderen Typ konvertieren
Funktionsergebnis	Datum, Objekt, Text, Zahl, Zeiger, Zeit	➲	Ergebnis der Ausführung

Beschreibung

Die Funktion **WA Evaluate JavaScript** führt im Web Bereich, definiert durch die Parameter * und *Objekt*, den in *jsCode* übergebenen JavaScript Code aus und gibt das Ergebnis zurück.
Sie muss nach dem Laden der Seite aufgerufen werden (das Formularereignis On End URL Loading muss generiert werden).

Standardmäßig gibt sie Werte als String zurück. Mit dem optionalen Parameter *Typ* können Sie einen anderen Typ definieren. Dafür können Sie eine der nachfolgenden Konstanten unter dem Thema **Feld und Variablentypen** verwenden:

Konstante	Typ	Wert
Is Boolean	Lange Ganzzahl	6
Is collection	Lange Ganzzahl	42
Is date	Lange Ganzzahl	4
Is longint	Lange Ganzzahl	9
Is object	Lange Ganzzahl	38
Is real	Lange Ganzzahl	1
Is text	Lange Ganzzahl	2
Is time	Lange Ganzzahl	11

Beispiel 1

Der folgende JavaScript Code bewirkt die Anzeige des vorigen URL:

```
$result:=WA Evaluate JavaScript(MyWArea;"history.back()")
```

Beispiel 2

Dieses Beispiel zeigt einige Bewertungen mit Konvertierung der empfangenen Werte.
In einer HTML Datei gesetzte JavaScript Funktionen:

```
<!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>
```

In der 4D Formularmethode schreiben Sie:

```
If(Form event=On Load)
   WA OPEN URL(*;"Web Area";"C:\\myDatabase\\index.html")
End if
```

Sie können dann den JavaScript Code von 4D aus ausführen:

```
$Eval1:=WA Evaluate JavaScript(*;"Web Area";"evalLong()";ls longint)
 //$Eval1 = 123
 //$Eval1 = "123" wenn Typ weggelassen
$Eval2:=WA Evaluate JavaScript(*;"Web Area";"evalText()";ls text)
 //$Eval2 = "456"
$Eval3:=WA Evaluate JavaScript(*;"Web Area";"evalObject()";ls object)
 //$Eval3 = {"a":1,"b":"hello world"}
$Eval4:=WA Evaluate JavaScript(*;"Web Area";"evalDate()";ls date)
 // $Eval4 = 06/21/13
 // $Eval4 = "2013-06-21T14:45:09.694Z" wenn Typ weggelassen
```

WA EXECUTE JAVASCRIPT FUNCTION

WA EXECUTE JAVASCRIPT FUNCTION ({* ;} Objekt ; jsFunktion ; Ergebnis | * {; Parameter}{; Parameter2 ; ... ; ParameterN})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
jsFunktion	String	➡	Name der auszuführenden JavaScript Funktion
Ergebnis | *	Variable	➡	* für eine Funktion ohne Ergebnis oder
		⬅	Funktionsergebnis (wenn erwartet)
Parameter	String, Zahl, Datum, Objekt, Collection	➡	Der zu übergebende Parameter

Beschreibung

Der Befehl **WA EXECUTE JAVASCRIPT FUNCTION** führt im Web Bereich, definiert durch die Parameter * und *Objekt*, die JavaScript Funktion *jsFunction* aus und gibt optional im Parameter *Ergebnis* das Ergebnis zurück.

Für eine Funktion ohne Ergebnis übergeben Sie * im Parameter *Ergebnis*.

Sie können in *Parameter* einen oder mehrere Strings mit den Parametern der Funktion übergeben.

Der Befehl unterstützt verschiedene Typen für Eingabe (*Parameter*) und Ausgabe (*Ergebnis*). Sie können Daten vom Typ String, Zahl, Datum, Objekt und Collection übergeben und erhalten.

Beispiel 1

Eine JavaScript Funktion mit 3 Parametern aufrufen:

```
$JavaScriptFunction:="AuszufuehrendeFunktion"
$Param1:="10"
$Param2:="wahr"
$Param3:="1,000.2"  ` "," als Trenner für Tausend und "." als Dezimaltrenner werten

WA EXECUTE JAVASCRIPT FUNCTION(MyWArea;$JavaScriptFunction;$Result;$Param1;$Param2;$Param3)
```

Beispiel 2

Die JavaScript Funktion "getCustomerInfo" empfängt eine ID Nummer als Parameter und gibt ein Objekt zurück:

```
C_OBJECT($Result)
C_LONGINT($ID)
$ID:=1000
WA EXECUTE JAVASCRIPT FUNCTION(*;"WA";"getCustomerInfo";$Result;$ID)
```

WA Forward URL available

WA Forward URL available ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	→	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	→	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	Boolean	←	Wahr, wenn es eine nächste URL in der Sequenz der geöffneten URLs gibt; sonst Falsch

Beschreibung

Die Funktion **WA Forward URL available** findet heraus, ob eine nächste URL in der Sequenz der geöffneten URLs im Web Bereich, definiert durch die Parameter * und *Objekt*, vorhanden ist.

Die Funktion gibt **Wahr** zurück, wenn eine URL vorhanden ist, sonst **Falsch**. Sie ist insbesondere in einer eigenen Oberfläche hilfreich, um die Schaltflächen zum Navigieren zu aktivieren bzw. deaktivieren.

WA Get current URL

WA Get current URL ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	String	⮌	Aktuell in den Web Bereich geladene URL

Beschreibung

Die Funktion **WA Get current URL** gibt die URL Adresse der im Web Bereich angezeigten Seite zurück, definiert durch die Parameter * und *Objekt*.

Ist die aktuelle URL nicht verfügbar, gibt die Funktion einen leeren String zurück.

Wurde die Web Seite vollständig geladen, ist der von der Funktion zurückgegebene Wert derselbe wie der Wert der Variablen "URL", die dem Web Bereich zugeordnet ist. Während dem Ladevorgang der Seite sind die Werte dagegen unterschiedlich: Die Funktion gibt die vollständig geladene URL zurück, während die Variable die URL enthält, die gerade geladen wird.

Beispiel

Die angezeigte Seite ist die URL "www.apple.com"; die Seite, die gerade geladen wird, ist "www.4d.com":

```
$url:=WA Get current URL(MyWArea)  `gibt "http://www.apple.com" zurück
  `Die zugewiesene URL Variable enthält "http://www.4d.com"
```

WA GET EXTERNAL LINKS FILTERS

WA GET EXTERNAL LINKS FILTERS ({* ;} Objekt ; FilterArr ; ErlaubenVerweigernArr)

Parameter	Typ		Beschreibung
*	Operator	⇒	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	⇒	Objektname (mit *) oder Variable (ohne *)
FilterArr	Array String	⇐	Filter Array
ErlaubenVerweigernArr	Array Boolean	⇐	Erlauben-Verweigern Array

Beschreibung

Der Befehl **WA GET EXTERNAL LINKS FILTERS** gibt in den Arrays *FilterArr* und *ErlaubenVerweigernArr* die Filter für externe Links des Web Bereichs zurück, definiert durch die Parameter * und *Objekt*. Ist kein Filter aktiv, werden die Arrays leer zurückgegeben.

Die Filter werden mit dem Befehl **WA SET EXTERNAL LINKS FILTERS** installiert. Werden die Arrays während der Sitzung erneut initialisiert, lassen sich die aktuellen Einstellungen über den Befehl **WA GET EXTERNAL LINKS FILTERS** herausfinden.

WA Get last filtered URL

WA Get last filtered URL ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	→	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	→	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	String	←	Zuletzt gefilterte URL

Beschreibung

Die Funktion **WA Get last filtered URL** gibt die zuletzt gefilterte URL im Web Bereich zurück, definiert durch die Parameter * und *Objekt*.

Wird die URL gefiltert, kann das folgende Gründe haben:

- Die URL wurde wegen eines Filters verweigert (Befehl **WA SET URL FILTERS**),
- Der Link wird im standardmäßigen Browser geöffnet (Befehl **WA SET EXTERNAL LINKS FILTERS**),
- Die URL versucht, ein PopUp-Fenster zu öffnen.

Sie sollten diesen Befehl im Rahmen der Formularereignisse On URL Filtering, On Open External Link und On Window Opening Denied aufrufen, um die gefilterte URL herauszufinden.

WA GET LAST URL ERROR

WA GET LAST URL ERROR ({* ;} Objekt ; Url ; Beschreibung ; Fehlernr)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Url	String	⬅	URL am Ursprung des Fehlers
Beschreibung	String	⬅	Fehlerbeschreibung (Mac OS)
Fehlernr	Lange Ganzzahl	⬅	Fehlernummer

Beschreibung

Der Befehl **WA GET LAST URL ERROR** ermöglicht, verschiedene Informationen über den zuletzt aufgetretenen Fehler im Web Bereich, definiert durch die Parameter * und *Objekt*, herauszufinden.

Diese Information wird in drei Variablen zurückgegeben:

- *Url*: URL, die den Fehler verursacht.
- *Beschreibung* (nur Mac OS): Text, der den Fehler beschreibt. Ist dem Fehler kein Text zugeordnet, wird ein leerer String zurückgegeben. Unter Windows wird dieser Parameter immer leer zurückgegeben.
- *Fehlernr*: Fehlernummer.
  - Ist die Nummer größer/gleich 400, bezieht sich der Fehler auf das HTTP Protokoll. Weitere Informationen dazu finden Sie unter:
    http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
  - Sonst wird ein Fehler von WebKit (Mac OS) oder ActiveX (Windows) zurückgegeben.

Wir empfehlen, diesen Befehl innerhalb des Formularereignisses On URL Loading Error aufzurufen, um die Ursache des gerade aufgetretenen Fehlers herauszufinden.

WA Get page content

WA Get page content ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	String	⮌	HTML Quellcode

Beschreibung

Die Funktion **WA Get page content** gibt den HTML Code der aktuellen Seite oder der Seite zurück, die im Web Bereich, definiert durch die Parameter * und *Objekt*, angezeigt wird.

Die Funktion gibt einen leeren String zurück, wenn der Inhalt der aktuellen Seite nicht verfügbar ist.

WA Get page title

WA Get page title ({* ;} Objekt) -> Funktionsergebnis

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Funktionsergebnis	String	⮌	Titel der aktuellen Seite

Beschreibung

Die Funktion **WA Get page title** gibt den Titel der aktuellen Seite oder der Seite, die im Web Bereich, definiert durch die Parameter * und *Objekt*, angezeigt wird. Der Titel entspricht dem HTML Tag "Title".

Die Funktion gibt einen leeren String zurück, wenn für die aktuelle URL kein Titel verfügbar ist.

WA GET PREFERENCE

WA GET PREFERENCE ({* ;} Objekt ; Selector ; Wert)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Selector	Lange Ganzzahl	➡	Zu erhaltende Voreinstellung
Wert	Variable	⬅	Aktueller Wert der Voreinstellung

Beschreibung

Die Funktion **WA GET PREFERENCE** gibt den aktuellen Wert der Voreinstellung im Web Bereich zurück, definiert durch die Parameter * und *Objekt*.

Übergeben Sie die Voreinstellung, deren Wert Sie im Parameter *Selector* erhalten wollen. Sie können eine Konstante unter dem Thema **Web Area** übergeben:

Konstante	Typ	Wert	Kommentar
WA enable contextual menu	Lange Ganzzahl	4	Erlaubt die Anzeige eines Standard Kontextmenü im Web Bereich
WA enable Java applets	Lange Ganzzahl	1	Erlaubt die Ausführung von Java applets im Web Bereich.
WA enable JavaScript	Lange Ganzzahl	2	Erlaubt die Ausführung von JavaScript Code im Web Bereich
WA enable plugins	Lange Ganzzahl	3	Erlaubt die Installation von Plug-Ins im Web Bereich
WA enable URL drop	Lange Ganzzahl	101	Erlaubt Ziehen einer URL oder Datei in Web Area (Standard = **False**)
WA enable Web inspector	Lange Ganzzahl	100	Erlaubt die Anzeige des Web Inspektors im Web Bereich

Weitere Informationen zu diesen Voreinstellungen finden Sie in der Beschreibung zum Befehl **WA SET PREFERENCE**.

Im Parameter *Wert* übergeben Sie eine Variable, die den aktuellen Wert der Voreinstellung empfängt. Die Art der Variable richtet sich nach der Voreinstellung. *Wert* ist immer vom Typ Boolean: Er enthält **True**, wenn die Voreinstellung aktiv ist, sonst **False**.

WA GET URL FILTERS

WA GET URL FILTERS ({* ;} Objekt ; FilterArr ; ErlaubenVerweigernArr)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
FilterArr	Array String	⬅	Filter Array
ErlaubenVerweigernArr	Array Boolean	⬅	ErlaubenVerweigern Array

Beschreibung

Der Befehl **WA GET URL FILTERS** gibt in den Arrays *FilterArr* und *ErlaubenVerweigernArr* die Filter zurück, die im Web Bereich, definiert durch die Parameter * und *Objekt*, aktiv sind. Sind keine Filter aktiv, werden die Arrays leer zurückgegeben.

Die Filter werden vom Befehl **WA SET URL FILTERS** installiert. Werden die Arrays während der Sitzung erneut initialisiert, können Sie über den Befehl **WA GET URL FILTERS** die aktuellen Einstellungen herausfinden.

WA GET URL HISTORY

WA GET URL HISTORY ({* ;} Objekt ; UrlsArr {; Richtung {; TitelArr}})

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
UrlsArr	Array String	⬅	Array der besuchten URLs
Richtung	Lange Ganzzahl	➡	0 oder ohne Wert=Liste der vorigen URLs, 1=Liste der nächsten URLs
TitelArr	Array String	⬅	Array der Fenstertitel

Beschreibung

Der Befehl **WA GET URL HISTORY** gibt ein oder zwei Arrays mit den URLs zurück, die während der Sitzung im Web Bereich, definiert durch die Parameter * und *Objekt*, aufgerufen werden. Er lässt sich auch zum Aufbau einer eigenen Navigationsoberfläche verwenden.

Die gelieferte Information betrifft die Sitzung; das ist die Navigation, die im gleichen Web Bereich ausgeführt wird, solange das Formular nicht geschlossen wird.

UrlsArr wird mit der Liste der aufgerufenen URLs gefüllt. Je nach dem Wert im Parameter *Richtung*, enthält das Array die Liste der vorigen URLs (Standardoperation) oder die Liste der nächsten URLs. Diese Listen entsprechen dem Inhalt der Standardschaltflächen **Zurück** and **Vor** von Browsern.

Die URLs sind chronologisch geordnet.

Übergeben Sie in *Richtung* einen Wert für die Richtung der angezeigten URLs. Sie können eine der folgenden Konstanten unter dem Thema **Web Area** verwenden:

Konstante	Typ	Wert
WA next URLs	Lange Ganzzahl	1
WA previous URLs	Lange Ganzzahl	0

Geben Sie in *Richtung* keinen Wert an, wird 0 (Null) übergeben.

Ist ein Wert angegeben, enthält der Parameter *TitelArr* die Liste der Fenstertitel, die den URLs zugeordnet sind. Dieses Array ist mit dem Array *UrlsArr* abgestimmt.

WA OPEN BACK URL

WA OPEN BACK URL ({* ;} Objekt)

Parameter	Typ		Beschreibung
*	Operator	⇒	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	⇒	Objektname (mit *) oder Variable (ohne *)

Beschreibung

Der Befehl **WA OPEN BACK URL** lädt die vorige URL in der Sequenz der geöffneten URLs im Web Bereich, definiert durch die Parameter * und *Objekt*.

Gibt es keine vorige URL, führt der Befehl nichts aus. Mit der Funktion **WA Back URL available** können Sie testen, ob eine vorige URL vorhanden ist.

WA OPEN FORWARD URL

WA OPEN FORWARD URL ({* ;} Objekt)

Parameter	Typ		Beschreibung
*	Operator	⇒	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	⇒	Objektname (mit *) oder Variable (ohne *)

Beschreibung

Der Befehl **WA OPEN FORWARD URL** lädt die nächste URL in der Sequenz der geöffneten URLs im Web Bereich, definiert durch die Parameter * und *Objekt*.

Gibt es keine nächste URL (mit anderen Worten, wenn der Benutzer nie zu einer vorigen URL zurückgegangen ist), führt der Befehl nichts aus. Mit der Funktion **WA Forward URL available** können Sie prüfen, ob eine nächste URL vorhanden ist.

WA OPEN URL

WA OPEN URL ({* ;} Objekt ; Url)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (string), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Url	String	➡	In den Web Bereich zu ladende URL

Beschreibung

Der Befehl **WA OPEN URL** lädt die URL, übergeben im Parameter *Url*, in den Web Bereich, definiert durch die Parameter * und *Objekt*.

Wird in *Url* ein leerer String übergeben, führt der Befehl nichts aus und es wird kein Fehler generiert. Um eine leere Seite in den Web Bereich zu laden, übergeben Sie in *Url* den String "about:blank".

Analog zum vorhandenen Befehl **OPEN URL** erlaubt **WA OPEN URL** im Parameter *Url* verschiedene Syntaxarten zum Bezeichnen der Dateien:

- Posix Syntax: "file:///c:/Meine%20Datei"
- System Syntax: "c:\MeinOrdner\MeineDatei" (Windows) oder "MeineFestplatte:MeinOrdner:MeineDatei" (Mac OS).

Hinweis: Zur Wahrung der Kompatibilität wird in 4D die Syntax "file://" (mit zwei "/") akzeptiert. Sie ist jedoch nicht konform zu RFC. Wir empfehlen, die RFC konforme Syntax "file:///" (mit drei "/") zu verwenden.

Unter Mac OS mit aktiviertem FileVault muss die Posix Syntax verwendet werden. Sie können Systempfade mit der Funktion **Convert path system to POSIX** umrechnen.

Dieser Befehl hat dieselbe Auswirkung wie Ändern des Wertes der Variablen "URL", die dem Bereich zugeordnet ist. Lautet z.B. die Variable des Bereichs MyWArea_url:

```
MyWArea_url:="http://www.4d.com/"
```

ist es dasselbe wie:

```
WA OPEN URL(MyWArea;"http://www.4d.com/")
```

WA REFRESH CURRENT URL

WA REFRESH CURRENT URL ({* ;} Objekt)

Parameter	Typ		Beschreibung
*	Operator	⇒	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	⇒	Objektname (mit *) oder Variable (ohne *)

Beschreibung

Der Befehl **WA REFRESH CURRENT URL** lädt erneut die aktuelle URL im Web Bereich, definiert durch die Parameter * und *Objekt*.

WA SET EXTERNAL LINKS FILTERS

WA SET EXTERNAL LINKS FILTERS ({* ;} Objekt ; FilterArr ; ErlaubenVerweigernArr)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
FilterArr	Array String	➡	Filter Array
ErlaubenVerweigernArr	Array Boolean	➡	Erlauben-Verweigern Array

Beschreibung

Der Befehl **WA SET EXTERNAL LINKS FILTERS** richtet für den Web Bereich, definiert durch die Parameter * und *Objekt*, einen oder mehrere Filter für externe Links ein. Diese Filter bestimmen, ob eine URL, die der aktuellen Seite über ein Link zugeordnet wird, im Web Bereich oder im standardmäßigen Web Browser des Rechners geöffnet wird.

Klickt der Benutzer auf einen Link in der aktuellen Seite, konsultiert 4D die Filterliste für externe Links, ob die angefragte URL im Browser des Rechners geöffnet werden soll. In diesem Fall wird die zur URL passende Seite im Web Browser angezeigt und das Formularereignis On Open External Link erzeugt. Sonst wird standardmäßig die zur URL passende Seite im Web Bereich angezeigt. Die URL wird gemäß dem Inhalt der Arrays *FilterArr* und *ErlaubenVerweigernArr* bewertet.

Die Arrays *FilterArr* und *ErlaubenVerweigernArr* müssen aufeinander abgestimmt sein.

- Jedes Element in *FilterArr* muss eine URL zum Filtern enthalten. Sie können den * als Joker einsetzen, um ein oder mehrere Zeichen zu ersetzen.
- Jedes entsprechende Element in *ErlaubenVerweigernArr* muss einen Boolean Wert haben, d.h. Wahr für Öfnen im Web Bereich, Falsch für Öffnen im Web Browser.

Bei einem Widerspruch auf der Konfigurationsebene (dieselbe URL ist erlaubt und verweigert), wird die letzte Einstellung berücksichtigt.

Um das Filtern der URL zu deaktivieren, rufen Sie den Befehl auf und übergeben ein leeres Array oder in den letzten Elementen von *FilterArr* und *ErlaubenVerweigernArr* jeweils die Werte "*" und Wahr.

Wichtig: Das Filtern der URLs über den Befehl **WA SET URL FILTERS** hat Vorrang vor dem Befehl **WA SET EXTERNAL LINKS FILTERS**. Wird also eine URL über einen Filter des ersten Befehls verweigert, lässt sie sich nicht im Browser öffnen, auch wenn sie explizit über einen Filter des Befehls **WA SET EXTERNAL LINKS FILTERS** erlaubt ist (siehe 2. Beispiel).

Beispiel 1

Dieses Beispiel öffnet die Web Sites in externen Browsern:

```
ARRAY STRING(0;$filters;0)
ARRAY BOOLEAN($AllowDeny;0)

APPEND TO ARRAY($filters;"*www.google.*")  ` "google" wählen
APPEND TO ARRAY($AllowDeny;False)
  `Falsch: Dieser Link wird in einem externen Browser geöffnet.
APPEND TO ARRAY($filters;"*www.apple.*")
APPEND TO ARRAY($AllowDeny;False)
  `Falsch: Dieser Link wird in einem externen Browser geöffnet
WA SET EXTERNAL LINKS FILTERS(MyWArea;$filters;$AllowDeny)
```

Beispiel 2

Dieses Beispiel kombiniert das Filtern beider Sites und externe Links:

```
ARRAY STRING(0;$filters;0)
ARRAY BOOLEAN($AllowDeny;0)
APPEND TO ARRAY($filters;"*www.google.*")  `"google" wählen
APPEND TO ARRAY($AllowDeny;False)  `Diesen Link verweigern
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)

ARRAY STRING(0;$filters;0)
ARRAY BOOLEAN($AllowDeny;0)
APPEND TO ARRAY($filters;"*www.google.*")  `"google" wählen
APPEND TO ARRAY($AllowDeny;False)
  `Falsch: Dieser Link sollte in einem externen Browser geöffnet werden,
  `diese Einstellung hat jedoch keine Auswirkung, da der Link durch URL Filter blockiert wird.

WA SET EXTERNAL LINKS FILTERS(MyWArea;$filters;$AllowDeny)
```

WA SET PAGE CONTENT

WA SET PAGE CONTENT ({* ;} Objekt ; Inhalt ; BasisURL)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Inhalt	String	➡	HTML Quellcode
BasisURL	String	➡	URL für relative Referenzen (Mac OS)

Beschreibung

Der Befehl **WA SET PAGE CONTENT** ersetzt die Seite, die im Web Bereich, definiert durch die Parameter * und *Objekt*, angezeigt wird, durch den HTML Code, der im Parameter *Inhalt* übergeben wird.

Über den Parameter *BasisURL* können Sie auf Mac OS eine Basis URL aufrufen, die vor alle relativen Links, die auf der Seite gefunden werden, gesetzt wird.

Unter Windows hat dieser Parameter keine Auswirkung, d.h. es wird keine Basis URL angegeben. Deshalb lassen sich auf dieser Plattform keine relativen Referenzen verwenden.

Hinweis: Unter Windows ist es zwingend, dass die Seite bereits im Web Bereich geladen ist, bevor dieser Befehl aufgerufen werden kann. Falls erforderlich, können Sie die URL "about:blank" übergeben, um eine leere Seite zu laden.

Beispiel

Anzeige des Satzes "Hello world!" und Definition einer URL "Datei:///" (nur Mac OS):

```
WA SET PAGE CONTENT(MyWArea;"<html><body><h1>Hello World!</h1></body></html>";"file:///")
```

WA SET PAGE TEXT LARGER

WA SET PAGE TEXT LARGER ({* ;} Objekt)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)

Beschreibung

Der Befehl **WA SET PAGE TEXT LARGER** erhöht die Textgröße im Web Bereich, definiert durch die Parameter * und *Objekt*.

Auf Mac OS gilt der Befehl während einer 4D Sitzung: die Einstellung wird nach dem Schließen der 4D Anwendung wieder aufgehoben.

Unter Windows ist seine Auswirkung global, d.h. die Einstellung bleibt nach dem Schließen der 4D Anwendung erhalten.

WA SET PAGE TEXT SMALLER

WA SET PAGE TEXT SMALLER ({* ;} Objekt)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)

Beschreibung

Der Befehl **WA SET PAGE TEXT SMALLER** verringert die Textgröße im Web Bereich, definiert durch die Parameter * und *Objekt*.

Auf Mac OS gilt der Befehl während einer 4D Sitzung: die Einstellung wird nach dem Schließen der 4D Anwendung wieder aufgehoben.

Unter Windows ist seine Auswirkung global, d.h. die Einstellung bleibt nach dem Schließen der 4D Anwendung erhalten.

WA SET PREFERENCE

WA SET PREFERENCE ({* ;} Objekt ; Selector ; Wert)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
Selector	Lange Ganzzahl	➡	Zu ändernde Voreinstellung
Wert	Boolean	➡	Wert der Voreinstellung (True = erlaubt, False = nicht erlaubt),

Beschreibung

Der Befehl **WA SET PREFERENCE** setzt verschiedene Voreinstellungen für den Web Bereich, definiert durch die Parameter * und *Objekt*.

Im Parameter *Selector* übergeben Sie die zu ändernde Voreinstellung, im Parameter *Wert* den zuzuweisenden Wert. In *Selector* können Sie eine der folgenden Konstanten unter dem Thema **Web Area** übergeben:

Konstante	Typ	Wert	Kommentar
WA enable contextual menu	Lange Ganzzahl	4	Erlaubt die Anzeige eines Standard Kontextmenü im Web Bereich
WA enable Java applets	Lange Ganzzahl	1	Erlaubt die Ausführung von Java applets im Web Bereich.
WA enable JavaScript	Lange Ganzzahl	2	Erlaubt die Ausführung von JavaScript Code im Web Bereich
WA enable plugins	Lange Ganzzahl	3	Erlaubt die Installation von Plug-Ins im Web Bereich
WA enable URL drop	Lange Ganzzahl	101	Erlaubt Ziehen einer URL oder Datei in Web Area (Standard = **False**)
WA enable Web inspector	Lange Ganzzahl	100	Erlaubt die Anzeige des Web Inspektors im Web Bereich

In *Wert* übergeben Sie für jede Voreinstellung **True**, um sie zu aktivieren, **False** um sie zu deaktivieren.

Web Plug-Ins und Java Applets

Wir empfehlen, Web Plug-Ins und Java Applets in Web Areas **nicht zu verwenden**, da dies zu instabiler Arbeitsweise von 4D führen kann, insbesondere beim Verwalten von Ereignissen.

Beispiel

Ziehen einer URL in den Web Bereich 'myarea' aktivieren:

```
WA SET PREFERENCE(*;"myarea";WA enable URL drop;True)
```

WA SET URL FILTERS

WA SET URL FILTERS ({* ;} Objekt ; FilterArr ; ErlaubenVerweigernArr)

Parameter	Typ		Beschreibung
*	Operator	➡	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	➡	Objektname (mit *) oder Variable (ohne *)
FilterArr	Array String	➡	Filter Array
ErlaubenVerweigernArr	Array Boolean	➡	Erlauben-Verweigern Array

Beschreibung

Der Befehl **WA SET URL FILTERS** richtet einen oder mehrere Filter für den Web Bereich ein, definiert durch die Parameter * und *Objekt*.

Vor dem Laden einer angefragten Seite konsultiert 4D die Filterliste, ob die Ziel-URL erlaubt ist oder nicht. Die Bewertung richtet sich nach dem Inhalt der Arrays *FilterArr* und *ErlaubenVerweigernArr*.

Nicht-erlaubte URLs werden nicht geladen und das Formularereignis On URL Filtering wird erzeugt.

Die Arrays *FilterArr* und *ErlaubenVerweigernArr* müssen aufeinander abgestimmt sein.

- Jedes Element von *FilterArr* muss eine URL zum Filtern enthalten. Sie können den * als Joker einsetzen, um ein oder mehrere Zeichen zu ersetzen.
- Jedes entsprechende Element im Array *ErlaubenVerweigernArr* muss einen Boolean Wert haben, d.h. **Wahr** für erlaubt, **Falsch** für verweigert.

Bei einem Widerspruch auf der Konfigurationsebene (dieselbe URL ist erlaubt und verweigert), wird die letzte Einstellung berücksichtigt.

Um das Filtern der URL zu deaktivieren, rufen Sie den Befehl auf und übergeben leere Arrays oder in den letzten Elementen der beiden Arrays *FilterArr* und *ErlaubenVerweigernArr* jeweils die Werte "*" und **Wahr**.

Nach Ausführen des Befehls werden die Filter dem Web Bereich als Eigenschaft zugeordnet. Werden die Arrays *FilterArr* und *ErlaubenVerweigernArr* gelöscht oder neu initialisiert, bleiben die Filter aktiv, solange der Befehl nicht erneut ausgeführt wird. Über den Befehl **WA GET URL FILTERS** können Sie den aktiven Filter für einen Bereich herausfinden.

Wichtig: Das Filtern der URLs durch diesen Befehl gilt für jede Anfrage zum Ändern der primären "URL" der Seite, sei es über den Benutzer, Javascript Code oder 4D Code, mit Ausnahme von **WA OPEN URL** und URLs, die mit "javascript:" beginnen.

Beispiel 1

Sie wollen den Zugriff auf alle Web Sites mit dem Kürzel .org, .net und .fr verweigern:

```
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)
```

Beispiel 2

Sie wollen den Zugriff auf alle Web Sites mit Ausnahme der russischen (.ru) verweigern:

```
ARRAY TEXT($filters;0)
ARRAY BOOLEAN($AllowDeny;0)

APPEND TO ARRAY($filters;"*")  `Alle auswählen
APPEND TO ARRAY($AllowDeny;False)  `Alle verweigern
APPEND TO ARRAY($filters;"www.*.ru")  `*.ru auswählen
APPEND TO ARRAY($AllowDeny;True)  `Erlauben
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)
```

Beispiel 3

Sie wollen nur den Zugriff auf die Web Sites von 4D erlauben (.com, .fr, .es, etc.):

```
ARRAY TEXT($filters;0)
ARRAY BOOLEAN($AllowDeny;0)
```

```
APPEND TO ARRAY($filters;"*")  `Alle auswählen
APPEND TO ARRAY($AllowDeny;False)  `Alle verweigern
APPEND TO ARRAY($filters;"www.4D.*")  `4d.fr, 4d.com... auswählen
APPEND TO ARRAY($AllowDeny;True)  `Erlauben
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)
```

Beispiel 4

Sie wollen nur lokalen Zugriff auf die Dokumentation erlauben (im Ordner C://doc):

```
ARRAY TEXT($filters;0)
ARRAY BOOLEAN($AllowDeny;0)

APPEND TO ARRAY($filters;"*")  `Alle auswählen
APPEND TO ARRAY($AllowDeny;False)  `Alle verweigern
APPEND TO ARRAY($filters;"file://C:/doc/*")
  `Pfad Datei:// erlaubt auswählen
APPEND TO ARRAY($AllowDeny;True)  `Erlauben
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)
```

Beispiel 5

Sie wollen den Zugriff auf alle Web Sites erlauben, mit einer Ausnahme, zum Beispiel die Site Elcaro:

```
ARRAY TEXT($filters;0)
ARRAY BOOLEAN($AllowDeny;0)

APPEND TO ARRAY($filters;"*")
APPEND TO ARRAY($AllowDeny;True)  `Alle erlauben
APPEND TO ARRAY($filters;"*elcaro*")  `Alle verweigern, die el caro enthalten
APPEND TO ARRAY($AllowDeny;False)
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)
```

Beispiel 6

Sie wollen den Zugriff auf bestimmte IP Adressen verweigern:

```
ARRAY TEXT($filters;0)
ARRAY BOOLEAN($AllowDeny;0)

APPEND TO ARRAY($filters;"*")  `Alle auswählen
APPEND TO ARRAY($AllowDeny;True)  `Alle erlauben
APPEND TO ARRAY($filters;"86.83.*")  `IP Adressen wählen, die mit 86.93 beginnen.
APPEND TO ARRAY($AllowDeny;False)  `Verweigern
APPEND TO ARRAY($filters;"86.1*")
  `IP Adressen wählen, die mit 86.1 beginnen (86.10, 86.135 etc)
APPEND TO ARRAY($AllowDeny;False)  `Verweigern
WA SET URL FILTERS(MyWArea;$filters;$AllowDeny)
  `Beachten Sie, dass die IP Adresse eines Domain variieren kann.
```

WA STOP LOADING URL

WA STOP LOADING URL ({* ;} Objekt)

Parameter	Typ		Beschreibung
*	Operator	⇒	Mit * ist Objekt ein Objektname (String), ohne * ist Objekt eine Variable
Objekt	Formularobjekt	⇒	Objektname (mit *) oder Variable (ohne *)

Beschreibung

Der Befehl **WA STOP LOADING URL** stoppt das Laden der Ressourcen der aktuellen URL des Web Bereichs, definiert durch die Parameter * und *Objekt*.

Web Server

- Web Server, Überblick
- Web Server konfigurieren und Verbindung verwalten
- Unterstützung von IPv6
- Sicherheit der Verbindung
- Datenbankmethode On Web Authentication
- Datenbankmethode On Web Connection
- Datenbankmethode On Web Close Process
- Web Sessions verwalten
- Halbdynamische Seiten
- URLs und Form Actions
- 4D Objekte mit HTML Objekten verbinden
- Web Server, Einstellungen
- Preemptive Web Prozesse verwenden
- Information über die Web Site
- Eigene HTTP Fehlerseiten definieren
- TLS Protokoll verwenden (HTTPS)
- XML und WML Unterstützung
- 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*

Web Server, Überblick

4D im lokalen Modus, 4D im remote Modus und 4D Server enthalten eine Web Server Engine, mit der Sie 4D Datenbanken oder jede Art von HTML Seiten im Internet publizieren können. Die Hauptmerkmale der 4D Web Server Engine sind:

- **Einfache Publikation**

Sie können die Veröffentlichung der Datenbank im Internet jederzeit starten oder stoppen. Dazu müssen Sie lediglich einen Menübefehl bzw. Befehl der Programmiersprache ausführen.

- **Datenbankmethoden für Web-Anfrage**

Datenbankmethode On Web Authentication und **Datenbankmethode On Web Connection** sind Einstiegspunkte (entry points) von Anfragen im Web Server; damit können Sie jede Art von Anfrage analysieren und weiterleiten (routen).

- **Einsatz spezieller Tags und URLs**

Der 4D Web Server bietet unzählige Mechanismen für die Interaktion mit Benutzeraktionen. Sie können:
- Spezifische Tags in Web Seiten einbauen, die die Bearbeitung durch den Web Server in dem Moment auslösen, wo sie an Browser gesendet werden.
- Spezifische URLs einrichten, die 4D aufrufen, um eine beliebige Aktion auszuführen.
- Diese URLs auch als Formularaktionen verwenden, um die Bearbeitung auszulösen, wenn der Benutzer HTML Formulare posted.

- **Benutzersitzungen verwalten**

Der 4D Web Server enthält vollautomatische Funktionalitäten zur einfachen Verwaltung von Web Sessions (Benutzersitzungen) über Cookies.

- **Zugriffssicherheit**

Es gibt verschiedene automatische Einstellungen, welche die Zugriffsberechtigung auf Web Browser regeln. Sie können auch das in 4D integrierte Kennwortsystem verwenden. Die Option "Allgemeiner Web Benutzer" vereinfacht die Zugriffsberechtigung innerhalb der Datenbank.
Mit der **Datenbankmethode On Web Authentication** können Sie jede Anfrage vor Bearbeitung durch den Web Server bewerten. Zusätzlich können Sie über den Standard HTML Root Ordner den Zugriff auf Dateien auf der Festplatte beschränken. Zu guter Letzt müssen Sie individuell die Projektmethoden zuweisen, die via Web ausgeführt werden sollen.

- **SSL Verbindungen**

Ihr 4D Web Server kann über das SSL Protokoll (Secured Socket Layer) im gesicherten Modus mit Browsern kommunizieren. Dieses Protokoll ist mit den meisten Web Browsern kompatibel, authentifiziert Sender und Empfänger und garantiert die Vertraulichkeit und Integrität der ausgetauschten Informationen.

- **Erweiterte Unterstützung für Internet Formate**

Der 4D Web Server ist kompatibel mit HTTP/1.1. und unterstützt XML Dokumente und WML (Wireless Markup Language) Technologie. Der 4D Web Server erweitert auch die Unterstützung der gzip Komprimierung: Nach einer "Absprache" zwischen Web Server und Client lässt sich jeder Datenaustausch potentiell komprimieren, um direkt die Performance zu steigern.

- **Simultane Operationen der Datenbank**
  - **4D im lokalen Modus und Web**
    Veröffentlichen Sie eine 4D Datenbank im Web mit 4D im lokalen Modus, können Sie gleichzeitig:
    - Die Datenbank mit 4D lokal benutzen
    - Sich mit Web Browsern an die Datenbank anschließen
  - **4D Server und Web**
    Veröffentlichen Sie eine 4D Datenbank im Web mit 4D Server, können Sie folgende Teile gleichzeitig an die Datenbank anschließen und damit arbeiten:
    - Remote 4D Arbeitsstationen
    - Web Browser
  - **Remote 4D und Web**
    Publizieren Sie eine 4D Datenbank über remote 4D im Web, können Sie sich gleichzeitig an die Datenbank anmelden und damit arbeiten:
    - via remote 4D Rechner
    - via Web Browser. In diesem Fall können Web Browser, wenn die Datenbank auch mit 4D Server veröffentlicht wird, sich an die veröffentlichte Datenbank über remote 4D oder 4D Server anschließen. Darüberhinaus sind verschiedene Zugriffsmodi auf die Daten möglich (öffentlich, Verwaltung, etc.).
    Ein 4D Web Server funktioniert im allgemeinen genauso wie der Web Server von 4D im remote Modus. Die Befehle der Programmiersprache funktionieren im allgemeinen auf dieselbe Weise, egal, ob sie auf 4D im lokalen Modus, 4D Server oder 4D im remote Modus ausgeführt werden. Der wichtigste Punkt ist, dass sie auf die Web Site des Rechners angewandt werden, auf dem sie ausgeführt werden. Das regeln Sie über die Befehle **Execute on server** / **EXECUTE ON CLIENT**.

- **Lastverteilung mit 4D:** Da jeder 4D Rechner im remote Modus als Web Server einsetzbar ist, können Sie mit einem Load Balancer ein dynamisches Web Server System einrichten. So haben Sie weitreichende Entwicklungsmöglichkeiten:
  - Ein Load-Balancer (Lastverteilung) optimiert die Leistung des 4D Web Server: Über einen Spiegel der Web Site, die auf jedem Web Server des 4D Client installiert ist, sendet ein Load Balancer (Hard- oder Software) Anfragen an Client Rechner gemäß ihrer verfügbaren Rechenleistung.

- Web Server mit Fehlertoleranz: Die 4D Web Site wird auf zwei oder mehr 4D Rechnern gespiegelt. Fällt ein Web Server von 4D aus, übernimmt ein anderer.
- Erstellen verschiedener Ansichten der gleichen Daten. Das kann sich z.B. nch dem Ursprung der Anwendung richten. Innerhalb eines Firmennetzwerks kann ein Web Server auf einem geschützten 4D Client Rechner für Intranet Anfragen dienen und ein Web Server auf einem anderen Client-Rechner, der hinter einer Firewall liegt, für Internet Anfragen.
- Aufgabenaufteilung zwischen Web Servern auf verschiedenen remote 4D Rechnern: Ein Web Server kann z.B. SOAP Anfragen bearbeiten, ein anderer Standardanfragen, usw.

Web Server konfigurieren und Verbindung verwalten

4D und 4D Server enthalten einen Web Server, auch HTTP Server genannt, mit dem Sie die Daten Ihrer Datenbank im Web transparent und dynamisch veröffentlichen können.

Dieser Abschnitt beschreibt die notwendigen Schritte zum Veröffentlichen von 4D Datenbanken und zum Anmelden an Browser sowie die Verwaltung der Verbindung.

Eine 4D Datenbank im Web nutzen

Um einen HTTP Server von 4D oder 4D Server zu starten, benötigen Sie folgende Elemente:

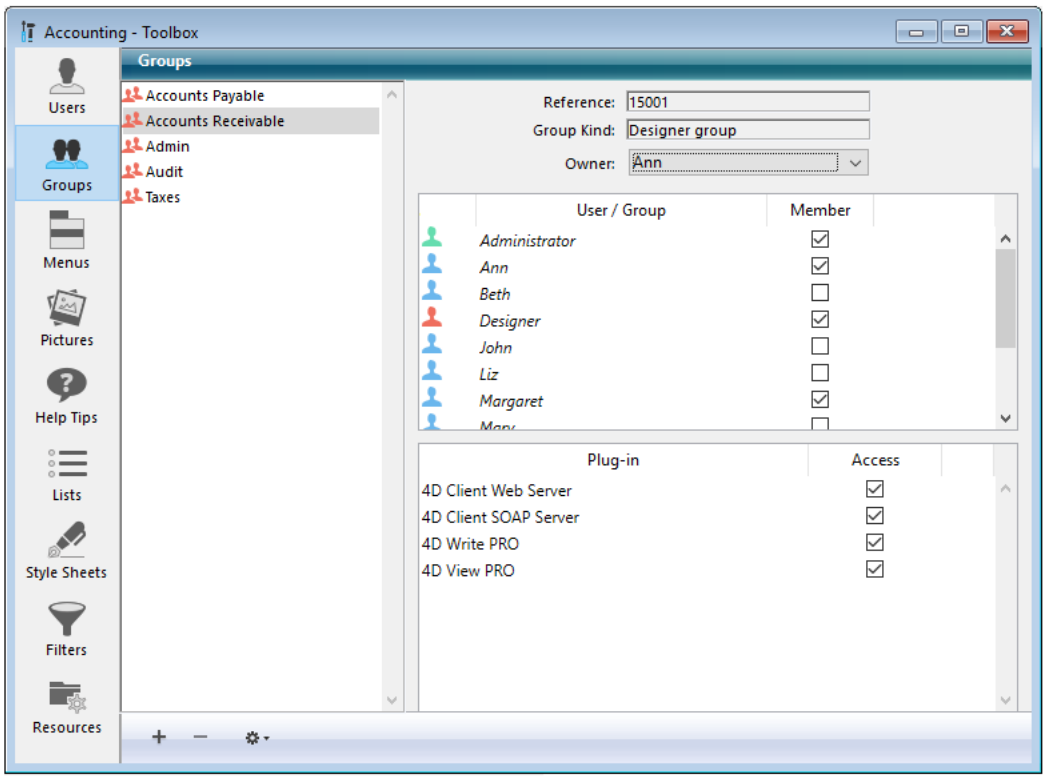
- Eine "4D Web Application" Lizenz. Weitere Informationen dazu finden Sie im *4D Installationshandbuch*.
- Web Verbindungen werden mit dem TCP/IP Protokoll über das Netzwerk gemacht. Folglich
  - müssen Sie TCP/IP auf Ihrem Rechner installiert und korrekt konfiguriert haben. Weitere Informationen finden Sie in der Dokumentation zu Ihrem Rechner oder Betriebssystem.
  - Wollen Sie SSL für Netzwerkverbindungen einsetzen, stellen Sie sicher, dass die erforderlichen Komponenten korrekt installiert sind. Weitere Informationen dazu finden Sie im Abschnitt **TLS Protokoll verwenden (HTTPS)**.
- Nach dem Installieren bzw. Prüfen der entsprechenden Elemente müssen Sie den Web Server von 4D aus starten. Das wird nachfolgend beschrieben.

Veröffentlichung erlauben (4D im remote Modus)

Standardmäßig können alle Client Rechner die Datenbank, an welche sie angeschlossen sind, im Web veröffentlichen. Jedoch können Sie diese Möglichkeit über das 4D Kennwortsystem für jeden 4D remote Rechner einzeln steuern.

4D Server betrachtet Web Lizenzen als Plug-In Lizenzen. So können Sie, wie für Plug-Ins, die Zugriffsberechtigung auf eine spezifische Benutzergruppe beschränken. Dazu zeigen Sie mit 4D in der **Toolbox** die Seite Gruppen an. Diesen Parameter können Sie nur mit entsprechender Zugriffsberechtigung ändern.

Wählen Sie im linken Bereich eine Gruppe und markieren dann im Plug-In Bereich neben der Option **Web Server** das Kästchen **Zugriff**.



Nur Benutzer, die zur Gruppe "4D Client Web Server" gehören, sind berechtigt, ihren 4D Rechner als Web Server zu veröffentlichen.

HelperTool auf Mac OS X konfigurieren

Auf macOS benötigen TCP/IP Ports, die zur Veröffentlichung im Web reserviert sind (Ports zwischen 0 und 1023), spezifische Zugriffsrechte (Administratorrechte). Damit diese Ports von Programmen ohne Anzeige des Systemkennwort-Dialogs genutzt werden können, bietet 4D ein Hilfsprogramm mit Namen HelperTool. Ist es installiert, läuft es unter dem Admin-Konto und kümmert sich automatisch um das Öffnen der Web Ports. Das funktioniert mit 4D in allen Modi, mit 4D Server und ausführbaren Anwendungen mit 4D Volume Desktop.

HelperTool ist in der 4D Software enthalten. Die Installation erfolgt automatisch beim ersten Öffnen eines Port <1024 auf dem Rechner. Der Benutzer erhält eine Meldung, das ein Tool installiert wird und wird aufgefordert, einen Namen und ein Administrator-Kennwort für den Rechner einzugeben. Diese Operation ist nur einmal erforderlich.
Die Anwendung wird umbenannt in "com.4D.HelperTool" und wird im Ordner "Library/PrivilegedHelperTools/" installiert. Nach diesem Vorgang lässt sich der 4D Web Server, unabhängig von der verwendeten 4D Version, transparent starten und stoppen.

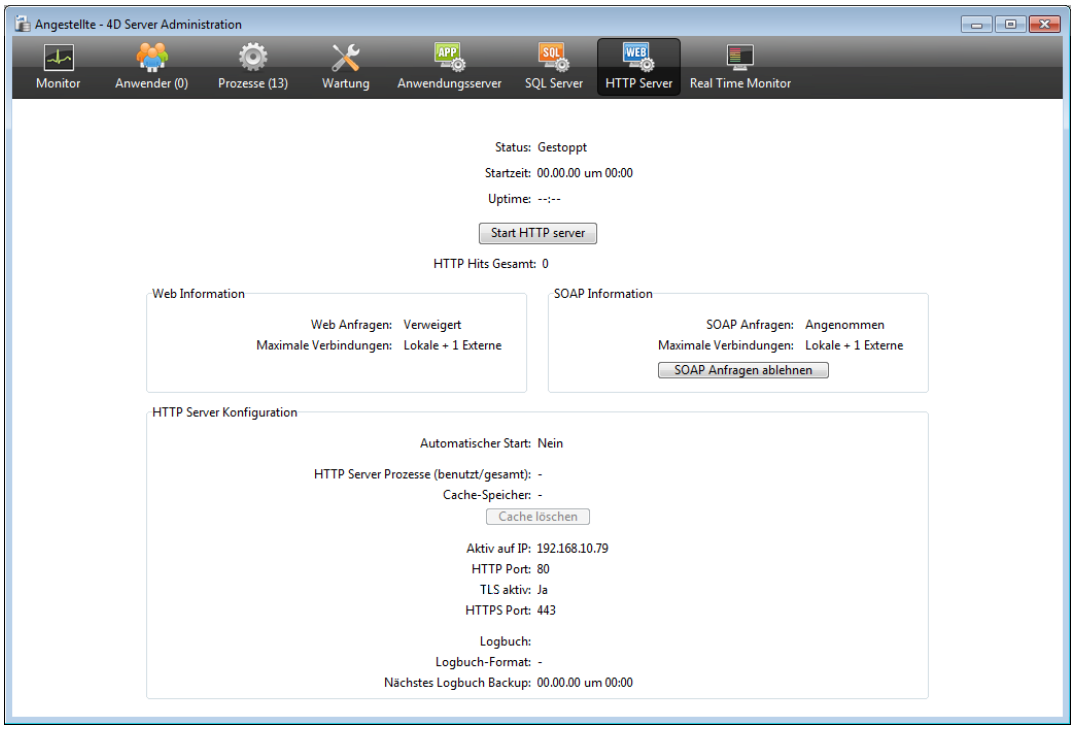
Web Server starten

Es gibt drei Möglichkeiten, den Web Server von 4D zu starten:

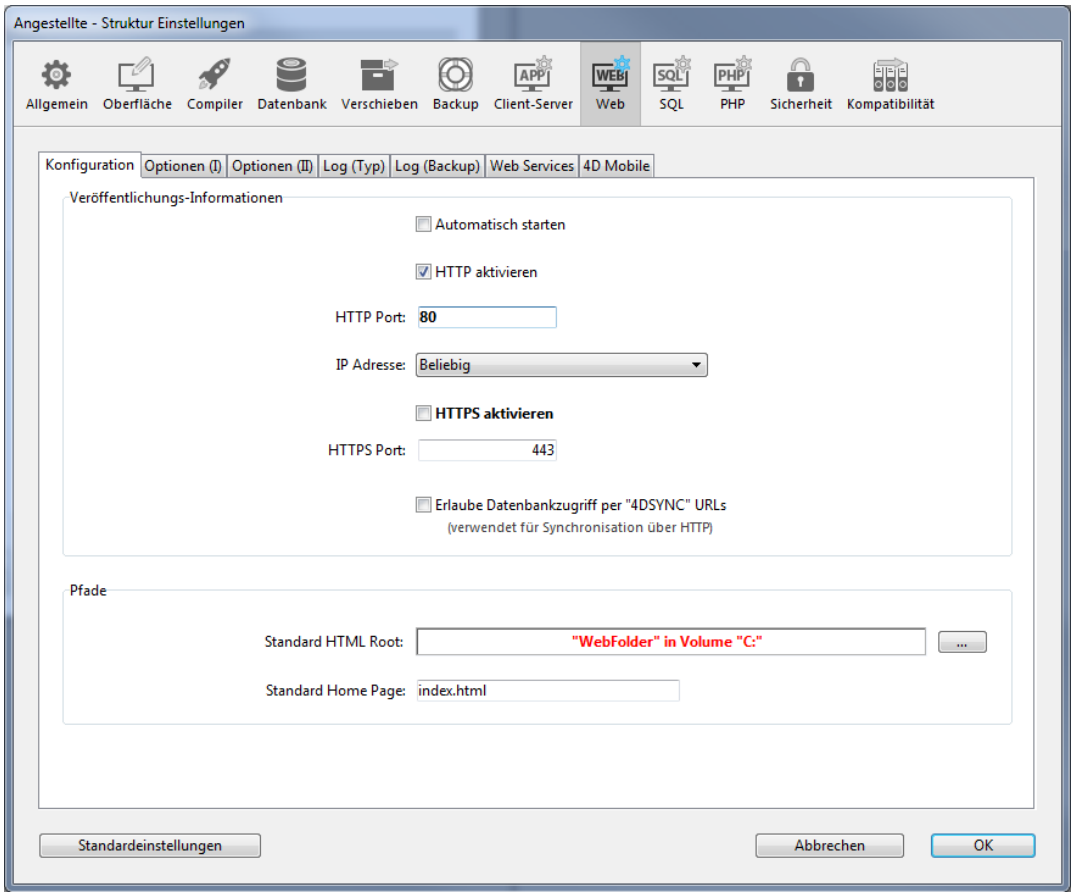
- Im Menü **Start** von 4D oder auf der Seite HTTP Server von 4D Server (Schaltfläche **Start HTTP Server**). Mit diesen Optionen können Sie den Web Server je nach Bedarf starten oder beenden:

4D:

4D Server:



- Um den Web Server beim Öffnen der 4D Anwendung immer automatisch zu starten, gehen Sie in den Datenbank-Eigenschaften auf die Seite **Web**>**Konfiguration**:



Markieren Sie unter **Veröffentlichungs-Informationen** das Kontrollkästchen **Automatisch starten** und klicken auf die Schaltfläche **OK**. Ihre Datenbank wird nun bei jedem Öffnen mit 4D oder 4D Server automatisch im Web publiziert.

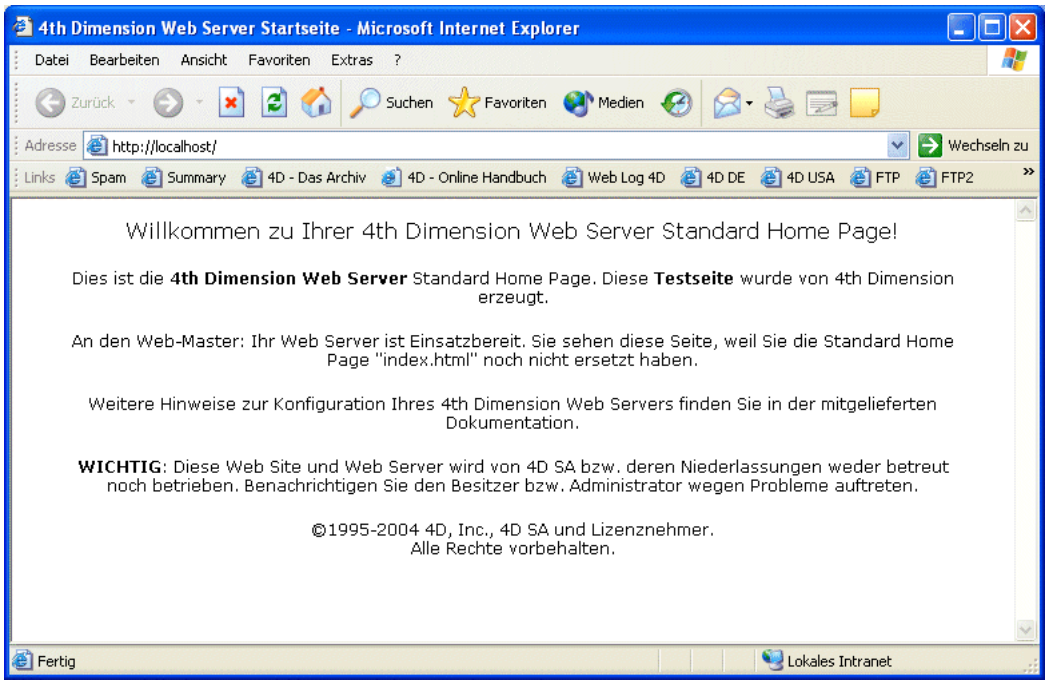
- Per Programmierung durch Aufrufen des Befehls **WEB START SERVER**.

Tipp: Sie müssen 4D nicht verlassen und die Datenbank erneut öffnen, um das Publizieren der Datenbank im Web zu starten bzw. zu stoppen. Sie können den Web Server beliebig oft unterbrechen und wieder starten: Entweder über das Menü **Start**, die Schaltfläche **Start HTTP Server** oder über die Befehle **WEB START SERVER** und **WEB STOP SERVER**.

Web Server testen

Über den Menübefehl **Test Web Server** können Sie sicherstellen, dass der integrierte Web Server korrekt arbeitet (nur 4D). Dieser Befehl ist im Menü **Start** verfügbar, wenn der Web Server gestartet ist:

Wählen Sie diesen Befehl, wird die Home Page der Web Site, die das 4D Programm veröffentlicht, in einem Fenster Ihres standardmäßigen Web Browser angezeigt:



Über diesen Befehl können Sie prüfen, ob Web Server, Anzeige der Home Page, etc. korrekt arbeiten.
Die Seite wird über die URL Localhost aufgerufen, das ist die Standardabkürzung für die IP Adresse des Rechners, auf dem der Web Browser ausgeführt wird. Der Befehl berücksichtigt die TCP Port-Nummer zur Veröffentlichung, die in den Datenbank-Eigenschaften angegeben ist.

Anschluss an eine 4D Datenbank, die im Web publiziert ist

Sobald eine 4D Datenbank im Web publiziert wird, können Sie über einen Web Browser darauf zugreifen. Gehen Sie folgendermaßen vor:

- Hat Ihre Web Site einen registrierten Namen (z.B. *www.RosenIntl.com*), geben Sie diesen Namen im Bereich Open, Address oder Location Ihres Browsers an. Drücken Sie dann die **Eingabetaste**.
- Hat Ihre Web Site keinen registrierten Namen, geben Sie die IP Adresse Ihres Rechners im Bereich Open, Address oder Location Ihres Browsers an (z.B. 123.4.567.89). Drücken Sie dann die **Eingabetaste**.

Ihr Browser sollte nun die Home Page Ihrer Web Site anzeigen. Bei Veröffentlichen der Datenbank mit der Standardeinstellung sollte die Home Page des 4D Web Servers erscheinen. Auf dieser Seite können Sie die Verbindung und die Server Operation prüfen. Es können auch folgende Situationen eintreten:

1. Die Verbindung schlägt fehlt und Sie erhalten eine Meldung wie "...der Server akzeptiert keinen Anschluss oder ist besetzt...". In diesem Fall prüfen Sie folgendes:

- Haben Sie den Namen oder die IP Adresse richtig eingegeben?
- Läuft 4D oder 4D Server und hat das Programm den Web Server gestartet?
- Ist die Datenbank für die Veröffentlichung nicht auf dem Standard Web TCP Port, sondern auf einem anderen TCP Port konfiguriert (siehe Punkt 4)?
- Ist TCP/IP sowohl auf dem Server- als auch auf dem Browser-Rechner korrekt konfiguriert? Beide Rechner müssen entweder auf dem selben Netz bzw. Unternetz liegen oder ihre Router müssen korrekt konfiguriert sein.
- Prüfen Sie die Anschlüsse Ihrer Hardware.
- Testen Sie nicht lokal Ihr eigenes Web, sondern wollen Sie eine Verbindung zu einer Web Datenbank herstellen, die von jemand anderem im Internet oder Intranet benutzt wird, kann die Meldung richtig sein: Der Server kann tatsächlich ausgeschaltet oder besetzt sein. Bauen Sie die Verbindung später wieder auf oder wenden Sie sich an den Web Provider.

2. Sie erhalten eine Verbindung, jedoch erscheint immer der Fehler HTTP 404 "Datei wurde nicht gefunden". Das bedeutet, dass die Home Page nicht definiert wurde. Prüfen Sie, ob die Home Page an der Stelle existiert, die in den Datenbank-Eigenschaften definiert wurde (siehe Abschnitt **Web Server, Einstellungen** oder verwenden Sie den Befehl **WEB SET HOME PAGE**.

3. Sie erhalten eine Verbindung, jedoch NICHT die erwartete Web Site! Dies kann vorkommen, wenn mehrere Server gleichzeitig auf demselben Rechner laufen. Beispiele:

- Sie haben nur eine 4D Web Datenbank auf einem Windows System laufen, das jedoch bereits einen eigenen Web Server in Betrieb hat.
- Sie haben mehrere 4D Web Datenbanken auf demselben Rechner laufen.

In diesem Fall müssen Sie die Nummer des TCP Port ändern, auf dem Ihre 4D Web Datenbank läuft. Die Vorgehensweise wird im Abschnitt **Web Server, Einstellungen** beschrieben.

Hinweis: Ist Ihre Datenbank durch Kennwortsystem geschützt, müssen Sie außerdem einen gültigen Benutzernamen und Kennwort eingeben. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

Web Prozesse

Immer wenn ein Web Browser versucht, sich an die Datenbank anzumelden, wird die Anfrage folgendermaßen verwaltet:

- Zuerst werden ein bzw. mehrere temporäre lokale 4D Prozesse, auch **Web Prozesse** genannt, erstellt, um die Verbindung mit dem Web Browser zu prüfen.
  Diese temporären Prozesse verwalten jede HTTP Anfrage. Sie werden schnell ausgeführt und dann annulliert oder auf

Warten gesetzt. Um den Web Server zu optimieren, friert 4D diesen "Pool" von Web Prozessen nach Bearbeiten der Anfrage für ein paar Sekunden ein, um sie bei zukünftigen Anfragen wieder aktivieren zu können. Diese Operation können Sie über die entsprechenden Konstanten des Befehls **SET DATABASE PARAMETER** steuern, d.h. Sie können das Timeout und die minimal bzw. maximal beibehaltene Anzahl der Prozesse im "Pool" festlegen.

- Der Web Prozess verwaltet die Bearbeitung der Anfrage und sendet bei Bedarf eine Antwort an den Browser. Der temporäre Prozess wird dann annulliert oder auf Warten gesetzt (siehe oben).

Unterstützung von IPv6

4D unterstützt ab Version 14 die Adressnotation IPv6. Dieses Feature betrifft die in 4D integrierten Server:

- Web Server und SOAP Server
- SQL Server

Hinweis: Weitere Informationen dazu finden Sie unter RFC 2460.

Die Unterstützung von IPv6 ist für Benutzer und 4D Entwickler transparent: Das Programm akzeptiert ohne Unterscheidung entweder IPv6 oder IPv4 Verbindungen, wenn die Liste "IP Adresse" auf dem Server auf **Alle** gesetzt ist (siehe **IP Adresse für HTTP Anfragen definieren** (HTTP Server) und **Configuration of 4D SQL Server** (SQL Server).

Sie sollten jedoch folgendes beachten:

- **Port Nummern angeben**
  Da die IPv6 Notation Doppelpunkte (:) verwendet, kann das Hinzufügen von Port Nummern für Probleme sorgen, zum Beispiel:

  ```
  2001:0DB8::85a3:0:ac1f:8001 // IPv6 Adresse
  2001:0DB8::85a3:0:ac1f:8001:8081 // IPv6 Adresse mit Port 8081
  ```

  Wir empfehlen, für mehr Klarheit die Notation [ ] zu verwenden, wenn Sie eine IPv6 Adresse mit einer Port Nummer kombinieren, wie z.B.:

  ```
  [2001:0DB8::85a3:0:ac1f:8001]:8081 //IPv6 Adresse mit Port 8081
  ```

- **Es gibt keine Warnung mehr, wenn der TCP Port besetzt ist**
  Ist der Server in 4D v14 auf "alle" IP Adressen eingestellt und wird der TCP Port von einer anderen Anwendung genutzt, wird das im Gegensatz zu bisherigen 4D Versionen beim Starten des Server nicht mehr angezeigt. 4D Server zeigt in diesem Fall keinen Fehler, da der Port für die IPv6 Adresse frei bleibt. Sie ist jedoch weder über eine IPv4 Adresse auf dem Rechner noch über die lokale Adresse 127.0.0.1 zugänglich.
  Scheint ihr 4D Server nicht auf dem definierten Port zu antworten, können Sie die Adresse [::1] auf dem Server testen (entspricht 127.0.0.1 für IPv6, zum Testen eines anderen Ports fügen Sie :portNum hinzu). Antwortet 4D, verwendet wahrscheinlich eine andere Applikation den Port in IPv4.
- **Anzeige von IPv4 Adressen in IPv6**
  Zur Standardisierung der Bearbeitung bietet 4D eine standardmäßige hybrid Darstellung von IPv4 Adressen in IPv6. Sie werden im IPv6 Format mit einem 96-bit Prefix geschrieben, gefolgt von 32-bits in der Punkt-Dezimal Notation von IPv4. Zum Beispiel stellt ::ffff:192.168.2.34 die IPv4 Adresse 192.168.2.34 dar.

Sicherheit der Verbindung

Die Sicherheit Ihres 4D Web Servers beruht auf folgenden Elementen:

- Kombination der Optionen für das Web Kennwort (BASIC oder DIGEST Modus) und der **Datenbankmethode On Web Authentication**
- Definition eines generischen Web-Anwenders
- Definition eines Standardordners HTML Root
- Definition der Eigenschaft "Zugang per 4D Tags und URLs (4DACTION...)" für jede Projektmethode der Datenbank
- Option zur spezifischen Unterstützung von Synchronisierungsanfragen durch HTTP

Hinweise:

- Die Sicherheit der Verbindung selbst lässt sich über das SSL Protokoll steuern. Weitere Informationen dazu finden Sie im Abschnitt **TLS Protokoll verwenden (HTTPS)**.
- Einen allgemeinen Überblick über die Sicherheitsvorkehrungen in 4D finden Sie im 4D Security guide.

Kennwort-Verwaltungssystem für Web Zugriff

BASIC Modus und DIGEST Modus

In den Einstellungen der Datenbank können Sie das Zugriffsystem für Ihren Web Server einrichten. Es gibt zwei Authentifizierungsmodi: BASIC Modus und DIGEST Modus. Je nach gewähltem Modus wird die Information zu Benutzername und Kennwort unterschiedlich gesammelt und bearbeitet:

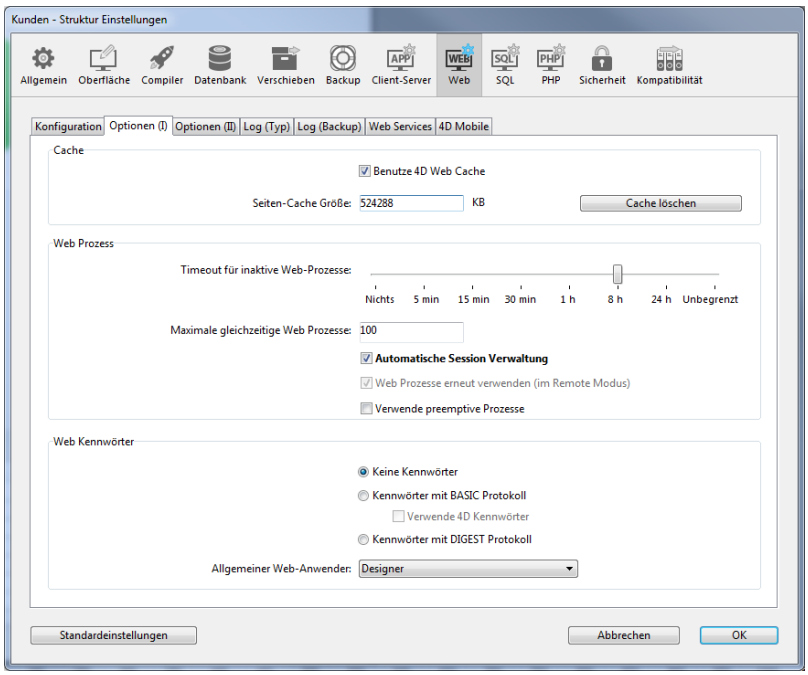
- Im BASIC Modus werden vom Benutzer eingegebener Name und Kennwort unverschlüsselt mit der HTTP Anfrage gesendet. Das bietet keine umfassende Systemsicherheit, da diese Information abgefangen und von dritter Stelle weiter verwendet werden kann.
- Der Digest Modus bietet eine höhere Sicherheitsstufe, da die Information durch einen nicht umkehrbaren Prozess, genannt Hashing, bearbeitet wird. Auf diese Weise ist es unmöglich, den jeweiligen Inhalt zu dechiffrieren.

Für den Benutzer ist der Authentifizierungsmodus unsichtbar.

Hinweise:

- Zur Wahrung der Kompatibilität verwenden 4D Datenbanken, die in Version 11 konvertiert wurden, standardmäßig den Basismodus zur Authentifizierung (wenn die Option “Benutze Kennwörter” in der vorigen Version markiert war). Sie müssen den Digest Modus explizit aktivieren.
- Die Digest Authentifizierung ist eine HTTP 1.1. Funktion und wird nicht von allen Browsern unterstützt. So akzeptiert Microsoft Internet Explorer diesen Modus ab Version 5.0. Sendet ein Browser, der diesen Modus nicht unterstützt, eine Anfrage an einen Web Server bei aktivierter Digest Authentifizierung, weist der Server die Anfrage zurück und sendet eine Fehlermeldung an den Browser.

Im Dialogfenster Datenbank-Eigenschaften legen Sie fest, welche Zugriffsoptionen für Ihren Web Server gelten sollen. Wählen Sie dazu unter dem Thema **Web** die Seite **Optionen I**:



Im Bereich "Web Kennwörter" gibt es folgende Optionen:

- **Keine Kennwörter:** Damit wird für Verbindungen an den Web Server keine Authentifizierung durchgeführt. Dann gilt folgendes:

    - Ist die **Datenbankmethode On Web Authentication** vorhanden, wird sie ausgeführt. Zusätzlich zu $1 und $2 werden die IP-Adressen von Browser und Server ($3 und $4) geliefert, Benutzername und Kennwort ($5 und $6) sind leer. In diesem Fall können Sie die Verbindungen anhand der IP-Adresse des Browsers bzw. der angeforderten IP-Adresse des Servers filtern.
    - Ist die **Datenbankmethode On Web Authentication** nicht vorhanden, werden Verbindungen automatisch akzeptiert.
- **Kennwörter mit BASIC Protokoll:** Standard Authentifizierung im Basis Modus. Meldet sich ein Benutzer an den Server an, erscheint ein Dialogfenster auf dem Browser, wo er Benutzername und Kennwort eingibt. Diese beiden Werte sowie die Verbindungsparameter (IP Adresse und Port, URL...) werden an die **Datenbankmethode On Web Authentication** gesendet, die nun ablaufen kann.
  Dieser Modus bietet Zugriff auf die Option **Benutze 4D Kennwörter**, so dass Sie an Stelle oder zusätzlich zu Ihrem eigenen Kennwortsystem das Kennwortsystem der 4D Datenbank (wie in 4D definiert) nutzen können.
- **Kennwörter mit DIGEST Protokoll**: Authentifizierung im Digest Modus. Benutzer müssen, analog zum BASIC Modus, Name und Kennwort eingeben. Diese beiden Werte sowie die Verbindungsparameter (IP Adresse und Port, URL...) werden verschlüsselt an die **Datenbankmethode On Web Authentication** gesendet. Sie müssen einen Benutzer mit der 4D Funktion **WEB Validate Digest** authentifizieren.

Hinweise:

- Sie müssen den Web Server neu starten, damit diese Parameter berücksichtigt werden.
- Beachten Sie beim Web Server von 4D Client, dass alle Sites, die Client-Rechner veröffentlichen, auf dieselbe Benutzerliste zugreifen. Die Bestätigung von Benutzern/Kennwörtern erfolgt über die 4D Server Anwendung.

BASIC Modus: Kombination von Kennwörtern und der Datenbankmethode On Web Authentication

In diesem Modus richtet sich das System, das die Verbindungen zum 4D Web Server filtert, nach der Kombination der beiden Parameter:

- Die Option **Web Kennwörter** im Dialogfenster Datenbank-Eigenschaften
- Vorhandensein der **Datenbankmethode On Web Authentication**

Sie erhalten je nach Einstellung unterschiedliche Ergebnisse:

Die Option "Kennwörter mit BASIC Protokoll" ist ausgewählt, "Verwende 4D Kennwörter" ist nicht ausgewählt

- Gibt es die **Datenbankmethode On Web Authentication**, wird sie ausgeführt und alle Parameter werden zurückgegeben. So können Sie die Verbindungen genauer filtern, d.h. nach Benutzername, Kennwort und/oder IP-Adressen der Browser oder Web Server.
- Gibt es die **Datenbankmethode On Web Authentication** nicht, wird die Verbindung automatisch zurückgewiesen. Der Browser erhält eine Meldung, dass die Datenbankmethode nicht vorhanden ist.

Hinweis: Ist der vom Browser gesendete Benutzername ein leerer String und gibt es die **Datenbankmethode On Web Authentication** nicht, erhält der Browser den Kennwortdialog.

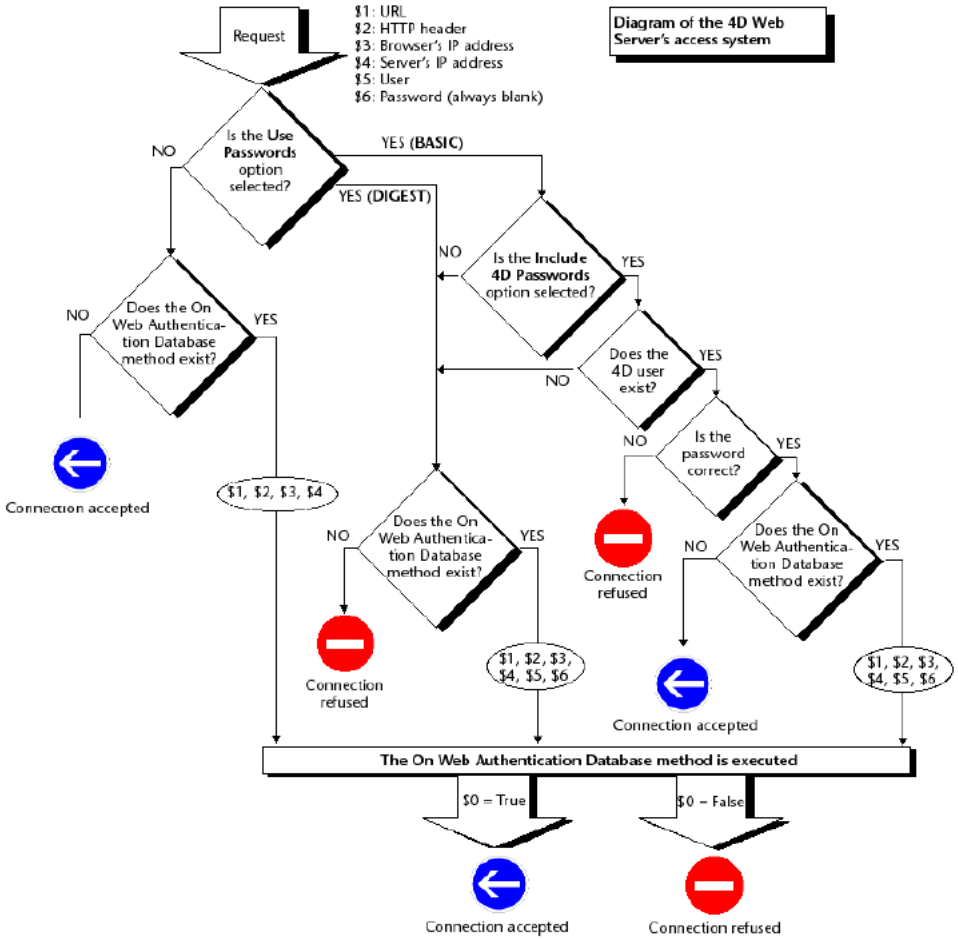
Die Optionen "Kennwörter mit BASIC Protokoll " und "Verwende 4D Kennwörter" sind ausgewählt

- Gibt es den vom Browser gesendeten Benutzernamen in der Tabelle der 4D Benutzer und ist das Kennwort korrekt, wird die Verbindung angenommen. Ist das Kennwort falsch, wird die Verbindung zurückgewiesen.
- Gibt es den vom Browser gesendeten Benutzernamen nicht in 4D, sind zwei Ergebnisse möglich:
    - Gibt es die **Datenbankmethode On Web Authentication**, werden die Parameter $1, $2, $3, $4, $5 und $6 zurückgegeben. Sie können dann die Verbindungen nach Benutzername, Kennwort und/oder IP Adresse des Browsers oder Web Servers filtern.
    - Gibt es die **Datenbankmethode On Web Authentication** nicht, wird die Verbindung zurückgewiesen.

DIGEST Modus

Der DIGEST Modus ist, im Gegensatz zum BASIC Modus, nicht mit den standardmäßigen 4D Kennwörtern kompatibel: Sie können 4D Kennwörter nicht als Web-Kennung verwenden. Bei diesem Modus erscheint die Option “4D Kennwörter einfügen” in Grauschrift.
Sie müssen die Kennung für Web Benutzer selbst verwalten, z.B. über eine Tabelle.
Ist der DIGEST Modus aktiviert, wird der Parameter $6 (Kennwort) in der **Datenbankmethode On Web Authentication** immer leer zurückgegeben. In diesem Modus läuft diese Information nicht als Klartext (unverschlüsselt) über das Netz. Deshalb müssen Verbindungsanfragen unbedingt mit der 4D Funktion **WEB Validate Digest** ausgewertet werden

Die Funktionsweise des 4D Web Server Zugriffssystems im Überblick:



Über Roboter (Sicherheitshinweis)

Bestimmte Roboter (Suchmaschinen, Netze...) scrollen durch die Web Server und statische Seiten. Sollen Roboter Zugriff auf Ihre ganze Web-Site erhalten, können Sie festlegen, auf welche URLs sie nicht zugreifen dürfen.
Legen Sie dazu die Datei ROBOTS.TXT auf den Root des Servers. Sie muss folgende Struktur haben:

```
User-Agent: <name>
Disallow: <URL> oder <beginning of the URL>
```

Zum Beispiel:

```
User-Agent: *
Disallow: /4D
Disallow: /%23%23
Disallow: /GIFS/
```

"User-Agent: *" bedeutet, dass alle Roboter betroffen sind.
"Disallow: /4D" bedeutet, dass Roboter nicht auf URLs zugreifen können, die mit /4D beginnen.
"Disallow: /%23%23" bedeutet, dass Roboter nicht auf URLs zugreifen können, die mit /%23%23 beginnen.
"Disallow: /GIFS/' bedeutet, dass Roboter nicht auf /GIFS/ Ordner oder dessen Unterordner zugreifen können.

Weiteres Beispiel:

```
User-Agent: *
Disallow: /
```

In diesem Fall können Roboter nicht auf die ganze Web-Site zugreifen.

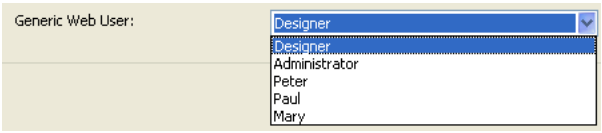
Allgemeiner Web-Anwender

Sie können einen Benutzer, der zuvor in der 4D Kennworttabelle definiert wurde, als allgemeinen Web-Anwender festlegen. In diesem Fall kann jeder Browser, der die Verbindung zur Datenbank herstellt, die diesem generischen Benutzer zugeordneten Zugriffsrechte und Einschränkungen nutzen. So können Sie den Zugriff des Browsers auf die verschiedenen Teile der Datenbank einfach steuern.

Hinweis: Verwechseln Sie diese Option nicht mit dem Steuerungssystem der Web Server Verbindung. Mit "allgemeiner Web Anwender" können Sie den Zugriff des Browsers auf verschiedene Teile der Datenbank einschränken (Tabellen, Menüs, etc.). Das Steuerungssystem der Web Server Verbindung arbeitet mit dem Kennwortsystem und der **Datenbankmethode On Web Authentication**.

Um einen allgemeinen Web-Anwender zu definieren, gehen Sie folgendermaßen vor:

1. Legen Sie im Designmodus im Kennworteditor einen Benutzer an. Sie können dem Benutzer bei Bedarf ein Kennwort zuordnen.
2. Legen Sie in den verschiedenen 4D Editoren die Zugriffsrechte bzw. -einschränkungen für diesen Benutzer fest.
3. Klicken Sie im Dialogfenster Datenbank-Eigenschaften unter dem Thema **Web** auf die Seite **Optionen (I)**. Standardmäßig ist der Designer der allgemeine Web-Anwender, die Browser können auf die gesamte Datenbank zugreifen.
4. Wählen Sie einen Benutzer in der Liste "Allgemeiner Web-Anwender" und bestätigen Sie den Dialog.



Alle Web Browser, die berechtigt sind, sich an die Datenbank anzumelden, können die diesem allgemeinen Web-Anwender zugewiesenen Zugriffsrechte bzw. -einschränkungen nutzen. Das gilt nur dann nicht, wenn die Optionen BASIC Modus und "Verwende 4D Kennwörter" markiert sind und der Benutzer, der die Verbindung herstellt, in der 4D Kennworttabelle nicht vorhanden ist. (siehe unten).

Interaktion mit dem BASIC Protokoll

Die Option "Kennwörter mit BASIC Protokoll" hat keinen Einfluss auf die Funktionsweise des allgemeinen Web-Anwenders. Unabhängig von dieser Option gelten alle Zugriffsrechte bzw. -einschränkungen des "allgemeinen Web-Anwender" für alle Web Browser, die berechtigt sind, sich an die Datenbank anzuschließen.

Ist dagegen die Option "Verwende 4D Kennwörter" ausgewählt, gibt es zwei Möglichkeiten:

- Es gibt Benutzername und Kennwort NICHT in der Kennworttabelle von 4D. Wird nun die Verbindung von der **Datenbankmethode On Web Authentication** angenommen, gelten die Zugriffsrechte des allgemeinen Web-Anwenders für den Browser.
- Es gibt Benutzername und Kennwort in der Kennworttabelle von 4D. Dann wird der Parameter "allgemeiner Web-Anwender" ignoriert. Der Benutzer meldet sich mit seinen eigenen Zugriffsrechten an.

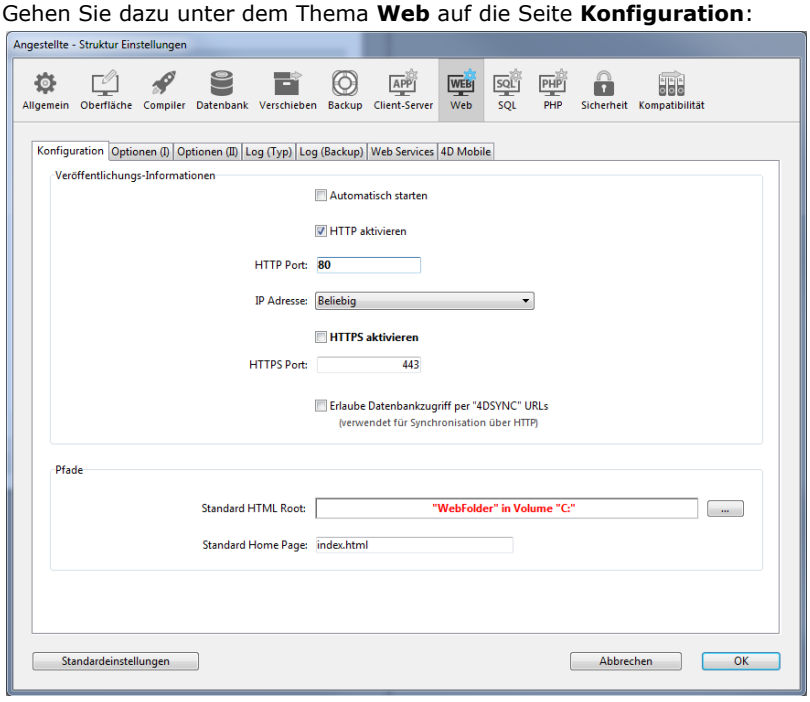
Standard HTML Root

Mit dieser Option bestimmen Sie den Ordner, in welchem 4D nach den statischen HTML Seiten und den Bildern sucht, die an die Browser gesendet werden sollen.
Darüberhinaus legt der Ordner HTML Root fest, bis zu welcher Ebene in der Hierarchie nicht mehr auf die Dateien zugegriffen werden kann.
Diese Einschränkung gilt für die URLs, die an Web Browser sowie an Befehle des 4D Web Server gesendet werden, wie z.B. **WEB SEND FILE**. Sendet der Browser eine URL an die Datenbank oder versucht ein 4D Befehl, auf eine Datei zuzugreifen, die in der Hierarchie oberhalb des Ordners HTML Root liegt, erhalten Sie eine Fehlermeldung. Sie gibt an, dass die Datei nicht gefunden wurde.

4D geht standardmäßig von einem Ordner HTML Root mit Namen **WebFolder** aus. Ist er noch nicht vorhanden, wird er physisch auf der Festplatte erstellt, wenn der Web Server zum ersten Mal gestartet wird.
Wollen Sie diese Standardeinstellungen beibehalten, wird er folgendermaßen angelegt:

- Mit 4D im lokalen Modus und 4D Server auf derselben Ebene wie die Strukturdatei der Datenbank.
  Hinweis: Bei einer kompilierten doppelklickbaren Anwendung liegt die Strukturdatei im **Database** Unterordner.
- Mit 4D im remote Modus im lokalen Ordner der 4D Datenbank (siehe Funktion **Get 4D folder**).
  Anschließend müssen Sie lediglich die erforderlichen Elemente, wie statische Seiten, Bilder, in diesen Ordner kopieren.

Wollen Sie den Ordner bewegen bzw. umbenennen oder einen anderen Ordner verwenden, müssen Sie die Datenbank-Eigenschaften anpassen.
Gehen Sie dazu unter dem Thema **Web** auf die Seite **Konfiguration**:



Geben Sie im Eingabebereich "Standard HTML Root" den neuen Zugriffspfad für den gewünschten Ordner an.
Der hier eingetragene Zugriffspfad ist relativ: Er wird von dem Ordner mit der Struktur der Datenbank festgelegt (4D im lokalen Modus oder 4D Server) oder vom Ordner mit der 4D Anwendung oder dem 4D Software Paket (4D im remote Modus). Damit Ihre

Datenbank auf mehreren Plattformen laufen kann, verwendet der 4D Web Server zum Beschreiben der Zugriffspfade spezifische Syntaxregeln:

- Ordner werden durch Schrägstrich ("/") voneinander getrennt
- Der Zugriffspfad darf nicht mit einem Schrägstrich ("/") enden
- Wollen Sie in der Ordnerhierarchie eine Ebene höher gehen, geben Sie vor dem Ordner zwei Punkte ("..") ein
- Der Zugriffspfad darf nicht mit einem Schrägstrich ("/") beginnen, außer der HTML Root Ordner soll der Datenbankordner oder der 4D remote Ordner sein (siehe unten).

Soll der Ordner HTML Root zum Beispiel der "Web" Unterordner im Ordner "4DDatenbank" sein, schreiben Sie *4DDatenbank/Web* Soll der Ordner HTML Root der Datenbank/4D remote Ordner sein, jedoch der Zugriff auf darüberliegende Ordner untersagt sein, geben Sie "/" in den Bereich ein. Soll der Zugriff auf alle Volumes möglich sein, lassen Sie den Bereich "Standard HTML Root" leer.

Warnung: Definieren Sie im Dialogfenster Datenbank-Eigenschaften keinen Standardordner HTML Root, wird der Ordner verwendet, der die Strukurdatei der Datenbank oder der 4D Anwendung enthält. Seien Sie vorsichtig, denn in diesem Fall **gibt es keine Zugriffsbeschränkungen**. Benutzer haben Zugriff auf alle Ordner.

Hinweise:

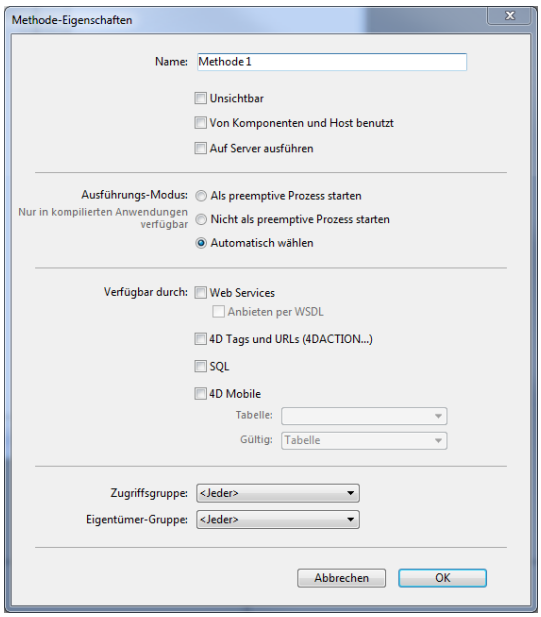
- Wird der HTML Root Ordner im Dialogfenster Datenbank-Eigenschaften geändert, wird der Cache geleert, damit keine Dateien mit beschränktem Zugriff gespeichert werden.
- Sie können den Ordner HTML Root auch dynamisch über den Befehl **WEB SET ROOT FOLDER** ändern. Die Änderung gilt dann für alle Web Prozesse für die Arbeitssitzung. Der Cache der HTML Seiten wird deshalb geleert.

Zugang per 4D HTML Tags und URLs

Mit den spezifischen URL *4DACTION*, sowie den Tags *4DSCRIPT*, *4DTEXT*, *4DHTML* (sowie den früheren Tags *4DVAR* und *4DHTMLVAR*) können Sie die Ausführung jeder Projektmethode in einer 4D Datenbank auslösen, die im Web veröffentlicht wird: Die Anfrage *http://www. server.com/4DACTION/Anzeigen* löst z.B. die Projektmethode *Anzeigen* aus, wenn diese vorhanden ist.

Dieser Mechanismus bedeutet jedoch ein Sicherheitsrisiko für die Datenbank, insbesondere, wenn ein Internet Benutzer absichtlich oder auch versehentlich eine Methode auslöst, die nicht über das Web ausgeführt werden soll. Es gibt drei Möglichkeiten, dies zu verhindern:

- Sie beschränken den Zugriff auf Projektmethoden über das 4D Kennwortsystem. Zur Erinnerung: Dieses System erfordert den Einsatz von 4D Kennwörtern und unterbindet jegliche Methodenausführung. Dazu gehört auch der Einsatz von HTML Tags.
- Sie filtern die über URLS aufgerufenen Methoden über die **Datenbankmethode On Web Authentication**. Beachten Sie jedoch, dass dieses System schwer zu verwalten ist, wenn die Datenbank eine große Anzahl Methoden enthält.
- Sie verwenden die Option "Verfügbar durch 4D HTML Tags und URLs (4DACTION...) aus dem Dialogfenster Methode-Eigenschaften:



Mit dieser Option können Sie jede Projektmethode einzeln bestimmen, die über die spezifischen URL *4D ACTION* oder die Tags *4DSCRIPT*, *4DTEXT* und *4DHTML* (*4DVAR* und *4DHTMLVAR*) aufrufbar sind. Ist sie nicht markiert, kann die entsprechende Projektmethode nicht über eine HTTP Anfrage ausgeführt werden, die ein spezielles URL oder Tag enthält. Bei anderen Aufrufen wie Formularen oder anderen Methoden sind sie dagegen ausführbar.

Die Option ist für erstellte Datenbanken standardmäßig inaktiv. Sie müssen Methoden, die über die Web URL *4DACTION* oder die Tags ausführbar sind, einzeln angeben.

Im Explorer erhalten Projektmethoden mit dieser Eigenschaft folgenden Icon:

Erlaube Datenbankzugriff per 4DSYNC URLs

Über diese Option auf der Seite "Web/Konfiguration" der Datenbank-Eigenschaften können Sie die Unterstützung von Anfragen mit /4DSYNC URLs steuern. Diese URLs dienen zum Synchronisieren von Daten über HTTP. Weitere Informationen dazu finden Sie unter **URL 4DSYNC/**.

Damit können Sie die spezifische Bearbeitung von Anfragen mit /4DSYNC aktivieren bzw. deaktivieren:

- Ist die Option nicht markiert, werden /4DSYNC Anfrage als standardmäßige Anfragen gewertet und erlauben keine spezifische Bearbeitung. So wird bei einer Synchronisationsanfrage eine Antwort vom Typ "404 - Ressource nicht verfügbar" gesendet.
- Ist die Option markiert, wird der Synchronisationsmechanismus aktiviert. /4DSYNC Anfragen werden dann als spezielle Anfragen gewertet und vom 4D HTTP Server analysiert.

Standardmäßig gilt:

- Diese Option ist **nicht markiert** in Anwendungen, die mit 4D ab Version 13 erstellt wurden .
- Zur Wahrung der Kompatibilität ist diese Option in Anwendungen **markiert**, die aus einer älteren 4D Version konvertiert wurden. Wir empfehlen, sie zu deaktivieren, falls Ihre Anwendung nicht die Funktion HTTP Replikation verwendet.

Die Reichweite dieser Option gilt lokal für die Anwendung. Der Web Server muss erneut gestartet werden, damit sie berücksichtigt wird.

Datenbankmethode On Web Authentication

Beschreibung

Die **Datenbankmethode On Web Authentication** verwaltet die Zugriffe auf den Web Server. Sie wird von 4D oder 4D Server aufgerufen, wenn eine Anfrage des Web Browser die Ausführung einer 4D Methode auf dem Server erfordert (Methode, die über eine URL *4DACTION* oder *4DCGI*, ein Tag *4DSCRIPT*, etc. aufgerufen wird).

Diese Methode empfängt die Text Parameter *$1*, *$2*, *$3*, *$4*, *$5* und *$6* und gibt einen Boolean Parameter $0 zurück:

Parameter	Typ	Beschreibung
$1	Text	URL
$2	Text	HTTP Kopfteil + HTTP body (32 KB maximum)
$3	Text	IP Adresse des Web Client (Browser)
$4	Text	IP Adresse des Server
$5	Text	Benutzername
$6	Text	Kennwort
$0	Boolean	Wahr = Anfrage angenommen, Falsch = Anfrage abgewiesen

Sie müssen diese Parameter folgendermaßen deklarieren:

```
` Datenbankmethode On Web Authentication

C_TEXT($1;$2;$3;$4;$5;$6)
C_BOOLEAN($0)
` Code für die Methode
```

Hinweis: Unter Umständen sind nicht alle Parameter der **Datenbankmethode On Web Authentication** angegeben. Die von der Datenbankmethode empfangene Information richtet sich nach den Optionen, die Sie zuvor im Dialogfenster Datenbank-Eigenschaften ausgewählt haben. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

- **URL**
  Der erste Parameter ($1) ist die **URL**, welche der Benutzer im Bereich Location seines Web Browsers eingibt, aus der die Host Adresse entfernt wurde.
  Nehmen wir als Beispiel eine Intranet Verbindung. Die IP Adresse Ihres 4D Web Server Rechners ist 123.4.567.89. Nachfolgende Tabelle zeigt die Werte von $1, je nachdem, welche **URL** im Web Browser eingegeben wurde:

URL im Bereich Location des Web Browsers	Wert des Parameters $
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Kunden	/Kunden
http://123.4.567.89/Kunden	/Kunden
http://123.4.567.89/Kunden/Hinzufügen	/Kunden/Hinzufügen
123.4.567.89/Führe_dieses_aus/Wenn_OK/	/Führe_dieses_aus/Wenn_OK
Führe_jenes_aus	/Führe_jenes_aus

- **Kopfteil der HTTP Anfrage**
  Der zweite Parameter ($2) ist der Kopfteil der HTTP Anfrage, die der Web Browser sendet. Beachten Sie, dass dieser Kopfteil komplett an Ihre **Datenbankmethode On Web Authentication** übergeben wird. Sein Inhalt variiert je nach Art des Web Browsers, der versucht, die Verbindung herzustellen.
  Verwendet Ihre Anwendung diese Information, entscheiden Sie selbst, ob der Kopfteil durchlaufen werden soll.

Hinweise:

- Aus Performance-Gründen darf die Datengröße, die über den Parameter $2 läuft, nicht größer als 32 KB sein. Was darüber hinaus geht, wird vom 4D HTTP Server abgeschnitten. Weitere Informationen zu diesem Parameter finden Sie im Abschnitt **Datenbankmethode On Web Connection**.

- **IP Adresse des Web Client**
  Der Parameter $3 empfängt die IP Adresse des Browser Rechners. Mit dieser Information können Sie zwischen Intranet- und Internet-Verbindungen unterscheiden.
  Hinweis: (*) 4D gibt IPv4 Adressen in einem hybrid IPv6/IPv4 Format mit einem 96-bit Prefix zurück, z.B. ::ffff:192.168.2.34 für die IPv4 Adresse 192.168.2.34. Weitere Informationen dazu finden Sie im Abschnitt **Unterstützung von IPv6**.

- **IP Adresse des Server**
  Der Parameter $4 empfängt die IP Adresse des 4D Web Servers. Dies ermöglicht Multi-Homing, d.h., Sie können Rechner mit mehr als einer IP Adresse nutzen. Weitere Informationen dazu finden Sie im Abschnitt **Web Server, Einstellungen**.

- **Benutzername und Kennwort**
  Die Parameter $5 und $6 empfangen Benutzername und Kennwort, die der Benutzer im Dialogfenster Standard-Identifikation des Browsers eingibt. Dieser Dialog erscheint für jede Verbindung, wenn im Dialogfenster Datenbank-Eigenschaften eine Option zur Kennwortverwaltung ausgewählt wurde (siehe Abschnitt **Sicherheit der Verbindung**).

Hinweis: Gibt es den vom Browser gesendeten Benutzernamen in 4D, wird der Parameter $6 (das Kennwort des Benutzers) aus Sicherheitsgründen nicht zurückgegeben.

- **Parameter $0**
  Die **Datenbankmethode On Web Authentication** gibt in $0 einen booleschen Wert zurück:
  - Ist $0 **True**, wird die Verbindung angenommen.
  - Ist $0 **False**, wird die Verbindung zurückgewiesen.

Die **Datenbankmethode On Web Connection** wird nur ausgeführt, wenn die Verbindung von **On Web Authentication** angenommen wurde.

Warnung: Übergeben Sie keinen Wert in $0 oder ist $0 in der **Datenbankmethode On Web Authentication** nicht definiert, wird die Verbindung als akzeptiert angesehen und die **Datenbankmethode On Web Connection** wird ausgeführt.

Hinweise:

- Rufen Sie in der **Datenbankmethode On Web Authentication** keine Elemente der Oberfläche auf (**ALERT**, **DIALOG**, etc.), denn das unterbricht die Datenbankmethode und die Verbindung wird zurückgewiesen. Dasselbe gilt, wenn beim Ausführen der Datenbankmethode ein Fehler auftritt.
- Sie können die Ausführung durch 4DACTION oder 4DSCRIPT für jede Projektmethode verbieten, wenn Sie im Dialogfenster Methode-Eigenschaften die Option "Zugang per 4D HTML Tags und URLs (4DACTION...)" markieren. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

Aufrufe der Datenbankmethode On Web Authentication

Die **Datenbankmethode On Web Authentication** wird automatisch aufgerufen, unabhängig vom Modus, wenn eine Anfrage oder Bearbeitung die Ausführung einer 4D Methode erfordert. Sie wird auch aufgerufen, wenn der Web Server eine ungültige statische URL empfängt, z.B. wenn die angefragte statische Seite nicht vorhanden ist.

Die Methode wird in folgenden Fällen aufgerufen:

- Wenn 4D eine URL empfängt, die mit 4DACTION/ beginnt
- Wenn 4D eine URL empfängt, die mit 4DCGI/ beginnt
- Wenn 4D eine URL empfängt, die mit 4DSYNC/ beginnt
- Wenn 4D eine URL empfängt, die eine statische Seite aufruft, die nicht vorhanden ist
- Wenn 4D eine URL mit Root Zugriff empfängt und in den Datenbank-Eigenschaften oder über den Befehl **WEB SET HOME PAGE** keine Home Page gesetzt wurde.
- Wenn 4D ein Tag 4DSCRIPT in einer halbdynamischen Seite abarbeitet
- Wenn 4D ein Tag 4D LOOP abarbeitet, das auf einer Methode in einer halbdynamischen Seite basiert.

Hinweis zur Kompatibilität: Die Datenbankmethode wird auch aufgerufen, wenn 4D eine URL empfängt, die mit 4DMETHOD/ beginnt. Diese URL ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten.

Beachten Sie, dass die **Datenbankmethode On Web Authentication** NICHT aufgerufen wird, wenn der Server eine URL empfängt, die nach einer statischen gültigen Seite fragt.

Beispiel 1

Beispiel für die **Datenbankmethode On Web Authentication** im BASIC Modus:

```
  `Datenbankmethode On Web Authentication
C_TEXT($5;$6;$3;$4)
C_TEXT($user;$password;$BrowserIP;$ServerIP)
C_BOOLEAN($4Duser)
ARRAY TEXT($users;0)
ARRAY LONGINT($nums;0)
C_LONGINT($upos)
C_BOOLEAN($0)

$0:=False

$user:=$5
$password:=$6
$BrowserIP:=$3
$ServerIP:=$4

  `Verweigere aus Sicherheitsgründen Namen mit @
If(WithWildcard($user)|WithWildcard($password))
   $0:=False
  `Die Methode WithWildcard folgt unten
Else
  `Prüfe, ob es ein 4D Benutzer ist
   GET USER LIST($users;$nums)
   $upos:=Find in array($users;$user)
   If($upos >0)
      $4Duser:=Not(Is user deleted($nums{$upos}))
   Else
      $4Duser:=False
   End if

   If(Not($4Duser))
  `Ist es kein 4D Benutzer, suche in der Tabelle WebUser
```

```
        QUERY([WebUser];[WebUser]User=$user;*)
        QUERY([WebUser]; & [WebUser]Password=$password)
        $0:=(Records in selection([WebUser])=1)
    Else
        $0:=True
    End if
End if
  `Ist es eine Intranet Verbindung?
If(Substring($BrowserIP;1;7)#"192.100.")
    $0:=False
End if
```

Beispiel 2

Beispiel für die **Datenbankmethode On Web Authentication** im DIGEST Modus:

```
  // Datenbankmethode On Web Authentication
C_TEXT($1;$2;$5;$6;$3;$4)
C_TEXT($user)
C_BOOLEAN($0)
$0:=False
$user:=$5
  // Aus Sicherheitsgründen Namen mit @ verweigern
If(WithWildcard($user))
    $0:=False
  // Die Methode <span class="rte4d_met">WithWildcard</span> folgt unten
Else
    QUERY([WebUsers];[WebUsers]User=$user)
    If(OK=1)
        $0:=WEB Validate digest($user;[WebUsers]password)
    Else
        $0:=False  // Benutzer existiert nicht
    End if
End if
```

Die Methode ***WithWildcard*** lautet:

```
  // WithWildcard Methode
  // WithWildcard ( String ) -> Boolean
  // WithWildcard ( Name ) -> Enthält ein Joker-Zeichen

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
```

Datenbankmethode On Web Connection

Die **Datenbankmethode On Web Connection** wird in folgenden Fällen aufgerufen:

- Der Web Server empfängt eine Anfrage, die mit der URL 4DCGI beginnt.
- Der Web Server empfängt eine ungültige Anfrage

Weitere Informationen dazu finden Sie im letzten Abschnitt **Datenbankmethode On Web Connection aufrufen**.
Die Anfrage sollte zuvor von der **Datenbankmethode On Web Authentication** – sofern vorhanden – angenommen worden sein und der Web Server muss gestartet sein.
Die **Datenbankmethode On Web Connection** erhält sechs von 4D übergebene Textparameter ($1, $2, $3, $4, $5, $6). das sind folgende Parametertypen:

Parameter	Typ	Beschreibung
$1	Text	URL
$2	Text	HTTP Kopfteil + HTTP Hauptteil (max. 32 Kb)
$3	Text	IP Adresse des Web Client (Browser)
$4	Text	IP Adresse des Server
$5	Text	Benutzername
$6	Text	Kennwort

Sie müssen diese Parameter folgendermaßen deklarieren:

```
` Datenbankmethode On Web Connection

C_TEXT($1;$2;$3;$4;$5;$6)

` Code für die Methode
```

- **URL Extra Daten**
  Der erste Parameter ($1) ist die URL, welche der Benutzer im Adressbereich seines Web Browsers eingegeben hat, ohne die Host-Adresse.
  Nehmen wir als Beispiel eine Intranet Verbindung. Die IP Adresse Ihres 4D Web Server Rechners ist 123.4.567.89.
  Nachfolgende Tabelle zeigt die Werte von $1 abhängig von der im Web Browser eingegebenen URL:

URL im Bereich Location des Web Browsers	Wert des Parameters $1
123.4.567.89	/
http://123.4.567.89	/
123.4.567.89/Kunden	/Kunden
http://123.4.567.89/Kunden	/Kunden
http://123.4.567.89/Kunden/Hinzufügen	/Kunden/Hinzufügen
123.4.567.89/Aktion1/Wenn_OK/Aktion2	/Aktion1/Wenn_OK/Aktion2

  Sie können diese Parameter nach Belieben nutzen. 4D ignoriert einfach den Wert, der nach dem Host-Teil der URL steht. Sie können zum Beispiel eine Konvention festlegen, nach der der Wert "/Kunden/Hinzufügen" bedeutet "gehe direkt zu der Aktion neuen Datensatz in der Tabelle [Kunden] hinzufügen". Wenn Sie den Web Benutzern Ihrer Datenbank eine Liste der möglichen Werte und/oder Standard-Bookmarks zur Verfügung stellen, können Sie auch Tastaturkürzel für die verschiedenen Teile Ihrer Anwendung einrichten. So können Web Benutzer schnell auf die Ressourcen im Web zugreifen und müssen nicht bei jedem Verbindungsaufbau erneut den gesamten Navigationspfad durchlaufen.

  Warnung: Um zu verhindern, dass ein Benutzer mit einem bei einer früheren Sitzung erstellten Lesezeichen erneut in die Datenbank gelangt, fängt 4D jede URL ab, die zu den Standard URLs von 4D gehört.

- **Kopfteil und Hauptteil der HTTP Anfrage**
  Der zweite Parameter ($2) ist Kopfteil und Hauptteil der HTTP Anfrage, die der Web Browser sendet. Beachten Sie, dass diese Information komplett in Ihre **Datenbankmethode On Web Connection** übernommen wird. Der Inhalt variiert je nach Art des Web Browsers, der versucht, die Verbindung aufzubauen.

  Mit Safari auf Mac OS könnte der Kopfteil folgendermaßen aussehen:

```
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
```

Mit Microsoft Internet Explorer 8 auf Windows könnte der Kopfteil folgendermaßen aussehen

```
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
```

Enthält Ihre Anwendung diese Information, können Sie selbst wählen, ob Sie den Kopfteil und den Hauptteil durchlaufen.
Hinweis: Aus Performance-Gründen dürfen diese Daten nicht größer als 32 KB sein. Wenn sie darüber liegen, werden sie vom 4D HTTP Server abgeschnitten.

- **IP Adresse des Web Client**
  Der Parameter $3 empfängt die IP Adresse des Browser Rechners. Mit dieser Information können Sie zwischen Intranet und Internet Verbindungen unterscheiden.
  Hinweis: 4D gibt IPv4 Adressen in einem hybrid IPv6/IPv4 Format mit einem 96-bit Prefix zurück, z.B. ::ffff:192.168.2.34 für die IPv4 Adresse 192.168.2.34. Weitere Informationen dazu finden Sie im Abschnitt **Unterstützung von IPv6**.

- **IP Adresse des Server**
  Der Parameter $4 empfängt die IP Adresse, an welche die HTTP Anfrage gesendet wurde. 4D ermöglicht 4D Multi-homing, d.h. den Einsatz von Rechnern mit mehr als einer IP Adresse. Weitere Informationen dazu finden Sie im Abschnitt **Web Server, Einstellungen**.

- **Benutzername und Kennwort**
  Die Parameter $5 und $6 empfangen Benutzernamen und Kennwort, die der Benutzer im vom Browser angezeigten Dialogfenster Standardidentifikation eingibt. Dieses Dialogfenster erscheint für jede Verbindung, wenn im Dialogfenster Datenbank-Eigenschaften die Option **Benutze Kennwort** ausgewählt wurde. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

  Hinweis: Gibt es den vom Browser gesendeten Benutzernamen in 4D, wird der Parameter $6 (Benutzerkennwort) aus Sicherheitsgründen nicht zurückgegeben.

Datenbankmethode On Web Connection aufrufen

Die **Datenbankmethode On Web Connection** lässt sich als Einstiegspunkt für den 4D Web Server verwenden, entweder über die spezielle URL /4DCGI/... oder über angepasste URL Befehle.

Warnung: Wird ein 4D Befehl aufgerufen, der ein Element der Oberfläche anzeigt (**DIALOG**, **ALERT**...), beendet das den Prozess.

Die **Datenbankmethode On Web Connection** wird in folgenden Fällen aufgerufen:

- Wenn 4D die URL /4DCGI empfängt. Die Datenbankmethode wird in $1 mit der URL /4DCGI/<action> aufgerufen.
- Wenn eine Web Seite aufgerufen wird und keine URL vom Typ <path>/<file> gefunden wird. Die Datenbankmethode wird mit der URL aufgerufen. (*)
- Wenn eine Web Seite mit einer URL vom Typ <file>/ aufgerufen wird und standardmäßig keine Home Page definiert wurde. Die Datenbankmethode wird mit der URL aufgerufen. (*)

(*) In diesen Sonderfällen startet die in $1 empfangene URL NICHT mit dem Zeichen "/".

Datenbankmethode On Web Close Process

Die **Datenbankmethode On Web Close Process** wird vom 4D Web Server immer beim Schließen einer Web Session aufgerufen. Eine Session wird in folgenden Fällen geschlossen:

- Die maximale Anzahl gleichzeitiger Sessions ist erreicht (standardmäßig 100, veränderbar über den Befehl **WEB SET OPTION**), und 4D muss neue anlegen (4D stoppt automatisch den Prozess der ältesten inaktiven Session),
- Die maximale Zeitspanne für Inaktivität der Session ist erreicht (standardmäßig 480 Minuten, veränderbar über den Befehl **WEB SET OPTION**),
- Der Befehl **WEB CLOSE SESSION** wurde aufgerufen.

Wird diese Datenbankmethode aufgerufen, bleibt der Session-Kontext (Variablen und vom Benutzer erzeugte Auswahlen) weiterhin gültig. Sie können also Daten, die zu dieser Session gehören, sichern und später wieder verwenden, insbesondere über die **Datenbankmethode On Web Connection**.

Hinweis: Im Kontext einer 4D Mobile Session (die mehrere Prozesse generieren kann) wird die **Datenbankmethode On Web Close Process** für jeden Web Prozess aufgerufen, der geschlossen wird. So können Sie alle Arten von Daten (Variablen, Auswahl, etc.) speichern, die der 4D Mobile Session Prozess generiert hat.

Ein Beispiel dazu finden Sie im Abschnitt **Web Sessions verwalten**.

Web Sessions verwalten

Der 4D Web Server bietet eine einfache und komplette Art zum Verwalten von Benutzersitzungen. Dieser automatische Mechanismus ermöglicht aufeinanderfolgenden Web Clients, von einer Anfrage zur nächsten denselben Kontext (Auswahlen und Variableninstanzen) erneut zu verwenden.

Das wird über ein privates Cookie ausgeführt, das 4D selbst setzt: "4DSID". Bei jeder Anfrage des Web Client prüft 4D den Wert des 4DSID Cookie:

- Hat das Cookie einen Wert, versucht 4D den Kontext zu finden, der dieses Cookie innerhalb der bestehenden Kontexte angelegt hat.
  - Wird der Kontext gefunden, wird er erneut für den Aufruf verwendet; die Methode ***Compiler_Web*** wird nicht ausgeführt
  - Wird kein Kontext gefunden, erstellt 4D einen neuen.
- Hat das Cookie keinen Wert, erstellt 4D einen neuen Kontext.

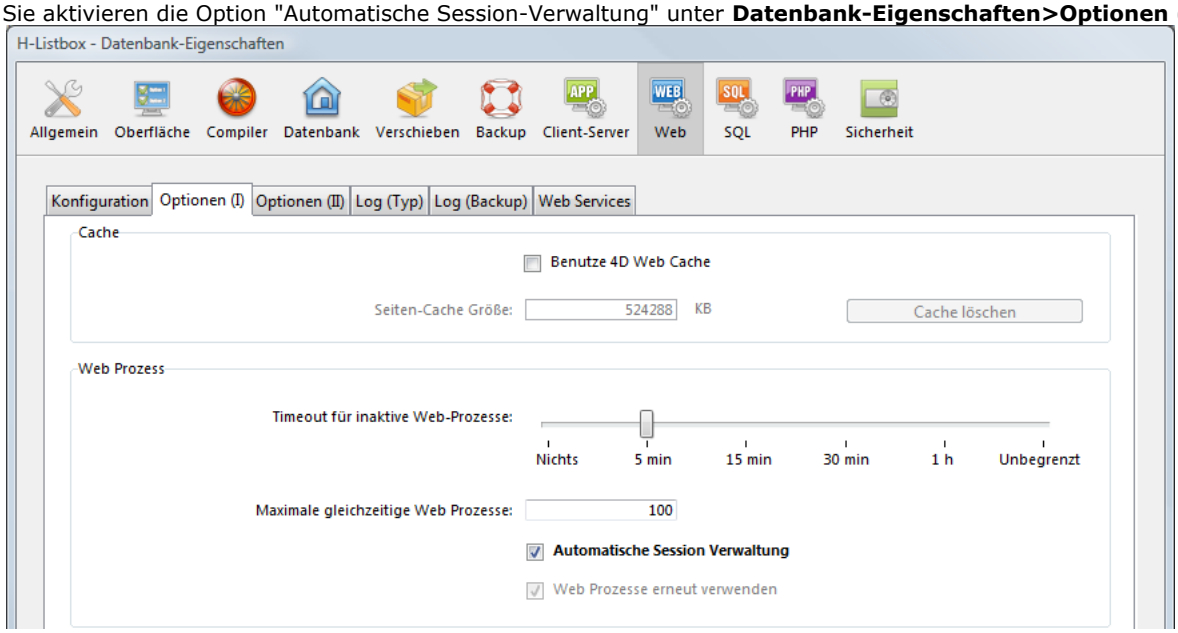
Automatische Session-Verwaltung

Damit Sie die automatische Session-Verwaltung in Ihrer Applikation verwenden können, müssen Sie diese in Ihrem 4D Web Server aktivieren.

Diese Funktionsweise wird in Anwendungen, die mit 4D v13 oder höher erstellt werden, automatisch aktiviert.
Zur Wahrung der Kompatibilität wird sie jedoch in Anwendungen, die aus früheren 4D Versionen konvertiert wurden, deaktiviert. Sie müssen sie explizit aktivieren, damit Sie die Vorteile der automatischen Verwaltung nutzen können.

Dafür gibt es zwei Möglichkeiten:

- Sie aktivieren die Option "Automatische Session-Verwaltung" unter **Datenbank-Eigenschaften>Optionen (I)**:



  In diesem Fall ist die Einstellung permanent, sie wird auf der Festplatte gesichert.
- Über die Konstante Web Keep session des Befehls **WEB SET OPTION**. In diesem Fall gilt die Einstellung nur für die Session und überschreibt die Einstellung, die in den Datenbank-Eigenschaften definiert wurde.

In beiden Fällen gilt die Einstellung lokal für den Rechner; sie kann also auf den Web Server in 4D Server anders sein als auf Rechnern mit remote 4D.

Ablaufen und Beibehalten von Cookies

Die Lebensdauer eines inaktiven Cookie beträgt standardmäßig 8 Stunden (480 Minuten), das lässt sich aber über den Befehl **WEB SET OPTION** verändern. Sie können eine andere Lebensdauer für Cookies (Option Web Inactive session timeout) in Prozessen festlegen, die mit Server Sessions verbunden sind (Option Web Inactive process timeout): Sie wollen z.B. für einen Warenkorb definieren, dass er 24 Stunden gültig bleibt. Aus Optimierungsgründen wollen Sie diesen Prozess jedoch nicht so lange beibehalten. Dafür können Sie z.B. eine Prozesslebensdauer von 4 Stunden festlegen. Am Ende dieser Zeitspanne wird die **Datenbankmethode On Web Close Process** aufgerufen und Sie können die Variablen und Auswahlen dieser Session speichern, bevor der Prozess gestoppt wird. Meldet sich der Web Client das nächste Mal an (bis zu 24 Stunden später), wird das Cookie an den Server zurückgesendet und Sie können die Informationen zur Session erneut in der **Datenbankmethode On Web Connection** laden (siehe Beispiel unten).

Falls erforderlich, können Sie über den Befehl **WEB CLOSE SESSION** jederzeit das Cookie beenden und so die Arbeitssitzung schließen.

Löschen inaktiver Kontexte

4D löscht automatisch den ältesten inaktiven Kontext, wenn die maximale Anzahl der beibehaltenen Kontexte erreicht ist. Standardmäßig ist die Nummer 100 festgelegt. Sie lässt sich über den Befehl **WEB SET OPTION** verändern.
Wird ein Kontext gerade gelöscht, d.h. der zugeordnete Web Prozess wird geschlossen, wird die **Datenbankmethode On Web Close Process** aufgerufen, so dass sie die Variablen und Auswahlen dieses Kontexts zur späteren Wiederverwendung sichern können.

Beispiel

Dieses Beispiel zeigt, wie einfach Sie Sessions über die **Datenbankmethode On Web Connection** und die **Datenbankmethode On Web Close Process** verwalten können.
Der Code der **Datenbankmethode On Web Connection** lautet:

```
  // On Web Connection (oder On Web Authentication)
C_TEXT(www_SessionID)
If(www_SessionID=WEB Get Current Session ID)
  // Compiler_Web wird nicht aufgerufen
  // Alle Variablen und Auswahlen sind bereits vorhanden
    ...
Else
  // Compiler_Web wurde gerade ausgeführt
  // Dies ist eine neue Session, es gibt keine Variable oder Auswahl
  // Sich die Session merken, die 4D gerade angelegt hat
    www_SessionID:=WEB Get Current Session ID

  // Initialisieren der Sitzung
  // Setup Auswahlen
  // Suche angemeldete Benutzer
    QUERY([User];[User]Login=www_Login)
    QUERY([prefs];[prefs]Login=www_Login)

  // Suche Angaben des Mitarbeiters
    QUERY([employees];[employees]Name=[user]name)
    QUERY([company];[company]Name=[user]company)

  // Setup Variablen
  // Erhalte prefs für diesen Benutzer
    SELECTION TO ARRAY([prefs]name;prefNames;[prefs]values;prefValues)
    www_UserName:=[User]name
    www_UserMail:=[User]mail

  // Die Session ist jetzt initialisiert
End if
```

Code für **Datenbankmethode On Web Close Process** :

```
  // On Web Session Suspend
  // Nach einer Periode ohne Aktivität oder bei Notwendigkeit schließt 4D die Session
C_TEXT(www_SessionID)
www_SessionID:=""
  // Wir speichern die Session Info
  // Wir sichern die Einstellungen des zuvor angemeldeten Benutzers
QUERY([prefs];[prefs]Login=www_Login)  // In der Session beibehalten
ARRAY TO SELECTION(prefNames;[prefs]name;prefValues;[prefs]values)

  // Wichtig: Der Prozess wird dann gestoppt
  // 4D löscht die Variablen, Auswahlen, etc.
```

Bei abgewiesenen Cookies

Da die Mechanismen der Session-Verwaltung auf Cookies basiert, kann der HTTP Server von 4D eine Session nicht aufrechterhalten, wenn der Web Client keine Cookies akzeptiert. In diesem Fall wird jede Anfrage als neue Anmeldung gewertet und die Methode ***Compiler_Web*** wird jedes Mal ausgeführt.
Über den Befehl **WEB GET HTTP HEADER** können Sie prüfen, ob Cookies unterstützt werden.

Session und IP Verwaltung

Der 4D HTTP Server merkt sich die IP, welche die Session gestartet hat. Versucht eine andere IP auf eine bestehende Session zuzugreifen, wird der generische Status 400 Client Fehler zurückgegeben.

Halbdynamische Seiten

Der Web Server von 4D ermöglicht die Verwendung **halbdynamischer Seiten**.

Diese Seiten sind HTML 'templates' mit **4D HTML Tags**, z.B. ein Mix aus statischem HTML Code und 4D Referenzen, die über Transformation Tags wie 4DHTML, 4DIF oder 4DINCLUDE hinzugefügt werden. Diese Tags werden in Form von (*<!--#Tag Contents-->*) in den HTML Quellcode eingefügt.

Hinweis: Unter bestimmten Bedingungen wird für 4DHTML, 4DTEXT und 4DEVAL Tags eine alternative Syntax mit $-Zeichen verwendet, um sie XML-kompatibel zu machen. Weitere Informationen dazu finden Sie im Abschnitt **Alternative Syntax für 4DTEXT, 4DHTML, 4DEVAL**.

Beim Senden über den HTTP Server werden diese Seiten analysiert und die darin enthaltenen Tags ausgeführt und mit den Ergebnisdaten ersetzt. Folglich sind die empfangenen Seiten eine Kombination aus statischen Elementen und Werten aus 4D.

Funktionsweise

Sie können per Programmierung Standardwerte für HTML Objekte festlegen. Sie setzen dazu *<!--#4DTEXT VarName-->* in das Feld **Wert** des HTML Objekts ein. *VarName* ist der Name der 4D Prozessvariablen, die im aktuellen Web Prozess definiert wurde. Der Name wird zwischen die Standardmarkierung für HTML Kommentare gesetzt *<!--#...-->*.

Hinweis: Einige HTML Editoren akzeptieren *<!--#4DTEXT VarName-->* nicht im Feld **Wert** von HTML Objekten. In diesem Fall müssen Sie den Ausdruck in HTML Code schreiben. Mit der Syntax *<!--#4DTEXT VarName-->* können Sie 4D Daten in der HTML Seite an beliebiger Stelle einsetzen. Schreiben Sie zum Beispiel <P>Welcome to *<!--#4DTEXT vtSiteName-->*!</P>, wird der Wert der 4D Variablen *vtSiteName* in die HTML Seite eingefügt.

Hier ein Beispiel:

```
 // Der folgende Teil des 4D Code weist "4D4D" der Prozessvariable vs4D zu
vs4D:="4D4D"
 // Dann sendet er die HTML Seite "AnyPage.HTM"
SEND HTML FILE("AnyPage.HTM")
```

Der Quellcode der HTML Seite *AnyPage.HTM* lautet:

```
<html> <head>  <title>AnyPage</title> <script language="JavaScript"><!-- function Is4DWebServer(){ return (document.frm.vs4D.value=="4D4D") } function HandleButton(){  if(Is4DWebServer()){  alert("Sie sind mit 4D Web Server verbunden!")  } else {  alert("Sie sind NICHT mit 4D Web Server verbunden!") } //--> </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>
```

Über das Tag *<!--#4DTEXT -->* lassen sich **4D Ausdrücke** (Felder, Array Elemente, etc.) in die gesendeten Seiten einfügen. Es funktioniert genauso wie mit Variablen. Über *4DHTML* Tags können Sie HTML Code in 4D Variablen einfügen. Über andere Tags wie *4DIF* lässt sich der ausgeführte Code steuern. Ausführliche Informationen dazu finden Sie im Abschnitt **4D HTML Tags**.

Tags bearbeiten

Der Inhalt von durch 4D gesendeten halbdynamischen Seiten wird analysiert, wenn die Befehle **WEB SEND FILE** (.htm, .html, .shtm, .shtml), **WEB SEND BLOB** (text/html vom Typ BLOB) oder **WEB SEND TEXT** bzw. gesendete Seiten über URL aufgerufen werden. Im letzten Fall werden zwecks Optimierung Seiten mit den Endungen .htm und .html NICHT analysiert. Um hier das Analysieren der HTML Seiten zu erzwingen, müssen Sie die Endung .shtm oder shtml anfügen, z.B. http://www.server.com/dir/page.shtm. Ein Beispiel dazu finden Sie in der Beschreibung zum Befehl **WEB GET STATISTICS**.

Über den Befehl **PROCESS 4D TAGS** können Sie das Analysieren auch außerhalb des Web Kontexts durchführen.

Der Parser arbeitet intern mit UTF-16 Strings, die zu analysierenden Daten könnten aber auch anders codiert sein. Bei Tags mit Text (z.B. 4DHTML) konvertiert 4D die Daten bei Bedarf, abhängig von Ursprung und verfügbarer Information (Charset). Nachfolgend die Fälle, in denen 4D die Tags in den HTML Seiten analysiert mit allen ausgeführten Konvertierungen:

Aktion	Analyse des Inhalts der gesendeten Seiten	Berücksichtigung der Syntax mit $ (*)	Verwendeter Zeichensatz
Über URLs aufgerufene Seiten	X, außer Seiten mit der Endung .htm” oder .html	X, außer Seiten mit der Endung .htm” oder .html	Verwendung von Charset, übergeben als Parameter des "Content-Type" Header der Seite. Gibt es keinen, wird nach einem Tag META-HTTP EQUIV mit einem Charset gesucht. Andernfalls wird der standardmäßige Zeichensatz für den HTTP Server verwendet.
Aufruf des Befehls **WEB SEND FILE**	X	-	Verwendung von Charset, übergeben als Parameter des "Content-Type" Header der Seite. Gibt es keinen, wird nach einem Tag META-HTTP EQUIV mit einem Charset gesucht. Andernfalls wird der standardmäßige Zeichensatz für den HTTP Server verwendet.
Aufruf des Befehls **WEB SEND TEXT**	X	-	Keine Konvertierung erforderlich
Aufruf des Befehls **WEB SEND BLOB**	X, wenn BLOB vom Typ "text/html" ist	-	Verwendung von Charset, gesetzt als Parameter des "Content-Type" Header der Antwort. Andernfalls wird der standardmäßig für den HTTP Server gesetzte Zeichensatz verwendet.
Einfügen über *<!--4DINCLUDE-->*	X	X	Verwendung von Charset, übergeben als Parameter des "Content-Type" Header der Seite. Gibt es keinen, wird nach einem Tag META-HTTP EQUIV mit einem Charset gesucht. Andernfalls wird der standardmäßige Zeichensatz für den HTTP Server verwendet.
Aufruf des Befehls **PROCESS 4D TAGS**	X	X	Text Daten: keine Konvertierung. BLOB Daten: Automatische Konvertierung vom Zeichensatz Mac-Roman zur Wahrung der Kompatibilität

(*) Die alternative Syntax mit $-Symbol ist für die Tags $DHTML, 4DTEXT und 4DEVAL verfügbar (siehe)

JavaScript Einbindung

4D unterstützt JavaScript Quellcode, sowie JavaScript*.js* Dateien, die in HTML Dokumente eingebunden sind (z.B. *<SCRIPT SRC="...">*).

Mit dem Befehl **WEB SEND FILE** oder **WEB SEND BLOB** senden Sie eine Seite, die Sie im HTML Quelleditor vorbereitet haben oder per 4D Programmierung erstellt und als Dokument auf der Festplatte gesichert haben. In beiden Fällen können Sie die Seite komplett steuern. Sie können im HEAD Bereich des Dokuments Skripte von JavaScript sowie Skripte mit dem FORM Marker verwenden. Im vorigen Beispiel bezieht sich das Skript auf das Formular "frm", da Sie das Formular benennen konnten. Sie können die Übertragung des Formulars in die Ebene FORM Marker auch auslösen, akzeptieren oder verweigern.

Hinweis: 4D unterstützt die Übertragung von Java Applets.

URLs und Form Actions

Der 4D Web Server bietet verschiedene "Form actions" in URL und HTML, um bestimmte Aktionen in Ihrer Datenbank zu integrieren.

Es gibt folgende URLs:

- *4DACTION/*, um ein beliebiges HTML Objekt mit einer Projektmethode Ihrer Datenbank zu verbinden.
- *4DCGI/*, um die **Datenbankmethode On Web Connection** von einem beliebigen HTML Objekt aus aufzurufen.
- *4DSYNC/*, um die Daten der Tabellen zu synchronisieren.

4D Web Server akzeptiert einige zusätzliche URLs:

- */4DSTATS*, */4DHTMLSTATS*, */4DCACHECLEAR* und */4DWEBTEST*. Damit erhalten Sie Informationen über die Funktionsweise Ihrer 4D Web Site. Weitere Informationen dazu finden Sie im Abschnitt **Information über die Web Site**.
- */4DWSDL*, um den Zugriff auf die Deklarationsdatei der Web Services zu erlauben, die auf dem Server veröffentlicht werden. Weitere Informationen dazu finden Sie im Abschnitt **Befehle für Web Services (Server)** und im Handbuch *4D Designmodus* im Abschnitt **Seite Web/ Web Services**.

URL 4DACTION/

Syntax: *4DACTION/MyMethod/Param*

Verwendung: URL oder "Form action"

Über diese URL können Sie ein HTML Objekt (Text, Schaltfläche...) mit einer 4D Projektmethode verbinden. Der Link lautet */4DACTION/MyMethod/Param*, wobei ***MyMethod*** der Name der auszuführenden 4D Projektmethode ist, wenn auf ein Link geklickt wird, *Param* ein optionaler Textparameter, der der Methode in $1 übergeben wird (siehe nachfolgenden Abschnitt "Textparameter für über URLs aufgerufene 4D Methoden").
Empfängt 4D die Anfrage */4DACTION/MyMethod/Param*, wird – sofern vorhanden – die **Datenbankmethode On Web Authentication** aufgerufen. Gibt diese **Wahr** zurück, wird die Methode ***MyMethod*** ausgeführt.
4DACTION/ kann einer URL in einer statischen Web Seite zugewiesen werden. Die Syntax der URL muss lauten:

```
<<A HREF="/4DACTION/MyMethod/Param">Etwas ausführen</A>
```

Die Projektmethode ***MyMethod*** sollte in der Regel eine "Response" zurückgeben, z.B. über **WEB SEND FILE** oder **WEB SEND BLOB**, etc. eine HTML Seite senden. Achten Sie auf eine möglichst kurze Ausführung, um den Browser nicht zu blockieren.

Hinweis: Eine über *4DACTION* aufgerufene Methode darf keine Oberflächenelemente aufrufen (**DIALOG**, **ALERT**...).

Achtung: Damit eine 4D Methode mit der URL *4DACTION/* ausgeführt werden kann, muss die Option "Zugang per 4D HTML Tags und URLs (4DAction...)" markiert sein. Sie ist im Dialogfenster Methode-Eigenschaften enthalten und standardmäßig inaktiv. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

Beispiel 1

Dieses Beispiel weist die URL *4DACTION/* einem HTML Bildobjekt zu, um auf der Seite ein Bild dynamisch anzuzeigen. Sie fügen in einer statischen Seite folgende Anweisungen ein:

```
<IMG SRC="/4DACTION/PICTFROMLIB/1000">
```

Die Methode ***PICTFROMLIB*** lautet:

```
C_TEXT($1) // Dieser Parameter muss immer angegeben sein
C_PICTURE($PictVar)
C_BLOB($BlobVar)
C_LONGINT($Number)
 // Wir finden die Bildnr. im String $1 wieder
$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 zum Übertragen von Formularen

Der 4D Web Server ermöglicht auch "gepostete" Formulare zu verwenden. Das sind statische HTML-Seiten, die Daten an den Web Server senden. Sie müssen vom Typ POST sein und die Aktion des Formulars muss zwingend mit */4DACTION/MethodName* starten.

Hinweis: Ein Formular lässt sich über zwei Methoden übertragen, beide sind mit 4D verwendbar:

- POST wird in der Regel verwendet, um im Web Server hinzugefügte Daten in der Datenbank hinzuzufügen.
- GET wird in der Regel verwendet, um im Web Server Daten anzufordern, die aus einer Datenbank stammen.

Erhält der Web Server ein "gepostetes" Formular, ruft er die **Datenbankmethode On Web Authentication** auf (wenn sie existiert). Gibt sie **Wahr** zurück, wird die Methode ***MethodName*** ausgeführt. In dieser Methode müssen Sie den Befehl **WEB GET VARIABLES** aufrufen, um die Namen und Werte aller Felder wiederzufinden, die in einer an den Server übertragenen HTML Seite enthalten sind.

Hinweis zur Kompatibilität: In konvertierten Anwendungen wird, wenn im **Seite Kompatibilität** die Option "Automatische Variablenzuweisung" markiert ist, zuerst die spezielle Projektmethode **COMPILER_WEB** (wenn vorhanden) aufgerufen, dann die **Datenbankmethode On Web Authentication**. 4D finden die im Formular vorhandenen HTML Felder und füllt die 4D Variablen in der aufgerufenen Methode mit deren Inhalt, wenn sie den denselben Namen haben. Diese Funktionsweise ist überholt. Weitere Informationen dazu finden Sie im Abschnitt **4D Objekte mit HTML Objekten verbinden**.

Die HTML Syntax für das Formular lautet:

- Um die Aktion in einem Formular zu definieren:

```
<FORM ACTION="/4DACTION/MethodName" METHOD=POST>
```

- Um ein Feld in einem Formular zu definieren:

```
<INPUT TYPE=Field type NAME=Field name VALUE="Standardwert">
```

4D setzt für jedes Feld im Formular den Wert des Feldes in die Variable mit demselben Namen.

Beispiel 2

In einer gestarteten 4D Web Datenbank sollen Browser in Datensätzen über eine statische HTML Seite suchen können. Diese Seite lautet "Suchen.htm". Die Datenbank enthält weitere statische Seiten, über welche Sie z.B. das Suchergebnis anzeigen können ("Ergebnisse.htm"). Der Typ POST wurde sowohl der Seite als auch der Aktion */4DACTION/SUCHE* zugewiesen.

Hier der HTML Code für diese Seite:

```
<FORM ACTION="/4DACTION/PROCESSFORM" METHOD=POST> <INPUT TYPE=TEXT NAME=VNAME VALUE=""><BR> <INPUT
TYPE=CHECKBOX NAME=EXACT VALUE="Wort">Ganzes Wort<BR> <INPUT TYPE=SUBMIT NAME=OK VALUE="Suchen"> </FORM>
```

Tippen Sie zur Dateneingabe im Dateneingabebereich "ABCD" ein, prüfen Sie die Option "Ganzes Wort" und bestätigen Sie durch Klicken auf die Schaltfläche **Suchen**.

In der an den Web Server gesendeten Anfrage:

```
VNAME="ABCD"
vEXACT="Wort"
OK="Suchen"
```

4D ruft die **Datenbankmethode On Web Authentication** (wenn vorhanden), dann die folgende Projektmethode PROCESSFORM:

```
C_TEXT($1) //zwingend für komplilierten Modus
C_LONGINT($vName)
C_TEXT(vNAME;vLIST)
ARRAY TEXT($arrNames;0)
ARRAY TEXT($arrVals;0)
WEB GET VARIABLES($arrNames;$arrVals) //Alle Variablen des Formulars finden
$vName:=Find in array($arrNames;"vNAME")
vNAME:=$arrVals{$vName}
If(Find in array($arrNames;"vEXACT")=-1) //Wurde die Option nicht markiert
   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("Ergebnisse.htm") //Die Liste zum Formular Ergebnisse.htm senden
 //mit einer Referenz auf die Variable vLIST
 //zum Beispiel <!--4DHTML vLIST-->
 //...
End if
```

URL 4DCGI/

Syntax: *4DCGI/<action>*

Verwendung: URL

Empfängt der 4D Web Server die URL */4DCGI/<action>*, wird die **Datenbankmethode On Web Authentication** aufgerufen (sofern vorhanden). Gibt sie **Wahr** zurück, ruft der Web Server die **Datenbankmethode On Web Connection** auf durch Senden der URL an $1, und zwar unverändert.

Die URL */4DCGI/* entspricht keiner Datei, ihre Aufgabe ist lediglich, 4D mit der **Datenbankmethode On Web Connection** aufzurufen. Der Parameter "<action>" kann Information jeglicher Art enthalten.

Mit dieser URL können Sie eine beliebige Aktion durchführen. Sie müssen lediglich den Wert von $1 in der **Datenbankmethode On Web Connection** oder in einer ihrer Untermethoden prüfen und 4D die entsprechende Aktion ausführen lassen. Sie können z.B. eigene statische HTML Seiten einrichten, um Datensätze hinzuzufügen, zu suchen oder zu sortieren bzw. GIF Bilder on-the-fly erstellen. Anwendungsbeispiele zu dieser URL finden Sie unter den Befehlen **_o_PICTURE TO GIF** und **WEB SEND HTTP REDIRECT**.

Am Ende einer Aktion muss über Befehle, die Daten senden, eine Antwort zurückgegeben werden (**WEB SEND FILE**, **WEB SEND BLOB**, etc.).

Warnung: Achten Sie darauf, dass die kürzest mögliche Aktion ausgeführt wird, um den Browser nicht unnötig warten zu lassen.

Textparameter für über URLs aufgerufene 4D Methoden

4D sendet Textparameter zu jeder 4D Methode, die über spezielle URLs (4DACTION/ und 4DCGI/) aufgerufen wird. Dabei gilt folgendes:

- Auch wenn Sie diesen Parameter nicht verwenden, müssen Sie ihn explizit mit dem Befehl **C_TEXT** angeben. Denn sonst treten Runtime Fehler auf, wenn Sie über das Web auf eine kompilierte Datenbank zugreifen. Die Meldung kann lauten wie folgt:
  "Fehler im dynamischen Code
  Ungültige Parameter in einem EXECUTE Befehl
  Methode Name
  Zeile Nr.
  Beschreibung: [<Datum und Zeit>]
  Dieser Runtime Fehler tritt auf, wenn der Textparameter $1 in der 4D Methode, die durch Klick auf den HTML Link aufgerufen wird, nicht deklariert wurde. Da der Ausführungskontext die aktuelle HTML Seite ist, bezieht sich der Fehler nicht auf die Methodenzeile. Solche Fehler werden durch explizites Deklarieren des Textparameters $1 behoben:

```
//Projektmethode M_SEND_PAGE
C_TEXT($1) // Dieser Parameter MUSS explizit deklariert werden
//...
WEB SEND FILE($mypage)
```

- Der Parameter $1 gibt die Extradaten am Ende der URL zurück. Sie können ihn als Platzhalter zum Übergeben von Werten aus der HTML Umgebung in die 4D Umgebung einsetzen.

Parameter, die in der aufgerufenen 4D Methode expliziert deklariert werden müssen

Je nach Art und Ursprung des Aufrufs einer Methode müssen Sie bestimmte Parameter angeben.

- **Datenbankmethode On Web Authentication** (sofern vorhanden) und **Datenbankmethode On Web Connection**.
  Sie müssen die sechs Parameter der Verbindung angeben:

```
//Datenbankmethode On Web Connection
C_TEXT($1;$2;$3;$4;$5;$6) //Diese Parameter MÜSSEN explizit deklariert werden.
```

- Über *4DACTION/* als URL aufgerufene Methode
  Sie müssen den Parameter *$1* angeben:

```
//Über URL 4DACTION/ aufgerufene Methode
C_TEXT($1) //Dieser Parameter MUSS explizit deklariert werden.
```

- Über *4DSCRIPT/* als HTML Kommentar in einem Dokument aufgerufene Methode
  Die Methode sollte in $0 einen Wert zurückgeben. Sie müssen den Parameter *$0* und *$1* angeben:

```
//Über 4DSCRIPT/ als HTML Kommentar aufgerufene Methode
C_TEXT($0;$1) //Diese Parameter MÜSSEN explizit deklariert werden.
```

URL 4DSYNC/

Syntax:
4DSYNC/$catalog{/TableName}
4DSYNC/TableName{/TableName}{/FieldName1,...,FieldNameN}{Params}

Verwendung: URL in POST oder GET Methoden

Diese URL synchronisiert die Daten in den Tabellen lokaler 4D Datenbanken mit einer remote Datenbank über HTTP. Sie kann z.B. eine 4D Datenbank mit einer Client Applikation synchronisieren, die auf einem Smartphone installiert ist, sowie andere HTTP Applikationen von Drittherstellern.

Die URL */4DSYNC/* wird in einer GET Methode verwendet, um die Daten der 4D Datenbank wiederherzustellen oder in einer POST Methode, um die Daten in der 4D Datenbank zu aktualisieren.

Folgende HTTP Anfragen können empfangen werden:

- *GET /4DSYNC/$catalog*
  Gibt die Liste der Tabellen der Datenbank zurück und wieviel Datensätze darin enthalten sind. Haben Sie z.B. eine Struktur mit zwei Tabellen: PERSONEN (2 Datensätze) und RECHNUNGEN (3 Datensätze) und verwenden die Syntax *http://localhost/4DSYNC/$catalog/*, wird im Browser PERSONEN 2 RECHNUNGEN 3 zurückgegeben.
- *GET /4DSYNC/$catalog/TableName*
  Gibt eine Beschreibung der Struktur *TableName* zurück (XML Format).
- *GET /4DSYNC/TableName/FieldName1{,FieldName2},...*
  Gibt die Daten des Feldes *FieldName* in der Tabelle *TableName* zurück
- *GET /4DSYNC/TableName/FieldName1?$stamp=0&$format=json*
  Gibt die Daten des Feldes *FieldName* in der Tabelle *TableName* zurück, beginnend mit *stamp* 0 und im Format *json*.
- *POST /4DSYNC/TableName/FieldName1{,FieldName2},...*
  Integriert die Änderungen, die auf dem Client Rechner gemacht wurden, in die 4D Datenbank.

Hinweis: Das Datenaustauschformat ist JavaScript Object Notation (JSON). Die komplette Syntax ist über den technischen Support von 4D, Inc erhältlich.

Hinweis: Um die Synchronisierungsmechanismus zu aktivieren, müssen Sie in den Datenbank-Eigenschaften auf der Seite **Web>Konfiguration** die Option **Erlaube Datenbankzugriff per "4DSYNC" URLs** markieren (siehe unten). Sonst funktionieren Anfragen, die 4D SYNC URLs enthalten, nicht.

Hinweise zur Synchronisation mit HTTP

Beim Verwenden der URL */4DSYNC/* müssen Sie folgende Prinzipien berücksichtigen:

- 4D arbeitet mit der virtuellen Struktur der Datenbank, wenn sie vorhanden ist. In diesem Fall entsprechen die Parameter *TabelleName* und *FeldName* den Tabellen- bzw. Feldnamen, die über die Befehle **SET FIELD TITLES** und **SET TABLE TITLES** definiert wurden. Beachten Sie, dass die Reichweite dieser Befehle die Sitzung ist. Bei Verwendung von 4D Server müssen Sie diese in einer Serverprozedur auf dem Server aufrufen.
- Die Replikation der Felder vom Typ BLOB und Bild wird nicht unterstützt
- Damit Daten synchronisierbar sind:
  - muss der HTTP Server gestartet sein.
  - Auf der Seite **Web>Konfiguration** der Datenbankeigenschaften muss die Option "Erlaube Datenbankzugriff per "4DSYNC" URLs" markiert sein (siehe Abschnitt **Sicherheit der Verbindung**).
  - Die Eigenschaft "Replikation erlauben" muss für jede Tabelle markiert sein, deren Daten Sie synchronisieren wollen. Wurde eine virtuelle Struktur definiert, müssen diese Tabellen darin enthalten sein.
    Warnung: Ist diese Option markiert, wird zusätzliche Information veröffentlicht; Sie müssen sicherstellen, dass der Zugriff auf Ihre Datenbank geschützt ist. Weitere Informationen dazu finden Sie unter **Sicherheit der Verbindung**.
- Empfängt 4D eine */4DSYNC* Anfrage, wird die **Datenbankmethode On Web Authentication** aufgerufen (außer das Kennwort ist inkorrekt, siehe Verbindungsdiagramm unter **Sicherheit der Verbindung**). Gibt sie **Wahr** zurück, wird die Anfrage ausgeführt; andernfalls wird sie zurückgewiesen.

4D Objekte mit HTML Objekten verbinden

Mit dem Web Server von 4D können Sie "gepostete" Daten wiederfinden, z.B. Daten, die Benutzer über Web Formulare eingegeben haben und über Schaltflächen oder Oberflächenelemente an den Server gesendet haben, im POST Modus oder im GET Modus.

Der Web Server akzeptiert einige spezifische URLs, die sich über Schaltflächen zuweisen lassen, so dass die Übertragung des Formulars die server-seitige Bearbeitung auslöst. Weitere Informationen dazu finden Sie im Abschnitt **URLs und Form Actions**.

Dieses Kapitel beschreibt die Vorgehensweisen, um an den Server zurückgegebene Daten in Formularen wiederzufinden.

Dynamische Werte empfangen

Empfängt der 4D Web Server ein gepostetes Formular, kann 4D die Werte jedes darin enthaltenen Objekts wiederfinden. Dieses Prinzip können Sie bei einem Web Formular verwenden, das z.B. mit **WEB SEND FILE** oder **WEB SEND BLOB** gesendet wurde, wo der Benutzer Werte eingibt oder ändern und dann auf die Schaltfläche **Bestätigen** klickt. In diesem Fall gibt es zwei Möglichkeiten, wie 4D die Werte der HTML Objekte in der Anfrage wiederfinden kann:

- Über den Befehl **WEB GET VARIABLES** oder
- Die Befehle **WEB GET BODY PART** und **WEB Get body part count**

Der Befehl **WEB GET VARIABLES** findet die Werte als Text, während die Befehle **WEB GET BODY PART** und **WEB Get body part count** die geposteten Formularen über BLOBs wiederfinden.

Hinweis zur Kompatibilität (4D v13.4): In bisherigen Versionen hat 4D die Werte von Variablen, die über ein gepostetes Web Formular oder eine URL vom Typ GET empfangen wurden, direkt in 4D Prozessvariablen kopiert (Im kompilierten Modus mussten diese Variablen in der Methode ***COMPILER_WEB*** deklariert werden). Diese Funktionsweise wurde ab 4D v13.4 entfernt; sie wird zur Wahrung der Kompatitibilität in konvertierten Datenbanken beibehalten, lässt sich aber über die Option **Automatische Zuweisung von Variablen** auf der **Seite Kompatibilität** der Datenbank-Eigenschaften deaktivieren. Wir empfehlen, diese Option zu deaktivieren und dafür den Befehl **WEB GET VARIABLES** oder **WEB GET BODY PART** zu verwenden.

Betrachten wir folgenden Quellcode für die HTML Seite:

```
<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 } <p>function LogOn(formObj){ if(formObj.vtUserName.value!=""){ return true } else { alert("Enter your name, then try again.") 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>Please enter your name:</b> <input name="vtUserName" value="<!--#4DTEXT vtUserName-->" 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>
```

Sendet 4D die Seite zu einem Web Browser, sieht sie so aus:

Die wichtigsten Features dieser Seite sind:

- Sie enthält drei Schaltflächen für die Übertragung: *vsbLogOn*, *vsbRegister* und *vsbInformation*.
- Klicken Sie auf Log On, wird die Übertragung des Formulars erst von der JavaScript Funktion *LogOn* bearbeitet. Ist kein Name eingegeben, wird das Formular gar nicht in 4D übertragen. JavaScript zeigt eine Meldung an.
- Das Formular hat sowohl eine POST 4D Methode als auch ein Übertragungsskript (*GetBrowserInformation*), das die Eigenschaften des Navigators in die vier verdeckten Objekte kopiert, deren Namen mit *vtNav_App* beginnen.
- Es enthält auch das Objekt *vtUserName*.

Prüfen wir die 4D Methode ***WWW_STD_FORM_POST***, die aufgerufen wird, wenn der Benutzer auf eine der Schaltflächen im HTML Formular klickt.

```
 // Wert der Variablen wiederfinden
ARRAY TEXT($arrNames;0)
ARRAY TEXT($arrValues;0)
WEB GET VARIABLES($arrNames;$arrValues)
C_TEXT($user)

Case of

 // Die Schaltfläche Log On wurde angeklickt
   :(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)
    // Die Methode WWW POST EVENT sichert die Information in einer Tabelle der Datenbank
        Else

            $0:=WWW Register
    // Die Methode WWW Register lässt einen neuen Web Benutzer sich registrieren
        End if

    // Die Schaltfläche Register wurde angeklickt
      :(Find in array($arrNames;"vsbRegister")#-1)
        $0:=WWW Register

    // Die Schaltfläche Information wurde angeklickt
      :(Find in array($arrNames;"vsbInformation")#-1)
        WEB SEND FILE("userinfos.html")
    End case
```

Die Features dieser Methode sind:

- Die 4D Variablen *vtNav_appName*, *vtNav_appVersion*, *vtNav_appCodeName* und *vtNav_userAgent* (verbunden mit den gleichnamigen HTML Objekten) werden über den Befehl **WEB GET VARIABLES** in den HTML Objekten gefunden, die über das JavaScript GetBrowserInformation erstellt wurde.
- Die 4D Variablen *vsbLogOn*, *vsbRegister* und *vsbInformation* sind mit den drei Schaltflächen für die Übertragung verbunden. Der Befehl **WEB GET VARIABLES** findet nur die Variable zur angeklickten Schaltfläche. Wird die Übertragung über eine dieser Schaltflächen ausgeführt, gibt der Browser den Wert der angeklickten Schaltfläche an 4D zurück. Auf diese Weise sehen Sie, welche Schaltfläche angeklickt wurde. Beachten Sie, dass 4D Schaltflächen in einem 4D Formular numerische Variablen sind. In HTML sind dagegen alle Objekte Textobjekte.

Bei *<SELECT...SELECT>* wird im Befehl **WEB GET VARIABLES** der Wert des markierten Elements im Objekt zurückgegeben und nicht die Position des Elements im Array wie in 4D.

WEB GET VARIABLES gibt immer Werte vom Typ Text zurück.

Unterstützung von chunked transfer encoding

Ab 4D v15 R3 unterstützt der in 4D integrierte Web Server Dateien von Web Clients, die mit "Chunked Transfer Encoding" (segmentierter Übertragungscodierung) zum Server übertragen werden. Das ist ein Mechanismus beim Datentransfert in HTTP/1.1. Er ermöglicht, die Daten in mehreren Teilen (chunks) zu übertragen, ohne die endgültige Datengröße in voraus zu kennen.

Hinweis: Der 4D Web Server unterstützt bereits "Chunked Transfer Encoding" vom Server zu Web Clients (siehe **WEB SEND RAW DATA**).

Weitere Informationen über Client-seitige Implementierung für segmentierte Übertragung finden Sie unter RFC2616 oder auf der entsprechenden Seite in Wikipedia.

Projektmethode COMPILER_WEB

Die Methode ***COMPILER_WEB*** wird, sofern vorhanden, systematisch aufgerufen, wenn der HTTP Server eine dynamische Anfrage empfängt und die 4D Engine aufruft. Das ist z.B. der Fall, wenn der 4D Web Server ein gepostetes Formular oder ein URL mit der Aktion 4DCGI/ empfängt. Diese Methode enthält die Direktiven zum Typisieren bzw. Initialisieren von Variablen während dem Web Austausch. Der Compiler verwendet sie beim Kompilieren der Datenbank. Die Methode ***COMPILER_WEB*** ist gängig für alle Web Formulare. Sie ist nicht standardmäßig vorhanden. Sie müssen sie explizit erstellen.

Web Services: Die Projektmethode ***COMPILER_WEB*** wird – sofern vorhanden – auch für jede akzeptierte SOAP Anfrage aufgerufen. Über diese Methode müssen Sie alle 4D Variablen deklarieren, die mit eingehenden SOAP Argumenten verknüpft sind. Bei Verwendung von Prozessvariablen in Methoden mit Web Services müssen diese vor Aufrufen der Methode deklariert werden. Weitere Informationen dazu finden Sie in der Beschreibung zum Befehl **SOAP DECLARATION**.

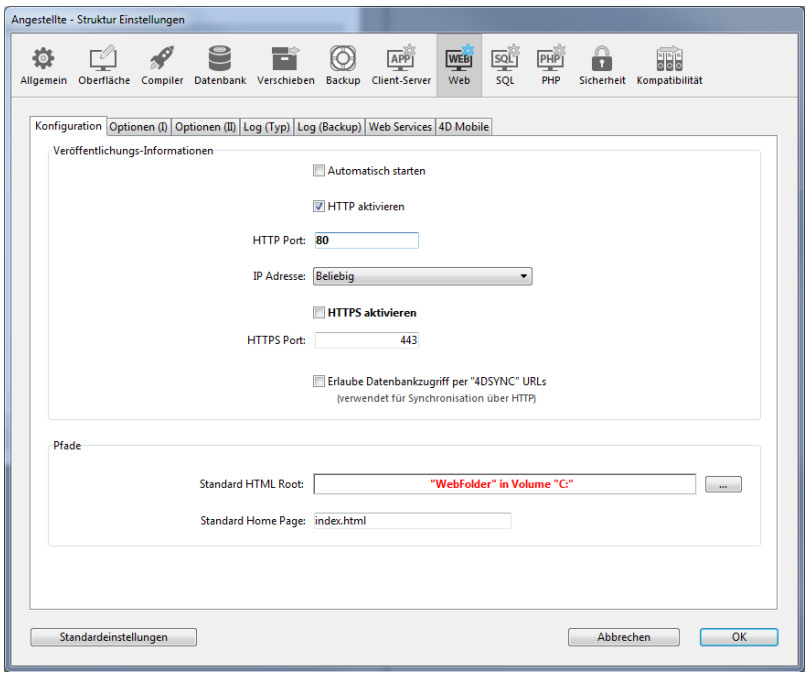
Web Server, Einstellungen

Sie können die Operation des 4D Web Server in den Datenbank-Eigenschaften unter dem Thema **Web** konfigurieren. Dieser Abschnitt beschreibt die Seiten **Konfiguration**, **Optionen (I)** und **Optionen (II)**.

- Die Beschreibung der Seiten zu **Log** finden Sie im Abschnitt **Information über die Web Site**
- Die Beschreibung der Seite **Web Services** finden Sie im Handbuch *4D Designmodus* im Abschnitt **Seite Web/ Web Services**.

Hinweis zur Kompatibilität: Einige Web Funktionalitäten früherer 4D Versionen sind heute überholt, z.B. *"/" von unbekannten URLs nicht entfernen*. Zur Wahrung der Kompatibilität sind diese Funktionalitäten in konvertierten Datenbanken weiterhin verwendbar. Sie können sie über die Seite **Kompatibilität** der Datenbank-Einstellungen aktivieren bzw. bei Bedarf deaktivieren.

Seite Konfiguration



Web Server bei Startup starten

Gibt an, ob der Web Server beim Starten der 4D Anwendung ebenfalls gestartet wird. Weitere Informationen dazu finden Sie im Abschnitt **Web Server konfigurieren und Verbindung verwalten**.

HTTP aktivieren

Gibt an, ob der Web Server nicht-sichere Verbindungen akzeptiert. Weitere Informationen dazu finden Sie im Abschnitt **Verbindungsmodus verwalten** auf der Seite **TLS Protokoll verwenden (HTTPS)** page.

HTTP Port

4D veröffentlicht eine Web Datenbank standardmäßig auf dem regulären Web HTTP Port (TCP Port), also Port 80. Wird dieser Port bereits von einem anderen Web Dienst benutzt, müssen Sie den von 4D verwendeten HTTP Port für diese Datenbank ändern. Ändern Sie den HTTP Port, können Sie den 4D Web Server unter Mac OS X starten, ohne der Root Benutzer des Rechners zu sein. Weitere Informationen dazu finden Sie im Abschnitt **Web Server konfigurieren und Verbindung verwalten**.
Gehen Sie zum Eingabebereich für den HTTP Port und geben Sie einen geeigneten Wert an, d.h. eine Nummer, die noch nicht von einem anderen TCP/IP Dienst auf demselben Rechner benutzt wird.

Hinweis: Tragen Sie Null (0) ein, verwendet 4D die standardmäßige TCP Portnummer 80.

Von einem Web Browser aus müssen Sie nun diese spezifische Nummer des HTTP Port in die Adresse eingeben, mit der Sie die Verbindung zur Web Datenbank aufbauen. Das Ende der Adresse nach dem Doppelpunkt ist die Port-Nummer. Beispiel: Ihre Nummer für den HTTP Port lautet 8080, Ihre Adresse ist demzufolge "123.4.567.89:8080.

Warnung: Wenn Sie nicht die Standardnummer 80 für den HTTP Port verwenden, also 80 für den Standardmodus und 443 für den SSL Modus, achten Sie darauf, dass Sie nicht die Standard Ports anderer Dienste verwenden, die Sie zur gleichen Zeit einsetzen wollen. Wollen Sie z.B. auf Ihrem Web Server Rechner auch das FTP Protokoll verwenden, benützen Sie nicht die Nummern 20 und 21, denn das sind die Standardnummern für dieses Protokoll. Weitere Informationen zu den Standardnummern für TCP Ports finden Sie im Handbuch *4D Internet Commands* unter **Appendix B, TCP Port Numbers**. Port Nummern unter 256 sind für allgemein bekannte Dienste reserviert, Port Nummern zwischen 256 und 1024 für spezifische Dienste auf UNIX Plattformen. Benutzen Sie deshalb als Port Nummern am besten höhere Zahlen, z.B. im zweitausender oder dreitausender Bereich.

IP Adresse für HTTP Anfragen definieren

Sie können die IP Adresse festlegen, über die der Web Server HTTP Anfragen empfangen muss.

Hinweis: Ab 4D v14 unterstützt der HTTP Server automatisch die Adressnotation IPv6, wenn in der Liste "IP Adresse" die Option **Alle** ausgewählt ist. Weitere Informationen dazu finden Sie im Abschnitt **Unterstützung von IPv6**.

Der Server antwortet standardmäßig allen IP Adressen (Option **Alle**).

Die DropDown-Liste zeigt automatisch alle auf dem Rechner verfügbaren IP Adressen an. Wählen Sie eine spezifische Adresse, beantwortet der Server nur Anfragen an diese Adresse.
Diese Funktionalität benötigen Sie für 4D Web Server auf Rechnern mit mehreren TCP/IP Adressen. Das ist für die meisten Internet Host Provider der Fall. Zum Einrichten eines solchen MultiHoming Systems sind auf dem Rechner mit dem Web Server spezifische Konfigurationen erforderlich:

- **Sekundäre IP Adressen auf Mac OS installieren**

Um ein MultiHoming System auf Mac OS einzurichten:
1. Öffnen Sie das Steuerfenster **TCP/IP**.
2. Wählen Sie im PopUp-Menü **Konfiguration** die Option **Manuell**.
3. Legen Sie ein Textdokument mit Namen "Sekundäre IP Adressen" an und sichern es in Ihrem Systemordner im Unterordner **Preferences**.
Jede Zeile von "Sekundäre IP Adressen" besteht aus einer sekundären IP Adresse, einer optionalen Subnetz-Maske und einer dazugehörigen Router-Adresse.

Weitere Informationen dazu finden Sie in der Dokumentation zu Apple.

- **Sekundäre IP Adressen unter Windows installieren**

Um ein MultiHoming System unter Windows einzurichten:
1. Wählen Sie folgende Befehlsfolge (oder die Entsprechungen je nach ihrer Windows-Version): Für Windows XP gilt: Menü **Start** > **Steuerung** > **Netzwerk und Internet-Verbindungen** > **Netzwerkverbindung** > **Local Area Connection** (Eigenschaften) > **Internet Protokoll (TCP/IP)** > Schaltfläche **Eigenschaften** > Schaltfläche **Erweitert**. Auf dem Bildschirm erscheint das Dialogfenster "Erweiterte TCP/IP Einstellungen".
2. Klicken Sie im Bereich "IP Adressen" auf die Schaltfläche **Hinzufügen** und fügen Sie zusätzliche IP Adressen ein. Sie können bis zu 5 verschiedene IP Adressen definieren.

Sie können bis zu 5 verschiedene IP Adressen definieren. Evtl. müssen Sie sich dafür an Ihren Systemadministrator wenden.

HTTPS aktivieren

Zeigt an, ob der Web Server sichere Verbindungen akzeptiert. Weitere Informationen dazu finden Sie auf der Seite **TLS Protokoll verwenden (HTTPS)** im Abschnitt **Verbindungsmodus verwalten**.

HTTPS Port

Damit können Sie die Nummer des TCP/IP Port verändern, welche der Web Server für über SSL gesicherte HTTP Verbindungen (HTTPS Protokoll) verwendet. Standardmäßig ist die Nummer 443 eingetragen.
Es gibt zwei Hauptgründe, diese Port-Nummer zu ändern:

- Aus Sicherheitsgründen: Hacker-Attacken auf Web Server konzentrieren sich in der Regel auf standardmäßige TCP Ports (80 und 443).
- Damit Standardbenutzer auf Mac OS X den Web Server im gesicherten Modus starten können. Auf Mac OS X sind für TCP/IP Ports, die für Veröffentlichungen im Web reserviert sind (0 bis 1023), bestimmte Zugriffsrechte erforderlich: Nur der Root Benutzer kann eine Verbindung über diese Ports öffnen. Damit auch Standardbenutzer den Web Server starten können, ist eine Lösung, die TCP/IP Port-Nummer zu ändern. Weitere Informationen dazu finden Sie im Abschnitt **Web Server konfigurieren und Verbindung verwalten**.
  Sie können jeden gültigen Wert übergeben. Unter Mac OS X sollten Sie zur Vermeidung von Zugriffskonflikten einen Wert größer als 1023 übergeben. Weitere Informationen zu TCP Port Nummern finden Sie im vorigen Abschnitt *TCP Ports*.

Erlaube Datenbankzugriff per "4DSYNC" URLs

Diese Option steuert die Unterstützung von Anfragen mit /4DSYNC URLs steuern. Diese URLs dienen zum Synchronisieren von Daten über HTTP. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

Standard HTML Root

Hier bestimmen Sie einen Standardordner, in dem die Dateien der Web Site abgelegt werden, sowie die hierarchische Ebene auf der Festplatte, ab der die Dateien nicht zugänglich sind. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

Standard Home Page definieren

Sie können für alle Browser, die sich an die Datenbank anmelden, eine Standard Home Page definieren. Sie kann statisch oder halbdynamisch sein.
Beim ersten Start des Web Servers erstellt 4D eine Home Page mit Namen "index.html" und legt sie in den Ordner HTML Root. Verändern Sie diese Konfiguration nicht, erhält jeder Browser, der sich an den Web Server anmeldet, folgende Seite:

Um die Standard Home Page zu verändern, ersetzen Sie einfach im Root Ordner der Datenbank den Ordner "index.html" durch Ihren eigenen Ordner oder tragen Sie im Eingabebereich den Zugriffspfad in Bezug auf den dafür vorgesehenen Ordner ein. Der Zugriffspfad muss in Bezug auf den Standard HTML Root Ordner eingerichtet werden.

Um die Plattformunabhängigkeit Ihrer Datenbanken zu gewährleisten, verwendet der 4D Web Server spezielle Schreibregeln für Zugriffspfade:

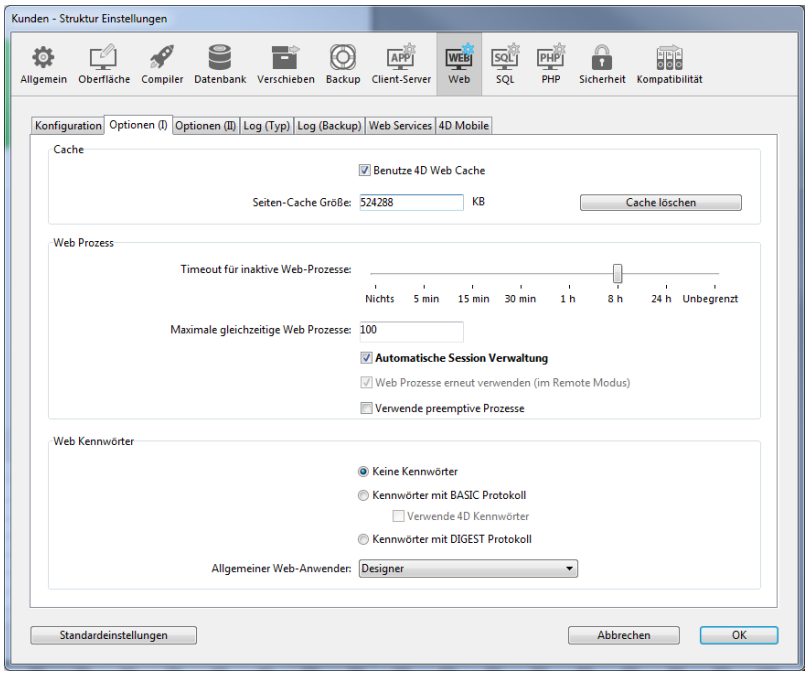
- Ordner werden durch Schrägstrich ("/") voneinander getrennt
- Der Zugriffspfad darf nicht mit einem Schrägstrich ("/") enden.
- Um in der Ordnerhierarchie eine Ebene höher zu gehen, geben Sie vor dem Ordnernamen zwei Punkte ("..") ein.
- Der Zugriffspfad darf nicht mit einem Schrägstrich ("/") beginnen.

Soll Ihre Standard Homepage zum Beispiel lauten "MyHome.htm" und liegt sie im Ordner "Web", geben Sie "Web/MyHome.htm" ein.

Hinweis: Mit dem 4D Befehl **WEB SET HOME PAGE** können Sie für jeden Web Prozess eine Standard Home Page definieren.

Legen Sie keine Standard Home Page an, wird die **Datenbankmethode On Web Connection** aufgerufen. Sie müssen dann selbst für die Übertragung einer Startseite sorgen.

Seite Optionen (I)



Cache für statische Seiten

Der 4D Web Server hat einen Cache, um angeforderte statische Seiten, GIF Bilder, JPEG Bilder (<512 kb) und Stilvorlagen (.css Dateien) in den Speicher zu laden.
Mit dem Cache steigern Sie die Leistung des Web Servers beim Senden von statischen Seiten.
Der Cache wird zwischen allen Web Prozessen aufgeteilt. Seine Größe legen Sie in den Einstellungen der Datenbank fest. Der Cache für statische Seiten ist standardmäßig nicht aktiviert. Um den Cache für statische Seiten zu aktivieren, markieren Sie die Option **Benutze 4D Web Cache**.

Im Bereich **Seiten Cache Größe** können Sie die Größe des Cache verändern. Sie richtet sich nach der Anzahl und der Größe der statischen Seiten Ihrer Web Site, sowie nach den jeweiligen Ressourcen der Host Rechner.

Hinweis: Während Sie mit Ihrer Web Datenbank arbeiten, können Sie mit dem 4D Befehl **WEB GET STATISTICS** die Leistung des Cache prüfen. Stellen Sie zum Beispiel fest, dass die Nutzungsrate des Cache fast 100% erreicht, sollten Sie erwägen, die zugewiesene Größe zu erhöhen.
Auch mit den URL /4DSTATS und /4DHTMLSTATS erhalten Sie Information über den Status des Cache. Weitere Informationen dazu finden Sie im Abschnitt **Information über die Web Site**.

Sobald der Cache aktiviert ist, sucht der 4D Web Server nach der vom Browser angeforderten Seite zuerst im Cache. Findet er die Seite, sendet er sie sofort. Findet er sie nicht, lädt 4D die Seite von der Festplatte und legt sie in den Cache. Ist der Cache voll und wird mehr Platz benötigt, entfernt 4D zuerst die am frühesten aufgerufenen Seiten.

Cache leeren

Sie können jederzeit Seiten und Bilder aus dem Cache entfernen, zum Beispiel, wenn Sie eine statische Seite geändert haben und diese wieder in den Cache laden wollen.
Klicken Sie dazu auf die Schaltfläche **Cache löschen**. Der Cache wird dann sofort geleert.

Hinweis: Sie können auch die spezifische URL **/4DCACHECLEAR** verwenden.

Timeout für interaktive Web-Prozesse

Hier definieren Sie das maximale Timeout zum Schließen inaktiver Web Prozesse auf dem Server.

Maximale gleichzeitige Web Prozesse

Die Option **maximale gleichzeitige Web Prozesse** gibt die Obergrenze für die gleichzeitig auf dem Server geöffneten Web Prozesse an. Sie definiert die maximale Anzahl aller Web Prozesse oder die zum "Pool" an Prozessen gehören. Dieser Parameter verhindert die Überlastung des 4D Server, die bei massiver Anzahl an Anfragen eintreten kann.

Standardmäßig ist der Wert 32000 vorgegeben. Sie können jede Zahl zwischen 10 und 32000 eingeben.
Ist die maximale Anzahl konkurrierender Web Prozesse (minus eins) erreicht, erstellt 4D keine neuen Prozesse und sendet für jede neue Anfrage die Meldung "Server nicht verfügbar" (Status HTTP 503 – Service Unavailable).

Hinweis: Sie können die maximale Anzahl auch über den Befehl **WEB SET OPTION** setzen.

Pool der Web Prozesse

Mit dem "Pool" der Web Prozesse können Sie die Reaktionszeit des Web Server steigern. Diese Reserve reicht vom Minimum (standardmäßig 0) bis zum Maximum (standardmäßig 10) der wiederherzustellenden Prozesse. Sie können diese Prozesse mit dem Befehl **SET DATABASE PARAMETER** ändern. Wurde die maximale Anzahl der Web Prozesse verändert und ist dieser Wert kleiner als der obere Grenzwert des "Pool", wird diese Grenze auf die maximale Anzahl der Web Prozesse herabgesetzt.

Passenden Wert bestimmen

In der Theorie ergibt sich die maximale Anzahl der Web Prozesse aus der Formel: Zugeteilter Arbeitsspeicher/Speichergröße des Web Prozesses (stack size) *
Sie können sich aber auch die Information zu den Web Prozessen ansehen, die der Runtime Explorer anzeigt: aktuelle Anzahl der Web Prozesse und erreichtes Maximum, seit der Web Server in Betrieb ist.

* 4D weist einen Web Prozess als Speichergröße ca. 512 KB für 64-bit Versionen und ca. 256 KB für 32-bit Versionen zu. Diese Werte können je nach Kontext variieren.

Automatische Session-Verwaltung

Aktiviert oder deaktiviert den internen Mechanismus zur automatischen Verwaltung von Benutzersitzungen durch den 4D HTTP Server. Weitere Informationen dazu finden Sie im Abschnitt **Web Sessions verwalten**.

Diese Funktionsweise wird in Anwendungen, die mit 4D v13 oder höher erstellt werden, automatisch aktiviert.
Zur Wahrung der Kompatibilität wird sie jedoch in Anwendungen, die aus früheren 4D Versionen konvertiert wurden, deaktiviert. Sie müssen sie explizit aktivieren, damit Sie die Vorteile der automatischen Verwaltung nutzen können.

Ist diese Option markiert, ist die Option "Web Prozesse erneut verwenden" automatisch markiert (und gesperrt).

Temporäre Kontexte erneut verwenden (im remote Modus)

Diese Option optimiert die Operation des Web Server im remote Modus. Dabei werden Web Prozesse, die zum Verwalten vorheriger Web Anfragen erstellt wurden, wiederhergestellt. Der Web Server von 4D benötigt zum Verwalten jeder Web Anfrage einen spezifischen Web Prozess; im remote Modus meldet sich dieser Prozess bei Bedarf an den 4D Server Rechner an, um auf die Daten und die Engine der Datenbank zuzugreifen. Er generiert dann über eigene Variablen, Auswahlen, etc. einen temporären Kontext. Dieser Prozess wird vernichtet, wenn die Anfrage abgewickelt ist.
Ist die Option **Web Prozesse erneut verwenden** markiert, behält 4D im remote Modus die spezifischen Web Prozesse bei und verwendet sie für nachfolgende Anfragen. Da der Prozess nicht neu aufgebaut werden muss, hat der Web Server eine bessere Performance. Sie müssen jedoch sicherstellen, dass dann die in 4D Methoden verwendeten Variablen systematisch initialisiert werden, damit keine unpassenden Ergebnisse geliefert werden. Außerdem müssen Sie alle aktuellen Auswahlen bzw. Datensätze aus der vorigen Anfrage löschen.

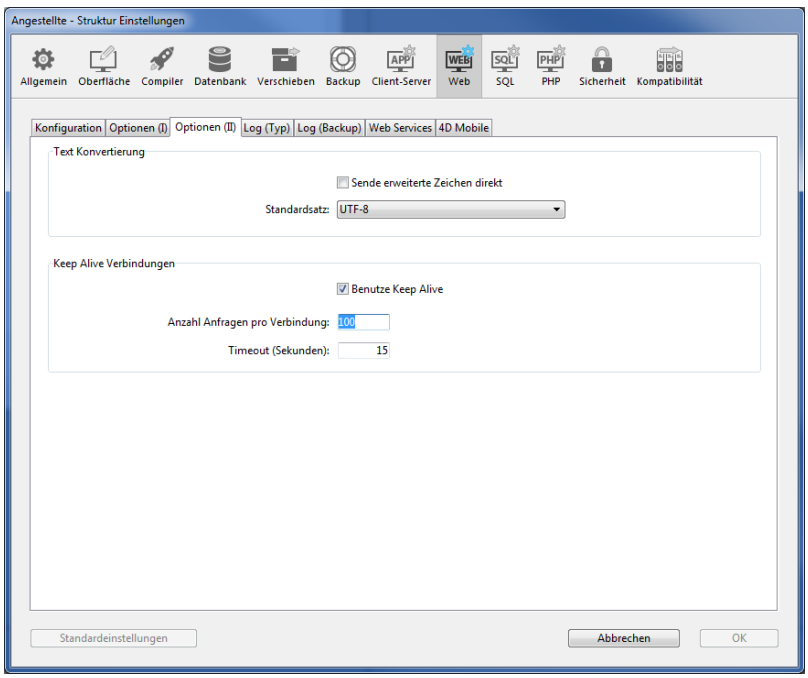
Hinweise:

- Diese Option ist automatisch markiert (und gesperrt), wenn die Option **Automatische Session-Verwaltung** markiert ist. Die Art der Session-Verwaltung basiert auf dem Prinzip Web Prozesse recyceln: Jede Session verwendet denselben Prozess, der während der Lebensdauer der Session beibehalten wird. Beachten Sie jedoch, dass Prozesse der Session von verschiedenen Sessions nicht gemeinsam genutzt werden können: Ist eine Session vorbei, wird der Prozess automatisch vernichtet und nicht wieder verwendet. Von daher ist es in diesem Fall nicht notwendig, die Auswahlen oder Variablen wiederherzustellen.
- Diese Option wirkt sich nur für einen 4D Web Server im remote Modus aus. Bei 4D im lokalen Modus werden alle Prozesse (andere als Session Prozesse) nach dem Verwenden beendet.

Web Kennwörter

Hier sichern Sie den Zugriff auf die Web Site über Kennwörter. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

Seite Optionen (II)



Erweiterte Zeichen direkt senden

Der 4D Web Server konvertiert die erweiterten Zeichen in dynamischen und statischen Web Seiten standardmäßig nach den HTML Standards, bevor er diese Seiten sendet. Browser können diese Zeichen dann interpretieren.
Sie können den Web Server aber auch so einstellen, dass Zeichen des erweiterten Satzes ohne Konvertierung in HTML Einheiten gesendet werden. Tests haben bei den meisten ausländischen Betriebssystemen eine Beschleunigung ergeben, insbesondere beim japanischen System. Damit der 4D Web Server erweiterte Zeichen direkt senden kann, wählen Sie die Option "Sende erweiterte Zeichen direkt".

Standard Zeichensätze

Markieren Sie die Option **Standardsatz**, um einen Zeichensatz für die Verwendung im 4D Web Server auszuwählen. Standardmäßig wird der Zeichensatz UTF-8 verwendet.

Hinweis: Diese Einstellung wird auch zum Generieren von Schnellberichten im HTML Format verwendet (siehe Abschnitt **Schnellbericht ausführen**).

Keep-Alive Verbindungen

Der Web Server von 4D kann keep-alive Verbindungen verwenden. So können Sie für den Austausch der Elemente zwischen Web Browser und Server eine einzige TCP Verbindung offenhalten. Das spart System-Ressourcen und optimiert die Übertragung.

Diese Option ermöglicht, das Aufrechterhalten von TCP Verbindungen für den Web Server zu aktivieren/deaktivieren. Sie ist standardmäßig aktiviert. Das ist für die meisten Fälle sinnvoll, das es den Austausch beschleunigt. Unterstützt der Web Browser nicht die Option keep-alive, wechselt der 4D Web Server automatisch zu HTTP/1.0.

Die Option keep-alive des 4D Web Server gilt für alle TCP/IP Verbindungen (HTTP, HTTPS). Beachten Sie jedoch, dass keep-alive Verbindungen nicht immer für alle 4D Web Prozesse verwendet werden. In einigen Fällen werden andere optimierte interne Funktionen ausgelöst. Keep-alive Verbindungen sind hauptsächlich für statische Seiten sinnvoll.

Es gibt folgende Optionen:

- **Anzahl Anfragen pro Verbindung:** Damit legen Sie die maximale Anzahl von Anfragen und Antworten fest, die keep-alive nutzen können. Diese Einschränkung verhindert, dass der Server durch eine hohe Anzahl hereinkommender Anfragen überflutet wird – eine von Hackern benutzte Technik.
  Standardmäßig ist der Wert 100 vorgegeben. Sie können ihn je nach den Ressourcen des Rechners, welcher den 4D Web Server hostet, erhöhen oder verringern.
- **Timeout:** Damit legen Sie die maximale Zeitspanne (in Sekunden) fest, für die der Web Server eine TCP Verbindung ohne Anfragen vom Web Browser offenhält. Ist die Zeit abgelaufen, schließt der Server die Verbindung.
  Sendet der Web Browser eine Anfrage nach Schließen der Verbindung, wird automatisch eine neue TCP Verbindung angelegt. Dieser Vorgang ist für den Benutzer unsichtbar.

Preemptive Web Prozesse verwenden

MIt dem integrierten Web Server von 4D 64-bit für Windows und OS X können Sie in Ihren kompilierten Anwendungen **preemptive 4D Code** schreiben und so die Vorteile von multi-core Computern nutzen. Sie können Ihren Web-basierten Code inkl. 4D Tags und Web Datenbankmethoden so konfigurieren, damit er gleichzeitig auf so vielen Cores wie möglich laufen kann. Weitere Informationen dazu finden Sie im Abschnitt **Preemptive 4D Prozesse**.

Voraussetzungen

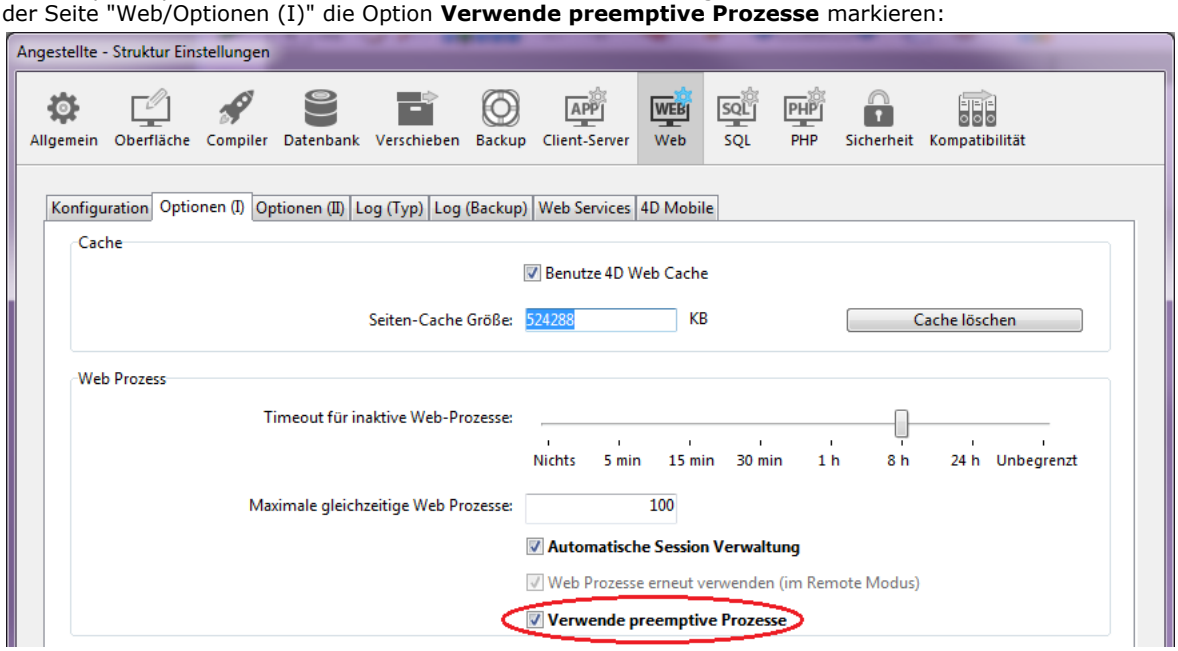
Der preemptive Modus für Web Prozesse lässt sich nur in folgenden Kontexten einsetzen:

- mit der 64-bit Version von 4D
- mit 4D Server oder 4D im lokalen Modus (4D im remote Modus unterstützt nicht den preemptive Modus)
- in kompilierten Anwendungen
- Markierte Datenbankeigenschaft **Verwende preemptive Prozesse** (siehe unten)
- Alle Web-basierten Datenbankmethoden und Projektmethoden wurden vom 4D Compiler als thread-safe bestätigt

Fehlt eine dieser Voraussetzungen, verwendet der Web Server kooperative Prozesse.

Preemptive Modus für den Web Server aktivieren

Um den preemptive Modus für Web Server Code Ihrer Anwendung zu aktivieren, müssen Sie in den Datenbank-Eigenschaften auf der Seite "Web/Optionen (I)" die Option **Verwende preemptive Prozesse** markieren:



Mit dieser Option bewertet der 4D Compiler automatisch die Eigenschaft thread-safe von Web-basiertem Code (siehe unten) und gibt Fehler zurück, wenn das nicht zutrifft.

Thread-safe Web Server Code schreiben

Jeder vom Web Server ausgeführte 4D Code muss thread-safe sein, wenn Ihre Web Prozesse im preemptive Modus laufen sollen. Ist die Option **Verwende preemptive Prozesse** in den Datenbank-Eigenschaften markiert, bewertet der 4D Compiler automatisch folgende Teile der Anwendung:

- Alle Web-basierten Datenbankmethoden:
  - **Datenbankmethode On Web Authentication**
  - **Datenbankmethode On Web Connection**
  - **Datenbankmethode On Web Close Process**
  - **Datenbankmethode On 4D Mobile Authentication**
- Die Projektmethode ***compiler_web*** (unabhängig von ihrer aktuellen Eigenschaft "Ausführungsmodus");
- Im Prinzip jeder Code, der über den Befehl **PROCESS 4D TAGS** im Web Kontext bearbeitet wird, z.B. über .shtml Seiten.
- Jede Projektmethode mit der Methodeneigenschaft "Verfügbar durch 4D HTML Tags und URLS (4DACTION...)"
- Trigger für Tabellen mit dem Attribut "Mit 4D Mobile Service veröffentlichen"
- Über 4D Mobile verfügbare Projektmethoden (markierte Eigenschaft "4D Mobile")

Der Compiler prüft für jede dieser Methoden und Code Teile, ob die thread-safety Vorgaben zutreffen und gibt bei Unstimmigkeiten Fehler zurück. Weitere Informationen dazu finden Sie im Abschnitt **Eine thread-safe Methode schreiben**.

Thread-safety von 4D Befehlen und Web URLs

Ab 4D v16 sind die meisten Web-basierten 4D Befehle, Datenbankmethoden und URLs thread-safe und im preemptive Modus verwendbar:

4D Befehle

Alle 4D Web-basierten Befehle sind thread-safe, z.B.:

- alle Befehle im Kapitel **Web Server**
- alle Befehle im Kapitel **HTTP Client**

Datenbankmethoden

Folgende Datenbankmethoden sind thread-safe und im preemptive Modus verwendbar:

- **Datenbankmethode On Web Authentication**
- **Datenbankmethode On Web Connection**
- **Datenbankmethode On Web Close Process**
- **Datenbankmethode On 4D Mobile Authentication**

Natürlich muss der Code, den diese Methoden ausführen, ebenfalls thread-safe sein.

Web Server URLs

Folgende 4D Web Server URLs sind thread-safe und im preemptive Modus verwendbar:

- 4daction/ (die aufgerufene Projektmethode muss auch thread-safe sein)
- 4dcgi/ (die aufgerufene Datenbankmethode muss auch thread-safe sein)
- 4dscript/ (als URL überholt, wird als tag verwendet)
- 4dwebtest/
- 4dblank/
- 4dstats/
- 4dhtmlstats/
- 4dcacheclear/
- rest/
- 4dimgfield/ (generiert von **PROCESS 4D TAGS** für Web Anfragen auf Bildfelder)
- 4dimg/ (generiert von **PROCESS 4D TAGS** für Web Anfragen auf Bildvariablen)

Folgende 4D Web Server URLs sind nicht-thread-safe und werden nicht im preemptive Modus unterstützt:

- 4dsync
- 4dsqlauth (überholt, verwendet für Flex 1.1)

Icon für preemptive Web Prozess

Im Fenster Runtime Explorer sowie 4D Server Administration sind preemptive Web Prozesse mit einem spezifischen Icon gekennzeichnet:

Prozesstyp	Icon
Preemptive Web Methode	

Information über die Web Site

4D bietet Informationen über die Funktionsweise Ihrer 4D Web Site.

- Sie können die Web Site über spezielle URLs steuern (*/4DSTATS*, */4DHTMLSTATS*, */4DCACHECLEAR* und */4DWEBTEST*).
- Sie können ein Logbuch von allen Anfragen erstellen.
- Sie können Informationen über den Web Server im Fenster **Runtime Explorer** auf der Seite **Überwachen** erhalten

Hinweis: Aus Sicherheitsgründen wird ab 4D v15 R2 im 4D Web Server die Methode HTTP TRACE standardmäßig deaktiviert, d.h. der 4D Web Server gibt den Fehler 405 ("Methode nicht erlaubt") zurück, wenn eine HTTP TRACE Anfrage empfangen wird. Um diese Methode explizit zu aktivieren, müssen Sie die Option Web HTTP TRACE mit dem Befehl **WEB SET OPTION** verwenden.

URLs zum Verwalten des Web Server

4D Web Server erlaubt vier spezielle URLs: */4DSTATS*, */4DHTMLSTATS*, */4DCACHECLEAR* und */4DWEBTEST*.

- */4DSTATS*, */4DHTMLSTATS*, */4DCACHECLEAR* stehen nur dem Designer und Administrator der Datenbank zur Verfügung. Ist das 4D Kennwortsystem der Datenbank nicht aktiviert, sind diese URLs für alle Benutzer verfügbar.
- */4DWEBTEST* ist immer verfügbar

/4DSTATS

Die URL */4DSTATS* gibt verschiedene Einträge in einer HTML Tabelle (im Browser anzeigbar) zurück:

- **Cache Current Size:** Aktuelle Größe des Web Server Cache (in Bytes)
- **Cache Max Size:** Maximale Größe des Cache (in Bytes)
- **Cached Object Max Size:** Maximale Größe jedes Objekts im Cache (in Bytes)
- **Cache Use:** Prozentsatz des verwendeten Cache
- **Cached Objects:** Anzahl der im Cache gefundenen Objekte (Seiten, Bilddateien, etc.)

Weitere Informationen über den Cache statischer Seiten und Bilder finden Sie im Abschnitt **Web Server, Einstellungen**.

Mit diesen Angaben können Sie die Funktionsweise Ihres Server prüfen und bei Bedarf die entsprechenden Parameter anpassen.

Hinweis: Über den Befehl **WEB GET STATISTICS** erhalten Sie auch Informationen, wie der Cache für statische Seiten eingesetzt wird.

/4DHTMLSTATS

Die URL */4DHTMLSTATS* gibt, ebenfalls als HTML Tabelle, dieselbe Information wie die URL */4DSTATS* zurück, mit dem Unterschied, dass das Feld **Cached Objects** nur die HTML Seiten zählt, d.h. ohne .Bilddateien. Sie gibt außerdem das Feld **Filtered Objects** zurück.

- **Filtered Objects:** Anzahl der Objekte im Cache, die nicht von der URL gezählt werden (insbesondere Bilder)

/4DCACHECLEAR

Die URL */4DCACHECLEAR* löscht sofort die statischen Seiten und Bilder im Cache. So erzwingen Sie die Aktualisierung, wenn Seiten geändert wurden.

/4DWEBTEST

Die URL */4DWEBTEST* dient zum Überprüfen des Web Server Status. Wird diese URL aufgerufen, gibt 4D eine Textdatei zurück, in der nur folgende HTTP Felder ausgefüllt sind:

- **Date:** Aktuelles Datum im Format RFC 822
  Beispiel: "Mon, 16 Jan 2012 13:12:50 GMT"
- **Server:** 4D_Version/interne Versionsnummer
  Beispiel: "4D_v12/12.3.0"
- **User Agent:** Name und Version @ IP Client Adresse
  Beispiel: "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:9.0.1) Gecko/20100101 Firefox/9.0.1 @ 127.0.0.1"

Log-Datei der Verbindung

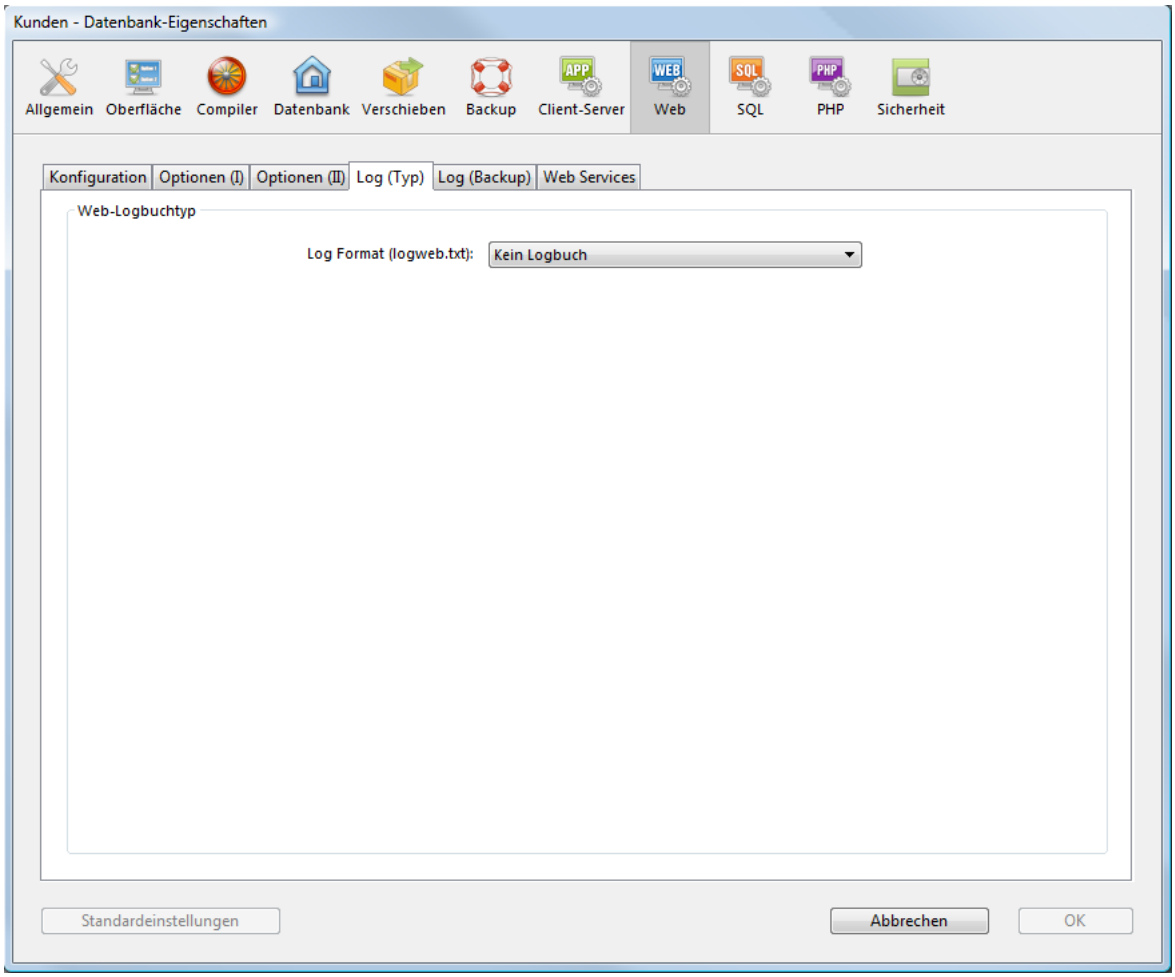
In 4D erhalten Sie eine Log-Datei der Anfragen.

Die Datei hat den Namen "weblog.txt" und wird automatisch abgelegt:

- In 4D im lokalen Modus und 4D Server im Ordner **Logs** neben der Strukturdatei der Anwendung
- In 4D im remote Modus im Unterordner **Logs** des Anwendungsordners der Arbeitsstation (Cache Ordner).

Aktivierung und Format

Die Aktivierung des Logbuchs und die Konfiguration seines Inhalts definieren Sie in den Datenbank-Eigenschaften auf der Seite **Web>Log (Typ)**:



Hinweis: Sie können das Web Logbuch auch per Programmierung über den Befehl **SET DATABASE PARAMETER** (4D v12) bzw. **WEB SET OPTION** (4D v13 und höher) aktivieren und deaktivieren.

Das PopUp-Menü Log Format im oberen Bereich bietet folgende Optionen:

- **Kein Logbuch:** Ist diese Option gewählt, erzeugt 4D kein Web Logbuch.
- **CLF (Common Log Format)**: Ist diese Option gewählt, wird das Web Logbuch im Format CLF erstellt. Im Format CLF Format stellt jede Zeile der Datei eine Anfrage dar, wie z.B.:
  host rfc931 user [DD/MMM/YYYY:HH:MM:SS] "query" state length
  Auf jedes Feld folgt ein Leerzeichen, jede Zeile endet mit der Folge CR/LF (Zeichen 13, Zeichen 10).
  - host: IP Adresse des Client (z.B. 192.100.100.10)
  - rfc931: Information, die nicht 4D generiert, ist immer - (Minuszeichen)
  - user: Benutzername wie zugelassen, oder - (Minuszeichen). Leerzeichen in Benutzernamen werden durch _ (Unterstrich) ersetzt.
  - DD: Tag, MMM: Monatsname, abgekürzt mit drei Buchstaben (Jan, Feb,...), YYYY: Jahr, HH: Stunde, MM: Minuten, SS: Sekunden

  Datum und Uhrzeit richten sich nach dem jeweiligen Server.
  - request: Vom Benutzer gesendete Anfrage (z.B. GET /index.htm HTTP/1.0)
  - state: Vom Server zurückgegebene Antwort.
  - length: Größe der zurückgegebenen Daten (ohne HTTP Kopfteil) oder 0.

  Hinweis: Diese Operationen werden zur Wahrung der Performance vor Übertragen auf die Festplatte im Pufferspeicher in 1 Kb Paketen gesichert. Sie werden auch auf die Festplatte geschrieben, wenn nicht innerhalb der nächsten 5 Sekunden eine Anfrage gesendet wird. Für den Status sind folgende Werte möglich:
  200: OK
  204: Kein Inhalt
  302: Redirektion
  304: Nicht geändert
  400: Inkorrekte Anfrage
  401: Authentifizierung erforderlich
  404: Nicht gefunden
  500: Interner Fehler
  Das CLF Format lässt sich nicht anpassen.

- **DLF (Combined Log Format)**: Ist diese Option gewählt, wird das Web Logbuch im Format DLF erzeugt. Das DLF Format ähnelt dem CLF Format und verwendet genau dieselbe Struktur. Es fügt lediglich am Ende jeder Anfrage zwei weitere HTTP Felder hinzu: referer und user-agent.
  - Referer: Enthält die URL der Seite, die auf das angefragte Dokument zeigt.
  - user-agent: Enthält Name und Version von Browser oder Software des Benutzers, der die Anfrage gestartet hat.

  Das DLF Format lässt sich nicht anpassen.
- **ELF (Extended Log Format)**: Ist diese Option gewählt, wird das Web Logbuch im Format ELF erzeugt. Dieses Format ist in der Welt der HTTP Browser weitverbreitet. Sie können damit ausgeklügelte Logbücher für spezielle Zwecke einrichten. Das ELF Format lässt sich also anpassen: Sie können auswählen, welche Felder gespeichert werden und in welcher Reihenfolge sie in die Datei eingefügt werden.
- **WLF (Webstar Log Format)**: Ist diese Option gewählt, wird das Web Logbuch im Format WLF erzeugt. Dieses Format wurde speziell für den WebSTAR Server entwickelt. Es ähnelt dem ELF Format, und hat lediglich ein paar zusätzliche Felder.

Es lässt sich ebenfalls anpassen.

Felder konfigurieren
Wählen Sie das Format ELF (Extended Log Format) oder WLF (WebStar Log Format), zeigt der Bereich "Web Log Token Auswahl" die für das gewählte Format verfügbaren Felder. Sie müssen jedes Feld auswählen, das in das Logbuch übernommen werden soll.

Hinweis: Sie können dasselbe Feld nicht zweimal auswählen

Nachfolgende Tabelle zeigt die für jedes Format verfügbare Felder in alphabetischer Reihenfolge und beschreibt den Inhalt:

Feld	ELF	WLF	Wert
BYTES_RECEIVED		X	Anzahl der vom Server empfangenen Bytes
BYTES_SENT	X	X	Anzahl der vom Server an den Client gesendeten Bytes
C_DNS	X	X	IP Adresse des DNS (ELF: Feld ist identisch zu C_IP field)
C_IP	X	X	IP Adresse des Client (z.B. 192.100.100.10)
CONNECTION_ID		X	ID Nummer der Verbindung
CS(COOKIE)	X	X	Information über Cookies in der HTTP Anfrage
CS(HOST)	X	X	Host Feld der HTTP Anfrage
CS(REFERER)	X	X	URL der Seite, die auf das angeforderte Dokument zeigt
CS(USER_AGENT)	X	X	Information über Software und Betriebssystem des Client
CS_SIP	X	X	IP Adresse des Server
CS_URI	X	X	URI auf der Anfrage erstellt wurde
CS_URI_QUERY	X	X	Suchparameter der Anfrage
CS_URI_STEM	X	X	Teil der Anfrage ohne Suchparameter
DATE	X	X	DD: Tag, MMM: Abkürzung für Monat mit drei Buchstaben (Jan, Feb, etc.), YYYY: Jahr
METHOD	X	X	HTTP Methode für die Anfrage an den Server
PATH_ARGS		X	CGI Parameter: String nach dem Zeichen "$"
STATUS	X	X	Vom Server gelieferte Antwort
TIME	X	X	HH: Stunde, MM: Minuten, SS: Sekunden
TRANSFER_TIME	X	X	Vom Server angefragte Zeit zum Erstellen der Antwort
USER	X	X	Benutzername wenn authentifiziert; sonst - (Minuszeichen) Enthält der Benutzername Abstände, werden sie durch Unterstriche ersetzt
URL	X		Vom Client angefragte URL

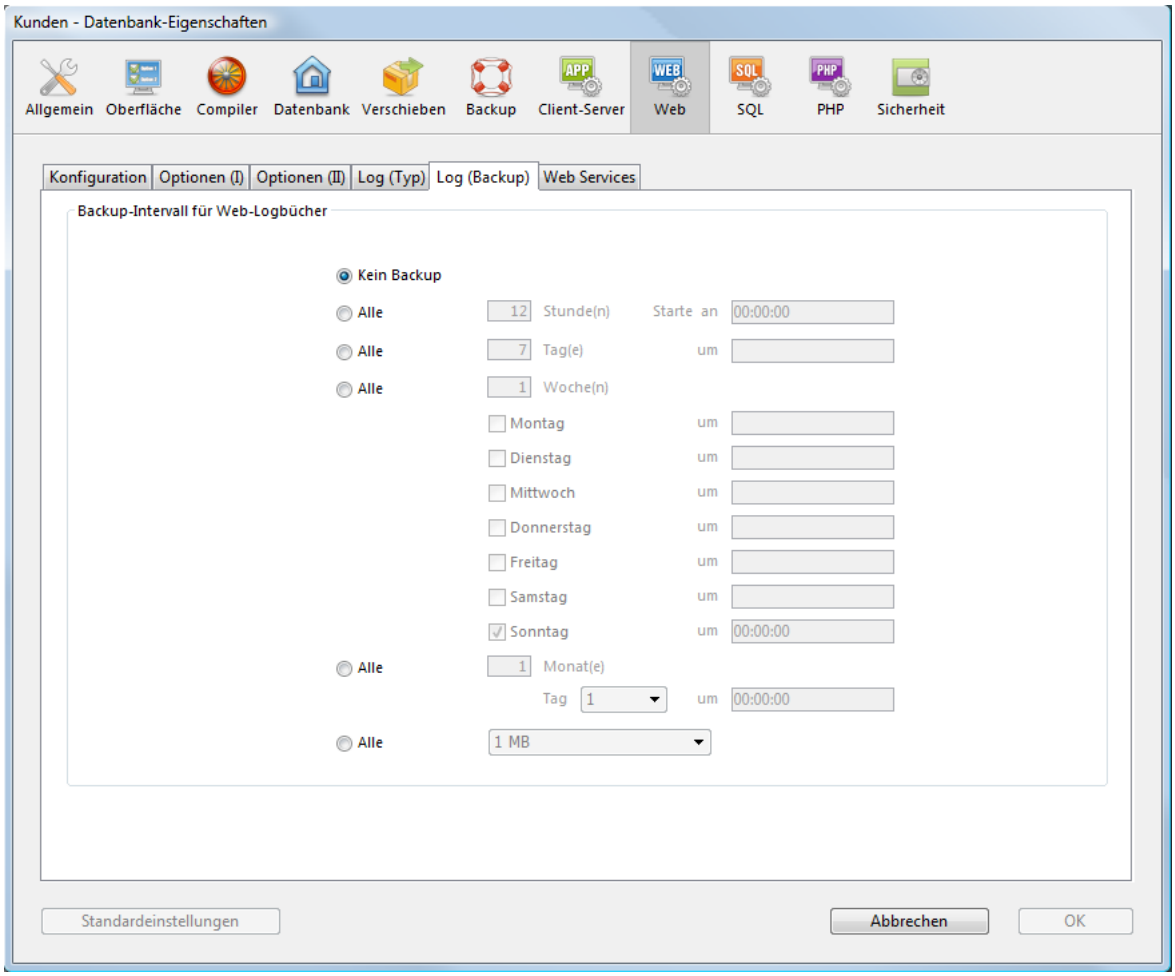
Hinweis: Datum und Zeit werden in GMT (Greenwich Mean Time) angegeben.

Backup-Intervall

Da ein Web-Logbuch eine beträchtliche Größe erreichen kann, können Sie eine automatische Archivierung einrichten. Die Ausführung eines Backup kann in einem bestimmten Zeitabstand erfolgen, ausgedrückt in Stunden, Tagen, Wochen oder Monaten; oder ab einer spezifischen Dateigröße. Sind der Zeitpunkt oder die Dateigröße erreicht, schließt und archiviert 4D automatisch das aktuelle Web-Logbuch und erstellt ein neues.
Das Backup des Web-Logbuchs wird in einem Ordner mit Namen "LogwebArchives" abgelegt, der auf derselben Ebene wie die Datei logweb.txt erstellt wird, d.h. neben der Strukturdatei der Datenbank.
Die archivierte Datei wird nach folgendem Muster benannt: "DYYYY_MM_DD_Thh_mm_ss.txt.
Beispiel: Die am 3. September 2007 um 3.57 p.m archivierte Datei erhält den Namen: D2007_09_03_T15_50_07.txt.

Backup Parameter

In den Datenbank-Eigenschaften auf der Seite **Web>Log (Backup)** können Sie automatische Backup Parameter für das Web Logbuch definieren:



Sie müssen zuerst durch Anklicken des entsprechenden Optionsfeldes das Backup-Intervall, also Tage, Wochen, etc. oder die max. Dateigröße festlegen. Dann definieren Sie den gewünschten Zeitpunkt für das Backup.

- **Kein Backup**: Damit ist ein planmäßiges Backup deaktiviert.
- **Alle X Stunden**: Damit wird ein Backup nach der definierten Stundenanzahl durchgeführt. Sie können Werte von 1 bis 24 eingeben.
  - **Start um**: Hier legen Sie die Zeit fest, zu der das erste Backup starten soll.
- **Alle X Tage um X**: Damit wird ein Backup nach der definierten Tagesanzahl durchgeführt. Tragen Sie 1 ein, wenn es täglich durchgeführt werden soll. Sie müssen auch die Zeit festlegen, zu der das erste Backup starten soll.
- **Alle X Wochen, Wochentag, um X**: Damit wird ein Backup nach der definierten Wochenanzahl durchgeführt. Tragen Sie 1 ein, wenn es wöchentlich durchgeführt werden soll. Sie müssen auch den Wochentag und die Zeit festlegen, zu der das Backup jeweils starten soll. Bei Bedarf können Sie auch mehrere Wochentage auswählen. Sie können z.B. zwei wöchentliche Backups festlegen, einmal am Mittwoch, einmal am Freitag.
- **Alle X Monate, X. Tag um X**: Damit wird ein Backup nach der definierten Monatsanzahl durchgeführt. Tragen Sie 1 ein, wenn es monatlich durchgeführt werden soll. Sie müssen auch den Tag und die Zeit festlegen, zu der das Backup jeweils starten soll.
- **Alle X MB:** Damit wird ein Backup nach der festgelegten Größe des aktuellen Web Logbuchs durchgeführt. Das Backup startet automatisch, wenn die Datei die definierte Größe erreicht. Sie können die Grenze 1, 10, 100 oder 1000 MB eintragen.

Hinweis: Ist ein planmäßiges Backup festgelegt und war der Web Server zum vorgesehenen Termin nicht gestartet, betrachtet 4D beim nächsten Start das Backup als gescheitert und führt die entsprechenden Operationen aus, so wie sie in den Datenbank-Eigenschaften festgelegt wurden.

Runtime Explorer Information

Die Seite **Überwachen** im Runtime Explorer zeigt unter "Web" Informationen zum Web Server, insbesondere:

- **Verwendung des Web Cache:** Zeigt die Anzahl Seiten im Web Cache sowie den verwendeten Prozentsatz. Diese Information ist nur bei aktivem Web Server verfügbar und nur, wenn die Cache Größe höher als 0 (Null) ist.
- **Verbrauchte Zeit Web Server:** Zeigt die verwendete Zeit des Web Server in Stunden:Minuten:Sekunden. Diese Information ist nur bei aktivem Web Server verfügbar.
- **Anzahl Web Hits:** Gibt die Gesamtanzahl der empfangenen HTTP Anfragen an, seit der Web Server in Betrieb ist, sowie die unmittelbare Anzahl der Anfragen pro Sekunde (Messung zwischen zwei Updates des Runtime Explorers). Diese Information ist nur bei aktivem Web Server verfügbar.

Weitere Informationen dazu finden Sie im Abschnitt **Seite Überwachen** des Handbuchs *4D Designmodus*.

Eigene HTTP Fehlerseiten definieren

Mit dem 4D Web Server können Sie an Clients gesendete HTTP Fehlerseiten, basierend auf dem Statuscode der Server Antwort, anpassen. Fehlerseiten zeigen folgendes:

- Statusmeldungen, die mit 4 beginnen (Client Fehler), z.B. 404
- Statusmeldungen, die mit 5 beginnen (Server Fehler), z.B. 501

Die komplette Beschreibung der HTTP Fehlerstatus-Codes finden Sie auf der Seite List of HTTP status codes (Wikipedia).

Funktionsweise

Um standardmäßige 4D Web Server Fehlerseiten mit Ihren eigenen Seiten zu ersetzen, gehen Sie folgendermaßen vor:

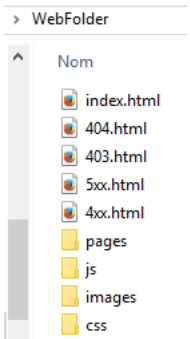
- Setzen Sie eigene HTML Seiten auf die erste Ebene des Web Ordners der Anwendung
- Nennen Sie die eigenen Seiten "{*statusCode*}.html" (z.B."404.html").

Sie können eine Fehlerseite pro Statuscode definieren bzw. eine generische Fehlerseite für eine Reihe von Fehlern mit Namen "{*number*}xx.html": z.B. "4xx.html" für generische Client Fehler. Der 4D Web Server sucht zuerst nach einer {*statusCode*}.html Seite, wenn sie nicht existiert, nach einer generischen Seite.
Gibt eine HTTP Antwort einen Statuscode 404 zurück, läuft folgendes ab:

1. 4D Web Server versucht eine Seite "404.html" aus dem Web Ordner der Anwendung zu senden
2. Wird keine gefunden, versucht 4D Web Server eine Seite "4xx.html" aus dem Web Ordner der Anwendung zu senden.
3. Wird keine gefunden, verwendet 4D Web Server dann seine standardmäßige Fehlerseite.

Beispiel

Sie haben folgende eigenen Seiten in Ihrem Web Ordner definiert:



- Die Seiten "403.html" oder "404.html" werden jeweils für 403 oder 404 HTTP Antworten verwendet
- Die Seite "4xx.html" wird für andere 4xx Fehlerstatus Meldungen, wie 400, 401, etc. verwendet
- Die Seite "5xx.html" wird für alle 5xx Fehlerstatus Meldungen verwendet.

TLS Protokoll verwenden (HTTPS)

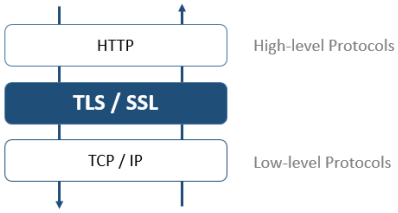
Der 4D Web Server kann über das TLS Protokoll (Transport Layer Security) - dem Nachfolger von SSL (Secured Socket Layer) - im gesicherten Modus kommunizieren. Standardmäßig ist die in 4D unterstützte Mindestversion TLS 1.2.

Definition

Das TLS Protokoll (Nachfolger von SSL) sichert die Kommunikation zwischen zwei Anwendungen ab, das sind hauptsächlich Web Server und Browser. Diese Art von Protokoll ist weit verbreitet und mit den meisten Web Browsern kompatibel.

In der Netzwerkarchitektur reiht sich das TLS Protokoll ein zwischen TCP/IP auf niederer Ebene und dem HTTP Protokoll auf hoher Ebene.

Netzwerkarchitektur mit TLS:



Hinweis: Das TLS Protokoll eignet sich auch zum Absichern von Standard Client/Server Verbindungen sowie SQL Server Verbindungen. Weitere Informationen dazu finden Sie im Abschnitt **Client/Server Verbindungen verschlüsseln** des Handbuchs *4D Server* und im Abschnitt **Configuration of 4D SQL Server** des Handbuchs *4D SQL Reference*.

Das TLS Protokoll bürgt für die Identität von Sender und Empfänger, sowie für die Vertraulichkeit und Vollständigkeit der ausgetauschten Informationen:

- **Authentifizierung:** Die Identität von Sender und Empfänger werden bestätigt.
- **Vertraulichkeit:** Die gesendeten Daten sind verschlüsselt, damit sie für unbefugte Dritte nicht lesbar sind.
- **Integrität:** Die empfangenen Daten wurden weder wissentlich noch versehentlich verändert.

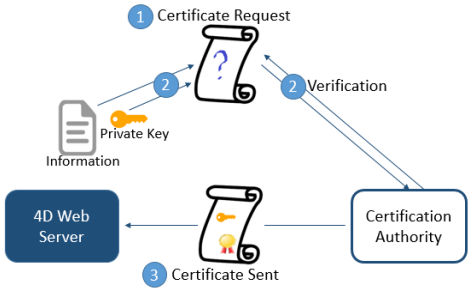
Das Sicherheitsprinzip von TLS basiert auf Verschlüsselungsalgorithmen mit einem Schlüsselpaar, d.h. einem privaten und einem öffentlichen Schlüssel.
Mit dem privaten Schlüssel werden die Daten verschlüsselt. Diesen bewahrt der Sender (Web Site) auf. Mit dem öffentlichen Schlüssel werden die Daten entschlüsselt. Er wird über das Zertifikat an den Empfänger weitergegeben (Web Browser). Für den Einsatz von TLS im Internet wird ein Herausgeber für das Zertifikat benötigt, z.B Verisign®. Der Herausgeber stellt der anfragenden Web Site gegen eine Gebühr ein Zertifikat aus, das die Identität des Servers garantiert und den öffentlichen Schlüssel zur Kommunikation im gesicherten Modus enthält.

Hinweis: Weitere Informationen über die Funktionsweise der Verschlüsselung und die Verwendung von öffentlichen und privaten Schlüsseln finden Sie unter dem Befehl **ENCRYPT BLOB**.

Zertifikat erhalten

Für einen Server im gesicherten Modus ist ein digitales Zertifikat notwendig, ausgestellt von einer Zertifizierungsstelle. Es enthält verschiedene Informationen, wie die ID der Web Site und den öffentlichen Schlüssel zur Kommunikation mit der Site. Dieses Zertifikat wird an die Clients (Web Browser) übertragen werden, die sich an die Web Site anmelden. Wurde es identifiziert und angenommen, erfolgt die Kommunikation im gesicherten Modus.

Hinweis: Ein Browser akzeptiert nur die Zertifikate einer Zertifizierungsstelle, wenn sie in seinen Eigenschaften registriert ist.



Die Auswahl der Zertifizierungsstelle hängt von mehreren Faktoren ab. Je bekannter der Herausgeber ist, desto mehr Browser erkennen die Zertifikate an, die er ausstellt, desto höher ist aber auch die zu zahlende Gebühr.

Um ein digitales Zertifikat zu erhalten:

1. Generieren Sie über den Befehl **GENERATE ENCRYPTION KEYPAIR** einen privaten Schlüssel.
   Warnung: Aus Sicherheitsgründen muss der private Schlüssel geheim bleiben. Er muss immer auf dem Server Rechner verbleiben. Für den Web Server muss die Datei **Key.pem** im Ordner mit der Struktur der Datenbank liegen.
2. Richten Sie über den Befehl **GENERATE CERTIFICATE REQUEST** eine Zertifikatsanfrage ein.
3. Senden Sie diese Anfrage an die ausgewählte Zertifizierungsstelle.Zum Ausfüllen der Anfrage müssen Sie sich unter Umständen mit der Zertifizierungsstelle in Verbindung setzen. Diese prüft die Echtheit der übermittelten Informationen. Die Anfrage des Zertifikats wird in einem BLOB im Format PEM (Privacy Enhanced Mail) generiert. Dieses Format autorisiert das Kopieren und Einfügen der Schlüssel in Textform und den Versand per E-Mail ohne das Risiko, dass ihr Inhalt verändert

wird. Sie können also zum Beispiel das BLOB mit der Zertifikatsanfrage über den Befehl **BLOB TO DOCUMENT** in einem Textdokument sichern, es dann öffnen, seinen Inhalt kopieren und in ein E-Mail bzw. ein Web Formular einfügen, das für die Zertifizierungsstelle bestimmt ist.

4. Haben Sie Ihr Zertifikat erhalten, erstellen Sie ein Textdokument mit Namen „cert.pem" und kopieren den Inhalt des Zertifikats in dieses Dokument.
   Sie erhalten das Zertifikat meist als E-Mail oder als HTML Formular. 4D akzeptiert alle für die jeweilige Plattform (PC, Mac OS, Linux...) kompatiblen Textformate. Das Zertifikat muss jedoch im PEM Format sein, z.B. PKCS codiert in base64.
   Hinweis: CR Zeichen nur für Zeilenende werden nicht unterstützt. Sie müssen CRLF oder LF verwenden.
5. Legen Sie das Dokument „cert.pem" an die passende Stelle. Für den Web Server ist es der Ordner mit der Strukturdatei der Datenbank. Der Web Server kann jetzt im gesicherten Modus arbeiten. Ein Zertifikat ist im allgemeinen 6 - 12 Monate gültig.

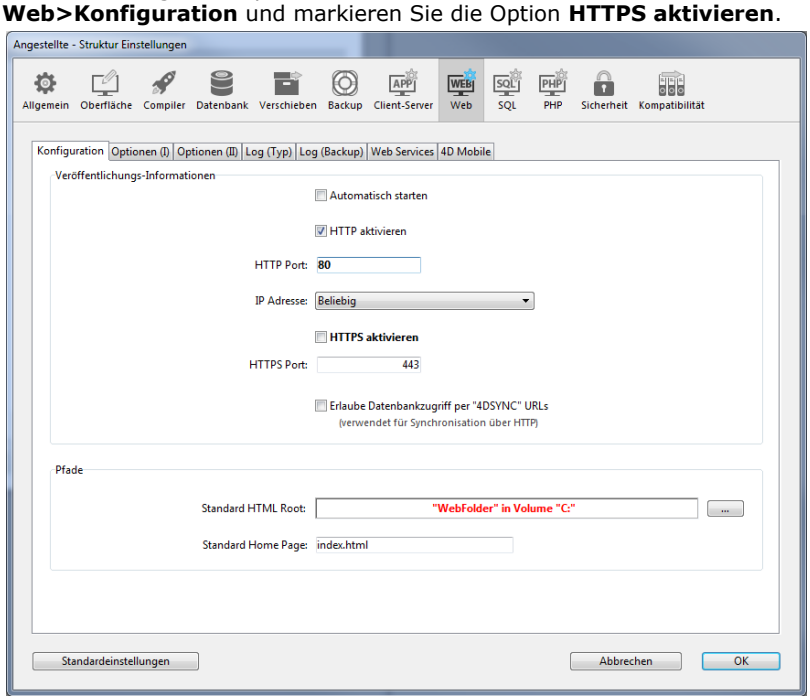
TLS in 4D installieren und aktivieren

Wollen Sie das TLS Protokoll mit dem 4D Web Server verwenden, müssen auf dem Server an verschiedenen Stellen folgende Komponenten installiert sein:

- *4DSLI.DLL* (Windows) oder *4DSLI.bundle* (Mac OS): Schnittstelle für die gesicherte Ebene (Secured Layer Interface) zum Verwalten von TLS. Diese Datei wird standardmäßig installiert, sie liegt:
  - Unter Windows neben der ausführbaren Datei von 4D oder 4D Server
  - Auf Mac im Unterordner *Native Components* des 4D bzw. 4D Server Pakets.
  Hinweis: *4DSLI.DLL* benötigen Sie auch für die Verschlüsselungsbefehle **ENCRYPT BLOB** und **DECRYPT BLOB**.

- *key.pem* (Dokument mit dem Schlüssel zur privaten Verschlüsselung) und *cert.pem* (Dokument mit dem Zertifikat):
  - in 4D im lokalen Modus oder 4D Server müssen diese Dateien neben der Strukturdatei der Anwendung liegen,
  - in 4D im remote Modus müssen diese Dateien im Ordner für lokale Ressourcen der 4D Anwendung auf dem remote Rechner liegen. Weitere Informationen dazu finden Sie im Abschnitt **Ordner 4D Client Database (Client Rechner)**. Sie müssen diese Dateien manuell auf den remote Rechner kopieren.
  Hinweis: 4D enthält standardmäßige *key.pem* und *cert.pem* Dateien. Für eine höhere Sicherheitsstufe raten wir dringend, diese Dateien durch ihre eigenen Zertifikate zu ersetzen.

Die Installation dieser Elemente ermöglicht, TLS für Verbindungen zum 4D Web Server zu verwenden. Damit der 4D Web Server TLS Verbindungen akzeptiert, müssen Sie HTTPS aktivieren. Gehen Sie dazu in den Datenbank-Eigenschaften auf die Seite **Web>Konfiguration** und markieren Sie die Option **HTTPS aktivieren**.



TLS Verbindungen sind standardmäßig erlaubt. Sie können diese Option deaktivieren, wenn Sie mit Ihrem Web Server keine TLS Funktionalitäten nutzen wollen oder auf demselben Rechner ein anderer Web Server mit gesicherter Verbindung operiert.

Für den HTTPS Port ist standardmäßig die Nummer 443 reserviert. Sie können diese Nummer im Eingabebereich **HTTPS Port** verändern, um beispielsweise die Sicherheit des Web Server zu erhöhen. Weitere Informationen dazu finden Sie im Abschnitt **Web Server, Einstellungen**. Der hier definierte HTTPS Port wird für die Standardverbindungen des Web Servers verwendet.

Hinweis: Die anderen Datenbank-Eigenschaften zum Verwalten des 4D Web Servers gelten immer, egal, ob der TLS Modus aktiviert bzw. nicht aktiviert ist. Das sind die Kennwörter, die Zeitspanne bis zum Abschalten der Verbindung (Timeout), die Größe des Cache, etc.

Perfect Forward Secrecy (PFS)

PFS bietet in Ihrer Kommunikation eine zusätzliche Sicherheitsebene. PFS benutzt keine vorgegebenen Austauschschlüssel, sondern erstellt Sitzungsschlüssel zwischen den kommunizierenden Parteien kooperativ mit Diffie-Hellman (DH) Algorithmen. Diese Art Schlüssel erstellt ein "gemeinsames Geheimnis" (shared secret), das eine Beeinträchtigung durch außenstehende Parteien unterbindet.

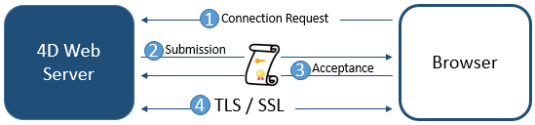
PFS ist automatisch aktiviert, wenn TLS auf dem 4D Web Server aktiv ist. Existiert die Datei *dhparams.pem* (Dokument mit dem privaten DH Schlüssel des Servers) noch nicht, generiert 4D sie automatisch mit der Schlüsselgröße 2048. Das kann beim ersten Erstellen dieser Datei ein paar Minuten dauern. Sie wird an dieselbe Stelle wie die Dateien *key.pem* und *cert.pem* gelegt (siehe Abschnitt **TLS in 4D installieren und aktivieren**).

Verwenden Sie eine eigene Cipher-Liste und wollen PFS aktivieren, müssen Sie prüfen, ob sie Eingaben mit DH oder ECDH (Elliptic-curve Diffie–Hellman) Algorithmen enthält.

Weitere Informationen dazu finden Sie unter dem Selektor SSL cipher list des Befehls **SET DATABASE PARAMETER**.

Verbindung mit Browsern in TLS

Damit eine Web Verbindung im gesicherten Modus ausgeführt wird, muss die vom Browser gesendete URL mit https statt http beginnen.
In diesem Fall erscheint im Browser eine Meldung. Klickt der Benutzer auf **OK**, sendet der Web Server das Zertifikat an den Browser.



Browser und Web Server bestimmen dann, welcher Verschlüsselungsalgorithmus für die Verbindung verwendet wird. Der Server bietet verschiedene symmetrische Verschlüsselungsalgorithmen. Von den gängigen Algorithmen wird immer der leistungsstärkste verwendet.

Warnung: Die erlaubte Verschlüsselungsebene hängt von der geltenden Rechtslage des jeweiligen Landes ab.

Verbindungsmodus verwalten

Für den Einsatz von TLS im 4D Web Server muss in den **Datenbank-Eigenschaften** auf der Seite **Web > Konfiguration** die Option "HTTPS aktivieren" markiert werden. Dann können Verbindungen mit dem Web Server im gesicherten Modus laufen. Ist jedoch auch die Option "HTTP aktivieren" markiert, müssen Sie berücksichtigen, dass der Browser weiterhin zwischen HTTPS und HTTP wechseln kann. Der Benutzer kann beispielsweise im URL Bereich des Browsers “HTTPS” durch “HTTP” ersetzen.

Hinweise:

- Der aktuelle Verbindungsmodus lässt sich mit der Funktion **WEB Is secured connection** abfragen.
- Mit dem Befehl **WEB SET OPTION** lassen sich HTTP und/oder HTTPS auch für eine Sitzung setzen.

Sie können Anfragen im nicht-gesicherten Modus verbieten oder umleiten:

- Um http Umleitungen zu verbieten, deaktivieren Sie die Option **HTTP aktivieren** oder verwenden **WEB SET OPTION** mit dem entsprechenden Selektor. Dann werden HTTP Anfragen des Client vom 4D Web Server ignoriert und es erscheint eine Fehlermeldung.
- Um HTTP Anfragen automatisch auf HTTPS umzuleiten, können Sie HTTP Strict Transport Security (HSTS) über den Befehl **WEB SET OPTION** mit dem Selektor Web HSTS enabled aktivieren. Dann ist die Verwendung von TLS zwingend und stellt sicher, dass alle Kommunikationen über HTTPS ablaufen. Weitere Informationen dazu finden Sie unter dem Befehl **WEB SET OPTION**.

XML und WML Unterstützung

WML

Der Web Server unterstützt WML (Wireless Markup Language) Technologie. Damit können Sie Daten in eine 4D Datenbank über Handy oder Personal Digital Assistents (PDA) und Web Appliances eingeben und lesen.

Hinweis: Die Seitenbeschreibungssprache WML, die dem WAP Protokoll (Wireless Application Protocol) zugeordnet ist, wird von verschiedenen Unternehmen entwickelt. WAP bietet eine Reihe von Werkzeugen für die Kommunikation über Netz, mit denen Handys und Personal Digital Assistents in Web Seiten veröffentlichte Texte lesen können. WML ist offen und lizenzfrei. Weitere Informationen dazu finden Sie z.B. auf der Web Site von Phone.com unter: http://www.phone.com/.
Das Lesen und Eingeben von Informationen erfolgt in WML Seiten über die Tags *4DTEXT* oder *4DSCRIPT*.

Der 4D Web Server lässt folgende WML Dokumente zu:

Extension	MIME Typ	Beschreibung
.wml	text/vnd.wap.wml	WML Seiten (immer von 4D unterstützt *)
.wmls	text/vnd.wap.wmlscript	WML Skripte (auf der Client-Ebene)
.wmlc	application/vnd.wap.wmlc	Binäre WML Seiten
.wmlsc	application/vnd.wap.wmlscript	Binäre WML Skripte
.wbmp	picture/vnd.wap.wbmp	Bitmap Bilder für Handys (wird nicht immer unterstützt)

* Ermöglicht das dynamische Einfügen der Daten über *4DTEXT* und *4DSCRIPT*.

XML

Der 4D Web Server unterstützt Dokumente mit den Endungen .xml,.xls und .dtd. Diese Dokumente werden jeweils mit dem MIME Typ “text/xml” und “text/xsl” gesendet.
4D analysiert deren Inhalt und verarbeitet die evtl. enthaltenen Tags *4DTEXT* oder *4DSCRIPT*, um ein dynamisches XML zu generieren.

WEB CLOSE SESSION

WEB CLOSE SESSION (SessionID)

Parameter	Typ		Beschreibung
SessionID	Text	➡	UUID Session

Beschreibung

Der Befehl **WEB CLOSE SESSION** macht die Session ungültig, die im Parameter *SessionID* angegeben ist. Existiert die Session nicht, führt der Befehl nichts aus.

Wird dieser Befehl über einen Web Prozess oder einen beliebigen anderen Prozess aufgerufen, passiert folgendes:

- Das Ablaufdatum des Cookie wird auf 0 gesetzt
- Die **Datenbankmethode On Web Close Process** wird aufgerufen, in der Sie die Informationen zur Session speichern können
- Auswahlen werden aufgehoben, Datensätze werden freigegeben und Variablen werden erneut gesetzt.

Sendet ein Web Client nach Ausführen dieses Befehls eine Anfrage mit einem ungültigen Cookie, wird eine neue Sitzung geöffnet und ein neues Cookie gesendet.

Hinweis: Im Kontext einer 4D Mobile Session schließt der Befehl **WEB CLOSE SESSION** die 4D Mobile Session mit der ID, die im Parameter *SessionID* übergeben ist. Da eine 4D Mobile Session mehrere Prozesse verwalten kann, fordert dieser Befehl alle Web Prozesse dieser Sitzung auf, ihre Ausführung zu beenden.

WEB GET BODY PART

WEB GET BODY PART (Teil ; Inhalt ; Name ; mimeTyp ; Dateiname)

Parameter	Typ		Beschreibung
Teil	Lange Ganzzahl	➡	Nummer des Teils
Inhalt	BLOB, Text	⬅	Inhalt des Teils
Name	Text	⬅	Name der Originaldatei
mimeTyp	Text	⬅	Mime Typ der übertragenen Datei
Dateiname	Text	⬅	Name der übertragenen Datei

Beschreibung

Der Befehl **WEB GET BODY PART** analysiert bei Aufrufen im Rahmen eines Web Prozesses den "body" Teil in einer Anfrage, die aus mehreren Teilen besteht.

Im Parameter *Teil* übergeben Sie die Nummer des zu analysierenden Teils. Über die Funktion **WEB Get body part count** erhalten Sie die Gesamtanzahl der Teile.

Der Parameter *Inhalt* erhält den Inhalt des Teils. Sind die zu findenden Teile Dateien, müssen Sie einen Parameter vom Typ BLOB übergeben. Werden in einem Webformular TEXT Variablen übertragen, können Sie einen Parameter vom Typ Text übergeben.

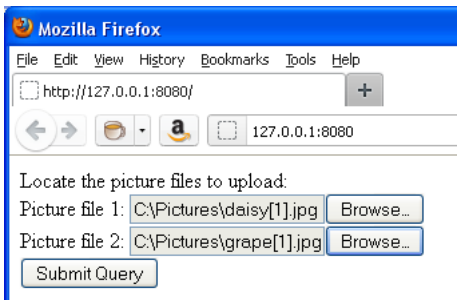
Der Parameter *Name* erhält den Variablennamen des HTTP Eingabefeldes.

Die Parameter *mimeTyp* und *Name* erhalten den Mime Typ und Namen der ursprünglichen Datei - sofern eine vorhanden ist. Ein *Name* wird nur empfangen, wenn die Datei als **<input type="file">** übertragen wird.
mimeTyp und *Name* sind optional, sie müssen jedoch zusammen übergeben werden.

Hinweis: Bei mehrteiligen Anfragen gibt das erste Array des Befehls **WEB GET VARIABLES** alle Teile des Formulars zurück, und zwar in derselben Reihenfolge wie der Befehl **WEB GET BODY PART**. Sie verwenden ihn, um die Position der Teile im Formular direkt zu erhalten.

Beispiel

In diesem Beispiel lädt ein Webformular verschiedene Bilder über einen Browser auf den HTTP Server und zeigt sie in der Seite an. Das Webformular sieht aus wie folgt:



Der Code für den <body> Teil der Seite lautet:

```
<body> <form enctype="multipart/form-data" action="/4DACTION/GetFile/" method="post"> Locate the picture files to upload: <br> Picture file 1: <input name="file1" type="file"><br> Picture file 2: <input name="file2" type="file"> <br> <input type="submit"> </form> <hr/> <!--4DSCRIPT/galleryInit--> <!--4Dloop aFileNames--> <img src="/photos/<!--4Dvar aFileNames{aFileNames}-->"/> <!--4Dendloop--> </body>
```

Diese Seite ruft zwei 4D Methoden auf:

- ***galleryInit*** beim Laden (Tag 4DSCRIPT) zeigt die im Zielordner gefundenen Bilder
- ***GetFile*** beim Senden der Daten (4DACTION URL des mehrteiligen Formulars) führt die Übertragung durch

Der Code für die Methode ***galleryInit*** lautet:

```
C_TEXT($vDestinationFolder)
ARRAY TEXT(aFileNames;0)
C_LONGINT($i)
$vDestinationFolder:=Get 4D folder(HTML Root folder)+"photos"+Folder separator //"WebFolder/photos" folder
DOCUMENT LIST($vDestinationFolder;aFileNames)
```

Der Code für die Methode ***GetFile*** lautet:

```
C_TEXT($vPartName;$vPartMimeType;$vPartFileName;$vDestinationFolder)
C_BLOB($vPartContentBlob)
C_LONGINT($i)
$vDestinationFolder:=Get 4D folder(HTML Root folder)+"photos"+Folder separator
```

```
For($i;1;WEB Get body part count)  //für jeden Teil
    WEB GET BODY PART($i;$vPartContentBlob;$vPartName;$vPartMimeType;$vPartFileName)
    If($vPartFileName#"")
        BLOB TO DOCUMENT($vDestinationFolder+$vPartFileName;$vPartContentBlob)
    End if
End for
WEB SEND HTTP REDIRECT("/")  // zurück zur Seite
```

WEB Get body part count

WEB Get body part count -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Lange Ganzzahl		Anzahl der Teile im Hauptteil (body)

Beschreibung

Die Funktion **WEB Get body part count** gibt die Anzahl der Teile im empfangenen Hauptteil (body) zurück.

Beispiel

Siehe Beispiel zum Befehl **WEB GET BODY PART**.

WEB Get Current Session ID

WEB Get Current Session ID -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Text		Session UUID

Beschreibung

Die Funktion **WEB Get Current Session ID** gibt die ID der Session zurück, die für die Web Anfrage geöffnet ist. 4D generiert diese ID automatisch als ein UUID.

Wird diese Funktion außerhalb einer Web Session aufgerufen, gibt sie einen leeren String "" zurück.

WEB GET HTTP BODY

WEB GET HTTP BODY (Body)

Parameter	Typ		Beschreibung
Body	BLOB, Text	⇐	Body der HTTP Anfrage

Beschreibung

Der Befehl **WEB GET HTTP BODY** gibt den Hauptteil der HTTP-Anfrage in Bearbeitung zurück. Der HTTP Body wird ohne Bearbeitung oder Durchlaufen (parsing) zurückgegeben.

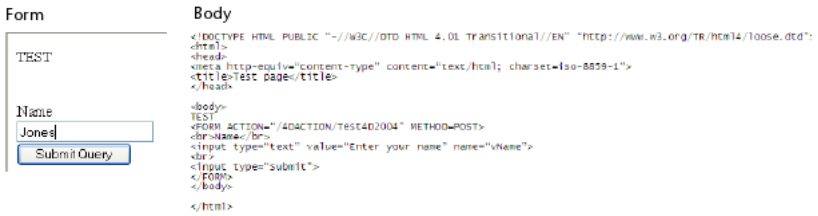
Dieser Befehl lässt sich über eine Web Datenbankmethode (**Datenbankmethode On Web Authentication**, **Datenbankmethode On Web Connection**) oder eine beliebige Web-Methode aufrufen.

In *Body* übergeben Sie ein Feld oder eine Variable vom Typ BLOB oder Text. Der Typ Text reicht normalerweise aus. Der Parameter *Body* kann bis zu 2 GB Text empfangen.

Mit diesem Befehl können Sie z.B. Suchläufe im Hauptteil von Anfragen ausführen. Versierte Benutzer können auch einen WebDAV Server innerhalb einer 4D Datenbank einrichten.

Beispiel

Im folgenden Beispiel wird an den 4D Web Server eine einfache Anfrage gesendet und der Inhalt des HTTP-Body im Debugger angezeigt. Hier sehen Sie das an den 4D Web Server gesendete Formular und den dazugehörigen HTML-Code:



Die Methode Test4D2004 lautet:

```
C_BLOB($request)
C_TEXT($requestText)

WEB GET HTTP BODY($request)
$requestText:=BLOB to text($request;UTF8 text without length)
WEB SEND FILE("page.html")
```

Hinweis: Diese Methode hat die Eigenschaft "Zugang per 4D HTML Tags und URLs (4DACTION...).

Wird das Formular an den Web Server übertragen, empfängt die Variable $requestText den Text aus dem Body-Teil der HTTP Anfrage.

WEB GET HTTP HEADER

WEB GET HTTP HEADER (Header/FeldArray {; WertArray})

Parameter	Typ		Beschreibung
Header/FeldArray	Text, Array Text	⇐	Anfrage HTTP Kopfteil oder Kopfteilfelder
WertArray	Array Text	⇐	HTTP Kopfteil Feldinhalt

Beschreibung

Der Befehl **WEB GET HTTP HEADER** gibt entweder einen String oder zwei Arrays mit dem HTTP Kopfteil für die aktuell ablaufende Anfrage zurück.
Dieser Befehl lässt sich von einer beliebigen Methode aus aufrufen (**Datenbankmethode On Web Authentication** oder **Datenbankmethode On Web Connection** aufgerufen über *'/4DACTION'*...), die in einem Web Prozess ausgeführt wird.

- **Erste Syntax: WEB GET HTTP HEADER (Kopfteil)**

Mit dieser Syntax gibt die Variable *Kopfteil* folgendes Ergebnis zurück:

```
"GET /page.html HTTP\1.0"+Char(13)+Char(10)+"User-Agent: browser"+Char(13)+Char(10)+"Cookie: C=HELLO"
```

Die Felder sind unter Windows und Mac OS durch CR+LF (Zeilenschaltung+Zeilenvorschub) voneinander getrennt.

- **Zweite Syntax: WEB GET HTTP HEADER (FeldArray; WertArray)**

Mit dieser Syntax werden in *FeldArray* und *WertArray* folgende Ergebnisse zurückgegeben:

FeldArray{1} = "X-METHOD"	WertArray{1} = "GET" *
FeldArray{2} = "X-URL"	WertArray{2} = "/page.html" *
FeldArray{3} = "X-VERSION"	WertArray{3} = "HTTP/1.0" *
FeldArray{4} = "User-Agent"	WertArray{4} = "browser"
FeldArray{5} = "Cookie"	WertArray{5} = "C=HELLO"

* Die drei ersten Elemente sind keine HTTP Felder. Sie gehören zur ersten Zeile der Anfrage.
In Anlehnung an den HTTP Standard werden Feldnamen immer in Englisch geschrieben.
Nachfolgend sehen Sie eine Liste mit HTTP Feldern für eine Anfrage:

- **Accept**: Inhalt, den der Browser zulässt.
- **Accept-Language**: Sprache(n), die der Browser verwenden kann (für Information). Damit lässt sich eine Web Site in der im Browser festgelegten Sprache auswählen.
- **Cookie**: Liste mit Cookies
- **From**: E-Mail Adresse des Browser Benutzers.
- **Host**: Server-Name oder -Adresse (verwenden Sie z.B. URL, http://mywebserver /mypage.html, nimmt Host den Wert «mywebserver» an). Damit können Sie mehrere Namen verwalten, die auf dieselbe IP Adresse zeigen (virtuelles Hosting).
- **Referer**: Ursprung der Anfrage (zum Beispiel http://mywebserver/mypage1.html), z.B. die Seite, welche beim Klicken auf die Schaltfläche **Previous** angezeigt wird.
- **User-Agent**: Browser oder Proxy-Name und Version.

Beispiel

Mit der folgenden Methode erhalten Sie den Inhalt eines beliebigen Feldes im Kopfteil einer HTTP Anfrage:

```
` Projektmethode GetHTTPField
` GetHTTPField (Text) -> Text
` GetHTTPField (HTTP header name) -> HTTP header content

C_TEXT($0;$1)
C_LONGINT($vlItem)
ARRAY TEXT($names;0)
ARRAY TEXT($values;0)
$0:=""
WEB GET HTTP HEADER($names;$values)
$vlItem:=Find in array($names;$1)
If($vlItem>0)
   $0:=$values{$vlItem}
End if
```

- Diese Projektmethode können Sie folgendermaßen aufrufen:

```
` Cookie header content
$cookie:=GetHTTPField("Cookie")
```

- Je nach der im Browser definierten Sprache können Sie verschiedene Seiten senden (zum Beispiel in der **Datenbankmethode On Web Connection**):

```
$language:=GetHTTPField("Accept-Language")
Case of
   :($language="@fr@")  `Französisch (siehe Liste ISO 639)
      WEB SEND FILE("index_fr.html")
   :($language="@sp@")  `Spanisch (siehe Liste ISO 639)
      WEB SEND FILE("index_es.html")
   Else
      WEB SEND FILE("index.html")
End case
```

Hinweis: In Web Browsern lassen sich standardmäßig mehrere Sprachen definieren. Sie sind im Feld "Accept-Language" durch Strichpunkt getrennt aufgeführt. Die Priorität richtet sich nach der Stelle innerhalb des String; deshalb empfiehlt es sich, die Position der Sprache im String zu testen.

- Hier ist ein Beispiel für virtuelle Hosts (zum Beispiel in der **Datenbankmethode On Web Connection**). Die Namen "home_site.com", "home_site1.com" und "home_site2.com" richten sich an dieselbe IP Adresse, zum Beispiel 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
      WEB SEND FILE("home_site.com")
End case
```

WEB GET OPTION

WEB GET OPTION (Selector ; Wert)

Parameter	Typ		Beschreibung
Selector	Lange Ganzzahl	➡	Code der Option, die geändert werden soll
Wert	Lange Ganzzahl, Text	⬅	Wert der Option

Beschreibung

Der Befehl **WEB GET OPTION** erhält den aktuellen Wert einer Option für die Operation des 4D Web Server.

Der Parameter *Selector* gibt die entsprechende Web Option an. In diesem Parameter übergeben Sie eine der Konstanten unter dem Thema **Web Server**:

Konstante	Wert	Kommentar
Web character set	17	**Reichweite:** 4D lokal, 4D Server **Beschreibung:** Damit können Sie sofort den Zeichensatz ändern, den der 4D Web Server mit 4D im lokalen Modus und 4D Server für die Kommunikation mit Browsern verwenden soll, die sich an die Datenbank anmelden. Die aktuelle Standardeinstellung richtet sich nach der Sprache des Betriebssystems. Dieser Parameter wird in den Einstellungen der Datenbank festgelegt. **Mögliche Werte:** Der Wert richtet sich nach dem jeweiligen Ausführungsmodus der Datenbank • **Unicode Modus:** Wird die Anwendung im Modus Unicode ausgeführt, müssen jetzt für diesen Parameter Zeichensatz Identifier übergeben werden. Das sind Identifier vom Typ MIBEnum Lange Ganzzahl oder Name String, definiert IANA (siehe unter http://www.iana.org/assignments/character-sets) Nachfolgend sehen Sie die Liste der Identifier, der dem von 4D Web Server unterstützten Zeichensatz entspricht: 4 = ISO-8859-1 12 = ISO-8859-9 13 = ISO-8859-10 17 = Shift-JIS 2026 = Big5 38 = euc-kr 106 = UTF-8 2024 = Windows-31J 2250 = Windows-1250 2251 = Windows-1251 2253 = Windows-1253 2255 = Windows-1255 2256 = Windows 1256 • **ASCII Kompatibilitätsmodus** 0: Western European 1: Japanisch 2: Chinesisch 3: Koreanisch 4: Benutzerdefiniert 5: Reserviert 6: Central European 7: Kyrillisch 8: Arabisch 9: Griechisch 10: Hebräisch 11: Türkisch 12: Baltisch
Web Client IP address to listen	23	**Reichweite**: Alle 4D remote Rechner **Mögliche Werte**: Siehe Selector 16 **Beschreibung**: Dient zur Angabe dieses Parameters für alle Rechner mit remote 4D, die als Web Server verwendet werden (gilt nur für den remote Web Server).
Web debug log	84	**Reichweite:** Lokaler Web Server **Hinweis:** Nach Neustart des HTTP Server wird ein neues Protokoll verwendet. **Beschreibung:** Ermöglicht, den Status des HTTP Anfrageprotokolls des 4D Web Server zu erhalten oder zu setzen. Ist er aktiviert, wird diese Datei mit Namen "**HTTPDebugLog_*nn*.txt**", im Ordner "Logs" der Anwendung gespeichert (*nn* ist die Dateinummer). Das ist hilfreich zum Debuggen bei Problemen mit dem Web Server. Die Datei protokolliert jede Anfrage und Antwort im Rohmodus. Die Anfragen werden mit Kopfteilen protokolliert; optional lassen sich auch Body-Bereiche mitprotokollieren. Weitere Informationen dazu finden Sie im **Anhang E: Beschreibung der Protokolldateien**. **Werte**: Eine der Konstanten mit der Vorsilbe "wdl" (siehe Beschreibung dieser Konstanten unter diesem Thema). **Standardwert**: 0 (nicht aktiviert)
Web HSTS enabled	86	**Reichweite**: 4D lokal, 4D Server. **Beschreibung**: HTTP Strict Transport Security (HSTS) Status. Über HSTS kann der 4D Web Server festlegen, dass Browser nur über sichere HTTPS Verbindungen mit ihm kommunizieren. Ist HSTS aktiviert, fügt der 4D Web Server automatisch in allen Antwort-Kopfteilen HSTS-bezogene Information hinzu. Browser speichern diese Information, wenn sie zum ersten Mal eine Antwort vom 4D Web Server empfangen. Alle nachfolgenden HTTP Anfragen werden dann automatisch in HTTPS Anfragen umgewandelt. Mit dem Selektor Web HSTS max age lässt sich angeben, wie lange diese Information vom Browser gespeichert wird. Für HSTS muss HTTPS auf dem Server aktiviert werden. HTTP muss ebenfalls aktiviert werden, um das erste Starten von Client Verbindungen zuzulassen. **Mögliche Werte**: 0 (deaktiviert, Standard) oder 1 (aktiviert) **Hinweis:** Die Einstellung wird erst nach Neustart des 4D Web Server übernommen.
Web HSTS max age	87	**Reichweite**: 4D lokal, 4D Server **Beschreibung**: Gibt die maximale Zeitspanne an (in Sekunden), die HSTS für jede neue Client-Verbindung aktiv bleibt. Diese Angabe wird auf Client-Seite für die angegebene Dauer gespeichert. **Mögliche Werte**: Lange Ganzzahl (Sekunden) **Standardwert**: 63072000 (2 Jahre) ***Warnung:*** *Ist HSTS aktiviert, verwenden Client Verbindungen diesen Mechanismus für die angegebene Dauer. Beim Testen Ihrer Anwendung empfehlen wir, einer kurze Dauer zu setzen, damit Sie bei Bedarf zwischen sicheren und nicht-sicheren Verbindungen wechseln können.*

Konstante	Wert	Kommentar
Web HTTP compression level	50	**Reichweite:** Lokaler Web Server **Mögliche Werte:** 1 bis 9 (1 = schneller, 9 = stärker komprimiert), -1 = beste Kombination **Beschreibung:** Setzt die Komprimierungsebene für jeden komprimierten HTTP Austausch für den 4D HTTP Server (Client Anfragen oder Server Antworten, Web und Web Service). Mit diesem Selektor können Sie den Austausch entweder über die Ausführungsgeschwindigkeit (weniger Komprimierung) oder die Komprimierungsmenge (weniger Geschwindigkeit) optimieren. Die Auswahl des passenden Wertes richtet sich nach der Größe und der Art der ausgetauschten Daten. Im Parameter *Wert* können Sie einen Wert von 1 bis 9 übergeben, wobei 1 die schnellste und 9 die höchste Komprimierung ist. Für einen Kompromiss zwischen Geschwindigkeit und Komprimierungsrate übergeben Sie -1. Standardmäßig ist als Komprimierungsebene 1 eingestellt (schnellere Komprimierung).
Web HTTP compression threshold	51	**Reichweite:** Lokaler HTTP Server **Mögliche Werte:** Jeder Wert vom Typ Lange Ganzzahl **Beschreibung:** Setzt den Schwellwert, bis zu dem der Austausch von Daten im Rahmen von 4D Web Services nicht komprimiert werden soll. Diese Einstellung ist hilfreich, um beim Austausch geringer Datenmengen zu verhindern, dass Rechenzeit für die Komprimierung beansprucht wird. In *Wert* übergeben Sie eine Größe in Bytes. Standardmäßig ist als Schwellwert für Komprimierung 1024 Bytes eingestellt.
Web HTTP enabled	88	**Reichweite**: 4D lokal, 4D Server **Beschreibung**: Status für Kommunikation über HTTP **Mögliche Werte**: 0 (deaktiviert) oder 1 (aktiviert)
Web HTTP TRACE	85	**Reichweite**: Lokaler Web Server **Wird zwischen 2 Sitzungen beibehalten**: Nein **Beschreibung**: Ermöglicht, die Methode HTTP TRACE im 4D Web Server zu aktivieren/deaktivieren. Ab 4D v15 R2 weist der 4D Web Server aus Sicherheitsgründen HTTP TRACE Anfragen standardmäßig mit dem Fehler 405 ab (HTTP TRACE ist deaktiviert). Bei Bedarf können Sie die Methode HTTP TRACE für die Sitzung aktivieren, indem Sie für diese Konstante den Wert 1 übergeben. Dann sendet der 4D Web Server bei HTTP TRACE Anfragen die Anfragezeile, Kopfteil und Hauptteil. **Mögliche Werte**: 0 (deaktiviert) oder 1 (aktiviert) **Standardwert**: 0 (deaktiviert)
Web HTTPS enabled	89	**Reichweite**: 4D lokal, 4D Server **Beschreibung**: Status für Kommunikation über HTTPS. **Mögliche Werte**: 0 (deaktiviert) oder 1 (aktiviert)
Web HTTPS port ID	39	**Reichweite:** 4D lokal, 4D Server **Mögliche Werte:** 0 bis 65535 **Beschreibung:** Mit diesem Selector können Sie per Programmierung die TCP Portnummer ändern, die der Web Server von 4D im lokalen Modus und von 4D Server für sichere Verbindungen via TLS (HTTPS protocol) verwendet. Die HTTPS Portnummer wird in den Datenbank-Eigenschaften auf der Seite **Web>Konfiguration** gesetzt. Der Wert ist standardmäßig 443. Für den Parameter *Wert* können Sie Konstanten unter dem Thema **TCP Port Nummern** einsetzen.
Web inactive process timeout	78	**Reichweite:** Lokaler Web Server **Mögliche Werte:** Lange Ganzzahl (Minuten) **Beschreibung:** Ändert die Lebensdauer des inaktiven Prozesses, der Sessions zugeordnet ist. Am Ende des Timeout wird der Prozess auf dem Server gestoppt, die **Datenbankmethode On Web Close Process** wird aufgerufen, dann wird der Session-Kontext gelöscht. **Standardwert:** 480 Minuten (zum Wiederherstellen des Standardwerts 0 übergeben)
Web inactive session timeout	72	**Reichweite:** Lokaler Web Server **Beschreibung:** Ändert die Lebensdauer inaktiver Sessions (Dauer wird in Cookie gesetzt). Endet diese Periode, läuft das Cookie der Session ab und wird nicht mehr vom HTTP Client gesendet. **Mögliche Werte:** Lange Ganzzahl (Minuten) **Standardwert:** 480 Minuten (zum Wiederherstellen des Standardwerts 0 übergeben)
Web IP address to listen	16	**Reichweite:** 4D lokal, 4D Server **Beschreibung:** IP Adresse, unter der der 4D Web Server HTTP Anfragen mit 4D im lokalen Modus und 4D Server empfängt. Standardmäßig ist keine bestimmte Adresse definiert (Wert = 0). Dieser Parameter lässt sich in den Datenbank-Eigenschaften auf der Seite **Web>Konfiguration** festlegen. Dieser Selector ist hilfreich für 4D Web Server mit einkompilierter 4D Volume Desktop, die keinen Zugriff auf den Designmodus zulassen. **Mögliche Werte**: IP Adresse String. Beide Formate werden unterstützt: IPv6, z.B. "2001:0db8:0000:0000:0000:ff00:0042:8329") und IPv4 (z.B. "123.45.67.89"). **Hinweis:** Zur Wahrung der Kompatibilität werden überholte IPv4 Adressen in Form von hexadezimalen Langen Ganzzahlen noch unterstützt.
Web keep session	70	**Reichweite:** Lokaler Web Server **Beschreibung:** Aktiviert oder deaktiviert den neuen Modus zum Verwalten der Sessions. Weitere Informationen dazu finden Sie im Abschnitt **Web Sessions verwalten** **Mögliche Werte:** 1 (Modus aktivieren) oder 0 (Modus deaktivieren) **Standardwert:** 1 für mit v13 erstellte Datenbanken, 0 für konvertierte Datenbanken. Beachten Sie, dass dieser Modus auch den Mechanismus zum Wiederverwenden temporärer Kontexte im remote Modus ermöglicht. Weitere Informationen dazu finden Sie im Abschnitt **Web Server, Einstellungen**

Konstante	Wert	Kommentar
Web log recording	29	**Reichweite:** 4D lokal, 4D Server **Beschreibung:** Startet oder stoppt das Speichern von Web Anfragen, die vom Web Server von 4D im lokalen Modus oder 4D Server empfangen werden. Der Standardwert ist 0, d.h. Anfragen werden nicht gespeichert. Das Logbuch von Web Anfragen wird als Textdatei mit Namen "logweb.txt" gespeichert, die automatisch in den Ordner Logs neben der Strukturdatei der Anwendung gesetzt wird. Das Format dieser Datei richtet sich nach dem übergebenen Wert. Weitere Informationen zu Formaten für Web Logfiles finden Sie im Abschnitt **Information über die Web Site** . Diese Datei lässt sich auch in den Datenbank-Eigenschaften auf der Seite **Log (Typ)** aktivieren. **Mögliche Werte:** 0 = Nicht speichern (Standard), 1 = In CLF Format speichern, 2 = In DLF Format speichern, 3 = In ELF Format speichern, 4 = In WLF Format speichern. **Warnung:** Format 3 und 4 sind individuell anpassbare Formate, d.h. Sie müssen den Typ zuvor in den Datenbank-Eigenschaften auf der Seite **Log (Typ)** definieren. Verwenden Sie diese Formate, ohne zuvor den Typ festzulegen, wird kein Logbuch angelegt.
Web max concurrent processes	18	**Reichweite:** 4D lokal, 4D Server **Werte:** Sie können jeden Wert zwischen 10 und 32 000 übergeben. Der Standardwert ist 100. **Beschreibung:** Damit setzen Sie die maximale Anzahl aller gleichzeitig laufenden Web Prozesse (kontextuell, nicht kontextuell oder die zum Pool der Prozesse gehören), die der 4D Web Server mit 4D im lokalen Modus und 4D Server unterstützt. Ist die maximale Anzahl erreicht, erstellt 4D keinen weiteren Prozess und gibt den HTTP Status 503 zurück - Dienst für weitere neue Anfragen nicht verfügbar. Dieser Parameter verhindert die Übersättigung des 4D Web Server. Sie kann eintreten, wenn gleichzeitig eine zu große Anzahl an Anfragen gesendet wird oder zu viele Kontext-Erstellungen angefordert werden. Theoretisch ist die max. Anzahl der Web Prozesse das Ergebnis der folgenden Formel: Verfügbarer Speicher/Stapelgröße der Web Prozesse. Sie können sich auch die Information über die Web Prozesse im Runtime Explorer ansehen: Er zeigt die aktuelle Anzahl der Web Prozesse und die max. erreichte Anzahl seit dem Hochfahren des Web Server an.
Web max sessions	71	**Reichweite:** Lokaler Web Server **Beschreibung:** Begrenzt die Anzahl gleichzeitiger Sessions. Bei Erreichen des Limits wird die älteste Session geschlossen (die **Datenbankmethode On Web Close Process** wird aufgerufen), wenn der Web Server eine neue erstellen muss. **Mögliche Werte:** Lange Ganzzahl. Die Anzahl gleichzeitiger Sessions kann nicht größer sein als die Gesamtzahl der Web Prozesse (Option Web Max Concurrent Processes, standardmäßig 100). **Standardwert:** 100 (zum Wiederherstellen des Standardwerts 0 übergeben)
Web maximum requests size	27	**Reichweite:** 4D lokal, 4D Server **Mögliche Werte:** 500 000 bis 2 147 483 648 **Beschreibung:** Maximale Größe (in Bytes) hereinkommender HTTP Anfragen (POST), die der Web Server akzeptiert. Standardmäßig ist der Wert 2 000 000 vorgegeben, z.B. etwas unter 2 MB. Übergeben Sie den maximalen Wert (2 147 483 648), wird praktisch keine Grenze gesetzt. Die Begrenzung sorgt dafür, dass der Web Server nicht überlastet wird durch zu große eingehende Anfragen. Erreicht eine Anfrage den Grenzwert, weist der 4D Web Server diese zurück.
Web port ID	15	**Reichweite:** 4D im lokalen Modus und 4D Server **Beschreibung:** Setzt oder erhält die Nummer des TCP Port, den 4D Web Server mit 4D im lokalen Modus und mit 4D Server (Lange Ganzzahl) verwendet. Standardmäßig ist der Wert 80. Die TCP Port Nummer wird in den Einstellungen der Datenbank auf der Seite "Web/Konfiguration" gesetzt. Für den Parameter *Wert* können Sie eine Konstante unter dem Thema **TCP Port Nummern** verwenden. Dieser Selector ist im Rahmen von 4D Web Servern mit einkompiliertem 4D Desktop hilfreich (kein Zugriff auf die Design-Umgebung). **Mögliche Werte:** Weitere Informationen über die TCP Port Nummer finden Sie im Abschnitt **Web Server, Einstellungen**. **Standardwert:** 80
Web session cookie domain	81	**Reichweite:** lokaler Web Server **Mögliche Werte:** Text **Beschreibung:** Setzt oder erhält den Wert des Feldes "domain" des Session Cookie (Text). Dieser Selektor, sowie Selektor 82 sind hilfreich zum Überprüfen der Reichweite von Session Cookies: Haben Sie z.B. für diesen Selektor den Wert "/*.4d.fr" gesetzt, sendet der Client ein Cookie nur, wenn die Anfrage an die Domaine ".4d.fr" gerichtet ist. Das schließt Server aus, die externe statische Daten hosten.
Web session cookie name	73	**Reichweite:** Lokaler Web Server **Beschreibung:** Setzt den Namen des Cookie zum Sichern der Session ID **Mögliche Werte:** Text **Standardwert:** "4DSID" (zum Wiederherstellen des Standardwerts leeren String übergeben)
Web session cookie path	82	**Reichweite:** lokaler Web Server **Mögliche Werte:** Text **Beschreibung:** Setzt oder erhält den Wert des Feldes "path" des Session Cookie (Text). Dieser Selektor, sowie Selektor 81 sind hilfreich zum Überprüfen der Reichweite von Session Cookies: Haben Sie z.B. für diesen Selector den Wert "/4DACTION" gesetzt, sendet der Client ein Cookie nur für dynamische Anfragen, die mit 4DACTION beginnen und nicht für Bilder, statische Seiten, etc.

Konstante	Wert	Kommentar
Web session enable IP address validation	83	**Reichweite**: Lokaler Web Server **Beschreibung**: Damit können Sie die Überprüfung der IP Adresse für Session Cookies aktivieren oder deaktivieren. Aus Sicherheitsgründen prüft der 4D Web Server standardmäßig die IP Adresse einer Anfrage mit einem Session Cookie und weist sie ab, wenn sie nicht zur IP Adresse passt, über die das Cookie erstellt wurde. Sie können diese Überprüfung bei bestimmten Applikationen deaktivieren und das Session Cookie akzeptieren, auch wenn die IP Adresse nicht dazu passt. Wechseln z.B. mobile Geräte zwischen Wifi und 3G/4G Netzwerken, ändert sich die IP Adresse. In diesem Fall übergeben Sie den Wert 0, damit Clients weiterhin ihre Web Sessions verwenden können, auch wenn sie die IP Adresse wechseln. Beachten Sie, dass diese Einstellung die Sicherheitstufe Ihrer Applikation herabsetzt. Die Einstellung ist sofort wirksam, d.h. Sie müssen den HTTP Server nicht neu starten. **Mögliche Werte**: 0 (deaktiviert) oder 1 (aktiviert) **Standardwert**: 1 (IP Adressen werden geprüft)

Mit dem *Selector* Web debug log erhalten Sie im Parameter *Wert* eine der folgenden Konstanten:

Konstante	Typ	Wert	Kommentar
wdl disable	Lange Ganzzahl	0	Web HTTP Fehlerprotokoll ist deaktiviert
wdl enable with all body parts	Lange Ganzzahl	7	Web HTTP Fehlerprotokoll ist aktiviert mit Body Bereichen der Anfrage und der Antwort
wdl enable with request body	Lange Ganzzahl	5	Web HTTP Fehlerprotokoll ist aktiviert nur mit Body Bereichen der Anfrage
wdl enable with response body	Lange Ganzzahl	3	Web HTTP Fehlerprotokoll ist aktiviert nur mit Body Bereichen der Antwort
wdl enable without body	Lange Ganzzahl	1	Web HTTP Fehlerprotokoll ist aktiviert ohne Body Bereiche (in diesem Fall wird die Body Größe angezeigt)

WEB Get server info

WEB Get server info {(CacheInfo)} -> Funktionsergebnis

Parameter	Typ		Beschreibung
CacheInfo	Boolean	→	Wahr, um Angaben zum Web Cache zurückzugeben. Standardmäßig werden keine Angaben zum Cache zurückgegeben.
Funktionsergebnis	Objekt	←	Angaben zum laufenden Web Server und SOAP Server

Beschreibung

Die Funktion **WEB Get server info** gibt ein Objekt mit detaillierten Angaben zur aktuellen 4D Web Server Sitzung im laufenden Betrieb zurück. Dazu gehört auch der SOAP Server.

Hinweis: Die Funktion gibt Information im laufenden Betrieb zurück. z.B. aktuelle Parameter, die der Web Server nutzt. Diese können sich von den Werten unterscheiden, die der Befehl **WEB GET OPTION** zurückgibt, da sie von Systemeinstellungen, verfügbaren Ressourcen, etc. abhängen.

Standardmäßig gibt die Funktion nicht die Eigenschaft "Cache" zurück, da dies sehr umfangreich sein kann. Um den Inhalt des Cache zu erhalten, übergeben Sie *Wahr* im optionalen Parameter *CacheInfo*.

Das zurückgegebene Objekt enthält folgende Eigenschaften. Beachten Sie, dass die Bezeichnungen Groß- und Kleinschreibung berücksichtigen:

Bezeichnung			Wertetyp	Beschreibung
started			Boolean	Wahr, wenn der http Server gestartet ist, sonst falsch
uptime			Zahl	Abgelaufene Zeit seit dem letzten Start des http Server
httpRequestCount			Zahl	Anzahl der empfangenen http hits seit Serverstart
startMode			String	"Automatisch" wenn in den **Datenbank-Eigenschaften** auf der Seite **Web** die Option "Web Server automatisch starten" markiert ist, sonst "manuell"
SOAPServerStarted			Boolean	Wahr, wenn der SOAP Server gestartet ist, sonst falsch
cache			Objekt	*Diese Eigenschaft ist nur enthalten, wenn der Wert des Parameters CacheInfo auf wahr gesetzt ist*. Beschreibt den Inhalt des Web Server Cache (siehe unten **Eigenschaft cache**)
security			Objekt	Aktueller Status der verschiedenen Sicherheitsoptionen
	properties		Objekt	Aktueller Status der verschiedenen Sicherheitseigenschaften
		cipherSuite	String	Von 4D verwendete Cipher Liste für das SSL Protokoll (entspricht dem Datenbankparameter [#cst id="844208"/])
		HTTPEnabled	Boolean	Wahr, wenn HTTP aktiviert ist
		HTTPSEnabled	Boolean	Wahr, wenn HTTPS aktiviert ist
		HSTSEnabled	Boolean	Wahr, wenn HSTSE auf dem Server aktiviert ist
		HSTSMaxAge	Zahl	Maximales Alter (in Sekunden) für HSTS. Standard sind 2 Jahre (63.072.000 Sekunden).
		minTLSVersion	String	Erforderliche TLS Mindestversion für Verbindungen (entspricht dem Datenbankparameter Min TLS version)
		openSSLVersion	String	Version der verwendeten OpenSSL Library
		perfectForwardSecrecy	Boolean	Wahr, wenn PFS auf dem Server verfügbar ist, sonst Falsch
options			Objekt	Aktueller Status der verschiedenen Optionen des Standard Web Server
	properties		Objekt	Aktueller Status der verschiedenen Eigenschaften der Optionen des Standard Web Server
		webCharacterSet	String	Name des Zeichensatzes (entspricht der Web Option Web character set)
		webHTTPCompressionLevel	Zahl	Komprimierungsebene für komprimierten HTTP Austausch (entspricht der Web Option Web HTTP compression level)
		webHTTPCompressionThreshold	Zahl	Komprimierungsrate (entspricht der Web Option Web HTTP compression threshold)
		webHTTPSPortID	Zahl	Vom Web Server benutzte TCP Port Nummer für sichere Verbindungen (entspricht Web HTTPS port ID)
		webInactiveProcessTimeout	Zahl	Lebensdauer der inaktiven Prozesse der Sitzung (entspricht der Web Option Web inactive process timeout)
		webInactiveSessionTimeout	Zahl	Lebensdauer der inaktiven Sitzungen (entspricht der Web Option Web inactive session timeout)
		webIPAddressToListen	Collection	IP Adresse(n) im definierten "Format", auf denen der Web Server http Anfragen erhält (entspricht der Web Option Web IP address to listen)
		webMaxConcurrentProcesses	Zahl	Maximale Anzahl gleichzeitig laufende Web Prozesse (entspricht der Web Option Web max concurrent processes)
		webPortID	Zahl	Vom Web Server benutzter TCP Port (entspricht der Web Option Web port ID)

4D Server: Die Funktion gibt Information über den lokalen Web Server zurück. Um den Web Server des 4D Server über remote 4D zu überwachen, müssen Sie der Methode die Eigenschaft "Auf Server ausführen" übergeben.

Eigenschaft cache

Übergeben Sie **wahr** im Parameter *CacheInfo*, gibt die Funktion die Objekteigenschaft "cache" mit folgendem Inhalt zurück:

Bezeichnung	Wertetyp	Beschreibung
cacheUsage	Zahl	Rate der Cache Verwendung
numOfLoads	Zahl	Anzahl der geladenen Objekte
currentSize	Zahl	Aktuelle Größe des Cache
maxSize	Zahl	Maximale Größe des Cache
objectMaxSize	Zahl	Maximale Größe der im Cache ladbaren Objekte
enabled	Boolean	Wahr, wenn der Cache des Web Server aktiviert ist
nbCachedObjects	Zahl	Anzahl der Objekte im Cache
cachedObjects	Collection	Objekt Collection im Cache. Jedes Cache Objekt ist durch verschiedene Eigenschaften definiert (url, mimeType, expirationType, lastModified, etc.)

Beispiel

Nach Ausführen des Code:

```
$webServerInfo:=WEB Get server info(True)
```

... kann $webServerInfo z.B. Beispiel folgende Angaben enthalten:

{ "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 (SessionID ; Ablaufdatum ; Ablaufzeit)

Parameter	Typ		Beschreibung
SessionID	Text	➡	UUID Session
Ablaufdatum	Datum	⬅	Ablaufdatum des Cookie
Ablaufzeit	Zeit	⬅	Ablaufzeit des Cookie

Beschreibung

Der Befehl **WEB GET SESSION EXPIRATION** gibt die Information zum Ablaufen der Cookie Session zurück, deren UUID in *SessionID* übergeben ist.

Der Parameter *Ablaufdatum* erhält das Ablaufdatum, der Parameter *Ablaufzeit* die Ablaufzeit des Cookie.

Hinweis: Bei jedem Senden einer Web Anfrage werden Ablaufdatum und -zeit neu gesetzt auf den Wert der Anfragezeit + Wert von Web Inactive session timeout. Zum Beispiel:
Erste Anfrage, Montag um 1:00
-> Sendet Cookie 4DSID xxxyyy mit Timeout I + 24h: Dienstag 01:00
Zweite Anfrage, Montag um 1:10
-> Sendet Cookie 4DSID xxxyyy mit Timeout I + 24h: Dienstag 01:10
Dritte Anfrage, Dienstag um 4:00 // Cookie ist abgelaufen
-> Sendet Cookie 4DSID aaabbb mit Timeout I + 24h: Mittwoch 01:00

WEB Get session process count

WEB Get session process count (sessionID) -> Funktionsergebnis

Parameter	Typ		Beschreibung
sessionID	Text	➡	Session UUID
Funktionsergebnis	Lange Ganzzahl	⮌	Number of processes attached to the session

Beschreibung

Die Funktion **WEB Get session process count** gibt die Anzahl der laufenden Prozesse innerhalb einer Session mit der in *SessionID* übergebenen UUID zurück.

Diese Funktion wurde im Rahmen der Funktionalität **4D Mobile Sessions verwalten** hinzugefügt, die mit 4D v15 R4 eingeführt wird. Sie dient hauptsächlich zur Angabe der Anzahl der Prozesse, die in einer 4D Mobile Session laufen.

- Für eine 4D Mobile Session gibt sie die aktuelle Anzahl der Prozesse zurück. In einer 4D Mobile Session können mehrere Prozesse laufen.
- Für eine reguläre Web Session gibt sie immer 1 zurück (eine Web Session = ein Prozess).

Beispiel

Die Information der aktuellen 4D Mobile Session in Arrays speichern:

```
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 (Seiten ; Hits ; Prozent)

Parameter	Typ		Beschreibung
Seiten	Array Text	⬅	Namen der häufig konsultierten Seite
Hits	Array Lange Ganzzahl	⬅	Anzahl Hits pro Seite
Prozent	Lange Ganzzahl	⬅	Prozent des benutzten Cache

Beschreibung

Der Befehl **WEB GET STATISTICS** liefert Informationen über die Seiten, die am häufigsten in den Cache des Web Servers geladen werden. Folglich betrifft diese Statistik nur statische Seiten, GIF Bilder, JPEG Bilder <100 KB und Stilvorlagen (.css).

Hinweis: Weitere Informationen dazu finden Sie im Abschnitt **QR DELETE COLUMN**.

Der Befehl füllt das Array *Seiten* mit den Namen der häufig konsultierten Seiten. Das Array *Hits* erhält die Anzahl "Hits" pro Seite. Der Parameter *Prozent* erhält die Prozentzahl des pro Seite benutzten Web Cache.

Beispiel

Sie möchten eine halb-dynamische Seite erzeugen, die die Statistik des Web Cache anzeigt. Dazu setzen Sie in einer statischen HTML Seite mit Namen "stats.shtm" (Seiten mit der Endung .shtm werden automatisch vom Web Server durchlaufen) das Tag <!--4DACTION/STATS-->, sowie als Referenzen auf die beiden 4D Variablen *vPages* und *vUsage*.

Beispiel:

```
<html> <head><title>4D Web Stats</title></head> <!--#4DSCRIPT/STATS--> <body> <strong>Prozentsatz des verwendeten Cache:
</strong> <!--#4DTEXT vUsage--> <hr> <strong>am häufigsten konsultierte Seiten: </strong> <!--#4DHTML vPages--> </body>
</html>
```

In der Projektmethode ***STATS*** schreiben Sie folgenden Code:

```
C_TEXT($1)
C_TEXT(vPages)
ARRAY TEXT(pages;0)
ARRAY LONGINT(hits;0)
C_LONGINT(vUsage)
```

```
WEB CACHE STATISTICS(pages;hits;vUsage)
For($i;1;Size of array(pages))
  ` Für jede im Cache vorhandene Seite
   vPages:=pages{$i}+"    "+String(hits{$i})+"<br>"
  ` Füge Namen der Seite und HTML Code ein
End for
```

Sie können die Seite "stats.shtm" über ein URL Link oder den Befehl **WEB SEND FILE** senden.

WEB GET VARIABLES

WEB GET VARIABLES (NameArray ; WertArray)

Parameter	Typ		Beschreibung
NameArray	Array Text	⇐	Variablennamen des Webformulars
WertArray	Array Text	⇐	Variablenwerte des Webformulars

Beschreibung

Der Befehl **WEB GET VARIABLES** füllt die Textarrays *NameArray* und *WertArray* mit den Variablennamen und -werten des übertragenen Webformulars (per Schaltfläche „Submit" gesendet).

Dieser Befehl erhält den Wert für alle Variablen, die in HTML Seiten einfügbar sind: Textbereich, Schaltfläche, Kontrollkästchen, Optionsfeld, PopUp-Menü, Auswahlliste...

Hinweis: Bei Kontrollkästchen werden Variablenname und Wert nur zurückgegeben, wenn es markiert ist.

Dieser Befehl gilt egal, welcher URL-Typ bzw. welches Formular (Methode POST oder GET) an den Web Server gesendet wird.

Dieser Befehl kann bei Bedarf in der **Datenbankmethode On Web Connection** oder jeder anderen 4D Methode, die sich aus einer Formularübertragung ergibt, aufgerufen werden.

Web Formulare und ihnen zugeordnete Aktionen

Jedes Formular enthält benannte Dateneingabebereiche (Textbereich, Schaltflächen, Kontrollkästchen).
Wird ein Formular übertragen (eine Anfrage wurde an den Web Server gesendet), enthält die Anfrage die Liste mit den Dateneingabebereichen und den dazugehörigen Werten.

Ein Formular lässt sich über zwei Methoden übertragen (beide sind mit 4D verwendbar):

- Mit POST werden normalerweise über den Web Server Daten in einer Datenbank hinzugefügt
- Mit GET werden normalerweise über den Web Server Daten aus einer Datenbank angefragt

Beispiel

Ein Formular enthält die beiden Felder *vName* und *vOrt* mit den Werten "ROBERT" und "BERLIN". Die dem Formular zugewiesene Aktion ist "/4DACTION/WEBFORM".

- Lautet die Formularmethode POST (am häufigsten verwendet), sind die eingegebenen Daten in der URL nicht sichtbar (http://127.0.0.1/4DACTION/WEBFORM).
- Lautet die Formularmethode GET, sind die eingegebenen Daten in der URL sichtbar (http://127.0.0.1/4DACTION/WEBFORM?vNAME=ROBERT&vORT=BERLIN).

Die Methode WEBFORM lautet:

```
ARRAY TEXT($anames;0)
ARRAY TEXT($avalues;0)
WEB GET VARIABLES($anames;$avalues)
```

Das Ergebnis ist dann:

```
$anames{1}="vNAME"
$anames{2}="vORT"
$avalues{1}="ROBERT"
$avalues{2}="BERLIN"
```

Die Variable vNAME enthält ROBERT, die Variable vORT enthält BERLIN

WEB Is secured connection

WEB Is secured connection -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	True = Die Web Verbindung ist abgesichert. False = Die Web Verbindung ist nicht abgesichert.

Beschreibung

Die Funktion **WEB Is secured connection** gibt ein Boolean Wert zurück, der anzeigt, ob die 4D Web Server Verbindung über TLS/SSL abgesichert ist (die Anfrage startet mit “https:” anstelle von “http:”).

- Erfolgt die Verbindung über TLS oder SSL, gibt die Funktion **True** zurück.
- Erfolgt die Verbindung ohne Absicherung, gibt die Funktion **False** zurück.

Hinweis: Weitere Informationen zum TLS Protokoll finden Sie im Abschnitt **TLS Protokoll verwenden (HTTPS)**.
Mit dieser Funktion können Sie z.B. Verbindungen ohne Absicherung verweigern.

WEB Is server running

WEB Is server running -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	True, wenn Web Server läuft, sonst False

Beschreibung

Die Funktion **WEB Is server running** gibt **True** zurück, wenn der in 4D integrierte Web Server läuft; **False**, wenn er nicht läuft. Diese Funktion gibt den Aktivitätsstatus des Web Servers an der Stelle zurück, wo er gerade ausgeführt wird:

Kontext der Ausführung	Zeigt den Status von
4D Anwendung im Einzelplatz	Lokaler 4D Web Server
4D Server	4D Server Web Server
4D remote Modus (lokaler Prozess)	Lokaler 4D Web Server
4D remote Modus (4D Server Serverprozedur)	4D Server Web Server
4D remote Modus (andere 4D remote Serverprozedur)	Remote 4D Web Server

Beispiel

Prüfen, ob der Web Server läuft:

```
If(WEB Is server running)
   ... //verschiedene Aktionen ausführen
End if
```

WEB SEND BLOB

WEB SEND BLOB (BLOB ; Typ)

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	An den Browser zu sendendes BLOB
Typ	String	➡	Datentyp des BLOB

Beschreibung

Der Befehl **WEB SEND BLOB** sendet *BLOB* an den Browser.
In *Typ* geben Sie den im BLOB enthaltenen Datentyp an. Sie können zwischen folgenden Typen wählen:

- *Typ* = **Leerer String** (""): In diesem Fall müssen Sie keine weiteren Informationen im BLOB zuweisen. Der Browser versucht, den Inhalt des BLOB zu interpretieren.
- *Typ* = **Dateierweiterung** (Beispiel: ".HTM", ".GIF", ".JPEG", etc.): In diesem Fall geben Sie den MIME Typ der im BLOB enthaltenen Daten mit der Dateierweiterung an. Das BLOB wird dann gemäß seiner Erweiterung interpretiert. Der Browser kann jedoch nur Standarderweiterungen korrekt interpretieren.
- *Typ* = **Mime/Typ** (Beispiel: "Text/html", "Bild/tiff", etc.): In diesem Fall geben Sie direkt den MIME Typ der im BLOB enthaltenen Daten an. Sie haben so mehr Freiheiten. Sie können zusätzlich zu den Standardtypen einen eigenen MIME Typ angeben, um Dokumente mit Eigentümer via Intranet zu senden. Dafür müssen Sie die Browser so konfigurieren, dass sie den gesendeten Typ erkennen und dadurch die entsprechende Anwendung öffnen. In *Typ* übergeben Sie den Wert "application/x-[TypName]". Im Browser der Arbeitsstationen verweisen Sie auf diesen Typ und ordnen ihn der Aktion "Anwendung starten" zu. Mit dem Befehl **WEB SEND BLOB** können Sie dann alle Dokumenttypen senden. Die Browser öffnen automatisch die dazugehörige Anwendung.
  Hinweis: Weitere Informationen zu MIME Typen finden Sie unter: http://www.iana.org/assignments/media-types.

Hier ist die Liste der gängigsten MIME Typen:

Erweiterung	Mime/Typ
.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

Hinweis: Die Liste der vom 4D HTTP Server unterstützten MIME Typen wird in der Datei "MimeTypes.xml" gesichert. Sie liegt in folgendem Ordner der 4D Applikation: *[Contents]\Native components\HTTPServer.bundle\Contents\Resources*.

Der Parameter *KeinKontext* wird lediglich zur Wahrung der Kompatibilität beibehalten. Er wird ab 4D Version 13 nicht mehr verwendet.

Die Verweise auf 4D Variablen und Tags vom Typ *4DSCRIPT* in der Seite werden immer analysiert.

Beispiel

Siehe Beispiel zum Befehl **PICTURE TO BLOB**.

WEB SEND FILE

WEB SEND FILE (htmlDatei)

Parameter	Typ		Beschreibung
htmlDatei	String	➡	HTML Pfadname für HTML Datei oder leerer String zum Beenden von SEND HTML FILE

Beschreibung

Der Befehl **WEB SEND FILE** sendet an den Web Browser die im Dokument gespeicherte HTML Seite oder Web Datei mit dem Pfadnamen *htmlDatei*.

4D sucht standardmäßig nach dem HTML Dokument im Root Ordner, der in den Datenbank-Eigenschaften definiert wurde.

Dieser Befehl akzeptiert als Parameter nur Pfadnamen in der Syntax des Systems oder in Posix Syntax, d.h. Verzeichnis- oder Ordnernamen müssen mit Schrägstrich ("/") voneinander getrennt werden.
Bei einem ungültigen Pfadnamen wird ein Fehler generiert, der sich auf die Dateiverwaltung Ihres Betriebssystems bezieht. Sie können diesen Fehler mit einer Methode abfangen, die über den Befehl **ON ERR CALL** installiert wird. Zeigt die Methode eine Warnung oder eine Meldung an, erscheint sie auf dem Rechner mit dem Browser.

Hinweis: Rufen Sie **WEB SEND FILE** von einen Prozess aus auf, der kein Web-Prozess ist, hat der Befehl keine Auswirkung und meldet auch keinen Fehler; der Aufruf wird einfach ignoriert.

Die Referenzen auf 4D Variablen und Tags vom Typ *4DSCRIPT* auf der Seite werden analysiert, wenn der Dokumenttyp dies zulässt, d.h. auf Text basiert.

Beispiel

Der HTML Root Ordner der Datenbank ist der Ordner **WebDocs**. Er enthält folgende Elemente:

```
..\WebDocs\HTM\MyPage.HTM
```

Das Senden der Web Seite *"MyPage.HTM"* muss folgendermaßen ausgeführt werden:

```
WEB SEND FILE("HTM/MyPage.HTM")
```

Systemvariablen und Mengen

Gibt es die zu sendende Datei und ist das timeout nicht abgelaufen, ist OK gleich 1, sonst gleich 0 (Null).

WEB SEND HTTP REDIRECT

WEB SEND HTTP REDIRECT (Url {; *})

Parameter	Typ		Beschreibung
Url	String	➡	Neue URL
*	Operator	➡	Mit * = URL wird nicht übersetzt, ohne * = URL wird übersetzt

Beschreibung

Der Befehl **WEB SEND HTTP REDIRECT** überführt eine URL in eine andere.

Der Parameter *Url* enthält die neue URL. Damit können Sie die Anfrage umleiten. Ist dieser Parameter eine URL auf eine Datei, muss sie den Verweis auf diese Datei enthalten, zum Beispiel: **WEB SEND HTTP REDIRECT**("/MyPage.HTM").

Dieser Befehl hat Vorrang vor Befehlen, die Daten senden (**WEB SEND FILE**, **WEB SEND BLOB**, etc.) und in derselben Methode liegen.

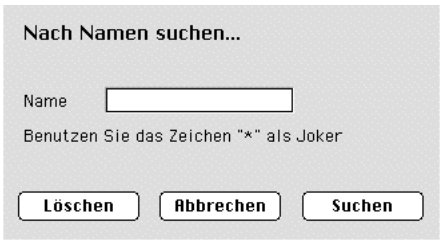
Mit diesem Befehl können Sie eine Anfrage auch auf einen anderen Web Server umleiten.

4D codiert automatisch die speziellen Zeichen der URL. Übergeben Sie den optionalen Parameter *, übersetzt 4D die Zeichen nicht.

Beachten Sie, dass die von diesem Befehl gesendete Anfrage den Status **302: Moved Temporarily** hat. Benötigen Sie den Status "moved permanently" (status 301), können Sie in den Kopfteil der Antwort ein Feld *HTTP X-.STATUS: 301* setzen.

Beispiel

Mit diesem Befehl können Sie in 4D über statische Seiten eigene Anfragen ausführen. Sie haben z.B. in einer statischen HTML Seite folgende Elemente gesetzt:



Hinweis: Die Aktion POST "/4dcgi/rech" wurde dem Textbereich und den Schaltflächen **OK** und **Abbrechen** zugewiesen.

In der **Datenbankmethode On Web Connection** fügen Sie folgenden Code ein:

```
Case of
   :($1="/4dcgi/rech")  //Empfängt 4D diese URL
 //Wurde die Schaltfläche OK angeklickt und enthält das Feld 'Name' einen Wert
      If((bOK="OK")&(name#""))
 //Ändere die URL zum Ausführen des Anfragecodes, die in derselben Methode an späterer Stelle liegt
         WEB SEND HTTP REDIRECT("/4dcgi/rech?"+Name)
      Else
 //Else kehrt zur beginnenden Seite zurück
         WEB SEND HTTP REDIRECT("/Seite1.htm")
      End if
      ...
   :($1="/4dcgi/rech?@")  //Wurde die URL umgeleitet
      ... //Setze den Anfragecode hier
End case
```

WEB SEND RAW DATA

WEB SEND RAW DATA (Daten {; *})

Parameter	Typ		Beschreibung
Daten	BLOB	➡	Zu sendende HTTP Daten
*	Operator	➡	Stückweise senden

Beschreibung

Der Befehl **WEB SEND RAW DATA** legt fest, dass der 4D Web Server HTTP Rohdaten sendet, die gestückelt werden können. *Daten* enthält die beiden Standardteile einer HTTP Antwort, das sind Header und Body. Der Server sendet die Daten ohne vorherige Formatierung.
4D überprüft jedoch Header und Body in der Antwort auf ihre Gültigkeit:

- Ist der Header unvollständig oder passt er nicht zu den Angaben des HTTP Protokolls, führt 4D die entsprechenden Änderungen aus.
- Ist die HTTP Anfrage unvollständig, fügt 4D die fehlende Information hinzu. Wollen Sie z.B. die Anfrage umleiten (Redirect), müssen Sie schreiben:

```
HTTP/1.1 302    Location: http://...
```

Übergeben Sie nur:

```
Location: http://...
```

vervollständigt 4D die Anfrage durch Hinzufügen von HTTP/1.1 302.

Mit dem optionalen Parameter * geben Sie an, dass die Antwort "stückweise" gesendet wird. Das Stückeln der Antwort ist hilfreich, wenn der Server eine Antwort sendet, ohne ihre Gesamtlänge zu kennen, z.B. wenn die Antwort nicht generiert wurde. Alle Browser, die mit HTTP/1.1- kompatibel sind, akzeptieren gestückelte Antworten (chunked responses).
Mit dem optionalen Parameter * bindet der Web Server automatisch das Feld **transfer-encoding: chunked** in den Header der Antwort ein, sofern erforderlich. Sie können den Header der Antwort natürlich auch manuell verwalten. Der Rest der Antwort berücksichtigt beim Formatieren ebenfalls die Option gestückelt. Gestückelte Antworten enthalten einen einzigen Header und eine undefinierte Anzahl Body "Stücke".

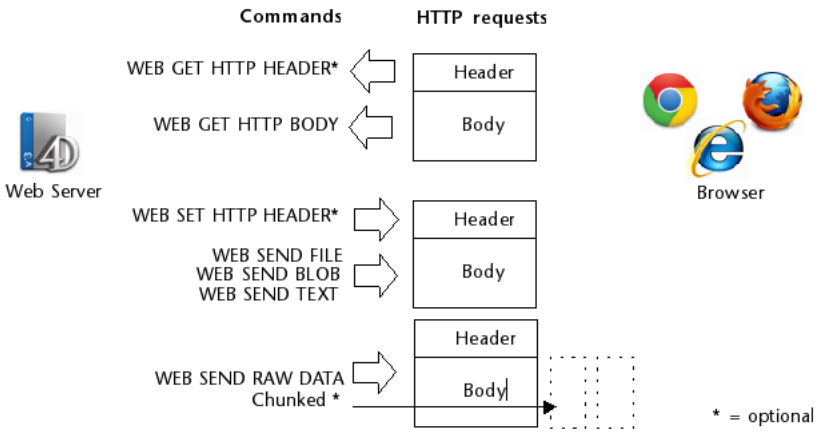
Alle Anweisungen **WEB SEND RAW DATA**, die auf die Ausführung von **WEB SEND RAW DATA** (Daten;*) in derselben Methode folgen, gelten als Teil der Antwort, unabhängig, ob mit oder ohne den Parameter *. Der Server beendet die stückweise Übersendung, wenn die Ausführung der Methode abgeschlossen ist.

Hinweis: Unterstützt der Web Client nicht HTTP/1.1, konvertiert 4D die Antwort automatisch in ein mit HTTP/1.0 kompatibles Format (die Daten werden nicht gestückelt gesendet). In diesem Fall entspricht das Ergebnis evtl. nicht Ihren Wünschen. Sie sollten deshalb prüfen, ob der Web Browser HTTP/1.1 unterstützt und eine entsprechende Antwort sendet.

Dafür können Sie folgenden Code verwenden:

```
C_BOOLEAN($0)
ARRAY TEXT(arFields;0)
ARRAY TEXT(arValues;0)
WEB GET HTTP HEADER(arFields;arValues)
$0:=False
If(Size of array(arValues)>=3)
   If(Position("HTTP/1.1";arValues{3})>0)
      $0:=True  ` Browser unterstützt HTTP/1.1; $0 gibt wahr zurück
   End if
End if
```

In Kombination mit dem Befehl **WEB GET HTTP BODY**, sowie anderen Befehlen im Kapitel "Web Server" steht 4D Entwicklern eine breite Palette an Werkzeugen zur Verfügung, um die Abarbeitung ein- und ausgehender HTTP-Anfragen vollkommen eigenständig zu gestalten. Die Übersicht zeigt die verschiedenen Werkzeuge:



Beispiel

Nachfolgendes Beispiel zeigt die Verwendung von **WEB SEND RAW DATA** mit der Option gestückelt. Die Daten – eine Folge von Nummern – wird in 100 Stücken gesendet, die unmittelbar in einer Schleife erzeugt werden. Beachten Sie, dass der Header der Antwort nicht explizit gesetzt wird. Der Befehl sendet ihn automatisch und fügt das Feld *transfer-encoding: chunked* ein, da der Parameter * verwendet wird.

```
C_LONGINT($cpt)
C_BLOB($my_blob)
C_TEXT($output)

For($cpt;1;100)
   $output:="["+String($cpt)+"]"
   TEXT TO BLOB($output;$my_blob;UTF8 text without length)
   WEB SEND RAW DATA($my_blob;*)
End for
```

WEB SEND TEXT

WEB SEND TEXT (htmlText {; Typ})

Parameter	Typ		Beschreibung
htmlText	Text	➡	HTML Textfeld oder Variable zum Senden an den Web Browser
Typ	Text	➡	MIME Typ

Beschreibung

Der Befehl **WEB SEND TEXT** sendet direkt Daten vom Typ Text im HTML Format.

Der Parameter *htmlText* enthält die Daten zum Senden. Da 4D den Inhalt des Parameters nicht überprüft, achten Sie darauf, dass die HTML Codierung korrekt ist.

Referenzen auf 4D Variablen und Tags vom Typ *4DSCRIPT* im Text werden immer analysiert.

Standardmäßig, d.h. ohne den Parameter *Typ*, nimmt 4D an, dass die gesendeten Daten vom Typ "text/html" sind. Der Befehl arbeitet dann exakt wie der Befehl **WEB SEND BLOB**, der ein BLOB vom Typ “html/txt” verwendet.
Im Parameter *Typ* können Sie auch den MIME Typ des zu sendenden Textes angeben. Weitere Informationen dazu finden Sie unter dem Befehl **WEB SEND BLOB**.

Beispiel

Folgende Methode:

```
TEXT TO BLOB("<html><head></head><body>"+String(Current time)+"</body></html>";$blob;UTF8 Text without length)
WEB SEND BLOB($blob;"text/html")
```

... lässt sich ersetzen durch eine einzelne Zeile:

```
WEB SEND TEXT("<html><head></head><body>"+String(Current time)+"</body></html>")
```

WEB SET HOME PAGE

WEB SET HOME PAGE (HomePage)

Parameter	Typ		Beschreibung
HomePage	String	➡	Seitenname oder HTML Zugriffspfad auf die Seite oder "", um die eigene Home Page nicht zu senden

Beschreibung

Der Befehl **WEB SET HOME PAGE** ermöglicht, die eigene Home Page für den aktuellen Web Prozess zu verändern.

Die definierte Seite ist mit dem Web Prozess verbunden. So können Sie verschiedene Home Pages festlegen, zum Beispiel, je nachdem, welcher Benutzer angemeldet ist. Diese Seite kann sowohl statisch als auch halb-dynamisch sein.

Im Parameter *HomePage* übergeben Sie den Namen der HTML Home Page oder den HTML Zugriffspfad für die Seite.

Hinweis: Existiert die in *HomePage* angegebene Seite nicht, wenn der Web Prozess zum ersten Mal darauf zugreift, erstellt der Web Server sie und weist ihr den Inhalt der Standard Home Page zu. Weitere Informationen dazu finden Sie im Abschnitt **Standard Home Page definieren**.

Soll *HomePage* für den aktuellen Web Prozess nicht länger als Home Page gesendet werden, übergeben Sie in *HomePage* einen leeren String ("").

Hinweis: Die Standard Home Page des Web Server wird in den Datenbank-Eigenschaften definiert.

WEB SET HTTP HEADER

WEB SET HTTP HEADER (Header/FeldArray {; WertArray})

Parameter	Typ		Beschreibung
Header/FeldArray	Text, Array Text	➡	Feld oder Variable mit der Anfrage HTTP Header oder HTTP Header Felder
WertArray	Array Text	➡	HTTP Kopfteil Feldinhalt

Beschreibung

Der Befehl **WEB SET HTTP HEADER** setzt die Felder in den HTTP Header der Antwort, die 4D an den Web Browser sendet. Der Befehl gilt nur in einem Web Prozess.
Mit diesem Befehl können Sie Cookies verwalten.

Es stehen zwei Syntaxarten zur Verfügung:

- Erste Syntax: **WEB SET HTTP HEADER** (Header)
  Im Parameter *FeldArray* übergeben Sie die gewünschten Felder vom Typ Text (Feld oder Variable) für den HTTP Header. Damit können Sie Header vom Typ "***HTTP/1.0 200 OK"+Char(13)+"Set-Cookie: C=HALLO***" schreiben.
  Die Felder müssen immer mit CR oder CR/LF (Zeilenschaltung/Zeilenvorschub) voneinander getrennt sein. 4D formatiert die Antwort

Beispiel für ein eigenes "Cookie":

```
C_TEXT($vTcookie)
$vTcookie:="Set-Cookie: USER="+String(Abs(Random))+"; PATH=/"
WEB SET HTTP HEADER($vTcookie)
```

Hinweis: Der Befehl erlaubt im Parameter *Header* keine Konstante vom Typ Text, es muss eine 4D Variable oder ein Feld sein.

Weitere Information über die Syntax finden Sie in der Dokumentation R.F.C (Request For Comments) unter der Internet Addresse: http://www.w3c.org.

- Zweite Syntax: **WEB SET HTTP HEADER** (FeldArray; WertArray)
  Der HTTP Header lässt sich durch die beiden Arrays *FeldArray* und *WertArray* definieren.
  Der Header lässt sich folgendermaßen schreiben:

```
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"
```

* Die beiden ersten Elemente sind die erste Zeile der Antwort. Werden diese eingegeben, sollten sie das erste und zweite Element des Array sein. Sie können diese aber auch übergehen und nur schreiben:

```
fieldArray{1}:="Set-Cookie"
valueArray{1}:="C=HELLO"
```

Legen Sie keinen Status fest, ist er automatisch HTTP/1.0 200 OK.

Die HTTP Feldnamen sind in Übereinstimmung mit dem HTTP Standard immer in Englisch. Das **Server** Feld ist standardmäßig "4D/<version>". Auch die Felder **Content-Length** und **Date** setzt 4D standardmäßig.

WEB SET OPTION

WEB SET OPTION (Selector ; Wert)

Parameter	Typ		Beschreibung
Selector	Lange Ganzzahl	➡	Code der Option
Wert	Lange Ganzzahl, Text	➡	Wert der Option

Beschreibung

Der Befehl **WEB SET OPTION** ändert den aktuellen Wert verschiedener Optionen zur Funktionsweise des 4D Web Server.

Änderungen an diesen Optionen bleiben erhalten, wenn der 4D Web Server beendet und wieder gestartet wird. Jedoch nicht, wenn die 4D Anwendung beendet und wieder gestartet wird.

Im Parameter *Selector* übergeben Sie eine Konstante unter dem Thema **Web Server** und in *Wert* den neuen Wert der Option:

Konstante	Wert	Kommentar
Web character set	17	**Reichweite:** 4D lokal, 4D Server **Beschreibung:** Damit können Sie sofort den Zeichensatz ändern, den der 4D Web Server mit 4D im lokalen Modus und 4D Server für die Kommunikation mit Browsern verwenden soll, die sich an die Datenbank anmelden. Die aktuelle Standardeinstellung richtet sich nach der Sprache des Betriebssystems. Dieser Parameter wird in den Einstellungen der Datenbank festgelegt. **Mögliche Werte:** Der Wert richtet sich nach dem jeweiligen Ausführungsmodus der Datenbank • **Unicode Modus:** Wird die Anwendung im Modus Unicode ausgeführt, müssen jetzt für diesen Parameter Zeichensatz Identifier übergeben werden. Das sind Identifier vom Typ MIBEnum Lange Ganzzahl oder Name String, definiert IANA (siehe unter http://www.iana.org/assignments/character-sets) Nachfolgend sehen Sie die Liste der Identifier, der dem von 4D Web Server unterstützten Zeichensatz entspricht: 4 = ISO-8859-1 12 = ISO-8859-9 13 = ISO-8859-10 17 = Shift-JIS 2026 = Big5 38 = euc-kr 106 = UTF-8 2024 = Windows-31J 2250 = Windows-1250 2251 = Windows-1251 2253 = Windows-1253 2255 = Windows-1255 2256 = Windows 1256 • **ASCII Kompatibilitätsmodus** 0: Western European 1: Japanisch 2: Chinesisch 3: Koreanisch 4: Benutzerdefiniert 5: Reserviert 6: Central European 7: Kyrillisch 8: Arabisch 9: Griechisch 10: Hebräisch 11: Türkisch 12: Baltisch
Web Client IP address to listen	23	**Reichweite**: Alle 4D remote Rechner **Mögliche Werte**: Siehe Selector 16 **Beschreibung**: Dient zur Angabe dieses Parameters für alle Rechner mit remote 4D, die als Web Server verwendet werden (gilt nur für den remote Web Server).
Web debug log	84	**Reichweite:** Lokaler Web Server **Hinweis:** Nach Neustart des HTTP Server wird ein neues Protokoll verwendet. **Beschreibung:** Ermöglicht, den Status des HTTP Anfrageprotokolls des 4D Web Server zu erhalten oder zu setzen. Ist er aktiviert, wird diese Datei mit Namen "**HTTPDebugLog_*nn*.txt**", im Ordner "Logs" der Anwendung gespeichert (*nn* ist die Dateinummer). Das ist hilfreich zum Debuggen bei Problemen mit dem Web Server. Die Datei protokolliert jede Anfrage und Antwort im Rohmodus. Die Anfragen werden mit Kopfteilen protokolliert; optional lassen sich auch Body-Bereiche mitprotokollieren. Weitere Informationen dazu finden Sie im **Anhang E: Beschreibung der Protokolldateien**. **Werte**: Eine der Konstanten mit der Vorsilbe "wdl" (siehe Beschreibung dieser Konstanten unter diesem Thema). **Standardwert**: 0 (nicht aktiviert)
Web HSTS enabled	86	**Reichweite**: 4D lokal, 4D Server. **Beschreibung**: HTTP Strict Transport Security (HSTS) Status. Über HSTS kann der 4D Web Server festlegen, dass Browser nur über sichere HTTPS Verbindungen mit ihm kommunizieren. Ist HSTS aktiviert, fügt der 4D Web Server automatisch in allen Antwort-Kopfteilen HSTS-bezogene Information hinzu. Browser speichern diese Information, wenn sie zum ersten Mal eine Antwort vom 4D Web Server empfangen. Alle nachfolgenden HTTP Anfragen werden dann automatisch in HTTPS Anfragen umgewandelt. Mit dem Selektor Web HSTS max age lässt sich angeben, wie lange diese Information vom Browser gespeichert wird. Für HSTS muss HTTPS auf dem Server aktiviert werden. HTTP muss ebenfalls aktiviert werden, um das erste Starten von Client Verbindungen zuzulassen. **Mögliche Werte**: 0 (deaktiviert, Standard) oder 1 (aktiviert) **Hinweis:** Die Einstellung wird erst nach Neustart des 4D Web Server übernommen.
Web HSTS max age	87	**Reichweite**: 4D lokal, 4D Server **Beschreibung**: Gibt die maximale Zeitspanne an (in Sekunden), die HSTS für jede neue Client-Verbindung aktiv bleibt. Diese Angabe wird auf Client-Seite für die angegebene Dauer gespeichert. **Mögliche Werte**: Lange Ganzzahl (Sekunden) **Standardwert**: 63072000 (2 Jahre) ***Warnung:*** *Ist HSTS aktiviert, verwenden Client Verbindungen diesen Mechanismus für die angegebene Dauer. Beim Testen Ihrer Anwendung empfehlen wir, einer kurze Dauer zu setzen, damit Sie bei Bedarf zwischen sicheren und nicht-sicheren Verbindungen wechseln können.*

Konstante	Wert	Kommentar
Web HTTP compression level	50	**Reichweite:** Lokaler Web Server **Mögliche Werte:** 1 bis 9 (1 = schneller, 9 = stärker komprimiert), -1 = beste Kombination **Beschreibung:** Setzt die Komprimierungsebene für jeden komprimierten HTTP Austausch für den 4D HTTP Server (Client Anfragen oder Server Antworten, Web und Web Service). Mit diesem Selektor können Sie den Austausch entweder über die Ausführungsgeschwindigkeit (weniger Komprimierung) oder die Komprimierungsmenge (weniger Geschwindigkeit) optimieren. Die Auswahl des passenden Wertes richtet sich nach der Größe und der Art der ausgetauschten Daten. Im Parameter *Wert* können Sie einen Wert von 1 bis 9 übergeben, wobei 1 die schnellste und 9 die höchste Komprimierung ist. Für einen Kompromiss zwischen Geschwindigkeit und Komprimierungsrate übergeben Sie -1. Standardmäßig ist als Komprimierungsebene 1 eingestellt (schnellere Komprimierung).
Web HTTP compression threshold	51	**Reichweite:** Lokaler HTTP Server **Mögliche Werte:** Jeder Wert vom Typ Lange Ganzzahl **Beschreibung:** Setzt den Schwellwert, bis zu dem der Austausch von Daten im Rahmen von 4D Web Services nicht komprimiert werden soll. Diese Einstellung ist hilfreich, um beim Austausch geringer Datenmengen zu verhindern, dass Rechenzeit für die Komprimierung beansprucht wird. In *Wert* übergeben Sie eine Größe in Bytes. Standardmäßig ist als Schwellwert für Komprimierung 1024 Bytes eingestellt.
Web HTTP enabled	88	**Reichweite**: 4D lokal, 4D Server **Beschreibung**: Status für Kommunikation über HTTP **Mögliche Werte**: 0 (deaktiviert) oder 1 (aktiviert)
Web HTTP TRACE	85	**Reichweite**: Lokaler Web Server **Wird zwischen 2 Sitzungen beibehalten**: Nein **Beschreibung**: Ermöglicht, die Methode HTTP TRACE im 4D Web Server zu aktivieren/deaktivieren. Ab 4D v15 R2 weist der 4D Web Server aus Sicherheitsgründen HTTP TRACE Anfragen standardmäßig mit dem Fehler 405 ab (HTTP TRACE ist deaktiviert). Bei Bedarf können Sie die Methode HTTP TRACE für die Sitzung aktivieren, indem Sie für diese Konstante den Wert 1 übergeben. Dann sendet der 4D Web Server bei HTTP TRACE Anfragen die Anfragezeile, Kopfteil und Hauptteil. **Mögliche Werte**: 0 (deaktiviert) oder 1 (aktiviert) **Standardwert**: 0 (deaktiviert)
Web HTTPS enabled	89	**Reichweite**: 4D lokal, 4D Server **Beschreibung**: Status für Kommunikation über HTTPS. **Mögliche Werte**: 0 (deaktiviert) oder 1 (aktiviert)
Web HTTPS port ID	39	**Reichweite:** 4D lokal, 4D Server **Mögliche Werte:** 0 bis 65535 **Beschreibung:** Mit diesem Selector können Sie per Programmierung die TCP Portnummer ändern, die der Web Server von 4D im lokalen Modus und von 4D Server für sichere Verbindungen via TLS (HTTPS protocol) verwendet. Die HTTPS Portnummer wird in den Datenbank-Eigenschaften auf der Seite **Web>Konfiguration** gesetzt. Der Wert ist standardmäßig 443. Für den Parameter *Wert* können Sie Konstanten unter dem Thema **TCP Port Nummern** einsetzen.
Web inactive process timeout	78	**Reichweite:** Lokaler Web Server **Mögliche Werte:** Lange Ganzzahl (Minuten) **Beschreibung:** Ändert die Lebensdauer des inaktiven Prozesses, der Sessions zugeordnet ist. Am Ende des Timeout wird der Prozess auf dem Server gestoppt, die **Datenbankmethode On Web Close Process** wird aufgerufen, dann wird der Session-Kontext gelöscht. **Standardwert:** 480 Minuten (zum Wiederherstellen des Standardwerts 0 übergeben)
Web inactive session timeout	72	**Reichweite:** Lokaler Web Server **Beschreibung:** Ändert die Lebensdauer inaktiver Sessions (Dauer wird in Cookie gesetzt). Endet diese Periode, läuft das Cookie der Session ab und wird nicht mehr vom HTTP Client gesendet. **Mögliche Werte:** Lange Ganzzahl (Minuten) **Standardwert:** 480 Minuten (zum Wiederherstellen des Standardwerts 0 übergeben)
Web IP address to listen	16	**Reichweite:** 4D lokal, 4D Server **Beschreibung:** IP Adresse, unter der der 4D Web Server HTTP Anfragen mit 4D im lokalen Modus und 4D Server empfängt. Standardmäßig ist keine bestimmte Adresse definiert (Wert = 0). Dieser Parameter lässt sich in den Datenbank-Eigenschaften auf der Seite **Web>Konfiguration** festlegen. Dieser Selector ist hilfreich für 4D Web Server mit einkompilierter 4D Volume Desktop, die keinen Zugriff auf den Designmodus zulassen. **Mögliche Werte**: IP Adresse String. Beide Formate werden unterstützt: IPv6, z.B. "2001:0db8:0000:0000:0000:ff00:0042:8329") und IPv4 (z.B. "123.45.67.89"). **Hinweis:** Zur Wahrung der Kompatibilität werden überholte IPv4 Adressen in Form von hexadezimalen Langen Ganzzahlen noch unterstützt.
Web keep session	70	**Reichweite:** Lokaler Web Server **Beschreibung:** Aktiviert oder deaktiviert den neuen Modus zum Verwalten der Sessions. Weitere Informationen dazu finden Sie im Abschnitt **Web Sessions verwalten** **Mögliche Werte:** 1 (Modus aktivieren) oder 0 (Modus deaktivieren) **Standardwert:** 1 für mit v13 erstellte Datenbanken, 0 für konvertierte Datenbanken. Beachten Sie, dass dieser Modus auch den Mechanismus zum Wiederverwenden temporärer Kontexte im remote Modus ermöglicht. Weitere Informationen dazu finden Sie im Abschnitt **Web Server, Einstellungen**

Konstante	Wert	Kommentar
Web log recording	29	**Reichweite:** 4D lokal, 4D Server **Beschreibung:** Startet oder stoppt das Speichern von Web Anfragen, die vom Web Server von 4D im lokalen Modus oder 4D Server empfangen werden. Der Standardwert ist 0, d.h. Anfragen werden nicht gespeichert. Das Logbuch von Web Anfragen wird als Textdatei mit Namen "logweb.txt" gespeichert, die automatisch in den Ordner Logs neben der Strukturdatei der Anwendung gesetzt wird. Das Format dieser Datei richtet sich nach dem übergebenen Wert. Weitere Informationen zu Formaten für Web Logfiles finden Sie im Abschnitt **Information über die Web Site** . Diese Datei lässt sich auch in den Datenbank-Eigenschaften auf der Seite **Log (Typ)** aktivieren. **Mögliche Werte:** 0 = Nicht speichern (Standard), 1 = In CLF Format speichern, 2 = In DLF Format speichern, 3 = In ELF Format speichern, 4 = In WLF Format speichern. **Warnung:** Format 3 und 4 sind individuell anpassbare Formate, d.h. Sie müssen den Typ zuvor in den Datenbank-Eigenschaften auf der Seite **Log (Typ)** definieren. Verwenden Sie diese Formate, ohne zuvor den Typ festzulegen, wird kein Logbuch angelegt.
Web max concurrent processes	18	**Reichweite:** 4D lokal, 4D Server **Werte:** Sie können jeden Wert zwischen 10 und 32 000 übergeben. Der Standardwert ist 100. **Beschreibung:** Damit setzen Sie die maximale Anzahl aller gleichzeitig laufenden Web Prozesse (kontextuell, nicht kontextuell oder die zum Pool der Prozesse gehören), die der 4D Web Server mit 4D im lokalen Modus und 4D Server unterstützt. Ist die maximale Anzahl erreicht, erstellt 4D keinen weiteren Prozess und gibt den HTTP Status 503 zurück - Dienst für weitere neue Anfragen nicht verfügbar. Dieser Parameter verhindert die Übersättigung des 4D Web Server. Sie kann eintreten, wenn gleichzeitig eine zu große Anzahl an Anfragen gesendet wird oder zu viele Kontext-Erstellungen angefordert werden. Theoretisch ist die max. Anzahl der Web Prozesse das Ergebnis der folgenden Formel: Verfügbarer Speicher/Stapelgröße der Web Prozesse. Sie können sich auch die Information über die Web Prozesse im Runtime Explorer ansehen: Er zeigt die aktuelle Anzahl der Web Prozesse und die max. erreichte Anzahl seit dem Hochfahren des Web Server an.
Web max sessions	71	**Reichweite:** Lokaler Web Server **Beschreibung:** Begrenzt die Anzahl gleichzeitiger Sessions. Bei Erreichen des Limits wird die älteste Session geschlossen (die **Datenbankmethode On Web Close Process** wird aufgerufen), wenn der Web Server eine neue erstellen muss. **Mögliche Werte:** Lange Ganzzahl. Die Anzahl gleichzeitiger Sessions kann nicht größer sein als die Gesamtzahl der Web Prozesse (Option Web Max Concurrent Processes, standardmäßig 100). **Standardwert:** 100 (zum Wiederherstellen des Standardwerts 0 übergeben)
Web maximum requests size	27	**Reichweite:** 4D lokal, 4D Server **Mögliche Werte:** 500 000 bis 2 147 483 648 **Beschreibung:** Maximale Größe (in Bytes) hereinkommender HTTP Anfragen (POST), die der Web Server akzeptiert. Standardmäßig ist der Wert 2 000 000 vorgegeben, z.B. etwas unter 2 MB. Übergeben Sie den maximalen Wert (2 147 483 648), wird praktisch keine Grenze gesetzt. Die Begrenzung sorgt dafür, dass der Web Server nicht überlastet wird durch zu große eingehende Anfragen. Erreicht eine Anfrage den Grenzwert, weist der 4D Web Server diese zurück.
Web port ID	15	**Reichweite:** 4D im lokalen Modus und 4D Server **Beschreibung:** Setzt oder erhält die Nummer des TCP Port, den 4D Web Server mit 4D im lokalen Modus und mit 4D Server (Lange Ganzzahl) verwendet. Standardmäßig ist der Wert 80. Die TCP Port Nummer wird in den Einstellungen der Datenbank auf der Seite "Web/Konfiguration" gesetzt. Für den Parameter *Wert* können Sie eine Konstante unter dem Thema **TCP Port Nummern** verwenden. Dieser Selector ist im Rahmen von 4D Web Servern mit einkompiliertem 4D Desktop hilfreich (kein Zugriff auf die Design-Umgebung). **Mögliche Werte:** Weitere Informationen über die TCP Port Nummer finden Sie im Abschnitt **Web Server, Einstellungen**. **Standardwert:** 80
Web session cookie domain	81	**Reichweite:** lokaler Web Server **Mögliche Werte:** Text **Beschreibung:** Setzt oder erhält den Wert des Feldes "domain" des Session Cookie (Text). Dieser Selektor, sowie Selektor 82 sind hilfreich zum Überprüfen der Reichweite von Session Cookies: Haben Sie z.B. für diesen Selektor den Wert "/*.4d.fr" gesetzt, sendet der Client ein Cookie nur, wenn die Anfrage an die Domaine ".4d.fr" gerichtet ist. Das schließt Server aus, die externe statische Daten hosten.
Web session cookie name	73	**Reichweite:** Lokaler Web Server **Beschreibung:** Setzt den Namen des Cookie zum Sichern der Session ID **Mögliche Werte:** Text **Standardwert:** "4DSID" (zum Wiederherstellen des Standardwerts leeren String übergeben)
Web session cookie path	82	**Reichweite:** lokaler Web Server **Mögliche Werte:** Text **Beschreibung:** Setzt oder erhält den Wert des Feldes "path" des Session Cookie (Text). Dieser Selektor, sowie Selektor 81 sind hilfreich zum Überprüfen der Reichweite von Session Cookies: Haben Sie z.B. für diesen Selector den Wert "/4DACTION" gesetzt, sendet der Client ein Cookie nur für dynamische Anfragen, die mit 4DACTION beginnen und nicht für Bilder, statische Seiten, etc.

Konstante	Wert	Kommentar
Web session enable IP address validation	83	**Reichweite**: Lokaler Web Server **Beschreibung**: Damit können Sie die Überprüfung der IP Adresse für Session Cookies aktivieren oder deaktivieren. Aus Sicherheitsgründen prüft der 4D Web Server standardmäßig die IP Adresse einer Anfrage mit einem Session Cookie und weist sie ab, wenn sie nicht zur IP Adresse passt, über die das Cookie erstellt wurde. Sie können diese Überprüfung bei bestimmten Applikationen deaktivieren und das Session Cookie akzeptieren, auch wenn die IP Adresse nicht dazu passt. Wechseln z.B. mobile Geräte zwischen Wifi und 3G/4G Netzwerken, ändert sich die IP Adresse. In diesem Fall übergeben Sie den Wert 0, damit Clients weiterhin ihre Web Sessions verwenden können, auch wenn sie die IP Adresse wechseln. Beachten Sie, dass diese Einstellung die Sicherheitstufe Ihrer Applikation herabsetzt. Die Einstellung ist sofort wirksam, d.h. Sie müssen den HTTP Server nicht neu starten. **Mögliche Werte**: 0 (deaktiviert) oder 1 (aktiviert) **Standardwert**: 1 (IP Adressen werden geprüft)

Verwenden Sie den *Selector* Web debug log, können Sie im Parameter *Wert* eine der folgenden Konstanten übergeben:

Konstante	Typ	Wert	Kommentar
wdl disable	Lange Ganzzahl	0	Web HTTP Fehlerprotokoll ist deaktiviert
wdl enable with all body parts	Lange Ganzzahl	7	Web HTTP Fehlerprotokoll ist aktiviert mit Body Bereichen der Anfrage und der Antwort
wdl enable with request body	Lange Ganzzahl	5	Web HTTP Fehlerprotokoll ist aktiviert nur mit Body Bereichen der Anfrage
wdl enable with response body	Lange Ganzzahl	3	Web HTTP Fehlerprotokoll ist aktiviert nur mit Body Bereichen der Antwort
wdl enable without body	Lange Ganzzahl	1	Web HTTP Fehlerprotokoll ist aktiviert ohne Body Bereiche (in diesem Fall wird die Body Größe angezeigt)

Beispiel

Das http Fehlerprotokoll ohne Body Bereiche aktivieren:

```
WEB SET OPTION(Web debug log;wdl enable without body)
```

Das Protokoll lautet wie folgt:

```
# 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 (RootOrdner)

Parameter	Typ		Beschreibung
RootOrdner	String	➡	Pfadname des Web Server Root Ordners

Beschreibung

Der Befehl **WEB SET ROOT FOLDER** ändert den Standard Root Ordner, in dem 4D nach der vom Web Server angeforderten HTML Datei sucht.

Dieser Befehl berücksichtigt nicht den standardmäßigen HTML Root Ordner, der evtl. in den Datenbank-Eigenschaften definiert wurde. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

Sie können den Ort des Root Ordners in HTML Syntax (Typ URL) oder der Syntax des Systems (absoluter Pfad) ausdrücken:

- HTML Syntax: Die Namen für Verzeichnisse bzw. Ordner sind - unabhängig von der Plattform - durch Schrägstrich (“/”) voneinander getrennt.
- System Syntax: Vollständiger Pfadname in der Schreibweise der aktuellen Plattform:
  - (Mac OS) Disk:Applications:myserv:folder
  - (Windows) C:\Applications\myserv\folder

Hinweise:

- Sie müssen den Web Server neu starten, damit der neue Root Ordner berücksichtigt wird.
- Mit der Funktion **Get 4D folder** können Sie jederzeit den Speicherort des aktuellen Root Ordners herausfinden.

Bei einem ungültigen Pfadnamen erhalten Sie einen OS-Systemfehler. Sie können diesen Fehler mit einer Methode **ON ERR CALL** abfangen. Zeigen Sie eine Warnung bzw. Meldung aus dieser Fehlermethode heraus an, erscheint er im Browser.

WEB START SERVER

WEB START SERVER
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **WEB START SERVER** startet den Web Server der 4D Anwendung, in der er ausgeführt wird (4D oder 4D Server). Die Datenbank wird dann über Ihr Intranet Netzwerk oder im Internet veröffentlicht.

Wurde der Web Server erfolgreich gestartet, hat die Systemvariable **OK** den Wert 1, sonst den Wert 0 (Null). Ist beispielsweise das TCP/IP Netzwerkprotokoll nicht korrekt konfiguriert, hat **OK** den Wert 0.

Systemvariablen und Mengen

Wurde der Web Server erfolgreich gestartet, hat die Systemvariable **OK** den Wert 1, sonst den Wert 0 (Null).

WEB STOP SERVER

WEB STOP SERVER
Dieser Befehl benötigt keine Parameter

Beschreibung

Der Befehl **WEB STOP SERVER** beendet den Web Server der 4D Anwendung, in der er ausgeführt wird (4D oder 4D Server).
Wurde der Web Server gestartet, werden alle Web Verbindungen abgebrochen und alle Web Prozesse beendet.
Wurde der Web Server nicht gestartet, hat der Befehl keine Auswirkung.

WEB Validate Digest

WEB Validate Digest (Benutzername ; Kennwort) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Benutzername	Text	➡	Benutzername
Kennwort	Text	➡	Benutzerkennwort
Funktionsergebnis	Boolean	⮌	Wahr=Authentifizierung OK, Falsch=Authentifizierung nicht OK

Beschreibung

Die Funktion **WEB Validate Digest** prüft die Gültigkeit der Identifizierung (Name und Kennwort), die der Benutzer beim Anmelden an den Web Server angibt. Diese Funktion muss in der **Datenbankmethode On Web Authentication** bei der Web Authentifizierung im Digest Modus verwendet werden. Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

In den Parametern *Benutzername* und *Kennwort* übergeben Sie die lokal gespeicherten Angaben zur Identifizierung. Der Befehl erstellt anhand dieser Informationen einen Wert und vergleicht ihn mit der Information, die der Web Browser sendet.
Sind beide Werte gleich, gibt die Funktion Wahr zurück, sonst Falsch.
Mit dieser Funktionsweise können Sie ihr eigenes Sicherheitssystem für Zugriff auf den Web Server per Programmierung verwalten und pflegen. Beachten Sie, dass sich Digest Validierung und 4D Kennwörter nicht gleichzeitig verwenden lassen.

Hinweis: Unterstützt der Browser nicht die Digest Authentifizierung, wird ein Fehler zurückgegeben (Authentifizierungsfehler).

Beispiel

Beispiel für **Datenbankmethode On Web Authentication** im Digest Modus:

```
  ` On Web Authentication Database Method
C_TEXT($1;$2;$5;$6;$3;$4)
C_TEXT($user)
C_BOOLEAN($0)
$0:=False
$user:=$5
  `Aus Sicherheitsgründen Namen mit @ ablehnen
If(WithWildcard($user))
   $0:=False
  `Die Methode WithWildcard ist im Abschnitt "On Web Authentication Database Method" beschrieben
Else
   QUERY([WebUsers];[WebUsers]User=$user)
   If(OK=1)
      $0:=WEB Validate digest($user;[WebUsers]password)
   Else
      $0:=False  `Benutzer existiert nicht
   End if
End if
```

_o_SET CGI EXECUTABLE

_o_SET CGI EXECUTABLE (url1 {; url2})

Parameter	Typ		Beschreibung
url1	String	➡	Zugriff URL
url2	String	➡	Zugriff URL

Befehl deaktiviert

Dieser Befehl ist ab Version 13 deaktiviert. Wenn er aufgerufen wird, wird Fehler 33 generiert ("Methode oder Funktion ist nicht implementiert").

_o_SET WEB DISPLAY LIMITS

_o_SET WEB DISPLAY LIMITS (AnzDatensätze {; AnzSeiten {; BildRef}})

Parameter	Typ		Beschreibung
AnzDatensätze	Lange Ganzzahl	➡	Max. Anzahl Datensätze pro HTML Seite
AnzSeiten	Lange Ganzzahl	➡	Max. Anzahl Seitenverweise am Ende jeder HTML Seite
BildRef	Lange Ganzzahl	➡	Bildreferenznummer für Schaltfläche Ganze Seite

Befehl deaktiviert

Dieser Befehl ist ab Version 13 deaktiviert. Wenn sie aufgerufen wird, wird Fehler 33 generiert ("Methode oder Funktion ist nicht implementiert").

_o_SET WEB TIMEOUT

_o_SET WEB TIMEOUT (Timeout)

Parameter	Typ		Beschreibung
Timeout	Lange Ganzzahl	➡	Timeout der Web Verbindungen in Sekunden

Befehl deaktiviert

Dieser Befehl ist ab Version 13 deaktiviert . Wenn er aufgerufen wird, wird Fehler 33 generiert ("Methode oder Funktion ist nicht implementiert").

_o_Web Context

_o_Web Context -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Boolean	True = Kontextueller Modus, False = Nicht kontextueller Modus

Funktion deaktiviert

Diese Funktion ist ab Version 13 deaktiviert. Wenn sie aufgerufen wird, wird Fehler 33 generiert ("Methode oder Funktion ist nicht implementiert").

Web Services (Client)

- Befehle für 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

Befehle für Web Services (Client)

4D unterstützt ab Version 2003 “Web Services”, d.h. Sie können Web Dienste aus Ihrer Datenbank veröffentlichen (Server Ebene) bzw. einsetzen (Client Ebene). Ein Web Service ist ein Satz Funktionen, der als Einheit zusammengefügt ist und in einem Netzwerk veröffentlicht werden kann. Jede Anwendung, die mit Web Services kompatibel ist und an ein Netzwerk angeschlossen ist, kann diese Funktionen aufrufen.
Web Services können ganz verschiedene Aufgaben durchführen, z.B. Bücher bestellen, E-Commerce abwickeln, Marktwerte überwachen, etc.

Weitere Informationen über Konzept und Arbeitsweise von Web Services finden Sie im Kapitel **Web Services publizieren und einsetzen** des Handbuchs *4D Designmodus*.

Die Veröffentlichung von Web Services mit 4D lässt sich mit den Optionen in den Methodeneigenschaften leicht ausführen. Das ist für die meisten Fälle ausreichend. Wollen Sie dagegen bestimmte Mechanismen verändern, mit Datenarrays arbeiten, etc., müssen Sie die Server SOAP Befehle von 4D verwenden.

Dieser Abschnitt beschreibt die Befehle zur Nutzung eines (im Netzwerk) zur Verfügung gestellten Web Service (**Client Ebene**) in 4D. Weitere Informationen zum Einrichten von Web Services in 4D finden Sie im Abschnitt **Befehle für Web Services (Server)** .

Hinweis: Die Begriffe “SOAP” und “Web Service” wurden zur Unterscheidung zwischen Befehls- und Konstantennamen auf Server-Ebene bzw. Client-Ebene eingeführt. Beide Konzepte beziehen sich auf dieselbe Technologie.

WEB SERVICE AUTHENTICATE

WEB SERVICE AUTHENTICATE (Auswahlname ; Kennwort {; AuthMethode} {; *})

Parameter	Typ		Beschreibung
Auswahlname	String	➡	Benutzername
Kennwort	String	➡	Benutzerkennwort
AuthMethode	Lange Ganzzahl	➡	Authentifizierungsmethode 0 oder weggelassen = nicht spezifiziert, 1 = BASIC, 2 = DIGEST
*	Operator	➡	Mit *: Authentifizierung über Proxy

Beschreibung

Der Befehl **WEB SERVICE AUTHENTICATE** fragt für den Web Service die Authentifizierung der Client Anwendung ab (einfache Authentifizierung). Die Methoden BASIC und DIGEST, sowie das Vorhandensein eines Proxy werden unterstützt.
Weitere Informationen dazu finden Sie im Abschnitt **Sicherheit der Verbindung**.

Die Parameter *Name* und *Kennwort* übergeben die erforderliche Information zur Identifikation (Benutzername und Kennwort). Diese Angaben werden verschlüsselt und in der HTTP Anfrage hinzugefügt, welche über den Befehl **WEB SERVICE CALL** an den Web Service gesendet wird. Zuerst müssen Sie jedoch den Befehl **WEB SERVICE AUTHENTICATE** aufrufen.

Der optionale Parameter *AuthMethode* gibt an, welche Authentifizierungsmethode für den nächsten Aufruf des Befehls **WEB SERVICE CALL** verwendet wird. Es gibt folgende Werte:

- 2 = Verwende die DIGEST Authentifizierungsmethode
- 1 = Verwende die BASIC Authentifizierungsmethode
- 0 (oder Parameter nicht angegeben) = Verwende die am besten geeignete Methode. In diesem Fall sendet 4D eine zusätzliche Anfrage, um die Authentifizierungsmethode abzuklären.

Mit dem optionalen Parameter * geben Sie an, dass die Authentifizierungsinformation an einen HTTP Proxy gesendet werden soll. Diese Konfiguration muss bei einem Proxy integriert werden, der die Authentifizierung zwischen dem Web Service Client und dem Web Service selbst erfordert. Ist der Web Service selbst authentifiziert, ist eine doppelte Authentifizierung erforderlich (siehe Beispiel).

Standardmäßig werden die Angaben zur Authentifizierung nach jeder Anfrage auf Null zurückgesetzt. Deshalb müssen Sie vor dem Aufrufen von **WEB SERVICE CALL** immer zuerst den Befehl **WEB SERVICE AUTHENTICATE** verwenden. Mit einer Option des Befehls **WEB SERVICE SET OPTION** können Sie die Angaben jedoch eine bestimmte Zeitspanne beibehalten. In diesem Fall müssen Sie nicht vor jedem Aufrufen von **WEB SERVICE CALL** den Befehl **WEB SERVICE AUTHENTICATE** ausführen.

Schlägt die Authentifizierung fehl, gibt der SOAP Server einen Fehler zurück, den Sie über die Funktion **WEB SERVICE Get info** herausfinden können.

Beispiel

Authentifizierung mit einen Web Service, der hinter einem Proxy liegt:

```
// Authentifizierung für Web Service im DIGEST Modus
WEB SERVICE AUTHENTICATE("SoapUser";"123";2)
// Authentifizierung für Proxy im Standardmodus
WEB SERVICE AUTHENTICATE("ProxyUser";"456";*)
WEB SERVICE CALL(...)
```

WEB SERVICE CALL

WEB SERVICE CALL (ZugriffURL ; SoapAktion ; Methodenname ; Namensraum {; KomplexTyp {; *}})

Parameter	Typ		Beschreibung
ZugriffURL	String	➡	Zugriff URL zu Web Service
SoapAktion	String	➡	Inhalt des Feldes SOAPAktion
Methodenname	String	➡	Name der Methode
Namensraum	String	➡	Namensraum
KomplexTyp	Lange Ganzzahl	➡	Konfiguration komplexer Typen (ohne Angabe einfache Typen)
*	Operator	➡	Mit *: Verbindung nicht schließen

Beschreibung

Der Befehl **WEB SERVICE CALL** ruft einen Web Service durch Senden einer HTTP Anfrage auf. Diese Anfrage enthält die SOAP Meldung, die zuvor mit dem Befehl **WEB SERVICE SET PARAMETER** erstellt wurde.

Jeder nachfolgende Aufruf von **WEB SERVICE SET PARAMETER** löst die Erstellung einer neuen Anfrage aus. Die Ausführung von **WEB SERVICE CALL** entfernt die Ergebnisse von zuvor aufgerufenen Web Services und ersetzt sie durch die neuen Ergebnisse.

In *ZugriffURL* übergeben Sie die vollständige URL für den Zugriff auf den Web Service. Verwechseln Sie diese URL nicht mit der für die WSDL Datei, welche den Web Service beschreibt.

- **Zugriff im gesicherten Modus (SSL)**: Wollen Sie einen Web Service im gesicherten Modus mit SSL verwenden, setzen Sie vor die URL https:// anstelle von http://. Dann läuft die Verbindung im gesicherten Modus.
  Beachten Sie, dass dieser Befehl ein Server Zertifikat verwenden kann (siehe Befehl **HTTP SET CERTIFICATES FOLDER**). Ist es ungültig (abgelaufen oder abgewiesen), wird die Systemvariable OK auf 0 gesetzt und Fehler 901 "Server Zertifikat ungültig" zurückgegeben. Dieser Fehler lässt sich mit einer Methode abfangen, die über den Befehl **ON ERR CALL** installiert wurde.

In *SoapAktion* übergeben Sie den Inhalt des Feldes SOAPAction der Anfrage. Es enthält in der Regel den Wert “ServiceName#MethodName”.

In *MethodenName* übergeben Sie den Namen der entfernten Methode, d.h. die zum Web Service gehört, welche Sie ausführen wollen.

In *Namensraum* übergeben Sie den XML Namensraum für die SOAP Anfrage. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.

Der optionale Parameter *KomplexTyp* spezifiziert die Konfiguration der empfangenen bzw. gesendeten Parameter des Web Service, die mit den Befehlen **WEB SERVICE SET PARAMETER** und **WEB SERVICE GET RESULT** definiert wurden. Der Wert von *KomplexTyp* richtet sich nach dem Publikationsmodus des Web Service (DOC oder RPC, siehe Handbuch *4D Designmodus*) und nach seinen anderen Parametern.

In *KomplexTyp* müssen Sie eine der vordefinierten Konstanten unter dem Thema **Web Services (Client)** übergeben:

Konstante	Typ	Wert
Web Service dynamic	Lange Ganzzahl	0
Web Service manual	Lange Ganzzahl	3
Web Service manual in	Lange Ganzzahl	1
Web Service manual out	Lange Ganzzahl	2

Jede Konstante entspricht der Einstellung eines Web Service. Das ist die Kombination aus Publikationsmodus (RPC/DOC) und integrierter Parameter (Eingabe/Ausgabe, einfach oder komplex).

Zur Erinnerung: Die Merkmale der Eingangs- bzw. Ausgangsparameter werden aus Sicht der Proxy-Methode/Web Service bewertet.

- Ein “input” Parameter ist ein Wert, welcher der Proxy-Methode, und so dem Web Service übergeben wird,
- Ein “output” Parameter wird vom Web Service und so von der Proxy-Methode zurückgegeben (in der Regel via $0).

Nachfolgende Tabelle zeigt die möglichen Einstellungen mit den dazugehörigen Konstanten:

	Eingabeparameter	
Ausgabeparameter	**Einfach**	**Komplex**
Einfach	Web Service Dynamic (RPC Modus)	Web Service Manual In (RPC Modus)
Komplex	Web Service Manual Out (RPC Modus)	Web Service Manual (RPC oder DOC Modus)

Sie können also diese Einstellungen implementieren. In allen Fällen verwaltet 4D die Formatierung der SOAP Anfrage, die an den Web Service gesendet wird und seinen „envelope“ (Umschlag). Es ist Ihre Aufgabe, den Inhalt der Anfrage gemäß der verwendeten Einstellung zu formatieren.

Hinweis: Auch wenn Daten-Arrays komplexe XML Typen sind, behandelt 4D diese als einfache Typen.

RPC Modus, einfache Ein- und Ausgabe

Das ist die am leichtesten verwendbare Einstellung. In diesem Fall enthält der Parameter *KomplexTyp* die Konstante Web Service Dynamic oder wird ausgelassen.

Die gesendeten Parameter und erhaltenen Antworten lassen sich direkt verwalten, ohne vorherige Bearbeitung.

Siehe Beispiel zum Befehl **WEB SERVICE GET RESULT**.

RPC Modus, komplexe Eingabe und einfache Ausgabe

In diesem Fall enthält der Parameter *KomplexTyp* die Konstante Web Service Manual In. Mit dieser Einstellung müssen Sie jedes XML Quellelement dem Web Service "manuell" als BLOB übergeben, und zwar über den Befehl **WEB SERVICE SET PARAMETER**.

Sie müssen selbst das Anfangs-BLOB als gültiges XML Element formatieren. Sein erstes Element muss das erste erscheinende "Child" Element aus dem Element <Body> der endgültigen Anfrage sein.

Beispiel

```
C_BLOB($1)
C_BOOLEAN($0)

WEB SERVICE SET PARAMETER("MyXMLBlob";$1)
WEB SERVICE CALL("http://my.domain.com/my_service";"MySoapAction";"TheMethod";"http://my.namespace.com/";Web Service manual in)
WEB SERVICE GET RESULT($0;"MyOutputVar";*)
```

RPC Modus, einfache Eingabe und komplexe Ausgabe

In diesem Fall enthält der Parameter *KomplexTyp* die Konstante Web Service Manual Out. Der Web Service gibt jeden Ausgabeparameter als XML Element, gespeichert in einem BLOB zurück. Sie finden diesen Parameter über den Befehl **WEB SERVICE GET RESULT** wieder. Sie können dann den Inhalt des erhaltenen BLOB mit den XML Befehlen von 4D durchlaufen.

Beispiel

```
C_BLOB($0)
C_BOOLEAN($1)

WEB SERVICE SET PARAMETER("MyInputVar";$1)
WEB SERVICE CALL("http://my.domain.com/my_service";"MySoapAction";"TheMethod";"http://my.namespace.com/";Web Service manual out)
WEB SERVICE GET RESULT($0;"MyXMLOutput";*)
```

RPC Modus, komplexe Eingabe und Ausgabe

In diesem Fall enthält der Parameter *KomplexTyp* die Konstante Web Service Manual. Jeder Eingabe- und Ausgabeparameter muss dann als XML Element in BLOBs gespeichert werden, wie in den beiden obigen Einstellungen beschrieben.

Beispiel

```
C_BLOB($0)
C_BLOB($1)

SET WEB SERVICE PARAMETER("MyXMLInputBlob";$1)
CALL WEB SERVICE("http://my.domain.com/my_service";"MySoapAction";"TheMethod"
"http://my.namespace.com/";Web Service manual)
GET WEB SERVICE RESULT($0;"MyXMLOutput";*)
```

DOC Modus

Ein Proxy, das eine Methode für ein DOC Web Service aufruft, ist ähnlich zum Proxy, das eine Methode für ein RPC Web Service über komplexe Eingabe- und Ausgabeparameter aufruft.

Der einzige Unterschied liegt in der Ebene des XML Inhalts der gesendeten und empfangenen BLOB Parameter. Aus Sicht von 4D sind Aufbau und Senden der SOAP Anfrage das gleiche.

Beispiel

```
C_BLOB($0)
C_BLOB($1)

WEB SERVICE SET PARAMETER("MyXMLInput";$1)
WEB SERVICE CALL("http://my.domain.com/my_service";"MySoapAction";"TheMethod";"http://my.namespace.com/";Web Service manual)
WEB SERVICE GET RESULT($0;"MyXMLOutput";*)
```

Hinweis: Für DOC Web Services spielt der Wert der als Parameter übergebenen Strings ("MyXMLInput" und "MyXMLOutput" s.o.) keine Rolle; Sie können auch leere Strings übergeben "". Tatsächlich verwendet die SOAP Anfrage mit dem XML Dokument diesen Wert nicht. Es ist jedoch zwingend, diese Parameter zu übergeben.

Um einen Web Service im DOC Modus bzw. RPC Modus mit komplexen Typen zu veröffentlichten, sollten Sie folgendermassen vorgehen:

- Erzeugen Sie die Proxy Methode mit dem Assistenten für Web Services auf Client-Ebene. Sie wird aufgerufen, wie folgt: *$XMLresultBlob:=$DOCproxy_Method($XMLparamBlob)*

- Machen Sie sich über einen Online-Test mit dem Inhalt von SOAP Anfragen an den Web Service vertraut, z.B. *http://soapclient.com/soaptest.html*. Dieses Tool dient zum Generieren von HTML Test Formularen, die auf dem WSDL von Web Service basieren.
- Kopieren Sie den erzeugten XML Inhalt aus dem ersten „child" Element von <body>.
- Schreiben Sie eine Methode, um reelle Werte von Parametern in XML Code zu schreiben; dieser Code muss dann in das BLOB *$XMLparamBlob* gesetzt werden.
- Zum Analysieren (Parsen) der Antwort können Sie wieder einen Online-Test einsetzen oder WSDL verwenden, welches die zurückgegebenen Elemente spezifiziert.

Der Parameter * dient zum Optimieren von Aufrufen. Ist dieser Parameter übergeben, schließt der Befehl nach seiner Ausführung nicht die von dem Prozess verwendete Verbindung. Der nächste Aufruf von **WEB SERVICE CALL** verwendet wieder dieselbe Verbindung, wenn der Parameter * übergeben ist, usw. Um die Verbindung zu schließen, führen Sie den Befehl ohne den Parameter * aus. Diese Vorgehensweise beschleunigt signifikant die Bearbeitung aufeinanderfolgender Aufrufe mehrerer Web Dienste auf demselben Server, insbesondere in einer WAN Konfiguration (z.B. über Internet). Beachten Sie, dass diese Vorgehensweise von der "keep-alive" Einstellung des Web Server abhängt. Diese Einstellung weist in der Regel pro Verbindung eine max. Anzahl für Anfragen zu und kann auch Anfragen abweisen. Ist die max. Anzahl Anfragen für **WEB SERVICE CALL** in derselben Verbindung erreicht oder sind keine keep-alive Verbindungen erlaubt, erstellt 4D für jede Anfrage eine neue Verbindung.

Systemvariablen und Mengen

Wurde die Anfrage korrekt geroutet und hat sie der Web Service akzeptiert, wird die Systemvariable OK auf 1 gesetzt. Sonst erhält sie 0 (Null) und ein Fehler wird zurückgegeben

WEB SERVICE Get info

WEB SERVICE Get info (InfoTyp) -> Funktionsergebnis

Parameter	Typ		Beschreibung
InfoTyp	Lange Ganzzahl	→	Zu findende Information
Funktionsergebnis	String	↩	Information über den letzten SOAP Fehler

Beschreibung

Die Funktion **WEB SERVICE Get info** gibt Informationen über jeden beim Ausführen der letzten SOAP Anfrage erzeugten Fehler zurück, die an einen remote Web Service gesendet wurde. Die Funktion sollte generell innerhalb einer Fehlerverwaltungsmethode aufgerufen werden, die mit dem Befehl **ON ERR CALL** installiert wurde.

Der Parameter *InfoTyp* gibt den gewünschten Informationstyp an. Sie müssen eine vordefinierte Konstante unter dem Thema **Web Services (Client)** verwenden.

Konstante	Typ	Wert	Kommentar
Web Service detailed message	Lange Ganzzahl	1	Detaillierte Meldung mit Fehlerbeschreibung. Die Art der Meldung unterscheidet sich je nach der Art des Hauptfehlers. - Ist der Hauptfehler = 9910 (Soap Fehler), wird der Grund des SOAP Fehlers zurückgegeben (z.B.: "the remote method does not exist"). - Ist der Hauptfehler = 9911 (Parser Fehler), wird die Stelle des Fehlers im XML Dokument zurückgegeben. - Ist der Hauptfehler = 9912 (HTTP Fehler): - Liegt der HTTP Fehler im Bereich [300-400] (Probleme mit der Stelle des angefragten Dokuments), wird die neue Stelle der angefragten URL zurückgegeben. - bei allen anderen HTTP Fehlercodes wird <body> zurückgegeben. - Ist der Hauptfehler = 9913 (Netzwerk Fehler), wird der Grund des Netzwerkfehlers zurückgegeben (z.B.: "ServerAddress: DNS lookup failure") - Ist der Hauptfehler = 9914 (interner Fehler), wird der Grund des internen Fehlers zurückgegeben.
Web Service error code	Lange Ganzzahl	0	Hauptfehler Code (von 4D definiert). Dieser Code wird auch in der Systemvariable Error zurückgegeben. Hier die Liste möglicher Fehler: 9910: Soap Fehler (siehe auch Web Service Fault Actor) 9911: Parser Fehler 9912: HTTP Fehler (siehe auch Web Service HTTP Error code) 9913: Netzwerkfehler 9914: Interner Fehler
Web Service fault actor	Lange Ganzzahl	3	Grund des Fehlers (vom SOAP Protokoll zurückgegeben — zu verwenden im Fall von Hauptfehler 9910). - Version passt nicht - Verstehen ist erforderlich (der Server konnte einen als zwingend definierten Parameter nicht interpretieren) - Sender Fehler - Receiver Fehler - Codierung unbekannt
Web Service HTTP error code	Lange Ganzzahl	2	HTTP Fehler-Code (bei Hauptfehler 9912 verwenden)

Ist keine Information verfügbar, wird ein leerer String zurückgegeben, insbesondere wenn die letzte SOAP Anfrage keine Fehler erzeugt hat.

WEB SERVICE GET RESULT

WEB SERVICE GET RESULT (WertZurück {; NameZurück {; *}})

Parameter	Typ		Beschreibung
WertZurück	Variable	⇐	Von Web Service zurückgegebener Wert
NameZurück	String	⇒	Name des zu findenden Parameters
*		⇒	Speicher frei machen

Beschreibung

Der Befehl **WEB SERVICE GET RESULT** übernimmt einen Wert, den der Web Service als Ergebnis der abgewickelten Anfrage zurücksendet.

Hinweis: Dieser Befehl darf nur nach dem Befehl **WEB SERVICE CALL** verwendet werden.

Der Parameter *WertZurück* empfängt den vom Web Service zurückgesendeten Wert. Übergeben Sie in diesem Parameter eine 4D Variable. Das ist in der Regel $0 und entspricht dem Wert, welchen die Proxy Methode zurückgibt. Sie können auch dazwischenliegende Variablen verwenden, es kann jedoch nur eine Prozessvariable sein.

Hinweis: Jede verwendete 4D Variable oder Array muss zuvor über einen Befehl im Kapitel **Compiler** bzw. **Arrays** deklariert werden.

Der optionale Parameter *NameZurück* gibt den Namen des zu findenden Parameters an. Da jedoch die meisten Web Services nur einen einzelnen Wert zurückgeben, fällt dieser Parameter in der Regel weg.

Der optionale Parameter * signalisiert dem Programm, den Speicher zum Abwickeln der Anfrage freizumachen. Sie müssen diesen Parameter übergeben, nachdem der zuletzt vom Web Service gesendete Wert gefunden wurde.

Beispiel

Nehmen wir an, ein Web Service gibt die aktuelle Zeit in einer beliebigen Stadt auf der Welt an. Der Web Service empfängt den Namen der Stadt und die Länderkennzahl und sendet im Gegenzug die entsprechende Zeit. Die aufrufende Proxy-Methode könnte folgendermaßen lauten:

```
C_TEXT($1)
C_TEXT($2)
C_TIME($0)

WEB SERVICE SET PARAMETER("city";$1)
WEB SERVICE SET PARAMETER("country_code";$2)

WEB SERVICE CALL("http://www.citiesoftheworld.com/WS";"WSTime#City_time";"City_time";
"http://www.citiesoftheworld.com/namespace/default")

If(OK=1)
    WEB SERVICE GET RESULT($0;"return";*)
End if
```

WEB SERVICE SET OPTION

WEB SERVICE SET OPTION (Option ; Wert)

Parameter	Typ		Beschreibung
Option	Lange Ganzzahl	➡	Code der zu setzenden Option
Wert	Lange Ganzzahl, Text	➡	Wert der Option

Vorbemerkung

Dieser Befehl ist optional. Er richtet sich an fortgeschrittene Nutzer von Web Services.

Beschreibung

Der Befehl **WEB SERVICE SET OPTION** setzt verschiedene Optionen, die während der nächsten SOAP Anfrage verwendet werden, die über den Befehl **WEB SERVICE CALL** ausgelöst werden.

Sie können diesen Befehl so oft aufrufen, wie es Optionen gibt, um vor **WEB SERVICE CALL** zu setzen.

In *Option* übergeben Sie die Nummer der Option, in *Wert* ihren neuen Wert. Für den Parameter *Option* können Sie auch eine vordefinierte Konstante unter dem Thema **Web Services (Client)** verwenden:

Konstante	Typ	Wert	Kommentar
Web Service display auth dialog	Lange Ganzzahl	4	*Wert* = 0 (Dialogfenster nicht anzeigen) oder 1 (Dialogfenster anzeigen) Diese Option verwaltet die Anzeige des Authentifizierungsdialogs beim Ausführen des Befehls **WEB SERVICE CALL**. Dieser Befehl zeigt standardmäßig nie den Dialog an; normalerweise müssen Sie dafür den Befehl **WEB SERVICE AUTHENTICATE** verwenden. Verwenden Sie diese Option, wenn der Authentifizierungsdialog erscheinen soll, damit Benutzer ihre Identifier eintragen können: Übergeben Sie 1 in Wert, um das Dialogfenster anzuzeigen, sonst 0. Das Dialogfenster erscheint nur, wenn der Web Service eine Authentifizierung benötigt.
Web Service HTTP compression	Lange Ganzzahl	6	*value* = Web Service Compression Damit lässt sich ein interner Kompressionsmechanismus für SOAP Anfragen aktivieren, um den Austausch zwischen 4D Anwendungen zu beschleunigen. Führen Sie die Anweisung **WEB SERVICE SET OPTION**(Web Service HTTP Compression; Web Service Compression) auf dem 4D Client des Web Service aus, werden die Daten der nächsten vom Client gesendeten SOAP Anfrage vor dem Senden an den 4D SOAP Server über einen standard HTTP Mechanismus komprimiert ("gzip" oder "deflate", je nach Inhalt der Anfrage). Der Server entkomprimiert und analysiert die Anfrage und antwortet dann automatisch über denselben Mechanismus. Das gilt nur für die Anfrage, die auf den Aufruf des Befehls **WEB SERVICE SET OPTION** folgt. Sie müssen deshalb diesen Befehl vor jedem Komprimieren erneut aufrufen. 4D komprimiert standardmäßig keine Web Service HTTP Anfragen. **Hinweis:** Dieser Mechanismus lässt sich nicht für Anfragen verwenden, die an einen 4D SOAP Server älter als Version 11.3 gesendet werden. Zur weiteren Optimierung dieser Funktionsweise können Sie zusätzliche Optionen wie Durchfluß- und Komprimierungsrate konfigurieren. Sie sind im Befehl **SET DATABASE PARAMETER** verfügbar.
Web Service HTTP timeout	Lange Ganzzahl	1	*Wert* = Timeout des Client Teils, angegeben in Sekunden. Timeout ist die Zeitspanne, die der Web Service Client wartet, wenn der Server nicht antwortet. Ist sie abgelaufen, schließt der Client die Sitzung und die Anfrage geht verloren. Das Timeout beträgt standardmäßig 180 Sekunden. Es kann für spezifische Fälle geändert werden (Netzwerkstatus, Web Service Eigenheiten, etc.).
Web Service reset auth settings	Lange Ganzzahl	5	*Wert* = 0 (Information nicht entfernen) oder 1 (Information entfernen) Damit können Sie für 4D angeben, ob die Information zur Authentifizierung des Benutzers (Benutzername, Kennwort, Methode, etc.) gespeichert werden soll, um sie in Folge erneut zu verwenden. Diese Information wird standardmäßig nach jeder Ausführung des Befehls **WEB SERVICE CALL** entfernt. Übergeben Sie 0 in *Wert*, um die Information zu speichern, 1 um sie zu entfernen. Beachten Sie, dass bei 0 die Information während der Sitzung beibehalten, aber nicht gespeichert wird.
Web Service SOAP header	Lange Ganzzahl	2	*Wert* = XML Root Element Referenz, die als Header in der SOAP Anfrage eingegeben werden soll. Damit können Sie einen Header in einer SOAP Anfrage eingeben, die über den Befehl **WEB SERVICE CALL** erstellt wurde. SOAP Anfragen enthalten standardmäßig keinen spezifischen Header. Bestimmte Web Services benötigen jedoch einen Header, z.B. zur Verwaltung von Parametern zur Identifizierung.
Web Service SOAP version	Lange Ganzzahl	3	*Wert* = Web Service SOAP 1 1 oder Web Service SOAP 1 2 Über diese Option können Sie die Version des in der Anfrage verwendeten SOAP Protokolls angeben. Übergeben Sie Web Service SOAP 1 1 für Version 1.1, Web Service SOAP 1 2 für Version 1.2.

Es spielt keine Rolle, in welcher Reihenfolge die Optionen aufgerufen werden. Wird dieselbe Option mehrmals gesetzt, wird nur der Wert des letzten Aufrufs berücksichtigt.

Beispiel 1

Die folgende Anweisung fügt in die SOAP Anfrage einen eigenen Header ein:

```
  ` Eine XML Referenz erstellen
C_TEXT(vRootRef;vElemRef)
vRootRef:=DOM Create XML Ref("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vElemRef:=DOM Create XML element(vRootRef;vxPath)
  `SOAP header mit Referenz ändern
WEB SERVICE SET OPTION(Web Service SOAP header;vElemRef)
```

Beispiel 2

Das folgende Beispiel verwendet das SOAP Protokoll Version 1.2:

```
WEB SERVICE SET OPTION(Web Service SOAP version;Web Service SOAP_1_2)
```

WEB SERVICE SET PARAMETER

WEB SERVICE SET PARAMETER (Auswahlname ; Wert {; SoapTyp})

Parameter	Typ		Beschreibung
Auswahlname	String	➡	Name des Parameters zum Einsetzen in die SOAP Anfrage
Wert	Variable	➡	4D Variable mit dem Wert des Parameters
SoapTyp	String	➡	SOAP Typ des Parameters

Beschreibung

Der Befehl **WEB SERVICE SET PARAMETER** definiert einen Parameter für eine SOAP Anfrage auf Client-Ebene. Rufen Sie diesen Befehl für jeden Parameter in der Anfrage auf (wie oft der Befehl aufgerufen wird, richtet sich nach der Anzahl der Parameter).

In *Prozessname* übergeben Sie den Namen des Parameters, wie er in der SOAP Anfrage erscheinen soll.

In *Wert* übergeben Sie die 4D Variable mit dem Wert dieses Parameters. Diese Variable ist für Proxy-Methoden generell $1, $2, $3, etc. gemäß dem 4D Parameter, der beim Aufrufen der Methode übergeben wird. Sie können aber auch dazwischengeschaltete Variablen verwenden.

Jede verwendete 4D Variable bzw. Array muss zuvor mit einem Befehl aus dem Kapitel **Compiler** bzw. **Arrays** deklariert werden.

4D bestimmt automatisch den am ehesten geeigneten SOAP Typ für den Parameter *Prozessname* gemäß dem Inhalt von *Wert*. Die Angabe des Typs wird in die Anfrage aufgenommen.

Sie können aber auch die Definition des SOAP Typs für einen Parameter vorschreiben. In diesem Fall können Sie einen Parameter *SoapTyp* aus der nachfolgenden Liste übergeben (primäre Datentypen):

SoapTyp	Beschreibung
String	String
int	Lange Ganzzahl
boolean	Boolean
float	32-bit Zahl
decimal	Zahl mit Dezimal
double	64-bit Zahl
duration	Zeit in Jahren, Monaten, Tagen, Stunden, Minuten, Sekunden, z.B.: P1Y2M3DT10H30M
datetime	Datum und Zeit in ISO8601 Format, z.B. 2003-05-31T13:20:00
time	Zeit, z.B. l 13:20:00
date	Datum, z.B. 2003-05-31
gyearmonth	Jahr und Monat (Gregorianische Zeitrechnung), z.B. 2003-05
gyear	Jahr (Gregorianische Zeitrechnung), z.B. 2003
gmonthday	Monat und Tag (Gregorianische Zeitrechnung), z.B. --05-31
gday	Tag (Gregorianische Zeitrechnung), z.B. ---31
gmonth	Monat (Gregorianische Zeitrechnung), z.B. --10--
hexbinary	Wert in hexadezimal
base64binary	BLOB
anyuri	Uniform Resource Identifier (URI), z.B. http://www.company.com/page
qname	Qualifizierter XML Name (Namensraum und lokaler Teil)
notation	Notation Attribute

Hinweis: Weitere Informationen zu XML Datentypen finden Sie Im Internet unter http://www.w3.org/TR/xmlschema-2/

Beispiel

Dieses Beispiel definiert zwei Parameter:

```
C_TEXT($1)
C_TEXT($2)
WEB SERVICE SET PARAMETER("Stadt";$1)
WEB SERVICE SET PARAMETER("Land";$2)
```

Web Services (Server)

- Befehle für Web Services (Server)
- SOAP DECLARATION
- SOAP Get info
- SOAP request
- SOAP SEND FAULT

Befehle für Web Services (Server)

4D unterstützt ab Version 2003 “Web Services”, d.h. Sie können Web Dienste aus Ihrer Datenbank veröffentlichen (Server Ebene) bzw. einsetzen (Client Ebene).

Dieser Abschnitt beschreibt die Befehle zum Veröffentlichen von Web Services (Server Ebene) in 4D. Weitere Informationen zum Einrichten von Web Services in 4D finden Sie im Abschnitt **Befehle für Web Services (Client)**.

Hinweis: Die Begriffe “SOAP” und “Web Service” wurden zur Unterscheidung zwischen Befehls- und Konstantennamen auf Server-Ebene bzw. Client-Ebene eingeführt. Beide Konzepte beziehen sich auf dieselbe Technologie.

SOAP DECLARATION

SOAP DECLARATION (Variablenname ; Typ ; Input_Output {; Alias})

Parameter	Typ		Beschreibung
Variablenname	Variable	➡	Variable für ein bzw. ausgehendes SOAP Argument
Typ	Lange Ganzzahl	➡	4D Typ, auf den das Argument zeigt
Input_Output	Lange Ganzzahl	➡	1 = SOAP Input, 2 = SOAP Output
Alias	String	➡	Name, der während SOAP Austausch für dieses Argument veröffentlicht wird

Beschreibung

Der Befehl **SOAP DECLARATION** deklariert explizit den SOAP Typ der Parameter in einer 4D Methode, die als Web Service veröffentlicht wird.

Wird eine Methode als ein Web Service veröffentlicht, werden die Standardparameter $0, $1... $n verwendet, um die Parameter des Web Service für die Außenwelt zu beschreiben, insbesondere in der WSDL Datei. Für das SOAP Protokoll müssen Parameter explizit benannt werden; 4D verwendet standardmäßig die Namen "FourD_arg0, FourD_arg1 ... FourD_argn".

Diese Standardoperation kann jedoch aus folgenden Gründen problematisch sein:

- Es ist nicht möglich, $0 oder $1, $2, etc. als Array zu deklarieren. Dafür sind Zeiger erforderlich; in diesem Fall muss jedoch zum Erzeugen der WSDL Datei die Art der Werte bekannt sein.
- Es kann hilfreich oder notwendig sein, die Parameternamen anzupassen (eingehend und ausgehend).
- Sie wollen Parameter wie XML Strukturen und DOM Referenzen verwenden.
- Es kann auch notwendig sein, Werte größer als 32 000 Zeichen zurückzugeben (Limit für Textargumente in Datenbanken, die im Nicht-Unicode Modus laufen).
- Schließlich macht es diese Operation unmöglich, in $0 mehr als einen Wert pro RPC Aufruf zurückzugeben.

Mit dem Befehl **SOAP DECLARATION** unterliegen Sie nicht diesen Einschränkungen. Sie können diesen Befehl für jeden ein- und ausgehenden Parameter ausführen und ihm einen Namen und einen Typ zuweisen.

Hinweis: Auch bei Verwendung dieses Befehls müssen Sie 4D Variablen und Arrays in der Methode "Compiler_Web" immer mit Befehlen aus dem Kapitel **Compiler** deklarieren.

In *Variablenname* übergeben Sie die 4D Variable, auf die im RPC Aufruf verwiesen wird.

Achtung: Sie können auf Prozessvariablen oder Parameter in 4D Methoden ($0 to $n) verweisen. Lokale und Interprozessvariablen können Sie nicht verwenden.

Da SOAP nur Texte vom Typ Argument verwenden kann, sind Antworten vom SOAP Server in Datenbanken, die im Nicht-Unicode Modus laufen, auf 32 KB begrenzt. Über BLOBs können Sie jedoch SOAP Argumente größer als 32 KB zurückgeben. Dazu müssen Sie lediglich vor Aufrufen von **SOAP DECLARATION** die Argumente als BLOBs deklarieren (siehe Beispiel 4).

Hinweis: Abonnieren Sie auf Client-Seite diese Art von Web Service mit 4D, generiert der Web Services Assistent eine Variable vom Typ Text. Um sie zu verwenden, müssen Sie lediglich die zurückgegebene Variable in der Proxy-Methode als BLOB typisieren.

In *Typ* übergeben Sie den entsprechenden 4D Typ. Sie können die meisten Arten von 4D Variablen und Arrays verwenden. Sie können die vordefinierten Konstanten unter dem Thema **Feld und Variablentypen** einsetzen:

Konstante	Typ	Wert
Boolean array	Lange Ganzzahl	22
Date array	Lange Ganzzahl	17
Integer array	Lange Ganzzahl	15
Is BLOB	Lange Ganzzahl	30
Is Boolean	Lange Ganzzahl	6
Is date	Lange Ganzzahl	4
Is integer	Lange Ganzzahl	8
Is longint	Lange Ganzzahl	9
Is real	Lange Ganzzahl	1
Is string var	Lange Ganzzahl	24
Is text	Lange Ganzzahl	2
Is time	Lange Ganzzahl	11
LongInt array	Lange Ganzzahl	16
Real array	Lange Ganzzahl	14
String array	Lange Ganzzahl	21
Text array	Lange Ganzzahl	18

sowie für XML Typen Konstanten unter dem Thema **Web Services (Server)**:

Konstante	Typ	Wert
Is DOM reference	Lange Ganzzahl	37
Is XML	Lange Ganzzahl	36

In *Input_Output* übergeben Sie einen Wert, der anzeigt, ob der bearbeitete Parameter "eingehend" (z.B. Wert, den eine Methode empfängt) oder "ausgehend" (z.B. Wert, den eine Methode zurückgibt) ist. Sie können die vordefinierten Konstanten für SOAP unter dem Thema **Web Services (Server)** einsetzen:

Konstante	Typ	Wert
SOAP input	Lange Ganzzahl	1
SOAP output	Lange Ganzzahl	2

XML Typen verwenden

Über die Konstanten Is XML und Is DOM reference können Sie Variablen vom Typ "XML Struktur" und "DOM Referenz" deklarieren, sowohl eingehend wie ausgehend. Bei Parametern dieses Typs erfolgt keine Bearbeitung oder Codierung. Die Daten werden unverändert gesendet (siehe Beispiel 5).

- Ausgehende Parameter:
  - Is XML gibt an, dass der Parameter eine XML Struktur enthält,
  - Is DOM reference gibt an, dass der Parameter die DOM Referenz einer XML Struktur enthält. In diesem Fall ist das Einfügen der XML Struktur in die SOAP Meldung dasselbe wie Ausführen des Befehls **DOM EXPORT TO VAR**.

Hinweis: Werden DOM Referenzen als ausgehende Parameter verwendet, sollten Sie globale Referenzen einsetzen, die z.B. beim Startup erstellt werden und sich schließen, wenn das Programm geschlossen wird. Denn eine DOM Referenz, die im Web Service selbst angelegt wird, lässt sich nicht mit **DOM CLOSE XML** schließen, da sonst der Web Service nichts mehr zurückgeben würde. Das mehrfache Aufrufen eines Web Dienstes führt deshalb auch dazu, dass mehrere nicht-geschlossene DOM Referenzen erstellt werden, was zu einer Speicherüberlastung führen kann.

- Eingehende Parameter:
  - Is XML gibt an, dass der Parameter ein vom SOAP Client gesendetes XML Argument empfangen soll.
  - Is DOM reference gibt an, dass der Parameter die DOM Referenz einer XML Struktur empfangen soll, die dem vom SOAP Client gesendeten XML Argument entspricht.

- Veränderung der WSDL Datei: 4D deklariert diese XML Strukturen in der WSDL als "jeder Typ" (unbestimmt). Wollen Sie eine XML Struktur genauer deklarieren, müssen Sie die WSDL Datei sichern und das gewünschte Datenschema im Abschnitt <types> der WSDL manuell hinzufügen.

COMPILER_WEB Methode

Eingehende SOAP Argumente, die über 4D Variablen, also nicht über Argumente von 4D Methoden aufgerufen werden, müssen Sie zuerst über die Projektmethode "Compiler_Web" deklarieren. Setzen Sie in Methoden für Web Services Prozessvariablen ein, müssen diese deklariert sein, bevor die Methode aufgerufen wird. Die Projektmethode "Compiler_Web" wird – sofern vorhanden – für jede akzeptierte SOAP Anfrage aufgerufen. Die Methode "Compiler_Web" ist standardmäßig nicht vorhanden. Sie müssen diese explizit anlegen.

Der 4D Web Server ruft "Compiler_Web" auch auf, wenn er "konventionelle" Web Anfragen vom Typ POST erhält. Weitere Informationen dazu finden Sie im Abschnitt **URLs und Form Actions**.

In *Alias* übergeben Sie den Argumentnamen, wie er in WSDL und SOAP Austauschaktionen erscheinen muss.

Achtung: Dieser Name muss im RPC Aufruf einmalig sein (Eingabe- und Ausgabeparameter zusammengenommen), ansonsten berücksichtigt 4D nur die letzte Deklaration.

Hinweis: Die Namen dürfen in Anlehnung an die XML Standards für Marker-Namen WEDER Leerzeichen NOCH Zeichen aus dem erweiterten ASCII-Satz enthalten. Nur Zeichen des Latin-Satzes sind zulässig ([A-Za-z0-9._] | '-')*.

Übergeben Sie keinen Parameter in *Alias*, verwendet 4D standardmäßig den Namen der Variablen oder *FourD_argN* für die Argumente der 4D Methode ($0 to $n).

Hinweis: Der Befehl **SOAP DECLARATION** muss innerhalb der Methode liegen, die als Web Service veröffentlicht wird. Er lässt sich nicht über eine andere Methode aufrufen.

Beispiel 1

Dieses Beispiel gibt den Parameternamen an:

```
` In der Methode COMPILER_WEB
C_LONGINT($1)

` In der Methode Web Service
` Beim Erzeugen der WSDL Datei und SOAP Aufrufe wird das Wort
` PLZ anstelle von fourD_arg1 verwendet.
C_LONGINT($1)
SOAP DECLARATION($1;ls longint;SOAP input;"PLZ")
```

Beispiel 2

Dieses Beispiel findet ein Array mit Postleitzahlen in Form von langen Ganzzahlen:

```
` Setzen Sie in die Methode "Compiler_Web"
ARRAY LONGINT(codes;0)

` In der Service Methode
SOAP DECLARATION(codes;LongInt array;SOAP input;"in_codes")
```

Beispiel 3

Dieses Beispiel verweist auf zwei zurückgegebene Werte ohne Angabe eines Argumentnamens:

```
SOAP DECLARATION(ret1;Is longint;SOAP output)
SOAP DECLARATION(ret2;Is longint;SOAP output)
```

Beispiel 4

Dieses Beispiel ermöglicht dem 4D SOAP Server, ein Argument größer als 32 KB in Datenbanken im Nicht-Unicode Modus zurückzugeben:

```
C_BLOB($0)
SOAP DECLARATION($0;Is text;SOAP output)
```

Beachten Sie, dass der Typ Is Text und nicht Is BLOB ist. So lässt sich das Argument korrekt bearbeiten.

Beispiel 5

Nachfolgendes Beispiel zeigt die Ergebnisse der verschiedenen Deklarationstypen:

```
ALL RECORDS([Contact])
  `XML Struktur aus der Auswahl Contact erstellen
  `und die XML in einem BLOB speichern
C_BLOB(ws_vx_xmlBlob)
getContactsXML(->ws_vx_xmlBlob)
  `Die XML Struktur in einer Textvariablen wiederfinden
C_TEXT(ws_vt_xml)
ws_vt_xml:=BLOB to text(ws_vx_xmlBlob;UTF8 text without length)
  `Eine DOM Referenz auf die XML Struktur wiederfinden
C_TEXT(ws_vt_xmlRef)
ws_vt_xmlRef:=DOM Parse XML variable(ws_vt_xml)
  `Die verschiedenen Deklarationen testen
SOAP DECLARATION(ws_vx_xmlBlob;Is BLOB;SOAP output;"contactListsX")
  `4D konvertiert XML in Base64
SOAP DECLARATION(ws_vt_xml;Is text;SOAP output;"contactListsText")
  `Die XML wird von 4D in Text konvertiert (< > become entities)
SOAP DECLARATION(ws_vt_xml;Is XML;SOAP output;"xmlContacts")
  `Die XML wird als XML Text übergeben
SOAP DECLARATION(ws_vx_xmlBlob;Is XML;SOAP output;"blobContacts")
  `Die XML wird als ein XML BLOB übergeben
SOAP DECLARATION(ws_vt_xmlRef;Is DOM reference;SOAP output;"contactByRef")
  `Die XML wird als Referenz übergeben
```

SOAP Get info

SOAP Get info (InfoNum) -> Funktionsergebnis

Parameter	Typ		Beschreibung
InfoNum	Lange Ganzzahl	→	Nummer des zu erhaltenden SOAP Infotyps
Funktionsergebnis	String	↩	SOAP Information

Beschreibung

Die Funktion **SOAP Get info** findet die verschiedenen Informationstypen einer SOAP Anfrage in Form von Zeichenketten.

Beim Abarbeiten einer SOAP Anfrage sind Zusatzinformationen über die Anfrage hilfreich. Sie können die Funktion z.B. in der **Datenbankmethode On Web Authentication** verwenden, um den Namen der angefragten Web Service Methode herauszufinden.

In *InfoNum* übergeben Sie die Nummer für die Art der gewünschten SOAP Information. Sie können die vordefinierten Konstanten unter dem Thema **Web Services (Server)** einsetzen:

Konstante	Typ	Wert	Kommentar
SOAP method name	Lange Ganzzahl	1	Name der auszuführenden Web Service Methode
SOAP service name	Lange Ganzzahl	2	Name des Web Service, zu dem die Methode gehört.

Hinweis: Sie können für die an 4D gesendeten Web Service Anfragen eine max. Größe festlegen. Das erstellen Sie mit dem Befehl **SET DATABASE PARAMETER**.

SOAP request

SOAP request -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	Boolean		Wahr, wenn Anfrage SOAP ist; sonst Falsch

Beschreibung

Die Funktion **SOAP request** gibt Wahr zurück, wenn der ausgeführte Code Teil einer SOAP Anfrage ist.
Sie können diese Funktion aus Sicherheitsgründen in der **Datenbankmethode On Web Authentication** verwenden, um die Art der empfangenen Anfragen zu bestimmen.

SOAP SEND FAULT

SOAP SEND FAULT (Fehlertyp ; Beschreibung)

Parameter	Typ		Beschreibung
Fehlertyp	Lange Ganzzahl	➡	1 = Client Fehler, 2 = Server Fehler
Beschreibung	String	➡	Fehlerbeschreibung zum Senden an SOAP Cients

Beschreibung

Der Befehl **SOAP SEND FAULT** gibt einen Fehler an einen SOAP Client zurück und gibt den Ursprung des Fehlers an: Client oder Server. Mit diesem Befehl können Sie dem Client einen Fehler melden, ohne ein Ergebnis liefern zu müssen.

Auf Client-Ebene tritt z.B. ein Fehler auf, wenn Sie einen "Quadratwurzel" Web Service veröffentlichen und ein Client eine Anfrage mit einem negativen Wert sendet. Sie können dann dem Client über diesen Befehl mitteilen, dass ein positiver Wert erforderlich ist.

Auf Server-Ebene kann ein Fehler auftreten, wenn während Ausführen einer Methode ein Speicherproblem auftritt.

In *Fehlertyp* übergeben Sie den Ursprung des Fehlers. Sie können die vordefinierten Konstanten unter dem Thema **Web Services (Server)** verwenden:

Konstante	Typ	Wert
SOAP client fault	Lange Ganzzahl	1
SOAP server fault	Lange Ganzzahl	2

In *Beschreibung* übergeben Sie die Fehlerbeschreibung. Ist die Client Implementation konform, kann der Fehler bearbeitet werden.

Beispiel

Für das obige Beispiel zu "Quadratwurzel" Web Service können Sie den Befehl zum Bearbeiten von Anfragen mit negativen Nummern folgendermaßen einsetzen:

```
SEND SOAP FAULT(SOAP client fault;"Positiver Wert erforderlich")
```

Werkzeuge

- 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 ({codierterText ;} BLOB)

Parameter	Typ		Beschreibung
codierterText	Text	➡	Text mit BLOB codiert im Format Base64
BLOB	BLOB	➡	BLOB codiert im Format Base64
		⬅	Decodiertes BLOB

Beschreibung

Der Befehl **BASE64 DECODE** decodiert das BLOB codiert in Base64, übergeben im Parameter *codierterText* oder *Blob*.

Übergeben Sie den Parameter *codierterText*, decodiert der Befehl seinen Inhalt und gibt ihn im Parameter *Blob* zurück. Das BLOB muss im Format Base64 codiert sein. In diesem Fall ignoriert der Befehl den ursprünglichen Inhalt des Parameters *Blob*.

Lassen Sie den Parameter *codierterText* weg, ändert der Befehl direkt das im Parameter *Blob* übergebene BLOB.

BASE64 DECODE überprüft nicht den Inhalt von *codierterText* oder *Blob*. Sie müssen selbst prüfen, ob die übergebenen Daten tatsächlich im Format Base64 codiert sind, sonst wird ein falsches Ergebnis zurückgegeben.

Beispiel

Dieses Beispiel zeigt, wie Sie ein Bild über ein BLOB übertragen können.

```
C_BLOB($blobSource)
C_PICTURE($mypicture)
$mypicture:=[people]photo
PICTURE TO BLOB($mypicture;$blobSource;"JPC")
C_TEXT($base64Text)
BASE64 ENCODE($blobSource;$base64Text)  //Text codieren
  // binary ist jetzt als Zeichenkette in $base64Text verfügbar

C_TEXT($base64Text)
C_BLOB($blobTarget)
BASE64 DECODE($base64Text;$blobTarget)  //Text decodieren
  // in base 64 codiertes binary ist jetzt als BLOB in $blobTarget verfügbar.
```

BASE64 ENCODE

BASE64 ENCODE (BLOB {; codierterText})

Parameter	Typ		Beschreibung
BLOB	BLOB	➡	BLOB für Codierung in Format Base64
		⬅	BLOB codiert im Format Base64
codierterText	Text	⬅	Ergebnis des BLOB, codiert im Format Base64

Beschreibung

Der Befehl **BASE64 ENCODE** codiert das BLOB, definiert in *BLOB* im Format Base64. Er ändert direkt das im Parameter *BLOB* übergebene BLOB.

Übergeben Sie den Parameter *codierterText*, empfängt er den codierten Inhalt des Blob als Text am Ende der Befehlsausführung. In diesem Fall verändert der Befehl den Parameter *BLOB* selbst nicht.
Lassen Sie den Parameter *codierterText* weg, funktioniert der Befehl wie in bisherigen Versionen.

Die Codierung Base64 ändert 8-bit codierte Daten, so dass sie nicht mehr als 7 sinnvolle Bits beibehalten. Diese Codierung ist zum Beispiel erforderlich, um BLOBs mit XML zu verwalten.

Choose

Choose (Kriterium ; Wert {; Wert2 ; ... ; WertN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Kriterium	Boolean, Ganzzahl	➡	Zu testender Wert
Wert	Ausdruck	➡	Mögliche Werte
Funktionsergebnis	Ausdruck	⮌	Wert des Kriteriums

Beschreibung

Die Funktion **Choose** gibt einen der Werte im Parameter *Wert* zurück, je nach dem im *Kriterium* übergebenen Wert.
Im Parameter *Kriterium* können Sie einen Wert vom Typ Boolean oder Ganzzahl übergeben:

- Ist *Kriterium* ein Boolean Wert, gibt **Choose** *Wert1* zurück, wenn Boolean Wahr ist und *Wert2*, wenn Boolean Falsch ist. In diesem Fall erwartet die Funktion genau drei Parameter: *Kriterium*, *Wert1* und *Wert2*.
- Ist *Kriterium* eine Ganzzahl, gibt **Choose** den *Wert* zurück, dessen Position dem Parameter *Kriterium* entspricht. Beachten Sie, dass die Nummerierung der Werte mit 0 beginnt, d.h. die Position von *Wert1* ist 0 (Null). In diesem Fall erwartet die Funktion mindestens zwei Parameter: *Kriterium* und *Wert1*.

Die Funktion erlaubt in *Wert* alle Datentypen bis auf Bilder, Zeiger, BLOBS und Arrays. Sie müssen jedoch sicherstellen, dass alle übergebenen Werte vom gleichen Typ sind. 4D führt hier keine Überprüfung durch.
Entspricht kein Wert *Kriterium*, gibt **Choose** einen "Null" Wert zurück in Bezug auf den in *Wert* übergebenen Typ (z.B. 0 für Zahl, "" für String, etc).
Mit dieser Funktion können Sie Code mit „Case of" verkürzen, der mehrere Zeilen einnimmt (siehe 2. Beispiel). Sie ist auch hilfreich an Stellen, wo Formeln ausführbar sind, z.B. Sucheditor, Anwendung einer Formel, Schnellberichteditor, in Listbox berechnete Spalte, usw.

Beispiel 1

Hier ein Beispiel für eine typische Verwendung mit dem Typ Booelan im Parameter *Kriterium*:

```
vTitle:=Choose([Person]Masculine;"Mr";"Ms")
```

Dieser Code entspricht exakt:

```
If([Person]Masculine)
   vTitle:="Mr"
Else
   vTitle:="Ms"
End if
```

Beispiel 2

Hier ein Beispiel für eine typische Verwendung mit dem Typ Zahl im Parameter *Kriterium*:

```
vStatus:=Choose([Person]Status;"Ledig";"Verheiratet";"Verwitwet";"Geschieden")
```

Dieser Code entspricht exakt:

```
Case of
   :([Person]Status=0)
      vStatus:="Ledig"
   :([Person]Status=1)
      vStatus:="Verheiratet"
   :([Person]Status=2)
      vStatus:="Verwitwet"
   :([Person]Status=3)
      vStatus:="Geschieden"
End case
```

Generate digest

Generate digest (Param ; Algorithmus) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Param	BLOB, Textvariable	➡	Blob oder Textvariable, deren Digest Schlüssel ermittelt werden soll
Algorithmus	Lange Ganzzahl	➡	Algorithmus des Rückgabeschlüssels: 0 = MD5 Digest, 1 = SHA1 Digest, 2 = 4D Digest, 3 = SHA-256 Digest, 4 = SHA-512 digest
Funktionsergebnis	Text	↩	Wert des Digest Schlüssels

Beschreibung

Die Funktion **Generate digest** gibt den Digest Schlüssel eines Feldes oder einer Variablen vom Typ BLOB oder Text nach Anwenden eines Verschlüsselungsalgorithmus zurück..

im Parameter *Param* übergeben Sie ein Feld oder eine Variable vom Typ BLOB oder Text. Die Funktion **Generate digest** gibt den Digest Schlüssel als String zurück.

Im Parameter *Algorithmus* übergeben Sie einen Wert für die entsprechende Hash Funktion. Verwenden Sie eine der Konstanten unter dem Thema :

Konstante	Typ	Wert	Kommentar
4D digest	Lange Ganzzahl	2	Interner Algorithmus von 4D. Dient zum Verschlüsseln von Benutzerkennwörtern. Dieser Algorithmus ist besonders im Rahmen der **Datenbankmethode On 4D Mobile Authentication** hilfreich, wenn Sie Ihre eigene Benutzerliste verwenden wollen.
MD5 digest	Lange Ganzzahl	0	*Message Digest 5* Algorithmus. Das ist eine Serie von 128 bits, zurückgegeben als String mit 32 hexadezimalen Zeichen.
SHA1 digest	Lange Ganzzahl	1	*Secure Hash 1* Algorithmus. Das ist eine Serie von 160 bits, zurückgegeben als String mit 40 hexadezimalen Zeichen.
SHA256 digest	Lange Ganzzahl	3	(SHA-2 Familie) SHA-256 ist eine Serie von 256 bits, zurückgegeben als String mit 64 hexadezimalen Zeichen.
SHA512 digest	Lange Ganzzahl	4	(SHA-2 Familie) SHA-512 ist eine Serie von 512 bits, zurückgegeben als String mit 128 hexadezimalen Zeichen.

Hinweis: Es wird nicht empfohlen, MD5 und SHA Algorithmen zum Verwalten von Kennwörtern verwenden. Dafür dienen die Funktionen **Generate password hash** und **Verify password hash**.

Der für dasselbe Objekt zurückgegebene Wert ist auf allen Plattformen gleich (Mac/Windows, 32 oder 64 bits). Die Berechnung wird in UTF-8 in Bezug auf den im Parameter übergebenen Text durchgeführt.

Hinweis: Verwenden Sie die Funktion mit einem leeren Text/BLOB, gibt er nicht leer zurück, sondern einen String Wert, wie z.B. d41d8cd98f00b204e9800998ecf8427e" für MD5.

Wird die Berechnung des Digest Schlüssels nicht korrekt ausgeführt, wird ein Fehler erzeugt, den Sie über den Befehl **ON ERR CALL** abfangen können und die Funktion gibt einen leeren String zurück.

Beispiel 1

Dieses Beispiel vergleicht zwei Bilder über den MD5 Algorithmus:

```
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("Diese beiden Bilder sind unterschiedlich.")
      Else
         ALERT("Diese beiden Bilder sind identisch.")
      End if
   End if
End if
```

Beispiel 2

Diese Beispiele zeigen, wie Sie den Digest Schlüssel eines Textes finden können:

```
$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"
```

Beispiel 3

Dieses Beispiel zur **Generate digest** akzeptiert nur den Benutzer "admin" mit dem Kennwort "123", der nicht mit einem 4D Benutzer übereinstimmt:

```
 // Datenbankmethode On REST Authentication
C_TEXT($1;$2)
C_BOOLEAN($0;$3)
 //$1: user
 //$2: password
 //$3: digest mode
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 (Kennwort {; Optionen}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Kennwort	String	→	Benutzerkennwort mit maximal 72 Zeichen
Optionen	Objekt	→	Ein Objekt mit Optionen
Funktionsergebnis	String	←	Gibt Kennwort-Hash zurück.

Beschreibung

Die Funktion **Generate password hash** gibt ein sicheres Kennwort-Hash zurück, generiert über einen kryptographischen Hash-Algorithmus.

Im Parameter *Kennwort* übergeben Sie einen Stringwert. **Generate password hash** gibt einen Hash-String für das Kennwort zurück. Mehrfaches Übergeben des gleichen Kennworts führt zu unterschiedlichen Hash-Strings.

Im Objekt *Optionen* übergeben Sie die Eigenschaften, die beim Generieren des Kennwort-Hash verwendet werden sollen. Es gibt folgende Eigenschaften:

Eigenschaft	Wertetyp	Beschreibung	Standardwert
algorithm	String	Algorithmustyp. Derzeit wird ausschließlich "bcrypt" unterstützt (Unterscheidung zwischen Groß- und Kleinschreibung).	bcrypt
cost	numerisch	Kostenfaktor/Geschwindigkeit. Die für bcrypt unterstützten Werte liegen zwischen 4 (schnell) und 31 (sehr langsam).	10

Hinweis: Ist ein Wert im Parameter *Optionen* ungültig, werden eine Fehlermeldung und ein leerer String zurückgegeben.

Fehlerverwaltung

Folgende Fehler können zurückgegeben werden. Über die Befehle **GET LAST ERROR STACK** und **ON ERR CALL** können Sie einen Fehler auswerten.

Nummer	Meldung
850	Password-Hash: Nicht unterstützter Algorithmus
852	Password-Hash: "bcrypt cost parameter" nicht verfügbar, bitte geben Sie einen Wert zwischen 4 und 31 an.

Über bcrypt

"bcrypt" ist eine Hash-Funktion für Kennwörter, die auf der Verschlüsselung Blowfish basiert. Sie integriert Salz (Salt), um gegen Attacken über eine Rainbow Table zu schützen und kann außerdem die Anzahl der Iterationen erhöhen, um sie gezielt langsamer zu machen. Auf diese Weise bleibt bcrypt selbst bei steigender Rechenleistung gegenüber brute-force Angriffen resistent, da sie bewusst langsam ist, was zuviel Zeit beansprucht und zu teuer wird.

Beispiel

Dieses Beispiel generiert ein Kennwort-Hash über bcrypt mit dem Kostenfaktor 4 (schnell).

```
C_TEXT($password)
C_TEXT($hash)
C_OBJECT($options)

$options:=New object("algorithm";"bcrypt";"cost";4)
$password:=Request("Bitte geben Sie Ihr Kennwort ein")

$hash:=Generate password hash($password;$options)
[Users]hash:=$hash
SAVE RECORD([Users])
```

Hinweis: Mehrfaches Übergeben des gleichen Kennworts führt zu unterschiedlichen Hash-Strings. Das ist ein Standardverhalten für Algorithmen wie bcrypt, da die beste Praxis ist, für jeden Hash ein neues zufälliges Salz zu erstellen. Unter der Funktion **Verify password hash** sehen Sie ein Beispiel zum Prüfen der Kennwörter.

Generate UUID

Generate UUID -> Funktionsergebnis

Parameter	Typ		Beschreibung
Funktionsergebnis	String		Neue UUID als nicht-kanonischer Text (32 Zeichen)

Beschreibung

Die Funktion **Generate UUID** gibt einen neuen UUID Identifier mit 32-Zeichen in nicht-kanonische Form zurück.

Ein UUID ist eine 16-byte Nummer (128 bits) mit 32 hexadezimalen Zeichen. Sie lässt sich in nicht-kanonischem Format, d.h. als Reihe mit 32 Buchstaben [A-F, a-f] und/oder Zahlen [0-9] darstellen, z.B. 550e8400e29b41d4a716446655440000) oder in kanonischem Format, d.h. als Gruppen mit 8,4,4,4,12 Stellen, z.B. 550e8400-e29b-41d4-a716-446655440000).

In 4D können Sie UUID Nummern in Feldern speichern. Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus* im Abschnitt .

Beispiel

Eine UUID in einer Variablen erzeugen:

```
C_TEXT(MyUUID)
MyUUID:=Generate UUID
```

GET ACTIVITY SNAPSHOT

GET ACTIVITY SNAPSHOT (arrAktivität | arrUUID ; arrStart ; arrDauer ; arrInfo {; arrDetails}{; *})

Parameter	Typ		Beschreibung
arrAktivität | arrUUID	Array Objekt, Array Text	⇐	Komplette Beschreibung der Operationen (Array Objekt) oder Operation UUIDs(Array Text)
arrStart	Array Text	⇐	Startzeit der Operation
arrDauer	Array Lange Ganzzahl	⇐	Dauer der Operation in Sekunden
arrInfo	Array Text	⇐	Beschreibung
arrDetails	Array Objekt	⇐	Kontext Details und Unteroperationen (sofern vorhanden)
*	Operator	⇒	Mit Stern: Erhalte Server Aktivität

Beschreibung

Der Befehl **GET ACTIVITY SNAPSHOT** gibt Arrays zurück, die mit 4D Daten ablaufende Operationen beschreiben. Diese Operationen zeigen in der Regel ein Ablaufdialog.

So erhalten Sie eine Momentaufnahme der Operationen, die am meisten Zeit benötigen bzw. am häufigsten auf dem Server laufen, wie z.B. Schreiben in den Cache oder Ausführen von Formeln.

Hinweis: Die vom Befehl **GET ACTIVITY SNAPSHOT** zurückgegebene Information ist dieselbe wie auf der Seite "Real-time monitor" (RTM) des Verwaltungsfensters von 4D Server. Weitere Informationen dazu finden Sie auf der **4D Server Verwaltungsfenster** des *4D Server Handbuchs*.

GET ACTIVITY SNAPSHOT bearbeitet standardmäßig Operationen, die lokal ausgeführt werden (mit 4D Einzelplatz, 4D Server oder 4D im remote Modus).
Mit 4D im remote Modus können Sie aber auch eine Momentaufnahme der Operationen erhalten, die auf dem Server ausgeführt werden. Dazu müssen Sie nur den Stern (*) als letzten Parameter übergeben. Dann werden die Server Daten auch lokal übermittelt.
Der Parameter * wird beim Ausführen des Befehls auf 4D Server oder im 4D Einzelplatz ignoriert.

Der Befehl akzeptiert zwei Syntaxarten:

- Syntax mit nur einem Array Objekt
- Syntax mit mehreren Arrays

Erste Syntax: GET ACTIVITY SNAPSHOT (arrAktivität {; *})

Mit dieser Syntax werden alle Operationen in strukturierter Form im 4D Array Objekt (*arrAktivität*) zurückgegeben. Jedes Element des Array ist ein Objekt in folgender Form:

```
[
    {
        "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",
            ...}
            ]
    },
    {...}
]
```

Hier die Beschreibung der Eigenschaften:

- message (Text): Art der Operation
- maxValue (Zahl): Anzahl Wiederholungen für diese Operation (-1 für nicht wiederholte Operationen)
- currentValue (Zahl): Aktuelle Wiederholung
- interruptible (Zahl): Operation kann durch Anwender abgebrochen werden (0=Ja, 1=Nein)
- remote (Zahl): Operation läuft auf Client und Server (0=Ja, 1=Nein)
- uuid (Text): UUID Identifier für Operation
- taskId (Zahl): Interne ID des Prozesses der die Operation gestartet hat
- startTime (Text): Start Zeit der Operation in Format "yyyy:mm:dd hh:mm:ss:mls"
- duration (Zahl): Dauer der Operation in Millisekunden
- *dbContextInfo* (Objekt): Information zu Operationen, die über die Datenbank-Engine verwalten werden. Es gibt folgende Eigenschaften:
  - *host_name* (String): Name des Host, der die Operation gestartet hat
  - *user_name* (String): Name des 4D Anwenders, dessen Sitzung die Operation gestartet hat
  - *task_name* (String): Name des Prozesses, der die Operation gestartet hat
  - *task_id* (Zahl): Nummer des Prozesses, der die Operation gestartet hat
  - *client_uid* (String): optional, uuid des Client, der die Operation gestartet hat
  - *is_remote_context* (Boolean, 0 oder 1): optional, gibt an, ob die Datenbankoperation von einem Client (Wert 1) oder vom Server über eine Serverprozedur (Wert 0)
  - *user4d_id* (num): Nummer des aktuellen 4D Anwenders auf der Client-Seite
  - *client_version* (String): vier Stellen für die 4D Engine der Anwendung, so wie vom Befehl **Application version** zurückgegeben.

  Hinweis: client_uid und is_remote_context sind nur in der Client/ServerUmgebung verfügbar. client_uid wird nur zurückgegeben, wenn die Datenbankoperation auf einen Client-Rechner gestartet wurde.
- *dbOperationDetails* (Objekt): Die Eigenschaft wird nur zurückgegeben, wenn die Operation die Datenbank-Engine verwendet. Das gilt z.B. für Suchläufe bzw. Sortierungen. Es ist ein Objekt mit spezifischen Informationen zur Operation ansich. Die verfügbaren Eigenschaften richten sich nach der ausgeführten Datenbankoperation. Sie können z.B. enthalten:
  - table (String): Name der Tabelle, die von der Operation verwendet wird
  - field (String): Name des Feldes, das von der Operation verwendet wird
  - queryPlan (String): Suchplan, der für die Operation definiert wurde
  - ...
- *subOperations* (Array): Array Objekt mit Unteroperationen für die aktuelle Operation (falls vorhanden). Die Struktur jedes Unterelements ist identisch zum Hauptobjekt. Gibt es keine Unteroperationen, ist der Eintrag leer.

Zweite Syntax: GET ACTIVITY SNAPSHOT (arrUUID ; arrStart ; arrDauer ; arrInfo {;arrUnterOp}{; *})

Mit dieser Syntax erscheinen alle Operationen in mehreren synchronisierten Arrays (bei jeder Operation wird ein Element in allen Arrays hinzugefügt). Folgende Arrays werden zurückgegeben:

- *arrUUID*: Enthält die UUIDs jeder Operation (entspricht der Eigenschaft *uuid* des Objekts *arrAktivität* in der oberen Syntax)
- *arrStart*: Enthält die Startzeiten jeder Operation (entspricht der Eigenschaft *startTime* des Objekts *arrAktivität* in der oberen Syntax)
- *arrDauer*: Enthält die Dauer jeder Operation in Millisekunden (entspricht der Eigenschaft *duration* des Objekts *arrAktivität* in der oberen Syntax)
- *arrInfo*: Enthält die Bezeichnungen, die jede Operation beschreiben (entspricht der Eigenschaft *message* des Objekts *arrAktivität* in der oberen Syntax)
- *arrDetails* (optional): Jedes Element dieses Array ist ein Objekt mit der Eigenschaft "Unteroperation" mit folgenden Eigenschaften:
  - "dbContextInfo" (Objekt): siehe oben
  - "dbOperationDetails" (Objekt): siehe oben
  - "Unteroperationen": Der Wert dieser Eigenschaft ist ein Array Objekt mit all den Unteroperationen zur aktuellen Operation. Hat die aktuelle Operation keine Unteroperationen, ist der Wert der Eigenschaft Unteroperationen ein leeres Array (entspricht der Eigenschaft Unteroperationen des Objekts *Aktivität*).

Beispiel

Diese Methode, die in einem eigenen Prozess in 4D oder 4D Server ausgeführt wird, liefert folgenden Schnappschuss der gerade ablaufenden Operationen:

```
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  // Debugger aufrufen
   End if
Until(False)  // Nicht beendete Schleife
```

Sie erhalten z.B. folgende Arrays:

Expression	Value
arrUUID	6 elements
arrUUID	0
arrUUID{0}	""
arrUUID{1}	"1858EB64D281A7429E2012CEEE78499A"
arrUUID{2}	"07BBF193B1C0184EA26F1D4235A89CE6"
arrUUID{3}	"AE18AFA9426B424FBFB1575554751E05"
arrUUID{4}	"715C5B028868E44BAC4062AB0B507601"
arrUUID{5}	"BEF371C7CFF45946A6B5FE3488692BD4"
arrUUID{6}	"5C51ED352AF1344AA3895D8236D9EC25"
arrStart	6 elements
arrStart	0
arrStart{0}	""
arrStart{1}	"12/3/2013 - 11:32:34"
arrStart{2}	"12/3/2013 - 11:32:35"
arrStart{3}	"12/3/2013 - 11:32:34"
arrStart{4}	"12/3/2013 - 11:32:34"
arrStart{5}	"12/3/2013 - 11:32:34"
arrStart{6}	"12/3/2013 - 11:32:35"
arrDuration	6 elements
arrDuration	0
arrDuration{0}	0
arrDuration{1}	5747
arrDuration{2}	5388
arrDuration{3}	5752
arrDuration{4}	5653
arrDuration{5}	5959
arrDuration{6}	5466
arrInfo	6 elements
arrInfo	0
arrInfo{0}	""
arrInfo{1}	"Array to selection: 9 of 100"
arrInfo{2}	"Loading data"
arrInfo{3}	"Array to selection: 7 of 100"
arrInfo{4}	"Sequential searching on Companies: 11 of 98167 records"
arrInfo{5}	"Deleting records: 6 of 10"
arrInfo{6}	"Sequential searching on Companies: 13 of 98167 records"

GET MACRO PARAMETER

GET MACRO PARAMETER (Selector ; TextParam)

Parameter	Typ		Beschreibung
Selector	Lange Ganzzahl	➡	Zu verwendender Selektor
TextParam	Text	⬅	Zurückgegebener Text

Beschreibung

Der Befehl **GET MACRO PARAMETER** gibt in *TextParam* Text oder Textteile der Methode zurück, in der er aufgerufen wird.

Im Parameter *Selector* können Sie die Art der zurückzugebenden Information setzen. Sie können eine der Konstanten unter dem Thema **4D Umgebung** verwenden:

Konstante	Typ	Wert
Full method text	Lange Ganzzahl	1
Highlighted method text	Lange Ganzzahl	2

Übergeben Sie in *Selector* Full method text, wird der gesamte Text der Methode in *TextParam* zurückgegeben. Mit Highlighted method text wird nur der ausgewählte Text der Methode zurückgegeben.

Beispiel

Siehe Beispiel zum Befehl **SET MACRO PARAMETER**.

LAUNCH EXTERNAL PROCESS

LAUNCH EXTERNAL PROCESS (DateiName {; EingStream {; AusgStream {; FehlerStream}}}{; pid})

Parameter	Typ		Beschreibung
DateiName	String	➡	Pfad und Argumente der zu startenden Datei
EingStream	String, BLOB	➡	Eingabe-Stream (stdin)
AusgStream	String, BLOB	⬅	Ausgabe-Stream (stdout)
FehlerStream	String, BLOB	⬅	Fehler-Stream (stderr)
pid	Lange Ganzzahl	⬅	Einmalige ID für externen Prozess

Beschreibung

Der Befehl **LAUNCH EXTERNAL PROCESS** startet unter Windows und OS X einen externen Prozess von 4D. Auf OS X gewährt dieser Befehl Zugriff auf jedes ausführbare Programm, das sich über das Terminal starten lässt.

In *DateiName* übergeben Sie den festen Dateipfad des auszuführenden Programms, sowie bei Bedarf die erforderlichen Argumente.
Auf OS X können Sie auch nur den Namen des Programms übergeben. 4D findet dann über die Umgebungsvariable *PATH* das ausführbare Programm.

Warnung: Dieser Befehl kann nur ausführbare Programme starten; er kann keine Anweisungen ausführen, die Teil der "Shell" (Befehls-Interpreter) sind. Auf Mac OS ist es z.B. nicht möglich, mit diesem Befehl die Anweisung *Echo* auszuführen oder Indirektionen.

EingStream (optional) enthält den *stdin* des externen Prozesses. Nach Ausführung des Befehls enthalten die Variablen *AusgStream* und *FehlerStream* – sofern übergeben – *stdout* und *stderr* des externen Prozesses. Sie können statt Parameter vom Typ String auch BLOB verwenden, wenn Sie Text größer als 32 KB oder binäre Daten, z.B. Bilder verwenden.

Hinweis: Verwenden Sie die Umgebungsvariable *_4D_OPTION_BLOCKING_EXTERNAL_PROCESS* mit dem Befehl **SET ENVIRONMENT VARIABLE** (asynchrone Ausführung), gibt er in den Parametern *AusgStream* und *FehlerStream* keinen Wert zurück.

Der Parameter *pid* gibt für den erstellten Prozess die ID auf Systemebene zurück, um den Befehl unabhängig vom Status der Option *_4D_OPTION_BLOCKING_EXTERNAL_PROCESS* zu starten. Diese Angabe erleichtert die Interaktion mit einem erstellten externen Prozess, z.B. um ihn zu stoppen. Schlägt Starten des Prozesses fehl, wird in *pid* kein Wert zurückgegeben.

Beispiele auf OS X

Die folgenden Beispiele verwenden das OS X Terminal aus dem Ordner Applications/Utilities.

1. Den Zugriff für eine Datei ändern (chmod ist die OS X Funktion zum Ändern des Dateizugriffs):

```
LAUNCH EXTERNAL PROCESS("chmod +x /folder/myfile.txt")
```

2. Eine Textdatei bearbeiten (cat ist die OS X Datei zum Bearbeiten von Dateien):

```
C_TEXT(input;output)
input:=""
LAUNCH EXTERNAL PROCESS("/bin/cat /folder/myfile.txt";input;output)
```

3. Den Inhalt eines Ordners anzeigen (ls -l is ist die Entsprechung unter UNIX für die Funktion dir in DOS):

```
C_TEXT($In;$Out)
LAUNCH EXTERNAL PROCESS("/bin/ls -l /Users";$In;$Out)
```

4. Um eine unabhängige "grafische" Anwendung zu starten, verwenden Sie besser den Systembefehl *Öffnen*. (In diesem Fall hat die Anweisung **LAUNCH EXTERNAL PROCESS** dieselbe Wirkung wie Doppelklick auf die Anwendung):

```
LAUNCH EXTERNAL PROCESS("open /Applications/Calculator.app")
```

Beispiele unter Windows

5. NotePad öffnen:

```
LAUNCH EXTERNAL PROCESS("C:\\WINDOWS\\notepad.exe")
```

6. Notepad und spezifisches Dokument öffnen:

```
LAUNCH EXTERNAL PROCESS("C:\\WINDOWS\\notepad.exe C:\\Docs\\new folder\\res.txt")
```

7. Microsoft® Word® Anwendung und ein bestimmtes Dokument öffnen (beachten Sie die Verwendung von zwei ""):

```
$mydoc:="C:\\Program Files\\Microsoft Office\\Office10\\WINWORD.EXE \"C:\\
Documents and Settings\\Mark\\Desktop\\MyDocs\\New folder\\test.xml\""
```

```
LAUNCH EXTERNAL PROCESS($mydoc;$tIn;$tOut)
```

8. Ein Perl Skript ausführen (benötigt ActivePerl):

```
C_TEXT($input;$output)
SET ENVIRONMENT VARIABLE("myvariable";"value")
LAUNCH EXTERNAL PROCESS("D:\\Perl\\bin\\perl.exe D:\\Perl\\eg\\cgi\\env.pl";$input;$output)
```

9. Einen Befehl in einem spezifischen Verzeichnis und ohne Anzeigen der Konsole starten:

```
SET ENVIRONMENT VARIABLE("_4D_OPTION_CURRENT_DIRECTORY";"C:\\4D_VCS")
SET ENVIRONMENT VARIABLE("_4D_OPTION_HIDE_CONSOLE";"true")
LAUNCH EXTERNAL PROCESS("meinBefehl")
```

10. Dem Benutzer ermöglichen, ein externes Dokument unter Windows zu öffnen:

```
$docname:=Select document("";"*.*";"Wähle Datei zum Öffnen";0)
If(OK=1)
   SET ENVIRONMENT VARIABLE("_4D_OPTION_HIDE_CONSOLE";"true")
   LAUNCH EXTERNAL PROCESS("cmd.exe /C start \"\" \""+$docname+"\"")
End if
```

11. Die Prozessliste unter Windows anfordern:

```
C_LONGINT($pid)
C_TEXT($stdin;$stdout;$stderr)

LAUNCH EXTERNAL PROCESS("tasklist";$pid)  //erhält nur PID
LAUNCH EXTERNAL PROCESS("tasklist";$stdin;$stdout;$stderr;$pid)  //erhält alle Angaben
```

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf, wie z.B. Datei nicht gefunden, Speicher reicht nicht aus, wird sie auf 0 (Null) gesetzt.

Load 4D View Document

Load 4D View Document (4DViewDokument) -> Funktionsergebnis

Parameter	Typ		Beschreibung
4DViewDokument	BLOB	➡	4D View Dokument
Funktionsergebnis	Objekt	⮌	Objektdarstellung des 4D View Dokuments

Beschreibung

Die Funktion **Load 4D View Document** ermöglicht, ein 4D View Dokument in ein 4D Objekt zu konvertieren.

Dazu sind weder eine gültige 4D View Lizenz, noch eine Instanz des bisherigen Plug-In 4D View in Ihrer Umgebung erforderlich.

Im Parameter *4DViewDokument* übergeben Sie eine Variable oder ein Feld vom Typ BLOB mit dem umzuwandelnden 4D View Dokument. Die Funktion gibt ein 4D *Objekt* zurück mit der Beschreibung der ursprünglich im 4D View Dokument gespeicherten Informationen:

- Struktur des Dokuments (Anzahl Zeilen und Spalten), Typ und Information (Version,Titel...)
- Zelleneigenschaften (Zellentyp, Wert, Formel, Name, Stil, Sicherheit...)
- Spalteneigenschaften (Breite, Stil, Typ, Sicherheit, Sichtbarkeit, Umbruch...)
- Zeileneigenschaften (Höhe, Stil, Typ, Sicherheit, Sichtbarkeit, Umbruch...)
- Stil, Rahmen und Fensterausschnitte

Mit dieser Funktion können Sie alle Daten, die in Ihren 4D View Dokumenten gespeichert sind, wiederherstellen und sie in einem offenen Format bearbeiten.

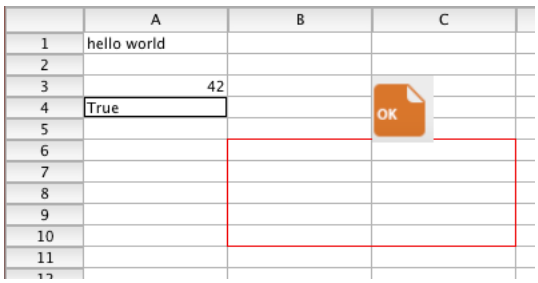
Hinweis: Für eine direkte Konvertierung von 4D View Dokumenten in 4D View Pro empfehlen wir, die spezifische Funktion **VP Convert from 4D View** zu verwenden.

Beispiel

Ein auf der Festplatte gespeichertes 4D View Dokument laden und konvertieren:

```
C_BLOB($blob)
C_OBJECT($object)
DOCUMENT TO BLOB("document.4PV";$blob)
$object:=Load 4D View document($blob)
ALERT("Document title is "+$object.title)
```

Ihr Dokument könnte z.B. wie folgt aussehen:



Sie erhalten folgendes Ergebnis (Hier Darstellung im JSON Format):

{ "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]" }] }

OPEN URL

OPEN URL (Pfad {; AppName}{; *})

Parameter	Typ		Beschreibung
Pfad	String	➡	Pfad des zu öffnenden Dokuments oder URL
AppName	String	➡	Name der Applikation
*	Operator	➡	Mit * = URL wird nicht übersetzt, Ohne * = URL wird übersetzt

Beschreibung

Der Befehl **OPEN URL** öffnet die in *Pfad* übergebene Datei oder URL mit dem Programm, definiert in *AppName*.

Der Parameter *Pfad* kann entweder eine standardmäßige URL oder den Pfadnamen einer Datei enthalten. Der Befehl erlaubt Doppelpunkte (':') auf OS X, linksgerichtete Schrägstriche ('\') unter Windows oder POSIX URL, die mit file:// beginnt.

Ohne den Parameter *AppName* versucht 4D zuerst, den Parameter *Pfad* als Pfadnamen einer Datei zu interpretieren. Trifft das zu, weist 4D das System an, die Datei mit dem am ehesten passenden Programm zu öffnen, z.B. ein Browser für .html Dateien, Word für .doc Dateien, etc. Der Parameter * wird in diesem Fall ignoriert.
Enthält der Parameter *Pfad* eine standardmäßige URL, wie die Protokolle mailto:, news:, http:, etc., startet 4D den standardmäßigen Web Browser und greift auf die URL zu. Gibt es unter den an den Rechner angeschlossenen Volumes keinen Browser, hat der Befehl keine Auswirkung.

Mit dem Parameter *AppName* sucht der Befehl im System nach einer installierten Applikation mit dem angegebenen Namen. Wird sie gefunden, startet er diese und die angegebene URL oder das Dokument wird geöffnet.
Unter Windows läuft das Erkennen des Applikationsnamens genauso ab wie über den Menübefehl 'Ausführen' im Menü 'Start'. Zum Beispiel:

- "iexplore" zum Starten des Internet Explorer
- "chrome" zum Starten von Chrome (wenn installiert)
- "winword" zum Starten von MS Word (wenn installiert)

Hinweis: Die Liste der erkannten Applikationen können Sie im Register unter folgendem Pfad abrufen:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths

Auf OS X wird die Finder Datenbank verwendet, die alle installierten Applikationen automatisch indiziert. Sie erkennt jede .app Applikation über ihren Bundle Namen (mit oder ohne die Endung .app). Zum Beispiel:

- "safari"
- "FireFox"
- "TextEdit"

Wird die Applikation *AppName* nicht gefunden, erscheint kein Fehler; der Befehl arbeitet dann ohne diesen Parameter. Das System wählt z.B. die am nächsten geeignete Applikation für den Dokumenttyp oder die URL.

4D decodiert automatisch die Sonderzeichen der URL. Übergeben Sie den Parameter *, übersetzt 4D die Sonderzeichen der URL nicht. Mit dieser Option können Sie auf URL vom Typ "*http://www.server.net/page.htm?q=something*" zugreifen bzw. solche URL senden.

Hinweis: Dieser Befehl funktioniert nicht, wenn er in einem Web Prozess aufgerufen wird.

Beispiel 1

Die folgenden Beispiele zeigen verschiedene String-Typen, die der Befehl als URL akzeptiert:

```
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")
```

Beispiel 2

Dieses Beispiel startet das am besten passende Programm:

```
$file:=Select document("";"";0)
If(OK=1)
   OPEN URL(Document)
End if
```

Beispiel 3

Sie können eine Textdatei mit verschiedenen Programmen öffnen:

```
OPEN URL("C:\\temp\\cookies.txt") //öffnet die Datei mit Notepad
OPEN URL("C:\\temp\\cookies.txt";"winword") //öffnet die Datei mit MS Word (wenn installiert)
```

```
OPEN URL("C:\\temp\\cookies.txt";"excel") //öffnet die Datei mit MS Excel (wenn installiert)
```

PROCESS 4D TAGS

PROCESS 4D TAGS (EingabeVorlage ; AusgabeDaten {; param}{; param2 ; ... ; paramN})

Parameter	Typ		Beschreibung
EingabeVorlage	Text, BLOB	➡	Vorlage mit Tags zur Bearbeitung
AusgabeDaten	Text, BLOB	⬅	Ergebnis der Vorlage
param	Ausdruck	➡	Parameter zur Übergabe in Vorlagen in Bearbeitung

Beschreibung

Der Befehl **PROCESS 4D TAGS** löst die Bearbeitung der 4D Transformation Tags aus, definiert in *EingabeVorlage* (Feld oder Variable vom Typ Text oder BLOB). Über den optionalen Parameter *param* können Sie spezifische Werte einfügen. Die Ergebnisdaten werden in *AusgabeDaten* zurückgegeben. Weitere Informationen dazu finden Sie im Abschnitt **4D Transformation Tags**.

Dieser Befehl ermöglicht, einen Text vom Typ Vorlage mit Tags und Referenzen auf 4D Ausdrücke bzw. Variablen auszuführen und das Ergebnis je nach Ausführungskontext bzw. den als Parameter übergebenen Werten auszuführen.
Sie können mit diesem Befehl z.B. HTML Seiten generieren und sichern, die auf halb-dynamischen Seiten mit 4D Transformation Tags basieren, ohne dass der 4D Web Server gestartet werden muss. Sie können damit auch E-Mails in HTML Format versenden, welche Tags oder Referenzen auf Daten aus der Datenbank enthalten, die über 4D Internet Commands abgearbeitet werden. Es lässt sich jede Art von Daten in Form von Text bearbeiten, wie XML, SVG oder Text mit Mehrfachstil.

In *EingabeVorlage* übergeben Sie die Daten mit Tags, die bearbeitet werden sollen. Der Parameter kann ein Feld oder eine Variable vom Typ Text oder BLOB sein. Der Typ Text reicht in der Regel aus (Parameter können bis zu 2 GB an Text empfangen).

Hinweis zur Kompatibilität: Ab 4D Version 12 geht der Befehl bei Parametern vom Typ BLOB automatisch davon aus, dass für BLOBs der Zeichensatz MacRoman ist. Für bessere Effizienz raten wir dringend, Parameter vom Typ Text zu verwenden, da sie im Unicode Modus ausgeführt werden.

Alle Transformation Tags von 4D werden unterstützt (*4DTEXT*, *4HTML*, *4DSCRIPT*, *4DLOOP*, *4D EVAL* etc.).

Hinweis: Bei Verwendung des Tag *4DINCLUDE* außerhalb des Bezugssystems von Web Server (Web Prozess) gilt folgendes:

- In 4D im lokalen Modus oder 4D Server ist der Standardordner der Ordner, der die Strukturdatei der Datenbank enthält
- In 4D im remote Modus ist der Standardordner der Ordner, der die 4D Anwendung enthält.

PROCESS 4D TAGS unterstützt eine undefinierte Anzahl zusätzlicher Parameter *param*, die sich in den ausgeführten Code einfügen lassen. Das können wie für Projektmethoden skalare Werte unterschiedlichen Typs (Text, Datum, Zeit, Lange Ganzzahl, Zahl...) sowie Objekte oder Collections sein. Über Array Zeiger können Sie auch Arrays verwenden. Diese Parameter sind während der Ausführung der Vorlage, wie für 4D Methoden, über gängige $1, $2... Argumente verfügbar (siehe Beispiel). Während der Ausführung von **PROCESS 4D TAGS** steht ein eigener Satz lokaler Variablen bereit. Diese Variablen lassen sich während der Bearbeitung schreiben oder lesen.

Hinweis zur Kompatibilität: In bisherigen 4D Versionen war das Nutzen lokaler Variablen der rufenden Methode während der Ausführung von **PROCESS 4D TAGS** möglich, jedoch ausschließlich im interpretierten Modus. Das ist ab 4D v14 R4 nicht mehr möglich, d.h. der Befehl verhält sich im interpretierten und im kompilierten Modus gleich.

Nach Ausführung des Befehls empfängt *AusgabeDaten* das Ausführungsergebnis aus *EingabeVorlage*, zusammen mit evtl. darin enthaltenen abgearbeiteten 4D Tags. Enthält *EingabeVorlage* keine 4D Tags, ist der Inhalt in *AusgabeDaten* identisch mit dem von *EingabeVorlage*. Der Parameter *AusgabeDaten* kann ein Feld oder eine Variable sein, er muss jedoch vom selben Typ wie der Parameter *EingabeVorlage* sein.

Hinweis: Dieser Befehl ruft nie die **Datenbankmethode On Web Authentication** auf.

Beispiel 1

Folgendes Beispiel lädt ein Dokument vom Typ "template", bearbeitet die darin enthaltenen Tags und speichert es dann:

```
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)
```

Beispiel 2

Dieses Beispiel erstellt einen Text aus Daten eines 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 (Name ; varWert)

Parameter	Typ		Beschreibung
Name	String	➡	Zu setzender Variablenname
varWert	String	➡	Wert der Variablen oder "" für Rücksetzen auf Standardwert

Beschreibung

Der Befehl **SET ENVIRONMENT VARIABLE** setzt den Wert einer Umgebungsvariablen unter Windows und Mac OS X. Er wird zusammen mit dem Befehl **LAUNCH EXTERNAL PROCESS** verwendet. Er funktioniert auch mit der Funktion **PHP Execute**.

In *varName* übergeben Sie den Namen der Variablen, in *varWert* ihren Wert.

- Die allgemeine Liste der Umgebungsvariablen und die möglichen Werte finden Sie in der technischen Dokumentation zu Ihrem Betriebssystem.
- Die Liste der für den Befehl **LAUNCH EXTERNAL PROCESS** verfügbaren Umgebungsvariablen finden Sie in der Beschreibung zu diesem Befehl. Beachten Sie, dass es zur Verwendung in diesem Kontext drei spezifische Umgebungsvariablen gibt:
  _4D_OPTION_CURRENT_DIRECTORY setzt das aktuelle Verzeichnis des zu startenden externen Prozesses. In *varWert* müssen Sie den Pfadnamen des Verzeichnisses übergeben (HFS Typ Syntax auf Mac OS und DOS unter Windows).
  _4D_OPTION_HIDE_CONSOLE (nur Windows) blendet das Fenster der DOS Konsole aus. Übergeben Sie in *varWert* "true" um die Konsole auszublenden, "false", um sie anzuzeigen.
  _4D_OPTION_BLOCKING_EXTERNAL_PROCESS führt einen externen Prozess in asynchronen Modus aus, d.h. er ist nicht für andere Anwendungen blockiert. Für die asynchrone Ausführung müssen Sie in *varWert* "false" übergeben, für die synchrone Ausführung "true" (Für diese Variable ist kein Standardwert möglich).
  Diese Variablen gelten im aktuellen Prozess für den nächsten Aufruf von **LAUNCH EXTERNAL PROCESS**.

Beispiel

Siehe Beispiele unter dem Befehl **LAUNCH EXTERNAL PROCESS**.

SET MACRO PARAMETER

SET MACRO PARAMETER (Selector ; TextParam)

Parameter	Typ		Beschreibung
Selector	Lange Ganzzahl	➡	Zu verwendender Selector
TextParam	Text	➡	Gesendeter Text

Beschreibung

Der Befehl **SET MACRO PARAMETER** fügt den Text aus *TextParam* in die Methode ein, in der er aufgerufen wurde.

Wurde in der Methode Text ausgewählt, können Sie im Parameter *Selector* festlegen, ob der Text *TextParam* in der Methode den gesamten Text oder nur den ausgewählten Text ersetzen soll. In *Selector* können Sie eine der Konstanten unter dem Thema **4D Umgebung** verwenden:

Konstante	Typ	Wert
Full method text	Lange Ganzzahl	1
Highlighted method text	Lange Ganzzahl	2

Wurde kein Text ausgewählt, wird *TextParam* in die Methode eingefügt.

Hinweis

Damit die Befehle **GET MACRO PARAMETER** und **SET MACRO PARAMETER** korrekt arbeiten, müssen Sie das neue Attribut "Version" im Makro selbst deklarieren, und zwar folgendermaßen:

```
<macro name="MyMacro" version="2">
--- Text of macro ---
</macro>
```

Beispiel

Dieses Macro erstellt einen neuen Text, der in der aufrufenden Methode zurückgegeben wird:

```
C_TEXT($input_text)
C_TEXT($output_text)
GET MACRO PARAMETER(Highlighted method text;$input_text)
  `Angenommen der gewählte Text ist eine Tabelle, z.B. "[Customers]"
$output_text:=""
$output_text:=$output_text+Command name(47)+"("+$input_text+")"  `Alle wählen ([Customers])
$output_text:=$output_text+"$i:="+Command name(76)+"("+$input_text+")"  `  $i:=Records in selection([Customers])
SET MACRO PARAMETER(Highlighted method text;$output_text)
  `Ersetzt den gewählten Text durch den neuen Code
```

Verify password hash

Verify password hash (Kennwort ; Hash) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Kennwort	String	➡	Benutzerkennwort mit max. 72 Zeichen
Hash	String	➡	Kennwort-Hash
Funktionsergebnis	Boolean	⮌	Gibt WAHR zurück, wenn Kennwort und Hash zusammenpassen, sonst FALSCH

Beschreibung

Die Funktion **Verify password hash** überprüft, ob der angegebene *Hash* zum angegebenen *Kennwort* passt.
Sie vergleicht *Kennwort* mit dem über die Funktion **Generate password hash** generierten *Hash*.

Fehlerverwaltung

Folgende Fehler können zurückgegeben werden. Über die Befehle **GET LAST ERROR STACK** und **ON ERR CALL** können Sie einen Fehler auswerten.

Nummer	Meldung
850	Password-hash: Nicht unterstützter Algorithmus
851	Password-hash: Konsistenzprüfungsfehler

Beispiel

Dieses Beispiel überprüft ein Kennwort-Hash, das zuvor mit **Generate password hash** erstellt und in einer Tabelle [Users] mit einem neu eingegebenen Kennwort gespeichert wurde:

```
C_TEXT($password)
$password:=Request("Bitte geben Sie Ihr Kennwort ein")

If(Verify password hash($password;[Users]hash))
   ALERT("Gutes Kennwort")
Else
   ALERT("Kennwortfehler")
End if
```

Hinweis: Das Kennwort selbst wird nie auf der Festplatte gespeichert, nur der Hash. Bei einer remote 4D Anwendung kann der Hash auf der Client-Seite produziert werden. Verwenden Sie dagegen ein auf JavaScript (oder ähnlich) basierendes Frontend, ist das beste Vorgehen für Sicherheit, ein Hash auf der Server-Seite zu erstellen. Sie sollten natürlich zur Sicherheit eine TLS verschlüsselte Netzwerkverbindung verwenden, da hier die Übertragung des Kennworts über das Netzwerk erforderlich ist.

XML

- Überblick über XML Tools
- XML DECODE
- XML GET ERROR
- XML GET OPTIONS
- XML SET OPTIONS
- *_o_XSLT APPLY TRANSFORMATION*
- *_o_XSLT GET ERROR*
- *_o_XSLT SET PARAMETER*

Überblick über XML Tools

Dieses Kapitel gruppiert die generischen XML Befehle von 4D zum Verwalten von Optionen und Fehlern sowie auf XSL spezialisierte Befehle.

Allgemeine Informationen zu XML (Überblick, Glossar) sowie die Unterschiede zwischen DOM und SAX finden Sie im Abschnitt **Überblick über XML DOM Befehle**.

Was ist SVG?

SVG (Scalable Vector Graphics) ist ein Dateiformat, das Vektorgraphiken mit der Endung .svg in XML beschreibt. Am häufigsten wird SVG beim Veröffentlichen von statistischen Daten oder Landkarten verwendet. Solche Dateien lassen sich in Web Browsern entweder nativ oder über Plug-Ins betrachten. 4D enthält eine SVG Rendering Engine, mit der Sie SVG Dateien in Bildfeldern oder Variablen anzeigen können. Mit dem Befehl **SVG EXPORT TO PICTURE** können Sie in 4D ein Bild erstellen, das auf SVG Beschreibung basiert. Beachten Sie auch, dass Sie mit dem Befehl **GRAPH** die Vorteile der in 4D integrierten SVG Engine nutzen können.
Weitere Informationen zu SVG finden Sie im Internet unter http://www.w3.org/Graphics/SVG/.

Über XSLT Befehle

Ab 4D v14 R4 sind die Befehle zum Bearbeiten von XSLT überholt. Die Befehlsnamen wurden mit der entsprechenden Vorsilbe gekennzeichnet:

Bisheriger Name	Geänderter Name ab 4D v14 R4
XSLT APPLY TRANSFORMATION	_o_XSLT APPLY TRANSFORMATION
XSLT GET ERROR	_o_XSLT GET ERROR
XSLT SET PARAMETER	_o_XSLT SET PARAMETER

Zur Wahrung der Kompatibilität werden XSL Transformationen in 4D noch unterstützt. Wir raten jedoch von einer weiteren Verwendung ab. Die Bearbeitung von XSLT wird in zukünftigen 4D Releases entfernt.

Hinweis zu 4D Server 64-bit OS X: XSLT ist in 4D Server 64-bit für OS X bereits nicht mehr verfügbar. Deshalb wird beim Aufrufen von XSLT Befehlen der Fehler 33 "Der Befehl oder die Methode ist nicht vorhanden" generiert.

Wir empfehlen die Nutzung von PROCESS 4D TAGS als Ersatz. Sein Leistungsumfang wurde mit 4D R4 stark erweitert.

Alternativ können Sie die XSLT Bearbeitung in Ihren Anwendungen durch das PHP Modul *libxslt* ersetzen, das seit Version 14.2 in 4D installiert ist. Ausführliche Informationen zur Verwendung von PHP XSL anstelle der überholten XSLT 4D Befehle finden Sie unter: Download the XSLT with PHP technical document (PDF)

XML DECODE

XML DECODE (xmlWert ; 4DObjekt)

Parameter	Typ		Beschreibung
xmlWert	Text	➡	Wert vom Typ Text, der aus einer XML Struktur stammt
4DObjekt	Feld, Variable	⬅	4D Variable oder Feld für Empfang der konvertierten XML Werte

Beschreibung

Der Befehl **XML DECODE** konvertiert einen Wert, gespeichert als ein XML String, in einen Wert vom Typ 4D. Die Konvertierung wird automatisch nach folgenden Regeln ausgeführt:

Wert	Beispiele	Konvertierung in englischem System
Zahl	<Preis>8,5</Preis> <Preis>8.5</Preis>	Zahl: 8.5
Boolean	<Doppel>1</Doppel> <Doppel>0</Doppel> oder <Doppel>true</Doppel> <Doppel>false</Doppel>	Boolean: True/False
BLOB		Base64 Decodierung
Bild		Base64 Decodierung + BLOB zu Bildbefehlen
Datum	2009-10-25T01:03:20+01:00	Löschen des Zeitteils und der Zeitzone: !25.10.2009!
Zeit	2009-10-25T01:03:20+01:00	Löschen des Datumteils *Warnung*: Zeitzone wird berücksichtigt, wenn unterschiedlich zur lokalen Zeit. Beispiel: "2009-10-25T01:03:20+05:00" ergibt ?21:03:20? in lokaler Zeit UTC+1

Beispiel

Daten aus einem XML Dokument importieren, in dem Werte als Attribute gespeichert sind.
Beispiel für XML Dokument:

```
<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)
                  :($attrName="Titel")
                     [CD]Work:=arrAttrValues{$i}
                  :($attrName="Preis")
                     XML DECODE(arrAttrValues{$i};[CD]Price)
                  :($attrName="Datum")
                     XML DECODE(arrAttrValues{$i};[CD]Date entered)
                  :($attrName="Dauer")
                     XML DECODE(arrAttrValues{$i};[CD]Total_duration)
                  :($attrName="Doppel")
                     XML DECODE(arrAttrValues{$i};[CD]Double_CD)
               End case
```

```
            End for
        End if
        ...
    End case
Until(MyEvent=XML End Document)
```

XML GET ERROR

XML GET ERROR (ElementRef ; FehlerText {; Zeile {; Spalte}})

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz des XML Elements
FehlerText	Variable	⬅	Fehlerbeschreibung
Zeile	Variable	⬅	Zeilennummer
Spalte	Variable	⬅	Spaltennummer

Beschreibung

Der Befehl **XML GET ERROR** gibt im Parameter *FehlerText* eine Beschreibung des gefundenen Fehlers zurück, der beim Bearbeiten des XML Elements auftritt, definiert durch *ElementRef*. Der erhaltene Text wird von der Xerces.DLL library übergeben.

Die optionalen Parameter *Zeile* und *Spalte* geben den Ort des Fehlers an: Sie finden die Zeilennummer und darin die Position des ersten Zeichens des Ausdrucks, der den Fehler hervorruft.

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf, wird sie auf 0 (Null) gesetzt.

XML GET OPTIONS

XML GET OPTIONS (ElementRef | Dokument ; Selector ; Wert {; Selector2 ; Wert2 ; ... ; SelectorN ; WertN})

Parameter	Typ		Beschreibung
ElementRef | Dokument	Text	➡	XML Root Element Referenz oder Referenz auf offenes Dokument
Selector	Lange Ganzzahl	➡	Zu erhaltende Option
Wert	Lange Ganzzahl	⬅	Aktueller Wert der Option

Beschreibung

Der Befehl **XML GET OPTIONS** erhält den aktuellen Wert von einem oder mehreren XML Parametern für die aktuelle Sitzung und den aktuellen Benutzer.

In *Selector* übergeben Sie eine Konstante unter dem Thema **XML**, welche die zu erhaltende Option angibt. Der aktuelle Wert der Option wird im Parameter *Wert* zurückgegeben.

Konstante	Typ	Wert	Kommentar
XML binary encoding	Lange Ganzzahl	5	Gibt an, wie binäre Daten konvertiert werden. **Mögliche Werte:** • XML Base64 (Standardwert): binäre Daten werden einfach in Base64 konvertiert. • XML Data URI scheme: binäre Daten werden in Base64 konvertiert und der Header "data:;base64" wird hinzugefügt. Dieses Format ermöglicht hauptsächlich einem Browser, ein Bild automatisch zu decodieren. Es ist auch zum Einfügen von SVG Bildern erforderlich. Weitere Information dazu siehe http://en.wikipedia.org/wiki/Data_URI_scheme.
XML date encoding	Lange Ganzzahl	2	Gibt an, wie 4D das Datum konvertiert, z.B. !01/01/2003! in der Paris Zeitzone. **Mögliche Werte:** • XML ISO (Standardwert): verwendet das Format xs:datetime ohne Angabe der Zeitzone. Ergebnis: "2003-01-01". Ist im 4D Wert (via SQL) ein Zeitteil enthalten, geht er verloren. • XML Local: verwendet das Format xs:date mit Angabe der Zeitzone. Ergebnis: "2003-01-01 +01:00". Ist im 4D Wert (via SQL) ein Zeitteil enthalten, geht er verloren. • XML Datetime local: verwendet das Format xs:dateTime (ISO 8601) mit Angabe der Zeitzone. Ist im 4D Wert (via SQL) ein Zeitteil enthalten, wird er beibehalten. Ergebnis: "<Date>2003-01-01T00:00:00 +01:00</Date>".. • XML UTC: verwendet das Format xs:date. Ergebnis: "2003-01-01Z". Ist im 4D Wert (via SQL) ein Zeitteil enthalten, geht er verloren. • XML Datetime UTC: verwendet das Format xs:dateTime (ISO 8601). Ist im 4D Wert (via SQL) ein Zeitteil enthalten, wird er beibehalten. Ergebnis: "<Date>2003-01-01T00:00:00Z</Date>"
XML DOM case sensitivity	Lange Ganzzahl	8	Definiert, ob Groß- und Kleinschreibung in Elementnamen für die Befehle **DOM Get XML element** und **DOM Count XML elements** berücksichtigt wird. **Mögliche Werte:** • XML case sensitive (Standardwert): Befehle unterscheiden zwischen Groß- und Kleinschreibung. • XML case insensitive: Befehle unterscheiden nicht zwischen Groß- und Kleinschreibung.
XML external entity resolution	Lange Ganzzahl	7	Steuert, ob externe Einheiten in XML Dokumente aufgelöst werden. Aus Sicherheitsgründen erlauben die DOM und SAX 4D XML Parser standardmäßig keine Auflösung externer Einheiten. **Mögliche Werte:** • XML enabled: Erlaubt die Auflösung externer Einheiten in XML Dokumente • XML disabled (Standardwert): Verweigert die Auflösung externer Einheiten (Die Deklaration generiert einen Parser/Analysefehler)
XML indentation	Lange Ganzzahl	4	Gibt die Einrückung des XML Dokuments an. **Mögliche Werte:** • XML with indentation (Standardwert): das Dokument ist eingerückt. • XML no indentation: Das Dokument ist nicht eingerückt; sein Inhalt wird in eine einzige Zeile gesetzt.
XML picture encoding	Lange Ganzzahl	6	Gibt an, wie Bilder konvertiert werden müssen (vor Codierung in Base64). **Mögliche Werte:** • XML Convert to PNG (Standardwert): Bilder werden vor der Codierung in Base64 in PNG konvertiert. • XML Native codec: Bilder werden vor Codierung in Base64 in ihr erstes native Speicher CODEC konvertiert. Sie müssen diese Optionen zum Codieren in SVG Bilder verwenden (siehe Beispiel zum Befehl **XML SET OPTIONS**.
XML string encoding	Lange Ganzzahl	1	Gibt an, wie 4D Strings in Elementwerte konvertiert werden. Das gilt nicht für die Konvertierung in Attribute, für die XML die Verwendung von Escape-Zeichen verlangt. **Mögliche Werte:** • XML With escaping (Standardwert): Konvertierung von 4D Strings in XML Elementwerte mit Ersetzen der Zeichen. Daten vom Typ Text werden automatisch analysiert, so dass verbotene Zeichen (<&>') durch XML Einheiten ersetzt werden (&&It;>'"). • XML Raw data: 4D Strings werden als reine Daten gesendet; 4D führt weder eine Codierung noch eine Analyse durch. 4D Werte werden nach Möglichkeit in XML Fragmente konvertiert und als Unterelement des Zielelements eingefügt. Kann ein Wert nicht als ein XML Fragment bewertet werden, wird es in Form von Rohdaten in einen neuen Knoten CDATA eingefügt.

Konstante	Typ	Wert	Kommentar
XML time encoding	Lange Ganzzahl	3	Gibt an, wie 4D die Zeit konvertiert, z.B. ?02/00/46? (Paris Zeit). Die Codierung ist unterschiedlich, je nachdem ob Sie eine Uhrzeit oder eine Zeitspanne ausdrücken wollen. **Mögliche Werte für Uhrzeit:** • XML Datetime UTC: Zeit ausgedrückt in UTC (Universal Time Coordinated). Beachten Sie, dass die Konvertierung in UTC automatisch ist. Ergebnis: "<Duration>0000-00-00T01:00:46Z</Duration>". • XML Datetime local: Zeit ausgedrückt mit der Zeitdifferenz des Rechners mit der 4D Engine. Ergebnis: "<Duration>0000-00-00T02:00:46+01:00</Duration>". • XML Datetime local absolute (Standardwert): Zeit ausgedrückt ohne Angabe der Zeitzone. Der Wert wird nicht geändert. Ergebnis: "<Duration>0000-00-00T02:00:46</Duration>". **Mögliche Werte für Zeitspanne:** • XML Seconds: Anzahl Sekunden ab Mitternacht; der Wert wird nicht geändert, da er eine Dauer ausdrückt. Ergebnis: "<Duration>7246</Duration>". • XML Duration: Dauer ausgedrückt in Übereinstimmung mit dem XML Schema Teil 2: Datentypen Zweite Edition. der Wert wird nicht geändert, da er eine Dauer ausdrückt. Ergebnis: "<Duration>PT02H00M46S</Duration>".

XML SET OPTIONS

XML SET OPTIONS (ElementRef | Dokument ; Selector ; Wert1...N {; Selector2 ; Wert1...N2 ; ... ; SelectorN ; Wert1...NN})

Parameter	Typ		Beschreibung
ElementRef | Dokument	Text	➡	XML Root Element Referenz oder Referenz auf offenes Dokument
Selector	Lange Ganzzahl	➡	zu setzende Option
Wert1...N	Lange Ganzzahl	➡	Wert der Option

Beschreibung

Der Befehl **XML SET OPTIONS** ändert den Wert einer oder mehrerer XML Optionen der Struktur, die im ersten Parameter übergeben wurde.

Dieser Befehl gilt für die XML Strukturen vom Typ "Baum" (DOM) oder "Dokument" (SAX). Im ersten Parameter können Sie entweder die Referenz auf ein Root Element (*ElementRef*) oder die Referenz auf ein offenes SAX Dokument (*Dokument*) übergeben.

Übergeben Sie die zu ändernde Option in *Selector* und den neuen Wert der Option in *Wert*. Sie können beliebig viele Paare *Selector/Wert* übergeben. Sie müssen die nachfolgend beschriebenen Konstanten unter dem Thema **XML** verwenden.

- Die gesetzten Optionen werden nur in der Richtung 4D zu XML genutzt. Der Befehl hat keine Auswirkung auf das Lesen von XML Werten in 4D. Folgende Befehle verwenden diese Optionen:
  - **DOM SET XML ATTRIBUTE**
  - **DOM SET XML ELEMENT VALUE**
  - **SAX ADD XML ELEMENT VALUE**

Konstante	Typ	Wert	Kommentar
XML binary encoding	Lange Ganzzahl	5	Gibt an, wie binäre Daten konvertiert werden. **Mögliche Werte:** ◦ XML Base64 (Standardwert): binäre Daten werden einfach in Base64 konvertiert. ◦ XML Data URI scheme: binäre Daten werden in Base64 konvertiert und der Header "data:;base64" wird hinzugefügt. Dieses Format ermöglicht hauptsächlich einem Browser, ein Bild automatisch zu decodieren. Es ist auch zum Einfügen von SVG Bildern erforderlich. Weitere Information dazu siehe http://en.wikipedia.org/wiki/Data_URI_scheme.
XML date encoding	Lange Ganzzahl	2	Gibt an, wie 4D das Datum konvertiert, z.B. !01/01/2003! in der Paris Zeitzone. **Mögliche Werte:** ◦ XML ISO (Standardwert): verwendet das Format xs:datetime ohne Angabe der Zeitzone. Ergebnis: "2003-01-01". Ist im 4D Wert (via SQL) ein Zeitteil enthalten, geht er verloren. ◦ XML Local: verwendet das Format xs:date mit Angabe der Zeitzone. Ergebnis: "2003-01-01 +01:00". Ist im 4D Wert (via SQL) ein Zeitteil enthalten, geht er verloren. ◦ XML Datetime local: verwendet das Format xs:dateTime (ISO 8601) mit Angabe der Zeitzone. Ist im 4D Wert (via SQL) ein Zeitteil enthalten, wird er beibehalten. Ergebnis: "<Date>2003-01-01T00:00:00 +01:00</Date>".. ◦ XML UTC: verwendet das Format xs:date. Ergebnis: "2003-01-01Z". Ist im 4D Wert (via SQL) ein Zeitteil enthalten, geht er verloren. ◦ XML Datetime UTC: verwendet das Format xs:dateTime (ISO 8601). Ist im 4D Wert (via SQL) ein Zeitteil enthalten, wird er beibehalten. Ergebnis: "<Date>2003-01-01T00:00:00Z</Date>"
XML indentation	Lange Ganzzahl	4	Gibt die Einrückung des XML Dokuments an. **Mögliche Werte:** ◦ XML with indentation (Standardwert): das Dokument ist eingerückt. ◦ XML no indentation: Das Dokument ist nicht eingerückt; sein Inhalt wird in eine einzige Zeile gesetzt.
XML picture encoding	Lange Ganzzahl	6	Gibt an, wie Bilder konvertiert werden müssen (vor Codierung in Base64). **Mögliche Werte:** ◦ XML Convert to PNG (Standardwert): Bilder werden vor der Codierung in Base64 in PNG konvertiert. ◦ XML Native codec: Bilder werden vor Codierung in Base64 in ihr erstes native Speicher CODEC konvertiert. Sie müssen diese Optionen zum Codieren in SVG Bilder verwenden (siehe Beispiel zum Befehl **XML SET OPTIONS**.
XML string encoding	Lange Ganzzahl	1	Gibt an, wie 4D Strings in Elementwerte konvertiert werden. Das gilt nicht für die Konvertierung in Attribute, für die XML die Verwendung von Escape-Zeichen verlangt. **Mögliche Werte:** ◦ XML With escaping (Standardwert): Konvertierung von 4D Strings in XML Elementwerte mit Ersetzen der Zeichen. Daten vom Typ Text werden automatisch analysiert, so dass verbotene Zeichen (<&>') durch XML Einheiten ersetzt werden (&&It;>'"). ◦ XML Raw data: 4D Strings werden als reine Daten gesendet; 4D führt weder eine Codierung noch eine Analyse durch. 4D Werte werden nach Möglichkeit in XML Fragmente konvertiert und als Unterelement des Zielelements eingefügt. Kann ein Wert nicht als ein XML Fragment bewertet werden, wird es in Form von Rohdaten in einen neuen Knoten CDATA eingefügt.
XML time encoding	Lange Ganzzahl	3	Gibt an, wie 4D die Zeit konvertiert, z.B. ?02/00/46? (Paris Zeit). Die Codierung ist unterschiedlich, je nachdem ob Sie eine Uhrzeit oder eine Zeitspanne ausdrücken wollen. **Mögliche Werte für Uhrzeit:** ◦ XML Datetime UTC: Zeit ausgedrückt in UTC (Universal Time Coordinated). Beachten Sie, dass die Konvertierung in UTC automatisch ist. Ergebnis: "<Duration>0000-00-00T01:00:46Z</Duration>". ◦ XML Datetime local: Zeit ausgedrückt mit der Zeitdifferenz des Rechners mit der 4D Engine. Ergebnis: "<Duration>0000-00-00T02:00:46+01:00</Duration>". ◦ XML Datetime local absolute (Standardwert): Zeit ausgedrückt ohne Angabe der Zeitzone. Der Wert wird nicht geändert. Ergebnis: "<Duration>0000-00-00T02:00:46</Duration>". **Mögliche Werte für Zeitspanne:** ◦ XML Seconds: Anzahl Sekunden ab Mitternacht; der Wert wird nicht geändert, da er eine Dauer ausdrückt. Ergebnis: "<Duration>7246</Duration>". ◦ XML Duration: Dauer ausgedrückt in Übereinstimmung mit dem XML Schema Teil 2: Datentypen Zweite Edition. der Wert wird nicht geändert, da er eine Dauer ausdrückt. Ergebnis: "<Duration>PT02H00M46S</Duration>".

Hinweise:

- Die Werte XML Local und XML Datetime local liefern keine Daten ausgedrückt in UTC (Universal Time Coordinated); sie werden ohne Änderung konvertiert, jedoch mit Angabe der Zeitdifferenz. Diese Formate sind sinnvoll bei sukzessiver und reziproker Konvertierung (round tripping).
- Die Werte XML UTC und XML Datetime UTC ähneln den vorigen in Bezug auf die Formatierung, werden jedoch in UTC ausgedrückt. Diese Formate sollten vorrangig verwendet werden, um die Interoperabilität zu sichern. Die Werte werden nicht geändert.

- Mit folgenden Optionen können Sie einige Standard Xml Parser Features ändern:

Konstante	Typ	Wert	Kommentar
XML DOM case sensitivity	Lange Ganzzahl	8	Definiert, ob Groß- und Kleinschreibung in Elementnamen für die Befehle **DOM Get XML element** und **DOM Count XML elements** berücksichtigt wird. **Mögliche Werte:** ◦ XML case sensitive (Standardwert): Befehle unterscheiden zwischen Groß- und Kleinschreibung. ◦ XML case insensitive: Befehle unterscheiden nicht zwischen Groß- und Kleinschreibung.
XML external entity resolution	Lange Ganzzahl	7	Steuert, ob externe Einheiten in XML Dokumente aufgelöst werden. Aus Sicherheitsgründen erlauben die DOM und SAX 4D XML Parser standardmäßig keine Auflösung externer Einheiten. **Mögliche Werte:** ◦ XML enabled: Erlaubt die Auflösung externer Einheiten in XML Dokumente ◦ XML disabled (Standardwert): Verweigert die Auflösung externer Einheiten (Die Deklaration generiert einen Parser/Analysefehler)

Beispiel

Ein SVG Bild einfügen

```
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 (xmlQuelle ; xslSheet ; Ergebnis {; KompiliereSheet})

Parameter	Typ		Beschreibung
xmlQuelle	String, BLOB	➡	Name oder Zugriffspfad des XML Quelldokuments oder BLOB mit der XML Quelle
xslSheet	String, BLOB	➡	Name oder Zugriffspfad des Dokuments mit XSL Stilvorlagen oder BLOB mit XSL Stilvorlagen
Ergebnis	String, BLOB	➡	Name oder Zugriffspfad des Dokuments mit Ergebnis der XSLT Transformation oder BLOB mit Ergebnis der XSLT Transformation
KompiliereSheet	Boolean	➡	Wahr = Optimiere XSLT Transformation, Falsch oder nicht übergeben = Keine Optimierung, entferne kompilierte XSL Datei (wenn vorhanden)

Hinweis zur Kompatibilität

Ab 4D v14 R4 sind die Befehle zum Bearbeiten von XSLT überholt. Zur Wahrung der Kompatibilität werden XSL Transformationen in 4D noch unterstützt. Wir raten jedoch von einer weiteren Verwendung ab. Die Bearbeitung von XSLT wird in zukünftigen 4D Releases entfernt. Weitere Informationen finden Sie im Abschnitt **Überblick über XML Tools***.

_o_XSLT GET ERROR

_o_XSLT GET ERROR (FehlerText {; Zeile {; Spalte}})

Parameter	Typ		Beschreibung
FehlerText	Variable	⇐	Text des Fehlers
Zeile	Variable	⇐	Zeilennummer
Spalte	Variable	⇐	Spaltennummer

Hinweis zur Kompatibilität

Ab 4D v14 R4 sind die Befehle zum Bearbeiten von XSLT überholt. Zur Wahrung der Kompatibilität werden XSL Transformationen in 4D noch unterstützt. Wir raten jedoch von einer weiteren Verwendung ab. Die Bearbeitung von XSLT wird in zukünftigen 4D Releases entfernt. Weitere Informationen finden Sie im Abschnitt **Überblick über XML Tools***.

_o_XSLT SET PARAMETER

_o_XSLT SET PARAMETER (ParamName ; ParamWert)

Parameter	Typ		Beschreibung
ParamName	String	➡	Name des zu suchenden Parameters im XSL Sheet
ParamWert	String	➡	Wert des zu verwendenden Parameters im umgewandelten Dokument

Hinweis zur Kompatibilität

Ab 4D v14 R4 sind die Befehle zum Bearbeiten von XSLT überholt. Zur Wahrung der Kompatibilität werden XSL Transformationen in 4D noch unterstützt. Wir raten jedoch von einer weiteren Verwendung ab. Die Bearbeitung von XSLT wird in zukünftigen 4D Releases entfernt. Weitere Informationen finden Sie im Abschnitt **Überblick über XML Tools***.

XML DOM

- Überblick über XML DOM Befehle
- 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

Überblick über XML DOM Befehle

4D bietet eine Befehlsgruppe zum Analysieren (parsen) von Objekten mit XML (eXtensible Markup Language).

Hinweis zum preemptive Modus: Von einem preemptive Prozess erstellte XML Referenzen lassen sich nur in diesem spezifischen Prozess verwenden. Im Unterschied dazu sind von einem kooperativen Prozess erstellte XML Referenzen von allen anderen kooperativen Prozessen verwendbar, jedoch nicht von einem preemptive Prozess.

Die XML Sprache

XML ist eine Standardsprache für den Datenaustausch. Sie arbeitet mit Tags, die eine exakte Beschreibung der ausgetauschten Daten sowie der Struktur ermöglichen. XML Dateien sind Textdateien, deren Inhalt die Programme beim Datenimport analysieren. Inzwischen unterstützen viele Programme dieses Format.

Weitere Informationen zu XML finden Sie im Internet unter http://xml.org und http://www.w3.org.

4D verwendet zur XML Unterstützung die Library mit Namen Xerces.dll, welche vom Unternehmen Apache Foundation entwickelt wurde.

Hinweis: 4D ermöglicht über den Import-/Exporteditor den direkten Im- und Export von Daten in XML Format.

SAX und DOM Befehle

4D bietet zwei verschiedene Sätze XML Befehle, jeweils mit der Vorsilbe DOM und SAX:
DOM (Document Object Model) und SAX (Simple API XML) sind zwei Möglichkeiten, um XML Dokumente logisch zu durchlaufen (parsen).

- Der DOM-Modus durchläuft eine XML-Quelle und erstellt deren Struktur, d.h. Hierarchie (“tree”) im Speicher. Auf diese Weise ist der Zugriff auf jedes Element der Quelle extrem schnell. Da jedoch die gesamte Baumstruktur im Speicher abgelegt ist, kann das Abarbeiten umfangreicher XML Dokumente die Speicherkapazität übersteigen und zu Fehlern führen.
- Der SAX-Modus erstellt keine Baumstruktur im Speicher. Hier werden Ereignisse, wie Start und Ende eines Elements beim Durchlaufen der Quelle erzeugt. So können Sie XML Dokumente jeglicher Größe durchlaufen, unabhängig von der verfügbaren Speicherkapazität. Die SAX Befehle sind im Kapitel **XML SAX** zusammengefasst. Weitere Informationen dazu finden Sie im Abschnitt **Überblick über XML SAX Befehle**.

Weitere Informationen zu XML Standards finden Sie in folgenden Websites: http://www.saxproject.org/?selected=event und http://www.w3schools.com/xml/.

XML Dokumente via DOM verwalten

Objekte, die von 4D DOM Befehlen erstellt, geändert oder analysiert werden, können Text, URLs, Dokumente oder BLOBs sein. Die DOM Befehle zum Öffnen von XML Objekten in 4D sind **DOM Parse XML source** und **DOM Parse XML variable**.

Es gibt viele Befehle zum Lesen, Analysieren und Schreiben von Elementen und Attributen. Fehler werden abgefangen mit dem Befehl **DOM Parse XML variable**. Er gilt für beide XML Standards.

Mit dem Befehl **XML GET ERROR** schließen Sie die Quelle am Ende wieder.

Hinweis zur Verwendung von XML BLOB Parametern: XML Strukturen basieren auf Daten vom Typ Text. Die 4D XML Befehle, z.B. **DOM Parse XML variable** verwenden jedoch zum Verwalten von XML Daten in der Regel Parameter vom Typ BLOB, unabhängig vom Originaltyp. Diese Vorgehensweise war in bisherigen Versionen von 4D notwendig, da die Größe von Variablen vom Typ Text auf 32 KB beschränkt war.
Ab 4D v11 können Variablen und Felder vom Typ Text bis zu 2 GB an Daten enthalten. Da die bisherige Einschränkung weggefallen ist, raten wir dringend, keine Texte mehr in BLOBs zu speichern. BLOBs sollten für die Bearbeitung binärer Daten reserviert bleiben. Zur besseren Konformität mit XML Spezifikationen werden in 4D v12 binäre Daten automatisch in Base64 codiert, selbst wenn das BLOB Text enthält.

XPath Notation verwalten (DOM)

Die drei DOM Routinen **DOM Create XML element**, **DOM Find XML element** und **DOM SET XML ELEMENT VALUE** akzeptieren die XPath Notation für den Zugriff auf XML Elemente.

XPath Notation stammt aus der XPath Sprache. Sie dient zum Navigieren in XML Strukturen. Sie können Elemente direkt über "Pfadname" vom Typ Syntax in eine XML Struktur setzen. Sie müssen nicht zwingend den kompletten Pfadnamen angeben, um das Element zu erreichen. Hierfür ein Beispiel:

```
<RootElement>  <Elem1>  <Elem2>  <Elem3 Font=Verdana Size=10> </Elem3>  </Elem2>
  </Elem1>  </RootElement>
```

Mit der XPath Notation können Sie über die Syntax */RootElement/Elem1/Elem2/Elem* auf Element 3 zugreifen.

4D akzeptiert über die Syntax *Element[NumElement]* auch indizierte XPath Elemente. Hierfür ein Beispiel:

```
<RootElement>  <Elem1>  <Elem2>aaa</Elem2>  <Elem2>bbb</Elem2>  <Elem2>ccc</Elem2>
  </Elem1>  </RootElement>
```

Mit der XPath Notation können Sie über die Syntax */RootElement/Elem1/Elem2[3]* auf den Wert "ccc" zugreifen.
Beispiele zur Darstellung der XPath Notation finden Sie unter den Funktionen **DOM Create XML element** und **DOM Find XML element**.

Zeichensätze

Die Befehle XML DOM und XML SAX von 4D unterstützen folgende Zeichensätze:

- ASCII
- UTF-8
- UTF-16 (Big/Small Endian)
- UCS4 (Big/Small Endian)
- EBCDIC Code Seiten IBM037, IBM1047 und IBM1140 Codierungen,
- ISO-8859-1 (oder Latin1)
- Windows-1252.

Terminologie

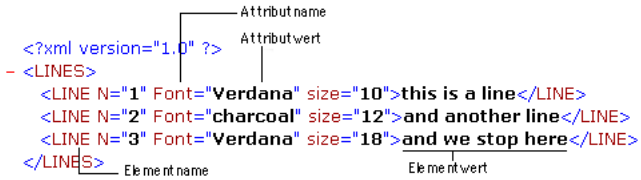
Die XML Sprache verwendet eine Reihe spezifischer Begriffe und Abkürzungen. Nachfolgende Übersicht erläutert die wichtigsten XML Konzepte, mit denen die Befehle und Funktionen von 4D arbeiten.

Attribut: Untergeordnetes XML Tag, das einem Element zugeordnet ist. Ein Attribut enthält immer einen Namen und einen Wert (siehe Schema).

Child (Kind): Element einer XML Struktur, das in der Hierarchie direkt unter einer höheren Ebene liegt.

DTD: Document Typ Declaration DTD speichert den Satz spezifischer Regeln und Eigenschaften für XML. Diese Regeln definieren Name und Inhalt jedes Tags sowie dessen Kontext. Über diese Formatierung lässt sich prüfen, ob ein XML Dokument den Regeln entspricht. In diesem Fall wird es als gültig (valid) deklariert.
DTD kann direkt im XML Dokument (internes DTD) oder in einem separaten Dokument (externes DTD) enthalten sein. Beachten Sie, dass DTD nicht zwingend ist.

Element: XML Tag. Ein Element enthält immer einen Namen und einen Wert. Ein Element kann optional auch Attribute enthalten (siehe Schema).



ElementRef: XML Referenz, über die XML Befehle von 4D eine XML Struktur definieren. Die Referenz besteht aus 8 hexadezimal codierten Zeichen, d.h. sie ist 16 oder 32 Zeichen lang, je nach dem, ob Sie ein 16- oder 32-bit System verwenden.
Wir empfehlen, XML Referenzen über die Direktive **C_TEXT** zu deklarieren.

Parent (Haupt): Element einer XML Struktur, das in der Hierarchie direkt über einer tieferen Ebene liegt.

Parsing, **parser**: Analyse eines strukturierten Objekts, um nützliche Information zu entnehmen. Über die Befehle unter dem Thema "XML" lässt sich der Inhalt jedes beliebigen XML Objekts analysieren.

Root: Element, das auf der obersten Ebene einer XML Struktur liegt.

Sibling (Geschwister): Element einer XML Struktur, das auf derselben Ebene wie ein anderes liegt.

XML Struktur: Strukturiertes XML Objekt. Das kann ein Dokument, eine Variable oder ein Element sein.

Validation: Der Parser betrachtet ein XML Dokument als gültig (validated), wenn es "well-formed" ist und mit den DTD Spezifikationen übereinstimmt.

Well-formed: Der Parser deklariert ein XML Dokument als "well-formed", wenn es die allgemeinen XML Spezifikationen erfüllt.

XML: eXtensible Markup Language. Weiterentwicklung von HTML. Dient zum exakten Beschreiben von Inhalten und ermöglicht damit auch den Datenaustausch mit Datenbanken. Die XML Sprache basiert auf Tags und einer spezifischen Syntax. Im Gegensatz zu HTML ermöglicht XML auch die Definition individuell gestalteter Tags.

XSL: eXtensible Stylesheet Language: Sprache zum Definieren von Vorlagen (style sheets), um den Inhalt eines XSL Dokuments zu bearbeiten und anzuzeigen.

Fehlerverwaltung

Viele Funktionen unter diesem Thema geben eine Referenz auf ein XML Element zurück. Tritt während der Ausführung ein Fehler auf, z.B. wenn die Referenz auf das Root Element ungültig ist, wird die Variable OK auf 0 gesetzt und ein Fehler erzeugt.

Zusätzlich ist die zurückgegebene Referenz in diesem Fall eine Sequenz von Nullen (16 Zeichen in 32 bits und 32 Zeichen in 64 bits).

DOM Append XML child node

DOM Append XML child node (ElementRef ; KindTyp ; KindWert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	Text	➡	XML Element Referenz
KindTyp	Lange Ganzzahl	➡	Typ des anzuhängenden Kindelements
KindWert	Text, BLOB	➡	Text oder Variable (Text oder BLOB), dessen Wert als Kindknoten eingefügt werden soll.
Funktionsergebnis	Text	↩	Referenz des Kind XML Elements

Beschreibung

Die Funktion **DOM Append XML child node** hängt den Wert *KindWert* an den in *ElementRef* definierten XML Knoten an.

Der Typ des erstellten Knotens wird im Parameter *KindTyp* definiert. Sie können eine der nachfolgenden Konstanten unter dem Thema **XML** übergeben:

Konstante	Typ	Wert
XML CDATA	Lange Ganzzahl	7
XML comment	Lange Ganzzahl	2
XML DATA	Lange Ganzzahl	6
XML DOCTYPE	Lange Ganzzahl	10
XML ELEMENT	Lange Ganzzahl	11
XML processing instruction	Lange Ganzzahl	3

In *KindWert* übergeben Sie die einzufügenden Daten. Sie können einen String oder eine 4D Variable übergeben (String oder BLOB). Der Inhalt dieses Parameters wird immer in Text konvertiert.

Hinweis: Gibt der Parameter *ElementRef* den Dokumentknoten an (Knoten auf der obersten Ebene), fügt die Funktion einen Knoten "Doctype" vor allen anderen Knoten ein. Das gleiche gilt auch für Arbeitsanweisungen und Kommentare. Sie werden immer vor dem Root Knoten, aber nach dem Doctype Knoten, eingefügt.

Beispiel 1

Knoten vom Typ Text hinzufügen:

```
Reference:=DOM Create XML element(elementRef;"myElement")
DOM SET XML ELEMENT VALUE(Reference;"Hallo")
temp:=DOM Create XML element(Reference;"br")
temp:=DOM Append XML child node(Reference;XML DATA;"New")
temp:=DOM Create XML element(Reference;"br")
temp:=DOM Append XML child node(Reference;XML DATA;"York")
```

Ergebnis:

```
<myElement>Hallo<br/>New<br/>York</myElement>
```

Beispiel 2

Knoten vom Typ Arbeitsanweisung hinzufügen:

```
$Txt_instruction:="xml-stylesheet type = \"text/xsl\" href=\"style.xsl\""
Reference:=DOM Append XML child node(elementRef;XML Processing Instruction;$Txt_instruction)
```

Ergebnis (vor dem ersten Element eingefügt):

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

Beispiel 3

Knoten vom Typ Kommentar hinzufügen:

```
Reference:=DOM Append XML child node(elementRef;XML Comment;"Hallo Welt")
```

Ergebnis:

```
<!--Hallo Welt-->
```

Beispiel 4

Knoten vom Typ CDATA hinzufügen:

```
Reference:=DOM Append XML child node(elementRef;XML CDATA;"12 < 18")
```

Ergebnis:

```
<element><![CDATA[12 < 18]]></element>
```

Beispiel 5

Knoten vom Typ Doctype declaration hinzufügen oder ersetzen:

```
Reference:=DOM Append XML child node(elementRef;XML DOCTYPE;"Books SYSTEM \"Book.DTD\"")
```

Ergebnis (vor dem ersten Element eingefügt):

```
<!DOCTYPE Books SYSTEM "Book.DTD">
```

Beispiel 6

Knoten vom Typ Element hinzufügen oder ersetzen:

- Ist der Parameter *KindWert* ein XML Fragment, wird er als Kindknoten eingefügt:

  ```
  Reference:=DOM Append XML child node(elementRef;XML ELEMENT;"<child>simon</child><child>eva</child>")
  ```

  Ergebnis:

  ```
  <parent>      <child>simon</child>      <child>eva</child> </parent>
  ```

- andernfalls wird ein neues leeres Kindelement angefügt

  ```
  Reference:=DOM Append XML child node(elementRef;XML ELEMENT;"tbreak")
  ```

  Ergebnis:

  ```
  <parent>      <tbreak/> </parent>
  ```

Ist der Inhalt von *KindWert* nicht gültig, wird ein Fehler zurückgegeben.

DOM Append XML element

DOM Append XML element (ZielElementRef ; QuellElementRef) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ZielElementRef	Text	→	Referenz des Eltern XML Elements
QuellElementRef	Text	→	Referenz auf das anzufügende XML Element
Funktionsergebnis	Text	←	Referenz des neuen XML Elements

Beschreibung

Die Funktion **DOM Append XML element** fügt ein neues XML Element zu den Kindern des XML Elements hinzu, mit der im Parameter *ZielElementRef* übergebenen Referenz.

Das anzufügende Element übergeben Sie in *QuellElementRef*. Es muss als Referenz eines vorhandenen XML Elements in einem DOM Baum übergeben werden. Es wird nach dem letzten vorhandenen Kindelement von *ZielElementRef* angefügt.

Beispiel

Siehe Beispiel zur Funktion **DOM Insert XML element**.

DOM CLOSE XML

DOM CLOSE XML (ElementRef)

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz des XML Root Elements

Beschreibung

Der Befehl **DOM CLOSE XML** gibt den Speicherplatz frei, den das XML Objekt, definiert durch *ElementRef*, belegt.
Ist *ElementRef* kein XML Root Objekt, wird ein Fehler erzeugt.

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf, wird sie auf 0 (Null) gesetzt.

DOM Count XML attributes

DOM Count XML attributes (ElementRef) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	→	Referenz auf XML Elemente
Funktionsergebnis	Lange Ganzzahl	←	Anzahl Attribute

Beschreibung

Die Funktion **DOM Count XML attributes** gibt die Anzahl der XML Attribute im XML Element zurück, definiert durch *ElementRef*. Weitere Informationen zu XML Attributen finden Sie im Abschnitt **Überblick über XML DOM Befehle**.

Beispiel

Vor Auffinden der Werte von Elementen im Array wollen Sie die Anzahl der Attribute im nachfolgenden XML Element wissen:

```
<?xml version="1.0" ?>
- <LINES>
    <LINE N="1" Font="Verdana" size="10">this is a line</LINE>
    <LINE N="2" Font="charcoal" size="12">and another line</LINE>
    <LINE N="3" Font="Verdana" size="18">and we stop here</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
```

Im obigen Beispiel ist $numAttributes gleich 3, tAttrib{1} enthält “Font”, tAttrib{2} enthält “N ”, tAttrib{3} enthält “size”, tValAttrib enthält "Verdana", "1" und "10".

Hinweis: Die Nummer entspricht nicht der Position des Attributs in der Ansicht der XML Datei. Bei dem in 4D integrierten XML-Parser gibt der Index seine jeweilige Position in alphabetischer Reihenfolge an. Sie richtet sich nach dem Namen des Attributs, aufsteigend sortiert.

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf, wird sie auf 0 (Null) gesetzt.

DOM Count XML elements

DOM Count XML elements (ElementRef ; ElementName) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	→	Referenz des XML Elements
ElementName	String	→	Name des zu zählenden XML Elements
Funktionsergebnis	Lange Ganzzahl	←	Anzahl der Elemente

Beschreibung

Die Funktion **DOM Count XML elements** gibt die Anzahl der "Kind" Elemente zum Hauptelement *ElementRef* mit Namen *ElementName* zurück.

Hinweis: Standardmäßig berücksichtigt **DOM Count XML elements** im Parameter *ElementName* Groß- und Kleinschreibung (in Übereinstimmung mit xml). Über den Selector XML DOM case sensitivity des Befehls **XML SET OPTIONS** können Sie die Groß- und Kleinschreibung der Funktion steuern.

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf, wird sie auf 0 (Null) gesetzt.

DOM Create XML element

DOM Create XML element (ElementRef ; xPath {; attrName ; attrWert} {; attrName2 ; attrWert2 ; ... ; attrNameN ; attrWertN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf Root XML Element
xPath	Text	➡	Pfad xPath des zu erstellenden XML Elements
attrName	String	➡	Zu setzendes Attribut
attrWert	String, Boolean, Lange Ganzzahl, Zahl, Zeit, Datum	➡	Wert des neuen Attributs
Funktionsergebnis	String	⮌	Referenz auf erstelltes XML Element

Beschreibung

Die Funktion **DOM Create XML element** erstellt ein neues Element im XML Element (definiert in *ElementRef*) im Pfad (definiert in *xPath)* und fügt bei Bedarf Attribute hinzu.

In *ElementRef* übergeben Sie die Referenz auf das Root Element, z.B. erstellt mit **DOM Create XML Ref**.

In *xPath* übergeben Sie den Zugriffspfad im XML Format des zu erstellenden Elements. In diesem Fall ist das XPath Format verwendbar. Weitere Informationen dazu finden Sie im Abschnitt **Überblick über XML DOM Befehle**. Gibt es keine Elemente des Pfads, werden sie angelegt.

Im Parameter *xPath* können Sie direkt den Namen eines Eintrags übergeben, um vom aktuellen Eintrag einen Untereintrag zu erstellen (siehe Beispiel 3).

Hinweis: Haben Sie einen oder mehrere Namensbereiche für den Baum gesetzt, definiert in *ElementRef* (siehe Befehl **DOM Create XML Ref**), muss dem Parameter *xPath* der Name des verwendeten Bereichs vorangestellt werden (zum Beispiel "MyNameSpace:MyElement").

In den optionalen Parametern *attrName* und *attrWert* können Sie Paare Attribute/Attributwerte in Form von Variablen, Feldern oder tatsächlichen Werten übergeben. Sie können beliebig viele Paare übergeben.

Sie können jetzt in *attrWert* auch einen Wert übergeben, der nicht vom Typ Text ist, das kann Boolean, Ganzzahl, Zahl, Datum oder Zeit sein. 4D steuert die Umwandlung in Text gemäß den folgenden Vorgaben:

Typ	Beispiel für konvertierten Wert
Boolean	"Wahr" oder "Falsch" (nicht übersetzt)
Ganzzahl	"123456"
Zahl	"12.34" (Dezimaltrenner ist immer ".")
Datum	"2006-12-04T00:00:00Z" (RFC 3339 Standard)
Zeit	"5233" (Anzahl Sekunden)

Die Funktion gibt als Ergebnis die XML Referenz des erstellten Elements zurück.

Beispiel 1

Sie wollen folgendes Element erstellen:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <RootElement>  <Elem1>  <Elem2>  <Elem3> </Elem3>
    <Elem3> </Elem3>  </Elem2>  </Elem1> </RootElement>
```

Dazu schreiben Sie:

```
C_TEXT(vRootRef;vElemRef)
vRootRef:=DOM Create XML Ref("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vElemRef:=DOM Create XML element(vRootRef;vxPath)
vxPath:="RootElement/Elem1/Elem2/Elem3[2]"
vElemRef:=DOM Create XML element(vRootRef;vxPath)
```

Beispiel 2

Sie wollen folgendes Element mit Attributen erstellen:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <RootElement>  <Elem1>  <Elem2>  <Elem3 Font=Verdana
Size=10> </Elem3>  </Elem2>  </Elem1> </RootElement>
```

Dazu schreiben Sie:

```
C_TEXT(vRootRef;vElemRef)
C_TEXT($aAttrName1;$aAttrName2;$aAttrVal1;$aAttrVal2)
$aAttrName1:="Font"
```

```
$aAttrName2:="Size"
$aAttrVal1:="Verdana"
$aAttrVal2:="10"

vRootRef:=DOM Create XML Ref("RootElement")
vxPath:="/RootElement/Elem1/Elem2/Elem3"
vElemRef:=DOM Create XML element(vRootRef;vxPath;$aAttrName1;$aAttrVal1;
$aAttrName2;$aAttrVal2)
```

Beispiel 3

Sie wollen folgende Struktur erstellen und exportieren:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <Root>      <Elem1>Hello</Elem1> </Root>
```

Dazu verwenden Sie die Syntax, die auf einem einfachen Eintragsnamen basiert. Dazu schreiben Sie:

```
C_TEXT($root)
C_TEXT($ref1)

$root:=DOM Create XML Ref("Root")
$ref1:=DOM Create XML element($root;"Elem1")
DOM SET XML ELEMENT VALUE($ref1;"Hello")
DOM EXPORT TO FILE($root;"mydoc.xml")
DOM CLOSE XML($root)
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

Fehlerverwaltung

Ein Fehler wird erzeugt, wenn:

- Die Referenz auf das Root Element ungültig ist.
- Der Name des zu erstellenden Elements ungültig ist, z.B. wenn er mit einer Zahl beginnt.

DOM Create XML element arrays

DOM Create XML element arrays (ElementRef ; xPath {; attrNameArray ; attrWerteArray} {; attrNameArray2 ; attrWerteArray2 ; ... ; attrNameArrayN ; attrWerteArrayN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	Text	➡	Referenz des XML Root Elements
xPath	Text	➡	Pfad XPath des zu erstellenden XML Elements
attrNameArray	Array String	➡	Array der Attributnamen
attrWerteArray	Array String	➡	Array der Attributwerte
Funktionsergebnis	Text	↩	Referenz des erstellten XML Elements

Beschreibung

Die Funktion **DOM Create XML element arrays** fügt im XML Element *ElementRef* ein neues Element hinzu, sowie optional Attribute und ihre Werte im Formular der Arrays.

Diese Funktion ist - mit Ausnahme der Unterstützung von Arrays - identisch mit der Funktion **DOM Create XML element**. Ausführliche Informationen dazu finden Sie in der Beschreibung zu dieser Funktion.

Optional können Sie diese Funktion verwenden, um mehrere Paare von Attributen und Attributwerte als Arrays in den Parametern *attrNamenArray* und *attrWerteArray* zu übergeben. Sie können Arrays vom Typ Text, Datum, Zeit, Zahl, BLOB und Bild übergeben. 4D führt automatisch die erforderlichen Konvertierungen durch; über den Befehl **XML SET OPTIONS** können Sie diese Konvertierungen verändern.

Die Arrays müssen zuvor erstellt worden sein und als Paar funktionieren. Sie können beliebig viele Array-Paare übergeben und in jedem Paar beliebig viele Elemente.

Beispiel

Wir wollen folgendes Element erstellen:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <RootElement>     <Elem1>          <Elem2>              <Elem3 Font="Verdana"
Size="10" Style="Bold"></Elem3>          </Elem2>     </Elem1> </RootElement>
```

Dafür können Sie einfach schreiben:

```
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 (Root {; Namensraum} {; NameBerName ; NameBerWert} {; NameBerName2 ; NameBerWert2 ; ... ; NameBerNameN ; NameBerWertN}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Root	String	➡	Name des Root Elements
Namensraum	String	➡	Wert von Namensbereich
NameBerName	String	➡	Name von Namensbereich
NameBerWert	String	➡	Wert von Namensbereich
Funktionsergebnis	String	⬅	Referenz auf Root XML Element

Beschreibung

Die Funktion **DOM Create XML Ref** erstellt im Speicher einen leeren XML Baum und gibt dessen Referenz zurück.

Im Parameter *Root* übergeben Sie den Namen des Root Elements im XML Baum.

Übergeben Sie im optionalen Parameter Namensbereich die Deklaration des Wertes für den Namensbereich des Baums, z.B. “http:www.4d.com”.
Beachten Sie, dass Sie dem Parameter *Root* den Namen des Namensbereichs, gefolgt von einem Doppelpunkt, voranstellen können, z.B. “MeinNamensbereich:MeinRoot“.
In diesem Fall ist der Parameter Namensbereich, der den Wert Namensbereich angibt, zwingend.

Hinweis: Über Namensbereich können Sie sicherstellen, dass die XML Variablennamen einmalig sind. Das ist in der Regel eine URL wie http://www.mysite.com/myurl. Die URL muss nicht unbedingt gültig sein, sie muss jedoch einmalig sein.

Sie können über das Paar *NameBerName*/*NameBerWert* einen oder mehrere zusätzliche Namensbereiche im angelegten XML Baum festlegen. Sie können beliebig viele Paare übergeben.

Wichtig: Benötigen Sie die Funktion nicht länger, denken Sie daran, den Befehl **DOM CLOSE XML** mit dieser Referenz aufzurufen, um den Speicher wieder freizumachen.

Beispiel 1

Einen einzelnen XML Baum erstellen:

```
C_TEXT(vElemRef)
vElemRef:=DOM Create XML Ref("MyRoot")
```

Diese Methode erzeugt folgendes Ergebnis:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?> <MyRoot/>
```

Beispiel 2

Einen XML Baum mit einem einzelnen Namensbereich erstellen:

```
C_TEXT(vElemRef)
$Root:="MyNameSpace:MyRoot"
$Namespace:="http://www.4D.com/tech/namespace"
vElemRef:=DOM Create XML Ref($Root;$Namespace)
```

Diese Methode erzeugt folgendes Ergebnis:

```
<MyNameSpace:MyRoot xmlns:MyNameSpace="http://www.4D.com/tech/namespace"/>
```

Beispiel 3

Einen XML Baum mit mehreren Namensbereichen erstellen:

```
C_TEXT(vElemRef)
C_TEXT($aNSName1;$aNSName2;$aNSValue1;$aNSValue2)
$Root:="MyNameSpace:MyRoot"
$Namespace:="http://www.4D.com/tech/namespace"
$aNSName1:="NSName1"
$aNSName2:="NSName2"
$aNSValue1:="http://www.4D.com/Prod/namespace"
```

```
$aNSValue2:="http://www.4D.com/Mkt/namespace"
vElemRef:=DOM Create XML Ref($Root;$Namespace;$aNSName1;$aNSValue1;$aNSName2;$aNSValue2)
```

Diese Methode erzeugt folgendes Ergebnis :

```
<MyNameSpace:MyRoot xmlns:MyNameSpace="http://www.4D.com/tech/nameSpace"
NSName1="http://www.4D.com/Prod/namespace" NSName2="http://www.4D.com/Mkt/namespace"/>
```

Systemvariablen und Mengen

Bei korrekt ausgeführter Funktion wird die Variable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird generiert.

DOM EXPORT TO FILE

DOM EXPORT TO FILE (ElementRef ; Dateipfad)

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf das Root XML Element
Dateipfad	Text	➡	Voller Zugriffspfad der Datei

Beschreibung

Der Befehl **DOM EXPORT TO FILE** speichert einen XML Baum in eine Datei auf der Festplatte.

In *ElementRef* übergeben Sie die Referenz auf das zu exportierende Root Element.

In *DateiPfad* übergeben Sie den kompletten Zugriffspfad der Exportdatei. Ist die Datei noch nicht vorhanden, wird sie erstellt.

Übergeben Sie nur den Dateinamen ohne Zugriffspfad, wird nach ihr gesucht oder die Datei neben der Strukturdatei angelegt.

Übergeben Sie einen leeren String (""), erscheinen der Standard-Öffnen Dialog und der Dialog für neue Datei.

Hinweise zum Bearbeiten von Zeichen für Zeilenende

In XML sind Zeilenumbrüche (CR) nicht signifikant, egal ob sie innerhalb oder zwischen XML Elementen liegen. XML verwendet intern standardmäßig Zeichen für Zeilenvorschub (LF) als Zeilentrenner.

Bei Import- und Exportoperationen lassen sich Zeichen für Zeilenvorschub (LF) konvertieren. Der XML Parser ersetzt während einem Import CRLF Zeichen (standard Zeilenumbruch unter Windows) mit Zeichen für Zeilenvorschub (LF). Während dem Export werden Zeichen für Zeilenvorschub (LF) auf MacOS durch CR Zeichen ersetzt, unter Windows durch CRLF Zeichen.

Wollen Sie Zeilenumbrüche (CR) beibehalten, müssen Sie sie in ein XML CDATA Element einbinden, so dass sie nicht vom XML Parser bearbeitet werden. Anstelle von CRLF Zeichen können Sie auch "
" verwenden. Das sind explizite Zeilenumbrüche, die nicht vom Parser bearbeitet werden.

Beispiel

Diese Anweisung speichert den Baum *vElemRef* in der Datei MyDoc.xml:

```
DOM EXPORT TO FILE(vElemRef;"C:\\folder\MyDoc.xml")
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

Fehlerverwaltung

Ein Fehler wird zurückgegeben, wenn:

- Die Elementreferenz ungültig ist,
- Der angegebene Zugriffspfad ungültig ist,
- Das Speicher-Volume einen Fehler zurückgibt, z.B. Platz auf Festplatte reicht nicht aus.

DOM EXPORT TO VAR

DOM EXPORT TO VAR (ElementRef ; vXmlVar)

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf das Root XML Element
vXmlVar	Text, BLOB	➡	Variable, die XML Baum empfangen muss

Beschreibung

Der Befehl **DOM EXPORT TO VAR** sichert einen XML Baum in einer Variablen vom Typ Text oder BLOB.

In *ElementRef* übergeben Sie die Referenz auf das Root Element.

In *vXmlVar* übergeben Sie den Namen der Variablen, die den XML Baum enthalten muss. Sie muss vom Typ Text oder BLOB sein. Der Typ richtet sich nach dem, was Sie als nächstes planen oder nach erreichbaren Größe für den Baum. Beachten Sie, dass Variablen vom Typ Text im Nicht-Unicode Modus auf 32.000 Zeichen, im Unicode Modus auf 2 GB begrenzt sind.

Verwenden Sie zum Speichern von *ElementRef* eine Textvariable im Nicht-Unicode Modus, wird der aktuelle Mac Zeichensatz verwendet, z.B. Mac Roman. Folglich verliert der zurückgegebene Text seine ursprüngliche Codierung (Codierung = xxx). In diesem Fall ermöglicht die Variable *vVarXml*, den Code zu sehen oder zu speichern, aber NICHT ein gültiges XML Dokument zu erstellen, z.B. über den Befehl **SEND PACKET**.

Im Unicode Modus wird die Original-Codierung in der Variable beibehalten.

Hinweise zum Bearbeiten von Zeichen für Zeilenende

In XML sind Zeilenumbrüche (CR) nicht signifikant, egal ob sie innerhalb oder zwischen XML Elementen liegen. XML verwendet intern standardmäßig Zeichen für Zeilenvorschub (LF) als Zeilentrenner.

Bei Import- und Exportoperationen lassen sich Zeichen für Zeilenvorschub (LF) konvertieren. Der XML Parser ersetzt während einem Import CRLF Zeichen (standard Zeilenumbruch unter Windows) mit Zeichen für Zeilenvorschub (LF). Während dem Export werden Zeichen für Zeilenvorschub (LF) auf MacOS durch CR Zeichen ersetzt, unter Windows durch CRLF Zeichen.

Wollen Sie Zeilenumbrüche (CR) beibehalten, müssen Sie sie in ein XML CDATA Element einbinden, so dass sie nicht vom XML Parser bearbeitet werden. Anstelle von CRLF Zeichen können Sie auch "
" verwenden. Das sind explizite Zeilenumbrüche, die nicht vom Parser bearbeitet werden.

Beispiel

Diese Anweisung speichert den Baum *vElementRef* in einer Text Variablen:

```
C_TEXT(vtMyText)
DOM EXPORT TO VAR(vElemRef;vtMyText)
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben, z.B. bei ungültiger Referenz auf das Element.

DOM Find XML element

DOM Find XML element (ElementRef ; xPath {; arrElementRefs}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf XML Element
xPath	Text	➡	Pfad xPath auf das zu suchende Element
arrElementRefs	Array String	⬅	Liste der gefundenen Element Referenzen (falls zutreffend)
Funktionsergebnis	String	➲	Referenz des gefundenen Elements (wenn zutreffend)

Beschreibung

Die Funktion **DOM Find XML element** sucht in einer XML Struktur nach einem spezifischen XML Element. Die Suche startet bei dem Element, definiert durch *ElementRef*.

In *xPath* setzen Sie den zu suchenden XML Knoten (node), ausgedrückt als "XPath Notation" (siehe **Überblick über XML DOM Befehle**). Auch indizierte Elemente lassen sich verwenden.

Hinweis: Suchläufe berücksichtigen in Übereinstimmung mit XML Standards Klein- und Großschreibung.

Die Funktion gibt die XML Referenz auf das gesuchte Element zurück.

Ist das Array *arrElementRefs* übergeben, füllt die Funktion es mit der Liste der gefundenen XML Referenzen und gibt als Ergebnis das erste Element des Array zurück. Dieser Parameter ist hilfreich, wenn an der Stelle, definiert durch den Parameter *xPath*, mehrere Elemente mit demselben Namen existieren.

Beispiel 1

Mit dieser Anweisung können Sie rasch nach einem XML Element suchen und seinen Wert anzeigen:

```
vFound:=DOM Find XML element(vElemRef;"Items/Book[15]/Title")
DOM GET XML ELEMENT VALUE(vFound;value)
ALERT("Der Wert des Elements ist: \""+value+"\"")
```

Dieselbe Suche lässt sich auch folgendermaßen ausführen:

```
vFound:=DOM Find XML element(vElemRef;"Items/Book[15]")
vFound:=DOM Find XML element(vFound;"Book/Title")
DOM GET XML ELEMENT VALUE(vFound;value)
ALERT("Der Wert des Elements ist: \""+value+"\"")
```

Hinweis: Wie Sie in obigem Beispiel sehen können, muss der Pfad *xPath* immer mit dem Namen des aktuellen Elements beginnen. Dieses Detail ist wichtig zur Verwaltung relativer *xPath* Pfade.

Beispiel 2

Wir gehen von folgender XML Struktur aus:

```
<Root>  <Elem1>  <Elem2>aaa</Elem2>  <Elem2>bbb</Elem2>  <Elem2>ccc</Elem2>  </Elem1> </Root>
```

Der folgende Code schreibt die Referenz auf jedes Element *Elem2* in Array *arrAfound*:

```
ARRAY TEXT(arrAfound;0)
vFound:=DOM Find XML element(vElemRef;"/Root/Elem1/Elem2";arrAfound)
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

Fehlerverwaltung

Ein Fehler wird zurückgegeben, wenn:

- Die Referenz auf das Element ungültig ist
- Der angegebene Pfad x Path ungültig ist.

DOM Find XML element by ID

DOM Find XML element by ID (ElementRef ; id) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	→	XML Element Referenz
id	String	→	Wert des id-Attributs des zu suchenden Elements
Funktionsergebnis	String	↩	Referenz des gefundenen Elements (falls zutreffend)

Beschreibung

Die Funktion **DOM Find XML element by ID** sucht in einem XML Dokument nach dem Element, dessen id-Attribut dem im Parameter *id* übergebenen Wert entspricht.

In *ElementRef* übergeben Sie die Referenz des Root-Elements aus dem XML Dokument. Die Suche wird, ausgehend vom Root-Element im gesamten XML Dokument durchgeführt. Übergeben Sie hier die Referenz eines anderen Elements, wird die Suche dennoch beim Root-Element gestartet.

Die Funktion gibt als Ergebnis die XML Referenz des gefundenen Elements zurück.

Warnung: In einem XML Dokument weist das id-Attribut jedem Element im XML Dokuments einen Wert zu. Der Wert des id-Attributs muss ein gültiger XML Name und innerhalb des XML Dokuments einmalig sein. Ist die Einmaligkeit nicht gegeben, ist das Ergebnis von **DOM Find XML element by ID** nicht garantiert. In der Regel wird bei nicht-einmaligen id-Attributen der erste Treffer des gesuchten Wertes gefunden.

DOM Get first child XML element

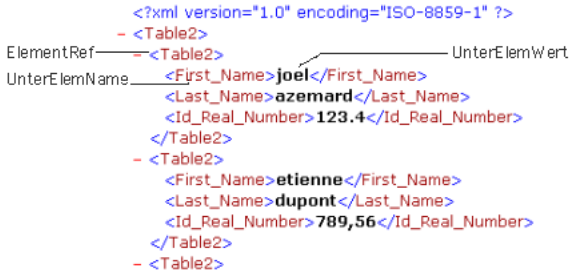
DOM Get first child XML element (ElementRef {; UnterElemName {; UnterElemWert}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf XML Element
UnterElemName	String	⬅	Name des gewählten XML Unterelements
UnterElemWert	String	⬅	Wert des XML Unterelements
Funktionsergebnis	String	↩	XML Referenz

Beschreibung

Die Funktion **DOM Get first child XML element** gibt die Referenz auf das erste „Kind" des als Referenz übergebene XML Element zurück. Sie können diese als Referenz in anderen XML Befehlen zum Parsen verwendet.

Die Parameter *UnterElemName* und *UnterElemWert* empfangen Name und Wert des Unterelements.



Beispiel 1

Referenz auf das erste XML Element des Eltern-Root wiederfinden. Die XML Struktur (C:\\import.xml) wird zuerst in ein BLOB geladen:

```
C_BLOB(myBlobVar)
C_TEXT($xml_Parent_Ref;$xml_Child_Ref)

DOCUMENT TO BLOB("c:\\import.xml";myBlobVar)
$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get first child XML element($xml_Parent_Ref)
```

Beispiel 2

Referenz, Name und Wert des ersten XML Elements des Eltern-Root wiederfinden. Die XML Struktur (C:\\import.xml) wird zuerst in ein BLOB geladen:

```
C_BLOB(myBlobVar)
C_TEXT($xml_Parent_Ref;$xml_Child_Ref)
C_TEXT($childName;$childValue)

DOCUMENT TO BLOB("c:\\import.xml";myBlobVar)
$xml_Parent_Ref:=DOM Parse XML variable(myBlobVar)
$xml_Child_Ref:=DOM Get first child XML element
($xml_Parent_Ref;$childName;$childValue)
```

Systemvariablen und Mengen

Wurde die Funktion korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

DOM Get last child XML element

DOM Get last child XML element (ElementRef {; UnterElemName {; UnterElemWert}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	→	Referenz auf XML Element
UnterElemName	String	←	Name des Kindelements
UnterElemWert	String	←	Wert des Kindelements
Funktionsergebnis	String	↩	Referenz auf XML Element

Beschreibung

Die Funktion **DOM Get last child XML element** gibt eine XML Referenz auf das letzte Unterelement (Kind) des XML Elements zurück, das als Referenz in *ElementRef* übergeben wurde. Diese Referenz lässt sich in den anderen XML Elementen zum Analysieren verwenden.

Sind die optionalen Parameter *UnterElemName* und *UnterElemWert* übergeben, erhalten sie jeweils Name und Wert des Unterelements (Kind).

Beispiel

Wiederfinden der Referenz des letzten XML Elements des Haupt-Root (parent root). Die XML Struktur (C:\\import.xml) wird zuvor in ein BLOB geladen.

```
C_BLOB(myBlobVar)
C_TEXT($ref_XML_Parent;$ref_XML_Child)
C_TEXT($childName;$childValue)

DOCUMENT TO BLOB("c:\\import.xml";myBlobVar)
$ref_XML_Parent:=DOM Parse XML variable(myBlobVar)
$ref_XML_Child:=DOM Get last child XML element($ref_XML_Parent;$childName;$childValue)
```

Systemvariablen und Mengen

Wurde die Funktion korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

DOM Get next sibling XML element

DOM Get next sibling XML element (ElementRef {; GeschwisterElemName {; GeschwisterElemWert}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf XML Element
GeschwisterElemName	String	⬅	Name des Geschwister XML Elements
GeschwisterElemWert	String	⬅	Wert des Geschwister XML Elements
Funktionsergebnis	String	↩	XML Referenz

Beschreibung

Die Funktion **DOM Get next sibling XML element** gibt eine Referenz auf das nächste Geschwisterelement des als Referenz übergebenen XML Elements zurück. Sie können diese als Referenz in anderen XML Befehlen zum Analysieren verwenden.

Die Parameter *GeschwisterElemName* und *GeschwisterElemWert* empfangen Name und Wert des Geschwisterelements.

Diese Funktion analysiert alle "Kinder" des als XML Element übergebenen Parameters. Nach dem letzten Geschwisterelement wird die Systemvariable OK auf 0 (Null) gesetzt.

Beispiel 1

Referenz des XML Elements wiederfinden, das auf das als Parameter übergebene Geschwisterelement folgt:

```
C_TEXT($xml_Parent_Ref;$next_XML_Ref)
$next_XML_Ref:=DOM Get next sibling XML element($xml_Parent_Ref)
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <Table2>
Hauptelement —— <Table2>
    <First_Name>joel</First_Name>
    <Last_Name>azemard</Last_Name>
    <Id_Real_Number>123.4</Id_Real_Number>
  </Table2>
Nächstes Element —— - <Table2>
    <First_Name>etienne</First_Name>
    <Last_Name>dupont</Last_Name>
    <Id_Real_Number>789,56</Id_Real_Number>
  </Table2>
- <Table2>
```

Beispiel 2

Referenz in einer Schleife auf alle XML Kinderelemente wiederfinden, die auf das als Parameter übergebene Hauptelement folgen, beginnend mit dem 1. Kind:

```
C_TEXT($xml_Parent_Ref;$first_XML_Ref;$next_XML_Ref)

$first_XML_Ref:=DOM Get first child XML element($xml_Parent_Ref)
$next_XML_Ref:=$first_XML_Ref
While(OK=1)
   $next_XML_Ref:=DOM Get next sibling XML element($next_XML_Ref)
End while
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <Table2>
Hauptelement —— <Table2>
    <First_Name>joel</First_Name>  —— 1. Element
    <Last_Name>azemard</Last_Name>  —— Nächstes Element (Schleife 1)
    <Id_Real_Number>123.4</Id_Real_Number>  —— Nächstes Element (Schleife 2)
  </Table2>
- <Table2>
    <First_Name>etienne</First_Name>
    <Last_Name>dupont</Last_Name>
    <Id_Real_Number>789,56</Id_Real_Number>
  </Table2>
- <Table2>
```

Systemvariablen und Mengen

Wurde die Funktion korrekt ausgeführt und ist das analysierte Element nicht das letzte Geschwisterelement des Hauptelements, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf oder ist das analysierte Element das letzte Geschwisterelement des Hauptelements, wird die Systemvariable auf 0 gesetzt.

DOM Get parent XML element

DOM Get parent XML element (ElementRef {; HauptElemName {; HauptElemWert}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	→	Referenz auf XML Element
HauptElemName	String	←	Name des XML Hauptelements
HauptElemWert	String	←	Wert des XML Hauptelements
Funktionsergebnis	String	↩	Referenz auf das XML Hauptelement

Beschreibung

Die Funktion **DOM Get parent XML element** gibt die XML Referenz auf das Hauptelement des XML Elements zurück, das als Referenz in *ElementRef* übergeben wird. Diese Referenz lässt sich in den anderen XML Elementen zum Analysieren verwenden.

Sind die optionalen Parameter *HauptElemName* und *HauptElemWert* übergeben, erhalten diese jeweils Name und Wert des Hauptelements.

Übergeben Sie in *ElementRef* ein Root Element, gibt die Funktion die Referenz "#document" zurück. Der Dokumentknoten ist das Hauptelement eines Root Elements.
Verwenden Sie diesen Befehl in einem Dokumentknoten, gibt die Funktion eine Nullreferenz ("0000000000000000") zurück und die OK Variable wird auf 0 gesetzt.

Systemvariablen und Mengen

Wurde die Funktion korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0.

DOM Get previous sibling XML element

DOM Get previous sibling XML element (ElementRef {; GeschwisterElemName {; GeschwisterElemWert}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	→	Referenz auf XML Element
GeschwisterElemName	String	←	Name des XML Geschwisterelements
GeschwisterElemWert	String	←	Wert des XML Geschwisterelements
Funktionsergebnis	String	↩	Referenz auf das XML Geschwisterelements

Beschreibung

Die Funktion **DOM Get previous sibling XML element** gibt eine Referenz auf das vorige Geschwisterelement des XML Elements zurück. Diese Referenz lässt sich in den anderen XML Elementen zum Analysieren verwenden.

Sind die optionalen Parameter *GeschwisterElemName* und *GeschwisterElemWert* übergeben, erhalten sie jeweils Name und Wert des vorigen Geschwisterelements.

Diese Funktion dient zum Navigieren in den Unterelementen/"Kindern" eines XML Elements.

Die Systemvariable OK wird vor dem ersten Geschwisterelement auf 0 (Null) gesetzt.

Systemvariablen und Mengen

Wurde die Funktion korrekt ausgeführt und ist das Element, auf das die Referenz verweist, nicht das erste Unterelement (Kind) der Struktur, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf, oder ist das analysierte Element das erste Unterelement (Kind) der Struktur, wird sie auf 0 (Null) gesetzt.

DOM Get Root XML element

DOM Get Root XML element (ElementRef) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	→	XML Element Referenz
Funktionsergebnis	String	←	Referenz des Root Elements oder "" bei einem Fehler

Beschreibung

Die Funktion **DOM Get Root XML element** gibt eine Referenz auf das Root Element des Dokuments zurück, zu dem das im Parameter übergebene XML Element gehört. Diese Referenz kann mit anderen Befehlen zur XML Analyse verwendet werden.

DOM GET XML ATTRIBUTE BY INDEX

DOM GET XML ATTRIBUTE BY INDEX (ElementRef ; attrIndex ; attrName ; attrWert)

Parameter	Typ		Beschreibung
ElementRef	String	➡	XML Element Referenz
attrIndex	Lange Ganzzahl	➡	Attribut Indexnummer
attrName	Variable	⬅	Attributname
attrWert	Variable	⬅	Attributwert

Beschreibung

Der Befehl **DOM GET XML ATTRIBUTE BY INDEX** erhält den Namen eines Attributs, definiert durch seine Indexnummer und durch seinen Wert.

In *ElementRef* übergeben Sie die Referenz eines XML Elements, in *attrIndex* die Indexnummer des Attributs, dessen Namen Sie wissen wollen. Der Name wird in *attrName* zurückgegeben, sein Wert in *attrWert*. 4D versucht, den erhaltenen Wert in denselben Typ umzuwandeln wie die als Parameter übergebene Variable.

Hinweis: Die Nummer entspricht nicht der Platzierung des Attributs in der Ansicht der XML Datei. Bei dem in 4D integrierten XML-Parser gibt der Index seine jeweilige Position in alphabetischer Reihenfolge an. Sie richtet sich nach dem Namen des Attributs, aufsteigend sortiert.

Ist der in *attrIndex* übergebene Wert größer als die Anzahl der Attribute im XML Element, wird ein Fehler zurückgegeben.

Beispiel

Siehe Beispiel unter dem Befehl **DOM Count XML attributes**.

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf, wird sie auf 0 (Null) gesetzt.

DOM GET XML ATTRIBUTE BY NAME

DOM GET XML ATTRIBUTE BY NAME (ElementRef ; attrName ; attrWert)

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf XML Element
attrName	String	➡	Attributname
attrWert	Variable	⬅	Attributwert

Beschreibung

Der Befehl **DOM GET XML ATTRIBUTE BY NAME** erhält den Wert eines Attributs, definiert durch *attrName*.

In *ElementRef* übergeben Sie die Referenz eines XML Elements, in *attrName* den Namen des Attributs, dessen Wert Sie wissen wollen. Der Wert wird in *attrWert* zurückgegeben. 4D versucht, den erhaltenen Wert in denselben Typ umzuwandeln wie die als Parameter übergebene Variable.

Gibt es kein Attribut *attrName* im XML Element, wird ein Fehler zurückgegeben. Bei mehreren gleichnamigen Attributen im XML Element wird nur der Wert des ersten Attributs zurückgegeben.

Beispiel

Diese Methode findet den Wert eines XML Attributs über seinen Namen:

```
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 GET XML ATTRIBUTE BY NAME($xml_Child_Ref;"N";$LineNum)
```

Wird diese Methode auf das Beispiel angewandt, enthält $LineNum den Wert 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>
```

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf, wird sie auf 0 (Null) gesetzt.

DOM GET XML CHILD NODES

DOM GET XML CHILD NODES (ElementRef ; KindTypenArr ; KnotenRefsArr)

Parameter	Typ		Beschreibung
ElementRef	Text	➡	XML Element Referenz
KindTypenArr	Array Lange Ganzzahl	⬅	Typen der Kindknoten
KnotenRefsArr	Array Text	⬅	Referenzen oder Werte der Kindknoten

Beschreibung

Der Befehl **DOM GET XML CHILD NODES** gibt die Typen und Referenzen bzw. Werte aller Kindknoten des in *ElementRef* definierten XML Elements zurück.
Die Typen der Kindknoten werden im Array *KindTypenArr* zurückgegeben. Sie können die vom Befehl zurückgegebenen Werte mit den nachfolgenden Konstanten unter dem Thema **XML** vergleichen:

Konstante	Typ	Wert
XML comment	Lange Ganzzahl	2
XML processing instruction	Lange Ganzzahl	3
XML DATA	Lange Ganzzahl	6
XML CDATA	Lange Ganzzahl	7
XML DOCTYPE	Lange Ganzzahl	10
XML ELEMENT	Lange Ganzzahl	11

Weitere Informationen dazu finden Sie unter der Funktion **DOM Append XML child node**.
Das Array *KnotenRefsArr* erhält je nach Typ die Werte bzw. Referenzen der Elemente (Inhalte oder Anweisungen).

Beispiel

Wir nehmen folgende XML Struktur:

```
<myElement>Hallo<br/>New<br/>York</myElement>
```

Nach Ausführen folgender Anweisungen:

```
elementRef:=DOM Find XML element($root;"myElement")
DOM GET XML CHILD NODES(elementRef;$typeArr;$textArr)
```

... enthalten die Arrays *$typeArr* und *$textArr* folgende Werte:
$typeArr{1}=6 $textArr{1} = "Hallo"
$typeArr{2}=11 $textArr{2} = "AEF1233456878977" (element reference
)
$typeArr{3}=6 $textArr{3} = "New"
$typeArr{4}=11 $textArr{4} = "AEF1237897734568" (element reference
)
$typeArr{5}=6 $textArr{5} = "York"

DOM Get XML document ref

DOM Get XML document ref (ElementRef) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	Text	→	Referenz des vorhandenen Elements im DOM Baum
Funktionsergebnis	Text	↩	Referenz des ersten Elements eines DOM Baums (Dokumentknoten)

Beschreibung

Der Befehl **DOM Get XML document ref** findet erneut die Referenz des Elements "Document" des DOM Baums mit der in *ElementRef* übergebenen Referenz. Dieses Element ist das erste Element eines DOM Baumes; es ist der Elternteil des Root Elements.

Über die Referenz des Elements "Document" können Sie die Knoten "Doctype" und "Arbeitsanweisung" verwalten. Sie lassen sich nur mit den Befehlen **DOM Append XML child node** und **DOM GET XML CHILD NODES** verwenden.

Auf dieser Ebene können Sie nur Arbeitsanweisungen und Kommentare anhängen oder den Knoten *Doctype* ersetzen. Sie können dagegen keine Knoten vom Typ *CDATA* oder *Text* erstellen.

Beispiel

In diesem Beispiel wollen wir die DTD Deklaration des XML Dokuments finden:

```
C_TEXT($rootRef)
$rootRef:=DOM Parse XML source("")
If(OK=1)
   C_TEXT($documentRef)
 // wir suchen nach dem Knoten Document, denn diesem ist der Knoten
 // DOCTYPE vor dem Root Knoten zugewiesen.
   $documentRef:=DOM Get XML document ref($rootRef)
   ARRAY TEXT($typeArr;0)
   ARRAY TEXT($valueArr;0)
 // in diesem Knoten suchen wir nach dem Knoten DOCTYPE unter den Kindknoten
   DOM GET XML CHILD NODES($refDocument;$typeArr;$valueArr)
   C_TEXT($text)
   $text:=""
   $pos:=Find in array($typeArr;XML DOCTYPE)
   If($pos>-1)
 // Wir finden die DTD Deklaration wieder in $text
      $text:=$text+"Doctype: "+$valueArr{$pos}+Char(Carriage return)
   End if
   DOM CLOSE XML($rootRef)
End if
```

DOM Get XML element

DOM Get XML element (ElementRef ; ElementName ; Index ; ElementWert) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	→	Referenz des XML Elements
ElementName	String	→	Name des zu erhaltenden Elements
Index	Lange Ganzzahl	→	Indexnummer des zu erhaltenden Elements
ElementWert	Variable	←	Wert des Elements
Funktionsergebnis	String	↩	XML Referenz (16 Zeichen)

Beschreibung

Die Funktion **DOM Get XML element** gibt eine Referenz auf das Kindelement zu den Parametern *ElementName* und *Index* zurück.

Der Wert des Elements wird auch im Parameter *ElementWert* zurückgegeben.

Wird der Parameter *ElementWert* übergeben, enthält er den Wert des Elements.

Hinweis: Standardmäßig berücksichtigt **DOM Get XML element** im Parameter *ElementName* Groß- und Kleinschreibung (in Übereinstimmung mit xml). Über den Selector XML DOM case sensitivity des Befehls **XML SET OPTIONS** können Sie die Groß- und Kleinschreibung der Funktion steuern.

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf, wird sie auf 0 (Null) gesetzt.

DOM GET XML ELEMENT NAME

DOM GET XML ELEMENT NAME (ElementRef ; ElementName)

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz des XML Elements
ElementName	Variable	⬅	Name des Elements

Beschreibung

Der Befehl **DOM GET XML ELEMENT NAME** gibt im Parameter *ElementName* den Namen des XML Elements zurück, definiert durch *ElementRef*. Weitere Informationen zu XML Elementnamen finden Sie im Abschnitt **Überblick über XML DOM Befehle**.

Beispiel

Diese Methode gibt den Namen des Elements $xml_Element_Ref zurück:

```
C_TEXT($xml_Element_Ref)
C_TEXT($name)

DOM GET XML ELEMENT NAME($xml_Element_Ref;$name)
```

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf, wird sie auf 0 (Null) gesetzt.

DOM GET XML ELEMENT VALUE

DOM GET XML ELEMENT VALUE (ElementRef ; ElementWert {; cDATA})

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz des XML Elements
ElementWert	Variable	⬅	Wert des Elements
cDATA	Variable	⬅	Inhalt des Bereichs cDATA

Beschreibung

Der Befehl **DOM GET XML ELEMENT VALUE** gibt im Parameter *ElementWert* den Wert des XML Elements zurück, definiert durch *ElementRef*. 4D versucht, den erhaltenen Wert in denselben Typ umzuwandeln wie die als Parameter übergebene Variable.

Der optionale Parameter *cDATA* ermöglicht, den Inhalt des Bereichs cDATA des XML Elements *ElementRef* zu finden. Analog zum Parameter *ElementWert* versucht 4D, den erhaltenen Wert in denselben Typ umzuwandeln wie die als Parameter übergebene Variable.

Hinweis: Ist das Element, definiert durch *ElementRef*, ein BLOB, das vom Befehl **DATABASE_PATH** bearbeitet wird, wurde es automatisch in base64 codiert. Deshalb versucht dieser Befehl automatisch, es in base64 zu decodieren.

Beispiel

Diese Methode gibt den Namen des Elements $xml_Element_Ref zurück:

```
C_TEXT($xml_Element_Ref)
C_REAL($value)

DOM GET XML ELEMENT VALUE($xml_Element_Ref;$value)
```

Systemvariablen und Mengen

Wurde der Befehl korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt. Tritt ein Fehler auf, wird sie auf 0 (Null) gesetzt.

DOM Get XML information

DOM Get XML information (ElementRef ; xmlInfo) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ElementRef	String	→	Referenz des XML Root Elements
xmlInfo	Lange Ganzzahl	→	Typ der zu erhaltenden Information
Funktionsergebnis	String	↩	Wert der XML Information

Beschreibung

Der Befehl **DOM Get XML information** findet diverse Informationen zum XML Element, definiert durch *ElementRef*.

In *xmlInfo* übergeben Sie einen Code für den zu findenden Informationstyp. Sie können die vordefinierten Konstanten unter dem Thema **XML** verwenden:

Konstante	Typ	Wert	Kommentar
DOCTYPE Name	Lange Ganzzahl	3	Name des Root Elements, wie im DOCTYPE Marker definiert
Document URI	Lange Ganzzahl	6	URI von der DTD
Encoding	Lange Ganzzahl	4	Verwendete Codierung (UTF-8, ISO...)
PUBLIC ID	Lange Ganzzahl	1	Öffentlicher Identifier (FPI) der DTD, zu dem das Dokument passt (sofern das tag DOCTYPE xxx PUBLIC vorhanden ist).
SYSTEM ID	Lange Ganzzahl	2	System Identifier
Version	Lange Ganzzahl	5	Zugelassene XML Version

DOM Insert XML element

DOM Insert XML element (ZielElementRef ; QuellElementRef ; KindIndex) -> Funktionsergebnis

Parameter	Typ		Beschreibung
ZielElementRef	Text	➡	Referenz des Eltern XML Element
QuellElementRef	Text	➡	Einzufügende Referenz auf XML Element
KindIndex	Lange Ganzzahl	➡	Index von Kind des Zielelements, über dem das neue Element eingefügt werden soll.
Funktionsergebnis	Text	⮌	Referenz des neuen XML Elements

Beschreibung

Die Funktion **DOM Insert XML element** fügt ein neues XML Element unter den Kindelementen des XML Elements ein mit der im Parameter *ZielElementRef* übergebenen Referenz.

Das einzufügenden Element übergeben Sie in *QuellElementRef*. Es muss als Referenz eines vorhandenen XML Elements in einem DOM Baum übergeben werden.

In *KindIndex* bestimmen Sie das Kind des Elternelements, vor dem das neue Element eingefügt werden soll. In diesem Parameter übergeben Sie eine Indexnummer. Bei einem ungültigen Wert, wenn z.B. kein Kindelement diesen Index hat, wird das neue Element vor dem ersten Kind des Elternelements hinzugefügt.

Die Funktion gibt die Referenz auf das erhaltene XML Element zurück.

Beispiel

In der folgenden Struktur wollen wir das erste und zweite Buch umstellen:

```
<?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>
```

Dazu führen wir folgenden Code aus:

```
C_TEXT($rootRef)
$rootRef:=DOM Parse XML source("")  //XML Dokument auswählen
If(OK=1)
   C_TEXT($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 (Dokumentname {; Gültigkeitsprüfung {; dtd | schema}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Dokumentname	String	➡	Zugriffspfad für Dokument
Gültigkeitsprüfung	Boolean	➡	Wahr = Bestätigung durch DTD, Falsch = Keine Bestätigung
dtd | schema	String	➡	Speicherort des DTD oder XML Schemas
Funktionsergebnis	String	⬅	Referenz des XML Elements

Beschreibung

Die Funktion **DOM Parse XML source** analysiert ein Dokument mit einer XML Struktur und gibt eine Referenz auf dieses Dokument zurück. Die Funktion kann je nach Ergebnis das Dokument über DTD oder ein XML Schema (XSD-Dokument = XML Schema Definition) bestätigen bzw. nicht bestätigen. Das Dokument kann auf der Festplatte oder im Internet/Intranet liegen.

Hinweis: Die Ausführung der Funktion **DOM Parse XML source** ist synchron.

Im Parameter *Dokumentname* übergeben Sie:

- entweder den vollständigen Standard-Zugriffspfad (unter Windows C:\Ordner\Datei\... auf Mac OS:Ordner:Datei)
- oder einen Unix-Pfad unter Mac OS (der mit "/" beginnen muss).
- oder einen Netzwerkpfad vomTyp http://www.site.com/Datei bzw. ftp://public.ftp.com...

Der Boolean Parameter *Gültigkeitsprüfung* gibt an, ob die Struktur bestätigbar ist oder nicht.

- Hat *Gültigkeitsprüfung* den Wert Wahr, wird die Struktur bestätigt. In diesem Fall versucht der Parser, die XML Struktur des Dokuments zu bestätigen, auf Basis der im Dokument definierten DTD oder XSD Referenz bzw. über die DTD bzw. das XML Schema des Parameters *dtd / schema*, wenn er übergeben ist.
- Hat *Gültigkeitsprüfung* den Wert Falsch, wird die Struktur nicht bestätigt.

Übergeben Sie im Parameter *Gültigkeitsprüfung* Wahr und lassen den dritten Parameter weg, versucht die Funktion, die XML Struktur über eine DTD oder XSD Referenz zu bestätigen, die in der Struktur selbst vorhanden ist. Diese Bestätigung kann indirekt sein: Enthält die Struktur eine Referenz auf eine DTD Datei, die wiederum eine Referenz auf eine XSD Datei enthält, versucht die Funktion, beide Bestätigungen auszuführen.

Der Parameter *dtd | schema* gibt die spezifische DTD oder ein XML Schema zum Durchlaufen des Dokuments an. Verwenden Sie diesen Parameter, berücksichtigt die Funktion nicht die DTD, die im XML Dokument angegeben wird.

Bestätigung über dtd
Es gibt zwei Wege zur Definition von DTD:

- Als eine **Referenz**. Dazu übergeben Sie im Parameter *dtd / schema* den vollständigen Zugriffspfad der neuen DTD (Endung .dtd). Enthält das angegebene Dokument keine gültige DTD, wird der Parameter ignoriert und ein Fehler angezeigt.
- Direkt in einem **Text**. Beginnt der Inhalt des Parameters mit "<?xml", betrachtet 4D das als DTD; sonst als Zugriffspfad.

Bestätigung über schema
Um das Dokument über ein XML Schema zu bestätigen, übergeben Sie im dritten Parameter eine Datei bzw. URL mit der Endung "xsd" anstatt "dtd". Die Bestätigung über XML Schema ist in der Regel flexibler und leistungsstärker als die Bestätigung über DTD, da die Sprache von XSD Dokumenten auf XML Sprache basieren. Außerdem unterstützen XML Schemas Datentypen. Weitere Informationen dazu finden Sie im Internet unter: http://www.w3.org/XML/Schema.

Ist keine Gültigkeitsprüfung möglich (kein DTD bzw. XSD, inkorrekte URL, etc.), wird ein Fehler generiert. Die Systemvariable Error gibt die Fehlernummer an. Sie können diesen Fehler mit einer Methode auffinden, die der Befehl **ON ERR CALL** aufruft.

Die Funktion gibt einen String mit 16 Zeichen zurück (*ElementRef*), der die Referenz im Speicher der virtuellen Struktur des Dokuments enthält. Diese Referenz müssen Sie zusammen mit anderen XML Befehlen zum Parsen verwenden.

Wichtig: Benötigen Sie die Funktion nicht länger, denken Sie daran, den Befehl **DOM CLOSE XML** mit dieser Referenz aufzurufen, um den Speicher wieder freizumachen.

Beispiel 1

XML Dokument auf Festplatte ohne Gültigkeitsprüfung öffnen:

```
$xml_Struct_Ref:=DOM Parse XML source("C:\\import.xml")
```

Beispiel 2

XML Dokument neben der Strukturdatei der Datenbank ohne Gültigkeitsprüfung öffnen:

```
$xml_Struct_Ref:=DOM Parse XML source("import.xml")
```

Beispiel 3

XML Dokument auf Festplatte und mit Gültigkeitsprüfung über eine DTD auf der Festplatte öffnen:

```
$xml_Struct_Ref:=DOM Parse XML source("C:\\import.xml";True;"C:\\import_dtd.xml")
```

Beispiel 4

XML Dokument in einer spezifischen URL ohne Gültigkeitsprüfung öffnen:

```
$xml_Struct_Ref:=DOM Parse XML source("http://www.4D.com/xml/import.xml")
```

Systemvariablen und Mengen

Wurde die Funktion korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

DOM Parse XML variable

DOM Parse XML variable (Variable {; Gültigkeitsprüfung {; dtd | schema}}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Variable	BLOB, Text	➡	Name der Variablen
Gültigkeitsprüfung	Boolean	➡	Wahr = Bestätigung durch DTD, Falsch = Keine Bestätigung
dtd | schema	String	➡	Position von DTD oder des XML Schemas
Funktionsergebnis	String	⬅	Referenz des XML Elements

Beschreibung

Die Funktion **DOM Parse XML variable** analysiert eine Variable vom Typ BLOB oder Text mit einer XML Struktur und gibt eine Referenz auf diese Variable zurück. Die Funktion kann je nach Ergebnis das Dokument bestätigen bzw. nicht bestätigen.

Im Parameter *Variable* übergeben Sie den Namen der Variablen vom Typ BLOB oder Text mit dem XML Objekt.

Der Boolean Parameter *Gültigkeitsprüfung* gibt an, ob die Struktur mit DTD bestätigbar ist oder nicht.

- Hat *Gültigkeitsprüfung* den Wert Wahr, wird die Struktur bestätigt. In diesem Fall versucht der Parser, die XML Struktur des Dokuments zu bestätigen, auf Basis der im Dokument definierten DTD oder XSD Referenz bzw. über die DTD bzw. das XML Schema des Parameters *dtd / schema*, wenn er übergeben ist.
- Hat *Gültigkeitsprüfung* den Wert Falsch, wird die Struktur nicht bestätigt.

Bestätigung über dtd
Der Parameter *dtd | schema* gibt die spezifische DTD bzw. das XML Schema zum Parsen des Dokuments an. Verwenden Sie diesen Parameter, berücksichtigt die Funktion nicht die DTD, die in der XML Variablen angegeben wird. Sie können die DTD definieren:

- Als eine **Referenz**: Dazu übergeben Sie im Parameter *dtd* den vollständigen Zugriffspfad der neuen DTD. Enthält das angegebene Dokument keine gültige DTD, wird *dtd* ignoriert und ein Fehler angezeigt.
- Direkt in **Text**: Beginnt der Inhalt des Parameters mit "<?xml", betrachtet 4D das als DTD; sonst als Pfadname.

Bestätigung über schema
Um das Dokument über ein XML Schema zu bestätigen, übergeben Sie im dritten Parameter eine Datei bzw. URL mit der Endung "xsd" anstatt "dtd". Die Bestätigung über XML Schema ist in der Regel flexibler und leistungsstärker als die Bestätigung über DTD, da die Sprache von XSD Dokumenten auf XML Sprache basieren. Außerdem unterstützen XML Schemas Datentypen. Weitere Informationen dazu finden Sie im Internet unter http://www.w3.org/XML/Schema.

Erfolgt keine Gültigkeitsprüfung (kein DTD, inkorrekte URL auf DTD, etc.), wird ein Fehler generiert. Die Systemvariable Error gibt die Fehlernummer an. Sie können diesen Fehler mit einer Methode auffinden, die der Befehl **ON ERR CALL** aufruft.

Die Funktion gibt einen String mit Zeichen (*ElementRef*) zurück, der die Referenz im Speicher der virtuellen Struktur des Dokuments enthält. Verwenden Sie diese Referenz zusammen mit anderen XML Befehlen zum Parsen.

Wichtig: Benötigen Sie die Funktion nicht länger, denken Sie daran, den Befehl **DOM CLOSE XML** mit dieser Referenz aufzurufen, um den Speicher wieder freizumachen.

Beispiel 1

XML Objekt in einer 4D Text Variablen ohne Gültigkeitsprüfung öffnen:

```
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
```

Beispiel 2

XML Dokument in einem 4D BLOB ohne Gültigkeitsprüfung öffnen:

```
C_BLOB(myBlobVar)
C_TEXT($ref_XML_Struct)

DOCUMENT TO BLOB(c:\\import.xml;myBlobVar)
$xml_Struct_Ref:=DOM Parse XML variable(myBlobVar)
```

Systemvariablen und Mengen

Wurde die Funktion korrekt ausgeführt, wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

DOM REMOVE XML ATTRIBUTE

DOM REMOVE XML ATTRIBUTE (ElementRef ; attrName)

Parameter	Typ		Beschreibung
ElementRef	Text	➡	XML Element Referenz
attrName	Text	➡	zu entfernendes Attribut

Beschreibung

Der Befehl **DOM REMOVE XML ATTRIBUTE** entfernt, sofern vorhanden, das in *attrName* angegebene Attribut aus dem XML Element mit der im Parameter *ElementRef* übergebenen Referenz.

Wurde das Attribut korrekt entfernt, wird die Systemvariable OK auf 1 gesetzt. Gibt es kein Attribut mit Namen *attrName* in *ElementRef*, wird ein Fehler zurückgegeben und die Systemvariable OK wird auf 0 gesetzt.

Beispiel

Nehmen wir folgende Struktur:

```
<?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>
```

Der nachfolgenden Code entfernt das erste Attribut "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)

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf XML Element

Beschreibung

Der Befehl **DOM REMOVE XML ELEMENT** entfernt das Element, angegeben in *ElementRef*.

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und es wird ein Fehler zurückgegeben, wenn die Referenz auf das Element nicht gültig ist.

DOM SET XML ATTRIBUTE

DOM SET XML ATTRIBUTE (ElementRef ; attrName ; attrWert {; attrName2 ; attrWert2 ; ... ; attrNameN ; attrWertN})

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf XML Element
attrName	String	➡	Zu setzendes Attribut
attrWert	String, Boolean, Lange Ganzzahl, Zahl, Zeit, Datum	➡	Neuer Attributwert

Beschreibung

Der Befehl **DOM SET XML ATTRIBUTE** fügt dem XML Element mit der in *ElementRef* übergebenen Referenz ein oder mehrere Attribute hinzu. Er setzt auch den Wert jedes definierten Attributs.

In *attrName* und *attrWert* übergeben Sie die zu setzenden Attribute mit den dazugehörigen Werten, und zwar in Form von Variablen, Feldern oder tatsächlichen Werten. Sie können beliebig viele Paare Attribut/Wert übergeben.

Sie können jetzt in *attrWert* auch einen Wert übergeben, der nicht vom Typ Text ist, das kann Boolean, Ganzzahl, Zahl, Datum oder Zeit sein. 4D steuert die Umwandlung in Text gemäß den folgenden Vorgaben:

Typ	Beispiel für konvertierten Wert
Boolean	"Wahr" oder "Falsch" (nicht übersetzt)
Ganzzahl	"123456"
Zahl	"12.34" (Dezimaltrenner ist immer ".")
Datum	"2006-12-04T00:00:00Z" (RFC 3339 Standard)
Zeit	"5233" (Anzahl Sekunden)

Beispiel

Die XML Quelle:

```
<Book>  <Title>The Best Seller</Title> </Book>
```

...führt folgenden Code aus:

```
vAttrName:="Font"
vAttrVal:="Verdana"
DOM SET XML ATTRIBUTE(vElemRef;vAttrName;vAttrVal)
```

...Sie erhalten:

```
<Book>  <Title Font=Verdana>The Best Seller</Title> </Book>
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

DOM SET XML DECLARATION

DOM SET XML DECLARATION (ElementRef ; Codierung {; Standalone {; Einzug}})

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf XML Element
Codierung	String	➡	Zeichensatz des XML Dokuments
Standalone	Boolean	➡	True = Dokument ist eigenständig; False (Standard) = Dokument nicht eigenständig
Einzug	Boolean	➡	*** Überholt, nicht verwenden ***

Beschreibung

Der Befehl **DOM SET XML DECLARATION** definiert hilfreiche Optionen zum Erstellen des XML Baums in *ElementRef*: *Codierung* und *Standalone*:

- *Codierung*: Gibt den im Dokument verwendeten Zeichensatz an. Standardmäßig, d.h. wenn der Befehl nicht aufgerufen wird, wird der Zeichensatz UTF-8 verwendet (komprimierter Unicode).
  Hinweis: Übergeben Sie einen Zeichensatz, der nicht von 4D XML Befehlen unterstützt wird, wird UTF-8 verwendet. Weitere Informationen dazu finden Sie im Abschnitt **Zeichensätze**. In den meisten Fällen wird jedoch UTF-8 empfohlen.
- *Standalone*: Gibt an, ob der Baum eigenständig (**Wahr**) ist oder andere Dateien bzw. externe Ressourcen benötigt (**Falsch**). Standardmäßig, d.h. wenn der Befehl nicht aufgerufen oder dieser Parameter weggelassen wird, ist der Baum nicht eigenständig.

Hinweis zur Kompatibilität: Der Parameter *Einzug* wird zur Wahrung der Kompatibilität mit früheren 4D Versionen beibehalten. Ab Version 12 wird die Verwendung jedoch nicht mehr empfohlen. Um den Einzug des Dokuments anzugeben, empfehlen wir dringend, den Befehl **XML SET OPTIONS** zu verwenden.

Beispiel

Nachfolgende Anweisung setzt für das Element in *ElementRef* die Codierung und die Option standalone fest:

```
DOM SET XML DECLARATION(ElementRef;"UTF-16";True)
```

DOM SET XML ELEMENT NAME

DOM SET XML ELEMENT NAME (ElementRef ; ElementName)

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf XML Element
ElementName	String	➡	Neuer Name des Elements

Beschreibung

Der Befehl **DOM SET XML ELEMENT NAME** ändert den Namen des Elements, definiert durch *ElementRef*.
In *ElementRef* übergeben Sie die Referenz auf das umzubenennende Element, in *ElementName* den neuen Elementnamen. Der Befehl übernimmt auch die Aktualisierung der Tags für Öffnen und Schließen des Elements.

Beispiel

Wird in der XML Quelle:

```
<Book>      <Title>The Best Seller</Title> </Book>
```

… der nachfolgende Code ausgeführt, wobei *vElementRef* die Referenz auf das Element 'Book' enthält:

```
DOM SET XML ELEMENT NAME(vElemRef;"BestSeller")
```

… erhalten Sie:

```
<BestSeller>      <Title>The Best Seller</Title> </BestSeller>
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

Fehlerverwaltung

Ein Fehler wird zurückgegeben, wenn:

- Die Referenz auf das Element ungültig ist
- Der neue Name des zu erstellenden Elements ungültig ist, z.B. wenn er mit einer Zahl beginnt.

DOM SET XML ELEMENT VALUE

DOM SET XML ELEMENT VALUE (ElementRef {; xPath}; ElementWert {; *})

Parameter	Typ		Beschreibung
ElementRef	String	➡	Referenz auf XML Element
xPath	Text	➡	Pfad xPath des XML Elements
ElementWert	String, Variable	➡	Neuer Wert des Elements
*	Operator	➡	Mit Stern: Setze Wert in CDATA

Beschreibung

Der Befehl **DOM SET XML ELEMENT VALUE** ändert den Wert des Elements, definiert durch *ElementRef*.

Mit den optionalen Parameter *xPath* legen Sie fest, dass auf das zu ändernde Element über die Notation XPath (siehe Absatz **XPath Notation verwalten (DOM)**) zugegriffen wird. In diesem Fall müssen Sie in *ElementWert* die Referenz auf ein Root XML Element übergeben und in *xPath* den Pfad XPath des zu ändernden Elements.

In *ElementRef* übergeben Sie die Referenz auf das zu ändernde Element, in *ElementWert* den neuen Wert:

- Übergeben Sie einen String, wird der Wert wie gehabt in der XML Struktur verwendet.
- Übergeben Sie eine Variable oder ein Feld, bearbeitet 4D den Wert je nach Typ von *ElementWert*. Sie können bis auf Arrays, Bilder und Zeiger alle Datentypen verwenden.

Ist der optionale Parameter Stern (*) übergeben, muss der Wert des Elements im Format *CDATA* gesetzt werden. In diesem Sonderformat lässt sich Ausgangstext unverändert schreiben (siehe Beispiel 2).

Hinweis: Ist in diesem Befehl das in *ElementRef* übergebene Element ein BLOB, wird es automatisch in base64 codiert. In diesem Fall führt der Befehl **DOM SET XML ELEMENT VALUE** automatisch die umgekehrte Operation aus.

Hinweis zum Bearbeiten von Zeichen für Zeilenende

In Anlehnung an die Regeln zum Bearbeiten von XML werden alle Zeichen für Zeilenende (CR und CRLF) in LF Zeichen konvertiert.

Beispiel 1

Wird in dieser XML Quelle:

```
<Book>	<Title>The Best Seller</Title> </Book>
```

…der nachfolgende Code ausgeführt, wobei *vElementRef* die Referenz auf das Element 'Title' enthält:

```
DOM SET XML ELEMENT VALUE(vElemRef;"The Loser")
```

…erhalten Sie:

```
<Book>	<Title>The Loser</Title> </Book>
```

Beispiel 2

Wir gehen von folgender XML Struktur aus:

```
<Maths>	<Postulate>1+2=3</Postulate> </Maths>
```

Wir wollen in der XML Struktur den Text "12 < 18" im Element <Postulate> schreiben. Dieser String lässt sich in dieser Form nicht in XML schreiben, da das Zeichen "<" nicht akzeptiert wird. Es muss geändert werden in "<" oder das Format CDATA muss verwendet werden.Gibt *vElemRef* den XML-Knoten *<Postulate>* an, gilt folgendes:

```
` Normales Formular
DOM SET XML ELEMENT VALUE(vRefElem;"12 < 18")
```

Wir erhalten:

```
<Maths>	<Postulate>12 < 18</Postulate> </Maths>
```

```
` Format CDATA
DOM SET XML ELEMENT VALUE(vRefElem;"12 < 18";*)
```

Wir erhalten:

```
<Maths>     <Postulate><![CDATA[12 < 18]]></Postulate> </Maths>
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben; z.B. bei ungültiger Referenz auf das Element.

XML SAX

- Überblick über XML SAX Befehle
- 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

Überblick über XML SAX Befehle

Dieses Kapitel enthält die XML SAX Befehle von 4D.

Für allgemeine Informationen zu XML (Überblick, Zeichensätze, Glossar), sowie die Unterschiede zwischen DOM und SAX finden Sie im Abschnitt **Überblick über XML DOM Befehle**.

Hinweis zum preemptive Modus: Von einem preemptive Prozess erstellte XML Referenzen lassen sich nur in diesem spezifischen Prozess verwenden. Im Unterschied dazu sind von einem kooperativen Prozess erstellte XML Referenzen von allen anderen kooperativen Prozessen verwendbar, jedoch nicht von einem preemptive Prozess.

XML Dokumente via SAX verwalten

Die SAX Befehle arbeiten mit den standardmäßigen Dokumentreferenzen von 4D (*DokRef*, Referenz vom Typ Zeit). Deshalb können Sie diese Befehle zusammen mit 4D Routinen zur Dokumentverwaltung verwenden, z.B **SEND PACKET** oder **Append document**.

XML Dokumente werden über die Funktionen **Create document** und **Open document** per Programmierung verwaltet. Folglich löst die Verwendung eines XML Befehls mit diesen Dokumenten automatisch die Einbindung von XML Mechanismen aus, wie Codierung. So wird der Header *<?xml version="1.0" encoding="… encoding …" standalone = "no "?>* automatisch in das Dokument geschrieben.

Hinweis: Dokumente, die von SAX Befehlen gelesen werden, müssen von der Funktion **Open document** im Nur-Lesen-Modus geöffnet werden. Dies verhindert Konflikte zwischen 4D und der Xerces library, wenn Sie gleichzeitig “Standard” und XML Dokumente öffnen. Führen Sie einen Befehl für SAX "Parsing" mit einem Dokument aus, das im Lesen-/Schreibmodus geöffnet ist, erscheint eine Fehlermeldung und "Parsing" ist nicht möglich.

Ein XML Dokument muss mit dem Befehl **CLOSE DOCUMENT** geschlossen werden. Alle evtl. offenen XML Elemente werden automatisch geschlossen.

SAX ADD PROCESSING INSTRUCTION

SAX ADD PROCESSING INSTRUCTION (DokRef ; Anweisung)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Anweisung	Text	➡	Anweisung zum Einfügen in das Dokument

Beschreibung

Der Befehl **SAX ADD PROCESSING INSTRUCTION** fügt im XML Dokument, definiert durch *DokRef*, in *Anweisung* die XML Bearbeitung hinzu.

In *Anweisung* geben Sie den Programmtyp und bei Bedarf weitere Parameter an, um eine externe nicht-geparste Einheit zu bearbeiten.

Der Befehl formatiert die Daten der Anweisung konform mit XML. Die Anweisung selbst wird dabei nicht geparst. Der Entwickler muss selbst sicherstellen, dass sie gültig ist.

Beispiel

Der Code:

```
vtInstruct:="xml-stylesheet type="+Char(Quotes)+"text/xsl"+Char(Quotes)+
"href="+Char(Quotes)+"style.xsl"+Char(Quotes)
SAX ADD PROCESSING INSTRUCTION($DocRef;vtInstruct)
```

... schreibt im Dokument folgende Zeile:

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

SAX ADD XML CDATA

SAX ADD XML CDATA (DokRef ; Daten)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Daten	BLOB, Text	➡	Text oder BLOB zum Einfügen in das Dokument zwischen CData Tags

Beschreibung

Der Befehl **SAX ADD XML CDATA** fügt im XML Dokument, definiert durch *DokRef*, Daten vom Typ Text oder BLOB hinzu. Diese Daten werden automatisch zwischen die Tags <![CData[und]]> gesetzt. Der XML Interpreter ignoriert Text zwischen solchen Tags.

Wollen Sie den Inhalt von Daten codieren, müssen Sie den Befehl **BASE64 ENCODE** verwenden. Dann müssen Sie natürlich in *Daten* ein BLOB übergeben.

Dieser Befehl arbeitet nur korrekt, wenn ein Element geöffnet ist. Sonst wird ein Fehler erzeugt.

Beispiel

Sie wollen folgende Zeilen in Ihr XML Dokument einfügen:

```
function matchwo(a,b) { if (a < b && a < 0) then        {        return 1        } else        {        return 0        } }
```

Dazu führen Sie folgenden Code aus:

```
C_TEXT(vtMytext)
... ` setze den Text in die Variable vtMytext hier
SAX ADD XML CDATA($DocRef;vtMytext)
```

Das Ergebnis ist:

```
<![CDATA[<br> function matchwo(a,b)<br> {<br> if (a < b && a < 0) then<br>      {<br>      return 1<br>      }<br> else<br>      {<br>
    return 0<br>      }<br> }<br> ]]>
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

SAX ADD XML COMMENT

SAX ADD XML COMMENT (DokRef ; Kommentar)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Kommentar	String	➡	Hinzuzufügender Kommentar

Beschreibung

Der Befehl **SAX ADD XML COMMENT** fügt im XML Dokument, definiert durch *DokRef*, einen *Kommentar* hinzu.
Ein XML Kommentar ist ein Text, dessen Inhalt der XML Interpreter nicht durchläuft. Er muss zwischen die Zeichen <!-- ... --> gestellt werden.

Beispiel

Die Anweisung:

```
vComment:="Erstellt von 4D"
SAX ADD XML COMMENT($DocRef;vComment)
```

... schreibt im Dokument folgende Zeile:

```
<!--Erstellt von 4D-->
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

Fehlerverwaltung

Gibt der Befehl einen Fehler zurück, lässt er sich über eine Fehlerverwaltungsmethode abfangen.

SAX ADD XML DOCTYPE

SAX ADD XML DOCTYPE (DokRef ; DokTyp)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
DokTyp	String	➡	Hinzuzufügender Dokumenttyp

Beschreibung

Der Befehl **SAX ADD XML DOCTYPE** fügt im XML Dokument, definiert durch *DokRef*, eine Anweisung *DokTyp* hinzu.
Mit *DokTyp* können Sie die Art des XML, in dem das Dokument verfasst wurde, sowie die verwendete Document Typ Declaration (DTD) angeben. Eine Anweisung *DokTyp* hat in der Regel folgende Form: *<!DOCTYPE XML_type "DTD_address">*.

Beispiel

Die Anweisung:

```
vDocType:="SYSTEM Books \"Book.DTD\""
SAX ADD XML DOCTYPE($DocRef;vDocType)
```

... schreibt im Dokument folgende Zeile:

```
<!DOCTYPE SYSTEM Books "Book.DTD">
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

Fehlerverwaltung

Gibt der Befehl einen Fehler zurück, lässt er sich über eine Fehlerverwaltungsmethode abfangen.

SAX ADD XML ELEMENT VALUE

SAX ADD XML ELEMENT VALUE (DokRef ; Daten {; *})

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf das offene Dokument
Daten	Text, Variable	➡	Text oder Variable zum Einfügen in das Dokument
*	Operator	➡	Mit *: Sonderzeichen werden codiert, ohne *: Keine Codierung

Beschreibung

Der Befehl **SAX ADD XML ELEMENT VALUE** fügt im XML Dokument, definiert durch *DokRef*, Daten direkt hinzu, d.h. ohne Konvertierung. Dieser Befehl entspricht z.B. dem Hinzufügen eines Anhangs im Hauptteil eines E-Mail.

Sie können in *Daten* entweder direkt eine Zeichenkette oder eine 4D Variable übergeben. Der Variableninhalt wird vor Einbinden in das XML Dokument in Text konvertiert.

Wollen Sie den Inhalt von *Daten* codieren, müssen Sie den Befehl **BASE64 ENCODE** verwenden. Dann müssen Sie natürlich in *Daten* ein BLOB übergeben.

Dieser Befehl codiert standardmäßig Sonderzeichen(< > ' ""...), die im Parameter *Daten* enthalten sind, außer Sie haben diesen Mechanismus für den aktuellen Prozess deaktiviert. Dazu verwenden Sie den Befehl **XML SET OPTIONS** und übergeben in der Option XML String encoding den Wert XML Raw data. Zum Beispiel:

```
XML SET OPTIONS($docRef;XML string encoding;XML raw data)
```

Um die Codierung dieser Parameter zu erzwingen, übergeben Sie einfach den optionalen Parameter *.

Dieser Befehl arbeitet nur korrekt, wenn ein Element geöffnet ist. Sonst wird ein Fehler erzeugt.

Beispiel

Diese Anweisung fügt im offenen XML Element die Datei *whitepaper.pdf* hinzu:

```
C_BLOB(vBMyBLOB)
DOCUMENT TO BLOB("c:\\whitepaper.pdf";vBMyBLOB)
SAX ADD XML ELEMENT VALUE($DocRef;vBMyBLOB)
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null).

SAX CLOSE XML ELEMENT

SAX CLOSE XML ELEMENT (DokRef)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument

Beschreibung

Der Befehl **SAX CLOSE XML ELEMENT** schreibt im XML Dokument, definiert durch *DokRef*, die notwendige Anweisung zum Schließen des letzten Elements, das mit dem Befehl **SAX OPEN XML ELEMENT** geöffnet wurde.

Dieser Befehl ist optional. Denn 4D fügt beim Schließen von XML Dokumenten automatisch die erforderlichen End Tags für alle nicht-geschlossenen Elemente hinzu.

Beispiel

Lautet das letzte offene Element <Book>, schreibt die Anweisung:

```
SAX CLOSE XML ELEMENT($DocRef)
```

... folgende Zeile im Dokument:

```
</Book>
```

SAX GET XML CDATA

SAX GET XML CDATA (DokRef ; Wert)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Wert	Text, BLOB	⬅	Wert des Elements

Beschreibung

Der Befehl **SAX GET XML CDATA** empfängt den *Wert* CDATA eines XML Elements aus dem XML Dokument, definiert durch *DokRef*. Er muss mit dem SAX Ereignis *XML CDATA* aufgerufen werden. Weitere Informationen über SAX Ereignisse finden Sie in der Beschreibung zum Befehl **SAX Get XML node**.

Übergeben Sie in *Wert* eine Variable vom Typ Text, um Daten zu finden, die größer als 32 KB sind. Dazu muss die Datenbank im Unicode Modus laufen.

Hinweis zur Kompatibilität: Ab 4D Version 12 wird CDATA Inhalt, der in base64 codiert wurde, von **SAX GET XML CDATA** automatisch decodiert. Von daher müssen Sie nicht den Befehl **BASE64 DECODE**aufrufen.

Beispiel

Betrachten wir folgenden Teil eines XML Code:

```
<RootElement>     <Child>MyText<![CDATA[MyCData]]</Child> </RootElement>
```

Folgender 4D Code gibt in *vTextData* "MyCData" zurück:

```
C_BLOB(vData)
C_TEXT(vTextData)
SAX GET XML CDATA(DocRef;vData)
vTextData:=BLOB to text(vData;UTF8 C string)
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

SAX GET XML COMMENT

SAX GET XML COMMENT (DokRef ; Kommentar)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Kommentar	String	⬅	XML Kommentar

Beschreibung

Der Befehl **SAX GET XML COMMENT** gibt einen *Kommentar* zurück, wenn im XML Dokument, definiert durch *DokRef*, ein XML Kommentar vom Typ SAX generiert wird. Weitere Informationen zu SAX Ereignissen finden Sie unter der Funktion **SAX Get XML node**.

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

SAX GET XML DOCUMENT VALUES

SAX GET XML DOCUMENT VALUES (DokRef ; Codierung ; Version ; Standalone)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Codierung	String	⬅	Dokument
Version	String	⬅	XML Version
Standalone	Boolean	⬅	True = Dokument ist eigenständig, sonst False

Beschreibung

Der Befehl **SAX GET XML DOCUMENT VALUES** erhält Grundinformation aus dem XML Header des XML Dokuments, definiert durch *DokRef*.

Die Parameter *Codierung*, *Version* und *Standalone* geben die Art der Codierung, der Eigenständigkeit und die Version für das Dokument zurück. Der Befehl muss mit dem SAX Ereignis XML Start Document aufgerufen werden. Weitere Informationen über SAX Ereignisse finden Sie unter der Funktion **SAX Get XML node**.

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

SAX GET XML ELEMENT

SAX GET XML ELEMENT (DokRef ; Auswahlname ; Vorsilbe ; attrName ; attrWert)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Auswahlname	String	⬅	Elementname
Vorsilbe	String	⬅	Namensbereich
attrName	Array String	⬅	Attributnamen
attrWert	Array String	⬅	Attributwerte

Beschreibung

Der Befehl **SAX GET XML ELEMENT** gibt verschiedene Informationen über das Element *Name* im XML Dokument, definiert durch *DokRef* zurück. Er muss mit den Ereignissen XML Start Element oder XML End Element aufgerufen werden. Bei XML End Element werden die Parameter für Attribute nicht verwendet. Weitere Informationen über SAX Ereignisse finden Sie in der Beschreibung zur Funktion **SAX Get XML node**.

Name enthält den Namen des Elements.

Vorsilbe gibt den Namensbereich des Elements zurück. Der Parameter ist leer, wenn kein Namensbereich mit dem Element verknüpft ist.

Der Befehl füllt das Array *attrName* mit den Attributnamen des Zielelements. Bei Bedarf erstellt er das Array automatisch und in der passenden Größe.

Der Befehl füllt auch das Array *attrWert* mit den Attributwerten des Zielelements. Bei Bedarf erstellt er das Array automatisch und in der passenden Größe.

Beispiel

Sehen wir uns folgenden Teil von XML Code an:

```
<RootElement>     <Child Att1="111" Att2="222" Att3="333">MyText</Child> </RootElement>
```

Nach Ausführung der folgenden Anweisung gilt:

```
SAX GET XML ELEMENT(DocRef;vName;vPrefix;tAttrNames;tAttrValues)
```

... *vName* enthält "Child"
vPrefix enthält ""
tAttrNames{1} enthält "Att1", *tAttrNames{2}* enthält "Att2", *tAttrNames{3}* enthält "Att3"
tAttrValues{1} enthält "111", *tAttrNames{2}* enthält "222", *tAttrValues{3}* enthält "333"

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

SAX GET XML ELEMENT VALUE

SAX GET XML ELEMENT VALUE (DokRef ; Wert)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Wert	Text, BLOB	⬅	Element Wert

Beschreibung

Der Befehl **SAX GET XML ELEMENT VALUE** empfängt den Wert eines XML Elements aus dem XML Dokument, definiert durch *DokRef*. Er muss mit dem SAX Ereignis XML DATA aufgerufen werden. Weitere Informationen dazu finden Sie in der Beschreibung zur Funktion **SAX Get XML node**.

In *Wert* übergeben Sie eine Variable vom Typ Text oder BLOB. Bei einem BLOB versucht die Funktion automatisch, es in base64 zu decodieren.

Beispiel

Sehen wir uns folgenden Teil von XML Code an:

```
<RootElement>     <Child Att1="111" Att2="222" Att3="333">MyText</Child> </RootElement>
```

Nachfolgende Anweisung gibt in *vValue* "MyText" zurück:

```
SAX GET XML ELEMENT VALUE(DocRef;vValue)
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

SAX GET XML ENTITY

SAX GET XML ENTITY (DokRef ; Auswahlname ; Wert)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Auswahlname	String	⬅	Name der Einheit
Wert	String	⬅	Wert der Einheit

Beschreibung

Der Befehl **SAX GET XML ENTITY** empfängt den *Wert* einer XML Einheit aus dem XML Dokument, definiert durch *DokRef*. Er muss mit dem SAX Ereignis XML Entity aufgerufen werden. Weitere Informationen zu SAX Ereignissen finden Sie in der Beschreibung zur Funktion **SAX Get XML node**.

Beispiel

Betrachten wir folgenden Teil eines XML Code:

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE body [  <!ELEMENT body (element*)>  <!ELEMENT element (#PCDATA)>
  <!ENTITY name "Replacement"> ]> <body>  <element>Entity updated by &name;</element> </body>
```

Die Anweisung gibt in *vName* "name" und in *vWert* "Replacement" zurück:

```
SAX GET XML ENTITY(DocRef;vName;vValue)
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

SAX Get XML node

SAX Get XML node (DokRef) -> Funktionsergebnis

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Funktionsergebnis	Lange Ganzzahl	➲	Zurückgegebenes Ereignis

Beschreibung

Die Funktion **SAX Get XML node** gibt eine Lange Ganzzahl zurück, die die Art des zurückgegebenen SAX Ereignisses angibt, während das XML Dokument, definiert durch *DokRef*, logisch durchlaufen wird.
Die zurückgebbaren Ereignisse sind als Konstanten unter dem Thema **XML** verfügbar:

Konstante	Typ	Wert
XML CDATA	Lange Ganzzahl	7
XML Comment	Lange Ganzzahl	2
XML DATA	Lange Ganzzahl	6
XML End Document	Lange Ganzzahl	9
XML End Element	Lange Ganzzahl	5
XML Entity	Lange Ganzzahl	8
XML Processing Instruction	Lange Ganzzahl	3
XML Start Document	Lange Ganzzahl	1
XML Start Element	Lange Ganzzahl	4

Beispiel

Folgende Methode bearbeitet ein Ereignis:

```
DocRef:=Open document("";"xml";Read Mode)
If(OK=1)
   Repeat
      MyEvent:=SAX Get XML node(DocRef)
      Case of
         :(MyEvent=XML Start Document)
            Etwas ausführen
         :(MyEvent=XML Comment)
            Etwas anderes ausführen
      End case
   Until(MyEvent=XML End Document)
   CLOSE DOCUMENT(DocRef)
End if
```

Systemvariablen und Mengen

Bei korrekt ausgeführtem Befehl wird die Systemvariable OK auf 1 gesetzt, sonst auf 0 (Null) und ein Fehler wird zurückgegeben.

SAX GET XML PROCESSING INSTRUCTION

SAX GET XML PROCESSING INSTRUCTION (DokRef ; Auswahlname ; Wert)

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Auswahlname	String	⬅	Name der Anweisung
Wert	String	⬅	Wert der Anweisung

Beschreibung

Der Befehl **SAX GET XML PROCESSING INSTRUCTION** gibt *Name* und *Wert* der XML Anweisung zurück, die im XML Dokument, definiert durch *DokRef*, bearbeitet wird. Er muss mit dem Ereignis XML Processing Instruction aufgerufen werden. Weitere Informationen zu SAX Ereignissen finden Sie in der Beschreibung zum Befehl **SAX Get XML node**.

Beispiel

Sehen wir uns folgenden Teil von XML Code an:

```
<?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">
```

Die Anweisung gibt "PI" in *vName* und "TextProcess" in *vValue* zurück:

```
SAX GET XML PROCESSING INSTRUCTION($DocRef;vName;vValue)
```

SAX OPEN XML ELEMENT

SAX OPEN XML ELEMENT (DokRef ; Tag {; attrName ; attrWert} {; attrName2 ; attrWert2 ; ... ; attrNameN ; attrWertN})

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Tag	String	➡	Name des zu öffnenden Elements
attrName	String	➡	Attributname
attrWert	String	➡	Attributwert

Beschreibung

Der Befehl **SAX OPEN XML ELEMENT** fügt im XML Dokument, definiert durch *DokRef*, ein neues Element hinzu, sowie optional Attribute und deren Werte.

Das hinzugefügte Element ist im Dokument "offen", d.h. es ist kein End Tag hinzugefügt. Um es zu schließen, müssen Sie:

- entweder den Befehl **SAX CLOSE XML ELEMENT** einsetzen
- oder das XML Dokument schließen. In diesem Fall fügt 4D bei Bedarf automatisch die schließenden XML Tags hinzu.

In *Tag* übergeben Sie den Namen des zu erstellenden Elements. Er darf nur Buchstaben, Zahlen und die Zeichen ".", "-","_" sowie ":" enthalten. Bei einem ungültigen Zeichen wird ein Fehler generiert.

Optional kann der Befehl für das erstellte Element ein/mehrere Paare von Attribut/Wert übergeben, und zwar in den Parametern *attrName* und *attrWert* in Form von Variablen, Feldern oder tatsächlichen Werten. Sie können beliebig viele Paare übergeben.

Beispiel

Die Anweisung:

```
vElement:="Book"
SAX OPEN XML ELEMENT($DocRef;vElement)
```

... schreibt im Dokument folgende Zeile:

```
<Book
```

Fehlerverwaltung

Wurde in *Tag* ein ungültiges Zeichen übergeben, wird ein Fehler generiert.

SAX OPEN XML ELEMENT ARRAYS

SAX OPEN XML ELEMENT ARRAYS (DokRef ; Tag {; attrNameArray ; attrWertArray} {; attrNameArray2 ; attrWertArray2 ; ... ; attrNameArrayN ; attrWertArrayN})

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Dokument
Tag	String	➡	Name des zu öffnenden Elements
attrNameArray	Array String	➡	Array mit Attributnamen
attrWertArray	Array String, Array Lange Ganzzahl, Array Datum, Array Zahl, Array Bild, Array Boolean	➡	Array mit Attributwerten

Beschreibung

Der Befehl **SAX OPEN XML ELEMENT ARRAYS** fügt ein neues Element im XML Dokument mit der Referenz *DokRef* hinzu, sowie optional, Attribute und deren Werte in Form von Arrays.
Dieser Befehl ist – bis auf die Unterstützung von Arrays – identisch mit **SAX OPEN XML ELEMENT**. Deshalb finden Sie die weitere Beschreibung unter diesem Befehl.

SAX OPEN XML ELEMENT ARRAYS erlaubt im Parameter *attrWertArray* Arrays vom Typ Datum, Zahl, Boolean und Bild. 4D führt automatisch die notwendigen Konvertierungen durch; Sie können diese Konvertierungen über den Befehl **XML SET OPTIONS** einstellen.

SAX OPEN XML ELEMENT ARRAYS kann in den optionalen Parametern *attrNameArray* und *attrWertArray* Paare Attribute/Attributwerte in Form von Arrays übergeben.
Die Arrays müssen vorher erstellt worden sein und in den Paaren Attribute/Attributwerte operieren. Sie können beliebig viele Arrays mit Paaren und pro Paar beliebig viele Einträge übergeben.

Beispiel

Folgende Methode:

```
ARRAY STRING(80;tAttrNames;2)
ARRAY STRING(80;tAttrValues;2)
vElement:="Book"
tAttrNames{1}:="Font"
tAttrValues{1}:="Arial"
tAttrNames{2}:="Style"
tAttrValues{2}:="Bold"
SAX OPEN XML ELEMENT ARRAYS($DocRef;vElement;tAttrNames;tAttrValues)
```

... schreibt im Dokument folgende Zeile:

```
<Book Font="Arial" Style="Bold">
```

SAX SET XML DECLARATION

SAX SET XML DECLARATION (DokRef ; Codierung {; Standalone {; Einzug}})

Parameter	Typ		Beschreibung
DokRef	DokRef	➡	Referenz auf offenes Dokument
Codierung	String	➡	Zeichensatz für XML Dokument
Standalone	Boolean	➡	Wahr = Dokument ist eigenständig, Falsch (Standard) = Dokument ist nicht eigenständig
Einzug	Boolean	➡	True (Standard) = Dokument hat Einzug, False = Dokument hat keinen Einzug

Beschreibung

Der Befehl **SAX SET XML DECLARATION** initialisiert das XML Dokument mit der Referenz *DokRef* mit den Werten der Parameter *Codierung* und *Standalone*.

- *Codierung*: Gibt den im Dokument verwendeten Zeichensatz an. Standardmäßig, d.h. wenn der Befehl nicht aufgerufen wird, wird der Zeichensatz UTF-8 verwendet.
  Hinweis: Übergeben Sie einen Zeichensatz, der nicht von 4D XML Befehlen unterstützt wird, wird UTF-8 verwendet. Weitere Informationen dazu finden Sie im Abschnitt **Zeichensätze**. In den meisten Fällen wird jedoch UTF-8 empfohlen.
- *Standalone*: Gibt an, ob das Dokument eigenständig (**True**) ist oder andere Dateien bzw. externe Ressourcen benötigt (**False**). Standardmäßig, d.h. wenn der Befehl nicht aufgerufen oder dieser Parameter nicht gesetzt wird, ist das Dokument nicht eigenständig.

Hinweis zur Kompatibilität: Der Parameter *Einzug* wird aus Kompatibilitätsgründen mit früheren 4D Versionen beibehalten. Er sollte ab 4D v12 nicht mehr verwendet werden. Wir empfehlen dringend, den Einzug des Dokuments über den Befehl **XML SET OPTIONS** zu definieren.

Dieser Befehl muss einmal pro Dokument und vor Setzen des ersten XML Befehls aufgerufen werden, sonst erscheint eine Fehlermeldung.

Beispiel

Der Code:

```
SAX SET XML DECLARATION($DocRef;"UTF-16";True)
```

... schreibt im Dokument folgende Zeile:

```
<?xml version="1.0" encoding="UTF-16" standalone="yes"?>
```

Zugriff Designobjekte

- Befehle für Zugriff auf Designobjekte
- 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

Befehle für Zugriff auf Designobjekte

In 4D können Sie in Ihren Anwendungen per Programmierung auf den Inhalt von Methoden zugreifen. Das Source Toolkit vereinfacht die Integration von Werkzeugen zur Code-Kontrolle und insbesondere zur Versionskontrolle (VCS). Sie können auch ausgeklügelte Systeme einrichten, um Code zu dokumentieren, einen eigenen Explorer anzulegen oder geplante Backups des Code zu organisieren, die als Datei gesichert werden.

Es wurden folgende Prinzipien integriert:

- Jede Methode und jedes Formular in einer 4D Anwendung hat eine eigene Adresse in Form eines Pfadnamens. So liegt beispielsweise die Triggermethode für Tabelle 1 unter "[trigger]/tabelle_1". Jeder Pfadname eines Objekts ist einmalig in der Anwendung.
  Hinweis: Damit die Einmaligkeit von Pfadnamen sichergestellt ist, erlaubt 4D nicht länger, auf verschiedenen Formularseiten Objekte mit demselben Namen zu erstellen. In konvertierten Datenbanken vor 4D v13 findet das MSC diese doppelten Namen und repariert sie.
- Über die Befehle dieses Kapitels können Sie auf Objekte in der 4D Anwendung zugreifen, z.B. **METHOD GET NAMES** oder **METHOD GET PATHS**.
- Die meisten Befehle dieses Kapitels funktionieren im interpretierten und kompilierten Modus. Befehle, die Eigenschaften verändern oder auf über Methoden ausführbaren Inhalt zugreifen sind nur im interpretierten Modus verwendbar (siehe Tabelle unten)
- Sie können einige Befehle dieses Kapitels mit 4D im lokalen oder im remote Modus verwenden.
  Beachten Sie jedoch, dass Sie diese Befehle nicht im kompilierten Modus verwenden können. Sie dienen vielmehr dazu, eigene Werkzeuge zur Entwicklungsunterstützung zu erstellen. Sie dürfen sie nicht einsetzen, um die Funktionsweise der Anwendung im laufenden Betrieb dynamisch zu verändern. Sie können z.B. nicht den Befehl **METHOD SET ATTRIBUTE** verwenden, um je nach Status des aktuellen Benutzers ein Attribut der Methode zu ändern.
- Wird ein Befehl dieses Kapitels von einer Komponente aufgerufen, greift er standardmäßig auf die Objekte der Komponente zu. Wollen Sie jedoch auf Objekte der Host Datenbank zugreifen, übergeben Sie den * als letzten Parameter. Beachten Sie, dass nur diese Syntax für Befehle möglich ist, die Objekte ändern, z.B. **METHOD SET ATTRIBUTE**, da Komponenten immer im Nur-Lesen Modus ausgeführt werden.

Verwendung im kompilierten Modus

Es liegt in der Natur des Kompilierungsprozesses, dass nur bestimmte Befehle dieses Kapitels im kompilierten Modus verwendbar sind. Nachfolgende Tabelle zeigt, für welche Befehle das gilt:

Befehl	Im kompilierten Modus verwendbar
Current method path	Ja
FORM GET NAMES	Ja
METHOD Get attribute	Ja
METHOD GET ATTRIBUTES	Ja
METHOD GET CODE	Nein (*)
METHOD GET COMMENTS	Ja
METHOD GET FOLDERS	Ja
METHOD GET MODIFICATION DATE	Ja
METHOD GET NAMES	Ja
METHOD Get path	Ja
METHOD GET PATHS	Ja
METHOD GET PATHS FORM	Ja
METHOD OPEN PATH	Nein (*)
METHOD RESOLVE PATH	Ja
METHOD SET ACCESS MODE	Ja
METHOD SET ATTRIBUTE	Nein (*)
METHOD SET ATTRIBUTES	Nein (*)
METHOD SET CODE	Nein (*)
METHOD SET COMMENTS	Nein (*)

(*) Bei Ausführung im kompilierten Modus wird der Fehler -9762 "Der Befehl lässt sich nicht in einer kompilierten Datenbank verwenden." generiert.

Pfadnamen erstellen

4D erstellt standardmäßig keine Datei auf der Festplatte. Pfadnamen für Objekte sind jedoch kompatibel mit der Dateiverwaltung des jeweiligen Betriebssystems; d.h. es kann Dateien über Ihre eigenen Import- bzw. Exportmethoden direkt auf der Festplatte erzeugen.
Insbesondere verbotene Zeichen, wie "?" werden in Methodennamen codiert. Bei Bedarf lassen sich generierte Dateien automatisch in ein System zur Versionskontrolle integrieren.
Es folgt die Liste der codierten Zeichen:

Zeichen	Codierung
"	%22
*	%2A
/	%2F
:	%3A
<	%3C
>	%3E
?	%3F
|	%7C
\	%5C
%	%25

Beispiele:
Form?1 wird codiert als *Form%3F1*
Button/1 wird codiert als *Button%2F1*

Current method path

Current method path -> Funktionsergebnis

Parameter	Typ	Beschreibung
Funktionsergebnis	Text	Ganzer interner Pfadname der ausführenden Methode

Beschreibung

Die Funktion **Current method path** gibt den internen Pfadnamen der ausführenden Datenbank-, Projekt-, Formular-, Objektmethode oder des Trigger zurück.

Hinweis: In den 4D Makro Befehlen wird das neue Tag <method_path> im Code beim Ausführen der Methode durch den vollständigen Pfadnamen ersetzt.

FORM GET NAMES

FORM GET NAMES ({Tabellenname ;} arrNamen {; Filter {; Marker}}{; *})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Referenz der Tabelle
arrNamen	Array Text	⬅	Array der Formularnamen
Filter	Text	➡	Filter für Namen
Marker	Lange Ganzzahl	➡	Marker für zurückzugebende Mindestversion
		⬅	Neuer Wert
*	Operator	➡	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der Befehl **FORM GET NAMES** füllt das Array *arrNamen* mit den Namen der Formulare in der Anwendung.
Übergeben Sie den Parameter *Tabellenname*, gibt der Befehl die Namen der Tabellenformulare zurück, die dieser Tabelle zugewiesen sind. Lassen Sie diesen Parameter weg, gibt er die Namen der Projektformulare der Datenbank zurück.

Sie können die Liste der Formulare einschränken, wenn Sie im Parameter *Filter* einen Vergleichs-String übergeben: In diesem Fall werden nur die Formulare zurückgegeben, deren Namen zum Filter passen. Sie können das Zeichen @ verwenden, um Filter vom Typ "beginnt mit", "endet mit" oder "enthält" zu übergeben. Übergeben Sie einen leeren String, wird der Parameter *Filter* ignoriert.

Sie können die Liste der Formulare auch über den optionalen Parameter *Marker* einschränken. Damit lassen sich in *arrNamen* zurückgegebene Formulare beschränken auf Formulare, die in einem bestimmten Zeitraum geändert wurden. Als Teil eines Systems zu Versionskontrolle ermöglicht dieser Parameter, nur Formulare zu aktualisieren, die seit dem letzten Backup geändert wurden.
Es funktioniert folgendermaßen: 4D führt intern einen Zähler der Änderungen der Datenbank-Ressourcen. Da Formulare Ressourcen sind, erhöht sich der Zähler bei jedem Erstellen oder Sichern eines Formulars.

Übergeben Sie den Parameter *Marker*, gibt der Befehl in *arrNamen* nur Formulare zurück, deren interner Marker größer oder gleich dem Wert von Marker ist. Übergeben Sie zusätzlich eine Variable in *Marker*, gibt der Befehl den neuen Wert des Zählers für Änderungen zurück, z.B. den höchsten in dieser Variable. Sie können diesen Wert dann sichern und im nächsten Aufruf des Befehls **FORM GET NAMES** verwenden, um nur neue oder geänderte Formulare wiederzufinden.

Wird der Befehl in einer Komponente ausgeführt, gibt er standardmäßig die Namen der Projektformulare der Komponente zurück. Übergeben Sie den Parameter *, enthält das Array die Formulare der Host Datenbank.

Hinweis: Formulare, die im Papierkorb liegen, werden nicht gelistet.

Beispiel

Beispiele für typische Anwendungen:

```
// Liste aller Projektformulare der Datenbank
FORM GET NAMES(arr_Names)

// Liste der Formulare der Tabelle [Employees]
FORM GET NAMES([Employees] ;arr_Names)

// Liste der Eingabeformulare der Tabelle [Employees]
FORM GET NAMES([Employees] ;arr_Names;"input_@")

// Liste spezifischer Projektformulare der Datenbank
FORM GET NAMES(arr_Names;"dialogue_@")

// Liste aller  Projektformulare der Datenbank, die seit der letzten Synchronisation geändert wurden
// vMarker enthält einen numerischen Wert
FORM GET NAMES(arr_Names;"";vMarker)

// Liste der Tabellenformulare aus einer Komponente
// Ein Zeiger ist notwendig, da der Tabellenname unbekannt ist
FORM GET NAMES(tablePtr->;arr_Names;*)
```

METHOD Get attribute

METHOD Get attribute (Pfad ; AttributTyp {; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
Pfad	Text	➡	Pfad der Projektmethode
AttributTyp	Lange Ganzzahl	➡	Typ des zu erhaltenden Attributs
*	Operator	➡	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)
Funktionsergebnis	Boolean	⮌	True = Attribut ist ausgewählt; sonst False

Beschreibung

Die Funktion **METHOD Get attribute** gibt den Wert des Parameters *AttributTyp* für die Projektmethode zurück, die im Parameter *Pfad* angegeben ist. Die Funktion funktioniert nur mit Projektmethoden. Übergeben Sie einen ungültigen Parameter *Pfad*, wird ein Fehler generiert.

Im Parameter *AttributTyp* übergeben Sie einen Wert, der den Typ des zu erhaltenden Attributs angibt. Sie können eine der folgenden Konstanten unter dem Thema **Zugriff Designobjekte** verwenden:

Konstante	Typ	Wert	Kommentar
Attribute executed on server	Lange Ganzzahl	8	Entspricht der Option "Auf Server ausführen"
Attribute invisible	Lange Ganzzahl	1	Entspricht der Option "Unsichtbar"
Attribute published SOAP	Lange Ganzzahl	3	Entspricht der Option "Zugang per Web Service"
Attribute published SQL	Lange Ganzzahl	7	Entspricht der Option "Zugang per SQL"
Attribute published Web	Lange Ganzzahl	2	Entspricht der Option "Zugang per 4D HTML Tags und URLs (4DACTION...)"
Attribute published WSDL	Lange Ganzzahl	4	Entspricht der Option "Anbieten per WSDL"
Attribute shared	Lange Ganzzahl	5	Entspricht der Option "Gemeinsam von Komponenten und Host benutzt"

Wird der Befehl in einer Komponente ausgeführt, gilt er standardmäßig für die Komponentenmethoden. Übergeben Sie den Parameter *, greift er auf die Methoden der Host Datenbank zu.

Die Funktion gibt **True** zurück, wenn ein Attribut ausgewählt ist; **False**, wenn es nicht ausgewählt ist.

METHOD GET ATTRIBUTES

METHOD GET ATTRIBUTES (Pfad ; Attribute {; *})

Parameter	Typ		Beschreibung
Pfad	Text, Array Text	➡	Methodenpfad(e)
Attribute	Objekt, Array Objekt	⬅	Attribut(e) für ausgewählte Methode(n)
*	Operator	➡	Mit Stern: Bei Ausführung über eine Komponente gilt der Befehl für die Host Datenbank (in anderem Kontext wird er ignoriert)

Beschreibung

Der Befehl **METHOD GET ATTRIBUTES** gibt in *Attribute* den aktuellen Wert aller Attribute für die Methode(n) zurück, angegeben im Parameter *Pfad*.

Dieser Befehl funktioniert nur mit Projektmethoden. Übergeben Sie einen ungültigen *Pfad*, wird ein Fehler generiert.

In *Pfad* übergeben Sie entweder einen Text mit einem Methodenpfad oder ein Array Text mit einem Array der Pfade. In *Attribute* müssen Sie jeweils denselben Parameter (String oder Array) übergeben, um die dazugehörigen Attribute zu erhalten.

In *Attribute* übergeben Sie ein Objekt oder ein Array, je nach Art des Parameters in *Pfad*. Alle Attribute für die Methode(n) werden als Objekteigenschaften zurückgegeben, die Werte "true"/"false" für Boolean Attribute, Text und bei Bedarf zusätzliche Werte, z.B. die 4D Mobile Eigenschaft "scope":"table".

Bei Ausführung über eine Komponente gilt der Befehl standardmäßig für die Komponentenmethode. Mit dem Parameter * greift er auf die Methoden der Host Datenbank zu.

Hinweis: Die Funktion **METHOD Get attribute** wird zur Wahrung der Kompatibilität weiterhin unterstützt. Da sie jedoch nur einfache Werte zurückgeben kann, lässt sie sich nicht für erweiterte Attribute wie 4D Mobile Eigenschaften verwenden.

Beispiel

Sie wollen die Attribute der Projektmethode *sendMail* erhalten. Sie schreiben wie folgt:

```
C_OBJECT($att)
METHOD GET ATTRIBUTES("sendMail";$att)
```

Nach Ausführen erhält $att zum Beispiel:

{ "invisible":false, "publishedWeb":false, "publishedSoap":false, "publishedWsdl":false, "shared":false, "publishedSql":false, "executedOnServer":false, "published4DMobile":{ "scope":"table", "table":"Table_1" } }

METHOD GET CODE

METHOD GET CODE (Pfad ; Code {; Option} {; *})

Parameter	Typ		Beschreibung
Pfad	Text, Array Text	➡	Text oder Text Array mit einem oder mehreren Methodenpfaden
Code	Text, Array Text	⬅	Code der angegebenen Methode(n)
Option	Lange Ganzzahl	➡	0 oder weggelassen = einfacher Export (ohne Token), 1 = Export mit Token
*	Operator	➡	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der Befehl **METHOD GET CODE** gibt im Parameter *Code* den Inhalt der Methode(n) zurück, die im Parameter *Pfad* angegeben sind. Dieser Befehl kann den Code aller Methodentypen zurückgeben: Datenbank-, Projekt-, Formular-, Objektmethoden und Trigger.

Sie können zwei Syntaxarten verwenden, die eine basiert auf Text Arrays, die andere auf Textvariablen:

```
C_TEXT(tVpath)  // Textvariablen
C_TEXT(tVcode)
METHOD GET CODE(tVpath;tVcode)  // Code einer einzelnen Methode
```

```
ARRAY TEXT(arrPaths;0)  // Text Arrays
ARRAY TEXT(arrCodes;0)
METHOD GET CODE(arrPaths;arrCodes)  // Code mehrerer Methoden
```

Sie können die beiden Syntaxarten nicht miteinander mischen.

Übergeben Sie einen ungültigen Pfadnamen, bleibt der Parameter *Code* leer und ein Fehler wird generiert.
Für Text von *Code*, der mit diesem Befehl erzeugt wird, gilt folgendes:

- Befehlsnamen werden in allen 4D Versionen in Englisch geschrieben, außer mit einer französischen 4D Version, wenn die Einstellung "Verwende regionale Systemeinstellungen" markiert ist (siehe **Seite Methoden**).
  Mit dem Parameter *Option* kann der Code Tokens enthalten, was ihn unabhängig von der 4D Programmiersprache und Version macht (siehe unten)
- Zur besseren Lesbarkeit von Code wird der Text mit Tabs eingerückt, basierend auf den Programmierstrukturen wie im Methodeneditor.
- Beim Import von Code mit Metadaten wird im Kopfteil eine Zeile hinzugefügt, z.B.:

  ```
  // %attributes = {"lang":"fr","invisible":true,"folder":"Web3"}
  ```

  Diese Zeile wird nicht mitimportiert, nur die angegebenen Attribute werden berücksichtigt (Nicht spezifizierte Attribute werden auf ihren Standardwert zurückgesetzt). Das Attribut "lang" setzt die Exportsprache und verhindert den Import in ein Programm in einer anderen Sprache. In diesem Fall wird ein Fehler erzeugt. Das Attribut "folder" enthält den Namen des Eltern-Ordners der Methode. Es erscheint nicht, wenn es für die Methode keinen Eltern-Ordner gibt.
  Es lassen sich zusätzliche Attribute definieren. Weitere Informationen dazu finden Sie unter dem Befehl **METHOD SET ATTRIBUTES**.

Mit dem Parameter *Option* können Sie festlegen, wie Code-Elemente in Methoden exportiert werden:

- Übergeben Sie 0 oder lassen den Parameter *Option* weg, wird der Code der Methode ohne Token exportiert, d.h. so wie er im Methodeneditor erscheint.
- Übergeben Sie 1 oder die Konstante Code with tokens, wird der Code der Methode mit Token exportiert, d.h. tokenisierte Elemente erscheinen mit ihrer internen Referenz im exportierten Inhalt in *Code*. So wird z.B. der Ausdruck "String(a)" als "String:C10(a)" exportiert, wobei "C10" die interne Referenz der Funktion **String** ist.

Folgende Elemente der Programmiersprache lassen sich als Token darstellen:

- 4D Befehle und Konstanten
- Tabellen- und Feldnamen
- 4D Plug-In Befehle

Mit Tokens exportierter Code ist unabhängig von nachfolgend umbenannten Elementen der Programmiersprache. Mit Hilfe von Tokens wird Code in Textform immer korrekt von 4D interpretiert, sei es über den Befehl **METHOD SET CODE** oder auch per Copy/Paste. Weitere Informationen dazu finden Sie im Abschnitt **Tokens in Formeln verwenden**.

Wird der Befehl in einer Komponente ausgeführt, gilt er standardmäßig für die Komponentenmethoden. Übergeben Sie den Parameter *, greift er auf die Methoden der Host Datenbank zu.

Beispiel 1

Siehe Beispiel unter dem Befehl **METHOD SET CODE**.

Beispiel 2

Dieses Beispiel zeigt die Auswirkung des Parameters *Option*.
Die folgende Methode "simple_init" exportieren:

```
Case of
   :(Form event=On Load)
      ALL RECORDS([Customer])
End case
```

Mit folgendem Code:

```
C_TEXT($code)
C_TEXT($contents)
$code:=METHOD Get path(Path project method;"simple_init")
METHOD GET CODE($code;$contents;0)  //keine Tokens
TEXT TO DOCUMENT("simple_init.txt";$contents)
```

erhalten Sie als Ergebnis:

```
  //%attributes = {"lang":"en"} comment added and reserved by 4D
Case of
   : (Form event=On Load)
      ALL RECORDS([Customer])
End case
```

Mit folgendem Code:

```
C_TEXT($code)
C_TEXT($contents)
$code:=METHOD Get path(Path project method;"simple_init")
METHOD GET CODE($code;$contents;Code with tokens)  //Tokens verwenden
TEXT TO DOCUMENT("simple_init.txt";$contents)
```

erhalten Sie als Ergebnis:

```
  //%attributes = {"lang":"en"} comment added and reserved by 4D
Case of
   : (Form event:C388=On Load:K2:1)
      ALL RECORDS:C47([Customer:1])
End case
```

METHOD GET COMMENTS

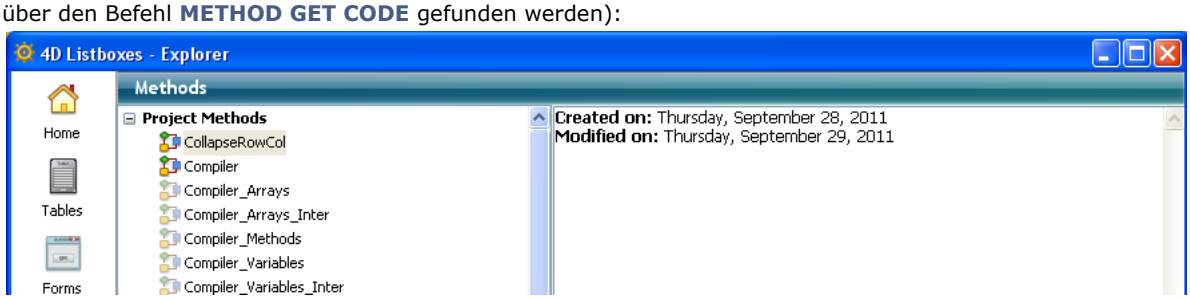
METHOD GET COMMENTS (Pfad ; Kommentare {; *})

Parameter	Typ		Beschreibung
Pfad	Text, Array Text	➡	Text oder Text Array mit einem oder mehreren Methodenpfaden
Kommentare	Text, Array Text	⬅	Kommentare der angegebenen Methode(n)
*	Operator	➡	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der Befehl **METHOD GET COMMENTS** gibt im Parameter *Kommentare* die Kommentare der Methode(n) zurück, die im Parameter *Pfad* angegeben sind.

Dieser Befehl findet Kommentare, die im 4D Explorer angegeben sind. (nicht zu verwechseln mit Kommentarzeilen im Code, die über den Befehl **METHOD GET CODE** gefunden werden):



Diese Kommentare lassen sich nur für Trigger, Projektmethoden und Formularmethoden erstellen. Sie können formatierten Text enthalten.

Hinweis: Formulare und Formularmethoden nutzen dieselben Kommentare gemeinsam.

Sie können zwei Syntaxarten verwenden, die eine basiert auf Textvariablen, die andere auf Text Arrays:

```
C_TEXT(tVpath)  // Textvariablen
C_TEXT(tVcomments)
METHOD GET COMMENTS(tVpath;tVcomments)  // Kommentare einer einzelnen Methode
```

```
ARRAY TEXT(arrPaths;0)  // Text Arrays
ARRAY TEXT(arrComments;0)
METHOD GET COMMENTS(arrPaths;arrComments)  // Kommentare mehrerer Methoden
```

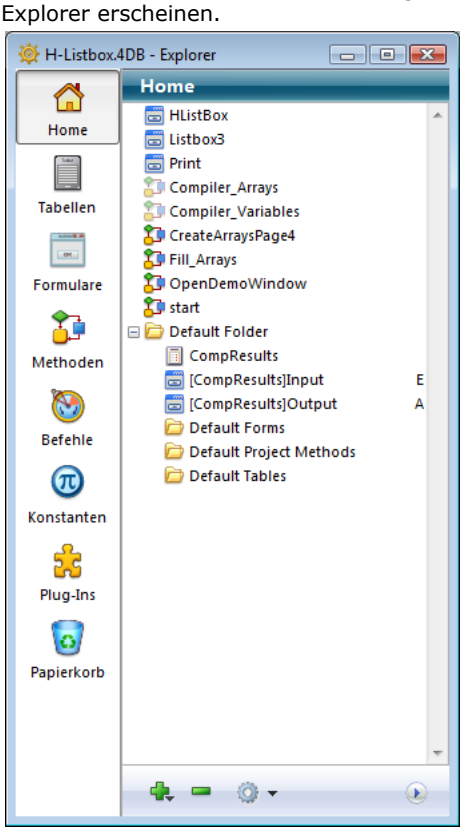
METHOD GET FOLDERS

METHOD GET FOLDERS (arrNamen {; Filter}{; *})

Parameter	Typ		Beschreibung
arrNamen	Array Text	⇐	Array der Ordnernamen der Seite Home
Filter	Text	⇒	Namen filtern
*	Operator	⇒	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der Befehl **METHOD GET FOLDERS** gibt im Array *arrNamen* die Namen der Ordner zurück, die auf der Seite Home des 4D Explorer erscheinen.



Da Ordnernamen einmalig sein müssen, wird die Hierarchie in diesem Array nicht zurückgegeben.

Sie können diese Liste auf bestimmte Ordner begrenzen, wenn Sie im Parameter *Filter* einen Vergleichsstring übergeben: Dann erscheinen nur Ordner, deren Namen zum Filter passen. Mit dem Jokerzeichen @ können Sie Filter vom Typ "beginnt mit", "endet mit" oder "enthält" übergeben. Übergeben Sie einen leeren String, wird der Parameter *Filter* ignoriert.

Wird der Befehl in einer Komponente ausgeführt, gilt er standardmäßig für die Komponentenmethoden. Übergeben Sie den Parameter *, greift er auf die Methoden der Host Datenbank zu.

METHOD GET MODIFICATION DATE

METHOD GET MODIFICATION DATE (Pfad ; modDatum ; modZeit {; *})

Parameter	Typ		Beschreibung
Pfad	Text, Array Text	➡	Text oder Text Array mit einem oder mehreren Methodenpfaden
modDatum	Datum, Array Datum	⬅	Änderungsdatum der Methode
modZeit	Zeit, Array Lange Ganzzahl	⬅	Änderungszeit der Methode
*	Operator	➡	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der Befehl **METHOD GET MODIFICATION DATE** gibt in den Parametern *modDatum* und *modZeit* Datum und Zeit der letzten Änderung in den Methoden zurück, die im Parameter *Pfad* angegeben sind.

Sie können zwei Syntaxarten verwenden, die eine basiert auf Variablen, die andere auf Arrays:

```
C_TEXT(tVpath)  // Variablen
C_DATE(vDate)
C_TIME(vTime)
METHOD GET MODIFICATION DATE(tVpath;vDate;vTime)  // Datum und Zeit einer einzelnen Methode
```

```
ARRAY TEXT(arrPaths;0)  // Arrays
ARRAY DATE(arrDates;0)
ARRAY LONGINT(arrTimes;0)
METHOD GET MODIFICATION DATE(arrPaths;arrDates;arrTimes)  // Datum und Zeit mehrerer Methoden
```

Sie können die beiden Syntaxarten nicht miteinander mischen.

Wird der Befehl in einer Komponente ausgeführt, gilt er standardmäßig für die Komponentenmethoden. Übergeben Sie den Parameter *, greift er auf die Methoden der Host Datenbank zu.

Beispiel 1

Änderungsdatum und -zeit für mehrere Methoden finden:

```
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)
```

Beispiel 2

Änderungsdatum für Methoden in einem Modul mit der Vorsilbe "Web_" erhalten. Sie können das Symbol "@" nicht in einem Pfad nutzen; aber Sie können schreiben:

```
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 (arrNamen {; Filter}{; *})

Parameter	Typ		Beschreibung
arrNamen	Array Text	⇐	Array mit Namen der Projektmethoden
Filter	Text	⇒	Namen filtern
*	Operator	⇒	Mit Stern: Befehl gilt für Host Datenbank bei Ausführung über eine Komponente. (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der Name **METHOD GET NAMES** füllt das Array *arrNamen* mit den Namen der Projektmethoden, die in der Anwendung erstellt wurden.

Standardmäßig werden alle Methoden aufgelistet. Sie können diese Liste einschränken, wenn Sie im Parameter *Filter* einen Vergleichsstring setzen. Dann gibt der Befehl nur Methoden zurück, deren Namen zum Filter passen. Um Filter vom Typ "beginnt mit", "endet mit" oder "enthält" zu setzen, müssen Sie das Zeichen @ setzen.

Wird dieser Befehl über eine Komponente ausgeführt, gibt er standardmäßig die Namen der Projektmethoden der Komponente zurück. Übergeben Sie den Parameter *, enthält das Array die Projektmethoden der Host Datenbank.

Hinweis: Methoden, die im Papierkorb liegen, werden nicht aufgelistet.

Beispiel

Hier ein paar typische Anwendungsbeispiele:

```
// Liste aller Projektmethoden der Datenbank
METHOD GET NAMES(t_Names)

// Liste der Projektmethoden, die mit einem bestimmten String beginnen
METHOD GET NAMES(t_Names;"web_@")

// Liste der Projektmethoden in der Host Datenbank, die mit einem bestimmten String beginnen
METHOD GET NAMES(t_Names;"web_@";*)
```

METHOD Get path

METHOD Get path (MethodeTyp {; Tabellenname}{; ObjektName{; FormularobjektName}}{; *}) -> Funktionsergebnis

Parameter	Typ		Beschreibung
MethodeTyp	Lange Ganzzahl	→	Selector des Objekttyps
Tabellenname	Tabelle	→	Referenz der Tabelle
ObjektName	Text	→	Name des Formulars bzw. der Datenbankmethode
FormularobjektName	Text	←	Name des Formularobjekts
*	Operator	→	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)
Funktionsergebnis	Text	↩	Kompletter Pfad des Objekts

Beschreibung

Die Funktion **METHOD Get path** gibt den kompletten internen Pfadnamen einer Methode zurück.
Im Parameter *MethodeTyp* übergeben Sie den Typ der Methode, deren Pfad Sie erhalten wollen.
Sie können dazu folgende Konstanten unter dem Thema **Zugriff Designobjekte** verwenden:

Konstante	Typ	Wert	Kommentar
Path database method	Lange Ganzzahl	2	Der Befehl gibt den Pfad der spezifizierten Datenbankmethoden (englische Namen) zurück. Liste dieser Methoden: [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	Lange Ganzzahl	4	Pfad der Projektformularmethoden und aller dazugehörigen Objektmethoden. Beispiele: [projectForm]/myForm/{formMethod} [projectForm]/myForm/button1 [projectForm]/myForm/my%2list [projectForm]/myForm/button1
Path project method	Lange Ganzzahl	1	Name der Methode Beispiel: MyProjectMethod
Path table form	Lange Ganzzahl	16	Pfad der Tabellenformularmethoden und aller dazugehörigen Objektmethoden. Beispiele: [tableForm]/table_1/Form1/{formMethod} [tableForm]/table_1/Form1/button1 [tableForm]/table_1/Form1/my%2list [tableForm]/table_2/Form1/my%2list
Path trigger	Lange Ganzzahl	8	Pfad der Datenbank-Trigger. Beispiele: [trigger]/table_1 [trigger]/table_2

In den Parametern *Tabellenname*, *ObjektName* und *FormularobjektName* übergeben Sie Werte gemäß dem Objekttyp, für den Sie den Pfadnamen der Methode erhalten wollen:

Objekttyp	*aTable*	*objectName*	*formObjectName*
Path Project form		X	X (optional)
Path Table form	X	X	X (optional)
Path Database method		X	
Path Project method		X	
Path Trigger	X		

Wird das Objekt nicht gefunden (Methodentyp unbekannt oder ungültig, fehlende Tabelle, etc.), wird ein Fehler erzeugt.
Wird der Befehl in einer Komponente ausgeführt, gibt er standardmäßig die Pfade der Komponentenmethoden zurück. Übergeben Sie den Parameter *, enthält das Array die Pfade der Methoden in der Host Datenbank.

Beispiel

```
//Pfadname der Datenbankmethode "On Startup" wiederfinden:
$path:=METHOD Get path(Path database method;"onStartup")

//Pfadname des Trigger zur Tabelle [Employees] wiederfinden:
```

```
$path:=METHOD Get path(Path trigger;[Employees])

 //Pfadname der Objektmethode "OK" des Eingabeformulars für die Tabelle [Employees] wiederfinden:
$path:=METHOD Get path(Path table form;[Employees];"input";"OK")
```

METHOD GET PATHS

METHOD GET PATHS ({OrdnerName ;} MethodeTyp ; arrPfade {; Zeitstempel}{; *})

Parameter	Typ		Beschreibung
OrdnerName	Text	➡	Name des Ordners der Seite Home
MethodeTyp	Lange Ganzzahl	➡	Selector des zu erhaltenden Methodentyps
arrPfade	Array Text	⬅	Array der Methodenpfade und Namen
Zeitstempel	Variable Lange Ganzzahl	➡	Mindestwert des Zeitstempels
		⬅	Neuer aktueller Wert
*	Operator	➡	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der **METHOD GET PATHS** füllt das Array *arrPfade* mit den internen Pfadnamen und Namen der Methoden vom Typ *MethodeTyp* in der Anwendung.

Ist Ihr Code im 4D Explorer (Seite Home) in "Ordnern" organisiert, können Sie im optionalen Parameter *OrdnerName* einen Ordnernamen übergeben. In diesem Fall enthält das Array *arrPfade* Pfade zu Methoden, die an dieser Stelle gefunden werden.

Hinweis: Im Parameter *OrdnerName* können Sie nicht das Zeichen "@" verwenden.

Im Parameter *MethodeTyp* übergeben Sie den Typ der Methode, deren Pfade Sie im Array *arrPfade* erhalten wollen.
Sie können die folgenden Konstanten - einzeln oder mehrere kombiniert - unter dem Thema **Zugriff Designobjekte**:

Konstante	Typ	Wert	Kommentar
Path all objects	Lange Ganzzahl	31	Der Befehl gibt die Pfade aller kombinierten Methoden zurück.
Path database method	Lange Ganzzahl	2	Der Befehl gibt den Pfad der spezifizierten Datenbankmethoden (englische Namen) zurück. Liste dieser Methoden: [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	Lange Ganzzahl	4	Pfad der Projektformularmethoden und aller dazugehörigen Objektmethoden. Beispiele: [projectForm]/myForm/{formMethod} [projectForm]/myForm/button1 [projectForm]/myForm/my%2list [projectForm]/myForm/button1
Path project method	Lange Ganzzahl	1	Name der Methode Beispiel: MyProjectMethod
Path table form	Lange Ganzzahl	16	Pfad der Tabellenformularmethoden und aller dazugehörigen Objektmethoden. Beispiele: [tableForm]/table_1/Form1/{formMethod} [tableForm]/table_1/Form1/button1 [tableForm]/table_1/Form1/my%2list [tableForm]/table_2/Form1/my%2list
Path trigger	Lange Ganzzahl	8	Pfad der Datenbank-Trigger. Beispiele: [trigger]/table_1 [trigger]/table_2

Mit dem Parameter *Zeitstempel* werden nur Pfade von Methoden zurückgegeben, die ab einem bestimmten Zeitpunkt geändert wurden. Als Teil eines Systems zur Versionskontrolle bedeutet dies, dass Sie nur Methoden updaten müssen, die seit dem letzten Backup geändert wurden.
Es funktioniert folgendermaßen: 4D verwaltet einen Zähler für Methodenänderungen, d.h. immer wenn eine Methode erstellt oder erneut gesichert wird, wird dieser Zähler erhöht und sein aktueller Wert im internen Zeitstempel der Methode gespeichert. Übergeben Sie den Parameter *Zeitstempel*, gibt der Befehl nur Methoden zurück, deren Stempel größer oder gleich dem hier übergebenen Wert ist. Außerdem gibt er in Zeitstempel den neuen aktuellen Wert des Zählers der Änderung zurück, z.B. den höchsten Wert. Sichern Sie diesen Wert, können Sie ihn beim nächsten Aufrufen des Befehls übergeben, so dass Sie nur neue bzw. geänderte Methoden erhalten.

Wird der Befehl in einer Komponente ausgeführt, gibt er standardmäßig die Pfade der Komponentenmethoden zurück. Übergeben Sie den Parameter *, enthält das Array die Pfade der Methoden in der Host Datenbank.

Findet der Befehl einen duplizierten Methodennamen, wird der Fehler -9802 generiert (Objektpfad ist nicht einmalig). In diesem Fall empfehlen wir, das MSC aufzurufen und die Datenbankstruktur zu überprüfen.

Beispiel 1

Projektmethoden im Ordner "web" wiederfinden:

```
METHOD GET PATHS("web"; Path Project method;arrPaths)
```

Beispiel 2

Datenbankmethoden und Trigger wiederfinden:

```
METHOD GET PATHS(Path trigger+Path database method;arrPaths)
```

Beispiel 3

Projektmethoden wiederfinden, die seit dem letzten Backup geändert wurden:

```
 // Wir laden den zuletzt gesicherten Wert
$stamp :=Max([Backups]cur_stamp)
METHOD GET PATHS(Path project method;arrPaths;$stamp)
 // Wir sichern den neuen Wert
CREATE RECORD([Backups])
[Backups]cur_stamp :=$stamp
SAVE RECORD([Backups])
```

Beispiel 4

Siehe Beispiel zum Befehl **METHOD SET CODE**.

METHOD GET PATHS FORM

METHOD GET PATHS FORM ({Tabellenname ;} arrPfade {; Filter}{; Zeitstempel}{; *})

Parameter	Typ		Beschreibung
Tabellenname	Tabelle	➡	Referenz der Tabelle
arrPfade	Array Text	⬅	Array mit Methodenpfaden und Namen
Filter	Text	➡	Namen filtern
Zeitstempel	Variable Lange Ganzzahl	➡	Mindestwert des Zeitstempel
		⬅	Neuer aktueller Wert
*	Operator	➡	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der Befehl **METHOD GET PATHS FORM** füllt das Array *arrPfade* mit den internen Pfadnamen und Namen aller Formularobjektmethoden sowie Formularmethoden. Formularmethoden werden bezeichnet mit {formMethod}.
Nur Objekte mit Code werden aufgelistet. So erscheinen beispielsweise keine Schaltflächen, denen nur eine Standardaktion zugeordnet ist.

Übergeben Sie den Parameter *Tabellenname*, gibt der Befehl die Objekte der Tabellenformulare zurück, die dieser Tabelle zugeordnet sind. Lassen Sie diesen Parameter weg, gibt er die Objekte der Projektformulare der Datenbank zurück.
Sie können die Liste der Formulare begrenzen, wenn Sie im Parameter *Filter* einen Vergleichsstring übergeben: Dann erscheinen nur Formulare, deren Namen zum Filter passen. Mit dem Jokerzeichen @ können Sie Filter vom Typ "beginnt mit", "endet mit" oder "enthält" übergeben. Übergeben Sie einen leeren String, wird der Parameter *Filter* ignoriert. Mit dem Parameter *Zeitstempel* werden nur Pfade von Methoden zurückgegeben, die nach einem bestimmten Zeitpunkt geändert wurden. Als Teil eines Systems zur Versionskontrolle bedeutet dies, dass Sie nur Methoden updaten müssen, die seit dem letzten Backup geändert wurden.
Es funktioniert folgendermaßen: 4D verwaltet einen Zähler für Methodenänderungen, d.h. immer wenn eine Methode erstellt oder erneut gesichert wird, erhöht sich der Zähler und sein aktueller Wert wird im internen Stempel der Methode gespeichert. Übergeben Sie den Parameter *Zeitstempel*, gibt der Befehl nur Methoden zurück, deren Stempel größer oder gleich dem hier übergebenen Wert ist. Außerdem gibt der Befehl in *Zeitstempel* den neuen aktuellen Wert des Zählers der Änderung zurück, z.B. den höchsten Wert. Sichern Sie diesen Wert, können Sie ihn beim nächsten Aufrufen des Befehls übergeben, so dass Sie nur neue bzw. geänderte Methoden erhalten.

Wird der Befehl in einer Komponente ausgeführt, gibt er standardmäßig die Pfade der Komponentenmethoden zurück. Übergeben Sie den Parameter *, enthält das Array die Pfade der Methoden in der Host Datenbank.

Hinweis: Der Befehl listet keine Objekte von vererbten Formularen bzw. Unterformularen.

Findet der Befehl einen duplizierten Methodennamen, wird der Fehler -9802 generiert (Objektpfad ist nicht einmalig). In diesem Fall empfehlen wir, das MSC aufzurufen und die Datenbankstruktur zu überprüfen.

Beispiel 1

Liste aller Objekte des Eingabeformulars zur Tabelle [Employees]. Beachten Sie, dass Tabellenformularmethoden und Projektformularmethoden als Objekte bearbeitet werden, die zu einem Formular gehören:

```
METHOD GET PATHS FORM([Employees];arrPaths;"input")
  // Inhalt von arrPaths (zum Beispiel)
  // [tableForm]/input/{formMethod} -> Formularmethode
  // [tableForm]/input/bOK -> Objektmethode
  // [tableForm]/input/bCancel -> Objektmethode
```

Beispiel 2

Liste der Objekte zum Projektformular "dial":

```
METHOD GET PATHS FORM(arrPaths;"dial")
```

Beispiel 3

Liste aller Objekte zum Eingabeformular der Tabelle [Employees] in einer Komponente:

```
METHOD GET PATHS FORM(([Employees];arrPaths;"input@";*)
```

METHOD OPEN PATH

METHOD OPEN PATH (Pfad {; *})

Parameter	Typ		Beschreibung
Pfad	Text	➡	Pfad der zu öffnenden Methode
*	Operator	➡	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der Befehl **METHOD OPEN PATH** öffnet im 4D Methodeneditor die Methode, deren interner Pfadname im Parameter *Pfad* übergeben ist.
Dieser Befehl kann alle Methodentypen (Datenbank, Projekt, Formular, Objekt oder Trigger) öffnen; die Methode muss allerdings bereits existieren. Entspricht der Parameter *Pfad* keiner vorhandenen Methode, wird der Fehler -9801 ("Methode lässt sich nicht öffnen") zurückgegeben.

Sie können diesen Befehl über eine Komponente ausführen, Sie müssen dann den Parameter * übergeben, da der Code der Komponente im Nur-Lesen Modus ist. Lassen Sie den Parameter * weg, wird der Fehler -9763 erzeugt.

METHOD RESOLVE PATH

METHOD RESOLVE PATH (Pfad ; MethodeTyp ; ZeigerTabelle ; ObjektName ; FormularobjektName {; *})

Parameter	Typ		Beschreibung
Pfad	Text	➡	Aufzulösender Pfad
MethodeTyp	Lange Ganzzahl	⬅	Selector vom Typ Objekt
ZeigerTabelle	Zeiger	⬅	Referenz der Tabelle
ObjektName	Text	⬅	Name des Formulars oder Datenbankmethode
FormularobjektName	Text	⬅	Name des Formularobjekts
*	Operator	➡	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der Befehl **METHOD RESOLVE PATH** analysiert den internen Pfadnamen, angegeben im Parameter *Pfad*, und gibt in den Parametern *MethodeTyp*, *ZeigerTabelle*, *ObjektName* und *FormularobjektName* die verschiedenen Bestandteile zurück.

Der Parameter *MethodeTyp* empfängt einen Wert, der den Typ der Methode angibt. Sie können diesen Wert mit folgenden Konstanten unter dem Thema **Zugriff Designobjekte** vergleichen:

Konstante	Typ	Wert	Kommentar
Path database method	Lange Ganzzahl	2	Der Befehl gibt den Pfad der spezifizierten Datenbankmethoden (englische Namen) zurück. Liste dieser Methoden: [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	Lange Ganzzahl	4	Pfad der Projektformularmethoden und aller dazugehörigen Objektmethoden. Beispiele: [projectForm]/myForm/{formMethod} [projectForm]/myForm/button1 [projectForm]/myForm/my%2list [projectForm]/myForm/button1
Path project method	Lange Ganzzahl	1	Name der Methode Beispiel: MyProjectMethod
Path table form	Lange Ganzzahl	16	Pfad der Tabellenformularmethoden und aller dazugehörigen Objektmethoden. Beispiele: [tableForm]/table_1/Form1/{formMethod} [tableForm]/table_1/Form1/button1 [tableForm]/table_1/Form1/my%2list [tableForm]/table_2/Form1/my%2list
Path trigger	Lange Ganzzahl	8	Pfad der Datenbank-Trigger. Beispiele: [trigger]/table_1 [trigger]/table_2

Der Parameter *ZeigerTabelle* enthält einen Zeiger auf eine Tabelle der Datenbank, wenn der Pfad auf eine Tabellenformularmethode oder einen Trigger verweist.

Der Parameter *ObjektName* enthält entweder:

- Den Namen eines Formulars, wenn der Pfad auf ein Tabellenformular oder Projektformular verweist.
- Den Namen einer Datenbankmethode, wenn der Pfad auf eine Datenbankmethode verweist.

Der Parameter *FormularobjektName* enthält den Namen eines Formularobjekts, wenn der Pfad auf eine Objektmethode verweist. Wird dieser Befehl über eine Komponente ausgeführt, geht er standardmäßig davon aus, dass *Pfad* eine Methode der Komponente angibt. Übergeben Sie den Parameter *, geht er davon aus, dass *Pfad* eine Methode der Host Datenbank angibt.

Beispiel 1

Auflösung des Pfads einer Datenbankmethode:

```
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: ""
```

Beispiel 2

Auflösung des Pfads für ein Objekt einer Tabellenformularmethode:

```
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 (Modus)

Parameter	Typ		Beschreibung
Modus	Lange Ganzzahl	➡	Zugriffsmodus für gesperrte Objekte

Beschreibung

Der Befehl **METHOD SET ACCESS MODE** setzt den Modus für 4D, wenn Sie versuchen, auf ein Objekt im Schreibmodus zuzugreifen, das bereits von einem anderen Benutzer oder Prozess geladen wurde. Die Reichweite dieses Befehls ist die Arbeitssitzung.

In *Modus* übergeben Sie eine der folgenden Konstanten unter dem Thema **Zugriff Designobjekte**:

Konstante	Typ	Wert	Kommentar
On object locked abort	Lange Ganzzahl	0	Laden des Objekts wird abgebrochen (Standard Funktionsweise)
On object locked confirm	Lange Ganzzahl	2	4D zeigt ein Dialogfenster, über das Sie die Operation erneut ausführen oder abbrechen können. Das wird im remote Modus nicht unterstützt, d.h. Laden wird abgebrochen
On object locked retry	Lange Ganzzahl	1	4D versucht das Objekt weiter zu laden, bis es freigegeben ist.

METHOD SET ATTRIBUTE

METHOD SET ATTRIBUTE (Pfad ; attrTyp ; attrWert {; attrTyp2 ; attrWert2 ; ... ; attrTypN ; attrWertN}{; *})

Parameter	Typ		Beschreibung
Pfad	Text	➡	Pfad der Projektmethode
attrTyp	Lange Ganzzahl	➡	Typ des Attributs
attrWert	Boolean, Text	➡	Wahr = Attribut wählen Falsch = Attribut abwählen oder Ordnername
*	Operator	➡	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der Befehl **METHOD SET ATTRIBUTE** setzt den Wert des Parameters *attrTyp* für die Projektmethode, angegeben im Parameter *Pfad*. Dieser Befehl funktioniert nur mit Projektmethoden. Übergeben Sie einen ungültigen Pfad, wird ein Fehler erzeugt.

Im Parameter *attrTyp* übergeben Sie einen Wert für den zutreffenden Attributtyp. Sie können folgende Konstanten unter dem Thema **Zugriff Designobjekte** verwenden:

Konstante	Typ	Wert	Kommentar
Attribute executed on server	Lange Ganzzahl	8	Entspricht der Option "Auf Server ausführen"
Attribute folder name	Lange Ganzzahl	1024	Name des Ordners für die Methode (Attribut "Ordner"). Übergeben Sie diese Konstante, müssen Sie in *attrWert* einen Ordnernamen übergeben: • Entspricht dieser Name einem gültigen Ordner, wird die Methode in diesen Elternordner gesetzt • Existiert der Ordner nicht, ändert der Befehl nichts auf der Ebene des Elternordners • Übergeben Sie einen leeren String, wird die Methode auf die Root-Ebene gesetzt
Attribute invisible	Lange Ganzzahl	1	Entspricht der Option "Unsichtbar"
Attribute published SOAP	Lange Ganzzahl	3	Entspricht der Option "Zugang per Web Service"
Attribute published SQL	Lange Ganzzahl	7	Entspricht der Option "Zugang per SQL"
Attribute published Web	Lange Ganzzahl	2	Entspricht der Option "Zugang per 4D HTML Tags und URLs (4DACTION...)"
Attribute published WSDL	Lange Ganzzahl	4	Entspricht der Option "Anbieten per WSDL"
Attribute shared	Lange Ganzzahl	5	Entspricht der Option "Gemeinsam von Komponenten und Host benutzt"

Im Parameter *attrWert* können Sie folgendes übergeben:

- Wahr, um die entsprechende Option auszuwählen, Falsch, um sie abzuwählen
- Ein String (Ordnername), wenn Sie in *attrTyp* die Konstante Attribute folder name verwendet haben

Sie können in einem einzelnen Aufruf mehrere Paare *attrTyp*;*attrWert* übergeben.

Sie können diesen Befehl über eine Komponente ausführen, Sie müssen dann den Parameter * übergeben, da der Code der Komponente im Nur-Lesen Modus ist. Lassen Sie den Parameter * weg, wird der Fehler -9763 erzeugt.

Hinweis:

Dieser Befehl lässt sich nicht im kompilierten Modus ausführen. Beim Aufrufen in diesem Modus wird der Fehler -9762 generiert.

Beispiel 1

Für die Projektmethode "Choose dialog" die Eigenschaft "Gemeinsam von Komponenten und Host benutzt" auswählen:

```
METHOD SET ATTRIBUTE("Choose dialog";Attribute shared;True)
```

Beispiel 2

Mehrere Paare Attribut/Wert setzen:

```
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 (Pfad ; Attribute {; *})

Parameter	Typ		Beschreibung
Pfad	Text, Array Text	➡	Methodenpfade
Attribute	Objekt, Array Objekt	➡	Attribute für die gewählte(n) Methode(n)
*	Operator	➡	Mit Stern: Bei Ausführung über eine Komponente gilt der Befehl für die Host Datenbank (in anderem Kontext wird er ignoriert).

Beschreibung

Der Befehl **METHOD SET ATTRIBUTES** setzt die *Attribute* für die Methode(n), angegeben im Parameter *Pfad*.

In *Pfad* übergeben Sie entweder einen Text mit einem Methodenpfad oder ein Array Text mit einem Array der Pfade. In *Attribute* müssen Sie jeweils denselben Parameter (String oder Array) übergeben, um die dazugehörigen Attribute zu setzen. Dieser Befehl funktioniert nur mit Projektmethoden. Übergeben Sie einen ungültigen *Pfad*, wird ein Fehler generiert.

In *Attribute* übergeben Sie ein Objekt oder ein Array, je nach Art des Parameters in *Pfad*, mit allen Attributen, die Sie für die Methode(n) setzen wollen.

Attribute zu Methoden müssen über die Befehle **OB SET** oder **OB SET ARRAY** gesetzt werden, mit den Werten "true"/"false" für Boolean Attribute oder spezifische Werte für erweiterte Attribute, z.B. die 4D Mobile Eigenschaft "scope":"table". In den Methodenattributen werden nur die im Parameter *Attribute* angegebenen Attribute aktualisiert.

Bei Ausführung über eine Komponente gilt der Befehl standardmäßig für die Komponentenmethode. Mit dem Parameter * greift er auf die Methoden der Host Datenbank zu.

Hinweis: Der Befehl **METHOD SET ATTRIBUTE** wird zur Wahrung der Kompatibilität weiterhin unterstützt. Da er jedoch nur einfache Werte setzen kann, lässt er sich nicht für erweiterte Attribute wie 4D Mobile Eigenschaften verwenden.

Folgende Attribute werden unterstützt:

{ "invisible" : false, // true, wenn sichtbar "preemptive" : "capable" // oder "incapable" oder "indifferent" "publishedWeb" : false, // true, wenn durch 4D tags und URLs verfügbar "publishedSoap": false, // true, wenn als Web Service angeboten "publishedWsdl": false, // true, wenn in WSDL veröffentlicht "shared" : false, // true, wenn gemeinsam mit Kompontenten und Host Datenbank genutzt "publishedSql" : false, // true, wenn über SQL verfügbar "executedOnServer" : false, // true, wenn auf dem Server ausgeführt "published4DMobile" : { "scope": "table", // "none" oder "table" oder "currentRecord" oder "currentSelection" "table": "aTableName" // erscheint, wenn die Reichweite anders als "none" ist } }

Hinweis: Das Attribut "published4DMobile" wird ignoriert, wenn der Wert "table" nicht existiert oder "scope" ungültig ist.

Beispiel 1

Ein einzelnes Attribut setzen:

```
C_OBJECT($attributes)
OB SET($attributes;"executedOnServer";True)
METHOD SET ATTRIBUTES("aMethod";$attributes)  //Nur das Attribut "executedOnServer" wird geändert
```

Beispiel 2

Eine Methode soll über 4D Mobile nicht verfügbar sein (das Attribut "scope" muss den Wert "none" erhalten):

```
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 (Pfad ; Code {; *})

Parameter	Typ		Beschreibung
Pfad	Text, Array Text	➡	Text oder Text Array mit einem oder mehreren Methodenpfaden
Code	Text, Array Text	➡	Code der angegebenen Methode(n)
*	Operator	➡	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der Befehl **METHOD SET CODE** ersetzt den Code der Methode(n), angegeben im Parameter *Pfad* mit dem im Parameter *Code* übergebenen Inhalt. Dieser Befehl kann auf den Code aller Methodentypen zugreifen: Datenbank-, Projekt-, Formular-, Objektmethoden und Trigger.

Bei einer Projektmethode gilt folgendes: Existiert diese Methode bereits in der Datenbank, wird ihr Inhalt ersetzt; existiert sie nicht, wird sie mit ihrem Inhalt erstellt. Sie können zwei Syntaxarten verwenden, die eine basiert auf Text Arrays, die andere auf Textvariablen:

```
C_TEXT(tVpath)  // Textvariablen
C_TEXT(tVcode)
METHOD SET CODE(tVpath;tVcode)  // Code einer einzelnen Methode
```

```
ARRAY TEXT(arrPaths;0)  // Text Arrays
ARRAY TEXT(arrCodes;0)
METHOD SET CODE(arrPaths;arrCodes)  // Code mehrerer Methoden
```

Sie können die beiden Syntaxarten nicht miteinander mischen. Übergeben Sie einen ungültigen Pfadnamen, führt der Befehl nichts aus.

Wird **METHOD SET CODE** aufgerufen, werden die Methodenattribute standardmäßig zurückgesetzt. Enthält jedoch die erste Zeile der Methode *Code* gültige Metadaten (in JSON), werden damit die Methodenattribute spezifiziert und die erste Zeile wird nicht eingefügt. Beispiel für Metadaten:

```
// %attributes = {"invisible":true,"lang":"fr","folder":"Security"}
```

Hinweis: Diese Metadaten werden automatisch vom Befehl **METHOD GET CODE** generiert. Weitere Informationen dazu finden Sie unter dem Befehl **METHOD SET ATTRIBUTES**.

Für die Eigenschaft "Ordner" der Metadaten gilt folgendes:

- Ist diese Eigenschaft vorhanden und entspricht einem gültigen Ordner, wird die Methode in ihren Explorer-Ordner gesetzt.
- Ist diese Eigenschaft nicht vorhanden oder existiert der Ordner nicht, macht der Befehl keine Änderung auf der Ebene des Explorer-Ordners.
- Enthält diese Eigenschaft einen leeren String, wird die Methode auf die Root Ebene gesetzt.

Sie können diesen Befehl über eine Komponente ausführen, Sie müssen dann den Parameter * übergeben, da der Code der Komponente im Nur-Lesen Modus ist. Lassen Sie den Parameter * weg, wird der Fehler -9763 erzeugt.

Beispiel

Dieses Beispiel exportiert und importiert alle Projektmethoden einer Anwendung:

```
$root_t:=Get 4D folder(Database folder)+"Methoden"+Folder separator
ARRAY TEXT($fileNames_at;0)
CONFIRM("Import oder Export der Methoden?";"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 (Pfad ; Kommentare {; *})

Parameter	Typ		Beschreibung
Pfad	Text, Array Text	➡	Text oder Text Array mit einem oder mehreren Methodenpfaden
Kommentare	Text, Array Text	➡	Kommentare der angegebenen Methode(n)
*	Operator	➡	Mit Stern: Befehl gilt für Host Datenbanken, wenn er von einer Komponente ausgeführt wird. (außerhalb dieses Kontexts wird der Parameter ignoriert.)

Beschreibung

Der Befehl **METHOD SET COMMENTS** ersetzt die Kommentare der Methode(n), angegeben im Parameter *Pfad*, durch die im Parameter *Kommentare* angegebenen Kommentare.

Die Kommentare, die dieser Befehl ändert, sind im 4D Explorer spezifiziert - nicht zu verwechseln mit den Kommentarzeilen im Code. Diese Kommentare lassen sich nur für Trigger, Projektmethoden und Formularmethoden verwenden. Sie enthalten formatierten Text.

Hinweis: Formulare und Formularmethoden verwenden die gleichen Kommentare gemeinsam.

Sie können zwei Syntaxarten verwenden, die eine basiert auf Text Arrays, die andere auf Textvariablen:

```
C_TEXT(tVpath)  // Textvariablen
C_TEXT(tVcomments)
METHOD SET COMMENTS(tVpath;tVcomments)  // Kommentare für eine einzelne Methode
```

```
ARRAY TEXT(arrPaths;0)  // Text Arrays
ARRAY TEXT(arrComments;0)
METHOD SET COMMENTS(arrPaths;arrComments)  // Kommentare für mehrere Methoden
```

Sie können die beiden Syntaxarten nicht miteinander mischen. Übergeben Sie einen ungültigen Pfadnamen, führt der Befehl nichts aus.

Sie können diesen Befehl über eine Komponente ausführen, Sie müssen dann den Parameter * übergeben, da der Code der Komponente im Nur-Lesen Modus ist. Lassen Sie den Parameter * weg, wird der Fehler -9763 erzeugt.

Beispiel

Sie können zwei Syntaxarten verwenden, die eine basiert auf Text Arrays, die andere auf Textvariablen:

```
METHOD GET COMMENTS("[trigger]/Table1";$comments)
$comments:="Modif:"+String(Current date)+"\r"+$comments
METHOD SET COMMENTS("[trigger]/Table1";$comments)
```

Konstantenthemen

- 4D Umgebung
- 4D Write Pro
- ASCII Codes
- Backup und Wiederherstellen
- Bild Anzeigeformate
- Bild Metadaten Namen
- Bild Metadaten Werte
- Bildkomprimierung
- Bildtransformation
- BLOB
- Cache Verwaltung
- Datenbank Engine
- Datenbankereignisse
- Datenbankparameter
- Datendatei Wartung
- Datum Anzeigeformate
- Digest Typ
- Druckoptionen
- Ereignisse (Inhalt)
- Ereignisse (Zusatztasten)
- Euro Währungen
- Farben
- Feld und Variablentypen
- Fenster
- Find window
- Formular Objekttypen
- Formularbereich
- Formularereignisse
- Formularobjekte (Eigenschaften)
- Formularobjekte (Zugriff)
- Formularoptionen
- Formulas
- Funktionstasten
- Graph Parameter
- Hierarchische Listen
- HTTP Client
- Indextyp
- Is license available
- ISO Latin Zeichensatz
- Kommunikation
- LDAP
- Listbox
- Listbox Fußteil Berechnung
- Log Ereignisse
- Mathematik
- Mehrfachstil Text
- Mehrfachstil Textattribute
- Menüzeilen Eigenschaften
- Objekte und Collections
- Open form window
- Open Window
- Pasteboard
- PHP
- Plattformeigenschaften
- Plattformoberfläche
- Prozesstatus
- Prozesstypen
- QR Ausgabeart
- QR Befehle
- QR Bereichseigenschaften
- QR Berichtstypen
- QR Dokumenteigenschaften
- QR Operatoren
- QR Rahmen
- QR Texteigenschaften
- QR Zeilen für Eigenschaften
- Ressourcen Eigenschaften
- Schriften Typliste
- Schriftstile
- SCREEN DEPTH
- SET RGB COLORS
- SQL
- Standard System Signaturen
- Standardaktion
- Strings
- Suchen

- Systemdokumente
- Systemformat
- Systemordner
- Tage und Monate
- Tastaturabkürzungen
- TCP Port Nummern
- Trigger Ereignisse
- Verknüpfungen
- Web Area
- Web Server
- Web Services (Client)
- Web Services (Server)
- Wertebereiche
- Wörterbücher
- XML
- Zeit Anzeigeformate
- Zugewiesene Standardaktion
- Zugriff Designobjekte

4D Umgebung

Konstante	Typ	Wert	Kommentar
_o_4D Interpreted desktop	Lange Ganzzahl	2	
_o_Extras folder	Lange Ganzzahl	2	Ordner mit angepasstem Inhalt, der auf jeden Client-Rechner geladen wird. **Hinweis zur Kompatibilität:** Ab 4D Version 11.2 SQL sollte der Ordner *Extras* nicht mehr für die angepasste Kommunikation zwischen Server und Remote-Rechnern verwendet werden. Wir empfehlen, dafür den Ordner *Resources* zu verwenden (Beschreibung siehe unten). 4D Server unterstützt zur Wahrung der Kompatibilität mit vorhandenen Anwendungen weiterhin den Ordner *Extras*. **Hinweis:** Existiert der Ordner *Extras* nicht für die Datenbank, wird er angelegt, wenn Sie die Funktion **Get 4D folder** mit der Konstanten Extras Folder ausführen.
_o_Full Version	Lange Ganzzahl	0	
4D Client database folder	Lange Ganzzahl	3	
4D Desktop	Lange Ganzzahl	3	
4D Local mode	Lange Ganzzahl	0	
4D Remote mode	Lange Ganzzahl	4	
4D Server	Lange Ganzzahl	5	
4D Volume desktop	Lange Ganzzahl	1	
64 bit version	Lange Ganzzahl	1	
Active 4D Folder	Lange Ganzzahl	0	
Backup configuration file	Lange Ganzzahl	1	Datei Backup.xml, im Ordner Preferences/Backup neben der Strukturdatei der Anwendung gespeichert
Backup log file	Lange Ganzzahl	13	Logbuch des aktuellen Backup. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde kein Logbuch angelegt oder existiert es nicht, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Build application log file	Lange Ganzzahl	14	Aktuelles Logbuch im xml Format des Application Builder. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde kein Logbuch angelegt oder existiert es nicht, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Compacting log file	Lange Ganzzahl	6	Logbuch der letzten Komprimierung der Anwendung, die mit dem Befehl **Compact data file** oder über das Maintenance und Security-Center (MSC) ausgeführt wurde. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde kein Logbuch angelegt oder existiert es nicht, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Current localization	Lange Ganzzahl	1	Aktuelle Sprache der Anwendung: Standardsprache oder Sprache, die über den Befehl **SET DATABASE LOCALIZATION** gesetzt wird.
Current resources folder	Lange Ganzzahl	6	
Data folder	Lange Ganzzahl	9	
Database folder	Lange Ganzzahl	4	
Database folder Unix syntax	Lange Ganzzahl	5	
Debug log file	Lange Ganzzahl	12	Mit dem Befehl **SET DATABASE PARAMETER**(Debug log recording) erstelltes Logbuch. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde kein Debug Loguch angelegt oder existiert es nicht, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Default localization	Lange Ganzzahl	0	Sprache, die 4D automatisch beim Starten setzt gemäß dem Ordner Resources und der Systemumgebung (nicht änderbar).
Demo version	Lange Ganzzahl	0	
Diagnostic log file	Lange Ganzzahl	11	Mit dem Befehl **SET DATABASE PARAMETER**(Diagnostic log recording) erstelltes Logbuch. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde keine Diagnose ausgeführt oder existiert kein Logbuch, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Full method text	Lange Ganzzahl	1	
Highlighted method text	Lange Ganzzahl	2	
HTML Root folder	Lange Ganzzahl	8	

Konstante	Typ	Wert	Kommentar
HTTP debug log file	Lange Ganzzahl	9	Mit dem Befehl **WEB SET OPTION**(Web debug log) erstelltes Logbuch. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde kein Debug Logbuch angelegt oder existiert es nicht, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Internal 4D localization	Lange Ganzzahl	3	Sprache, die 4D zum Sortieren und für Textvergleiche verwendet (definiert in den Einstellungen der Anwendung).
Last backup file	Lange Ganzzahl	2	Letzte Backup Datei mit Namen <baseName>[bkpNum].4BK, an einem eigenen Ort gespeichert
Licenses folder	Lange Ganzzahl	1	
Logs folder	Lange Ganzzahl	7	
Merged application	Lange Ganzzahl	2	Version ist eine Anwendung mit einkompilierter 4D Volume Desktop
Processes only	Lange Ganzzahl	1	Gibt nur die Prozessliste zurück
Repair log file	Lange Ganzzahl	7	Protokoll der ausgeführten Reparaturen der Datenbank im Maintenance und Security Center (MSC). Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde keine Reparatur ausgeführt oder existiert kein Logbuch, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Request log file	Lange Ganzzahl	10	Logbuch mit standardmäßigen Client-/Server Anfragen (ohne Web Anfragen). das mit **SET DATABASE PARAMETER**(4D Server log recording) oder **SET DATABASE PARAMETER**(Client log recording) erstellt wurde. Bei Ausführung auf dem Server wird das Server Log zurückgegeben (gespeichert im Ordner Logs auf dem Server). Bei Ausführung auf dem Client wird das Client Log zurückgegeben (gespeichert im lokalen Ordner Logs des Clients). Existiert kein Logbuch, wird ein leerer Pfad zurückgegeben.
Sessions only	Lange Ganzzahl	2	Gibt nur die Liste Benutzersitzung zurück
Structure settings	Lange Ganzzahl	0	Zugriff auf die "Struktur-Einstellungen" (Standardwert, wenn Parameter weggelassen). In diesem Modus sind die für *Selector* verwendeten Pfade identisch mit denen im Standardmodus.
User settings	Lange Ganzzahl	1	Zugriff auf "Benutzer-Einstellungen". In diesem Modus sind im Parameter *Selector* nur bestimmte Pfade möglich.
User settings file	Lange Ganzzahl	3	Datei Settings.4DSettings für alle Datendateien, falls aktiviert, im Ordner Preferences neben der Strukturdatei der Anwendung gespeichert
User settings file for data	Lange Ganzzahl	4	Datei Settings.4DSettings für aktuelle Datendatei, im Ordner Preferences neben der Datendatei der Anwendung gespeichert
User settings for data	Lange Ganzzahl	2	Zugriff auf "Benutzereinstellungen für Datendatei", d.h. Benutzereinstellungen auf derselben Ebene wie die Datendatei gespeichert. In diesem Modus lassen sich im Parameter *Selector* nur bestimmte Pfade verwenden (gleiche Untermenge wie Benutzereinstellungen)
User system localization	Lange Ganzzahl	2	Sprache, die der aktuelle Systembenutzer setzt.
Verification log file	Lange Ganzzahl	5	Logbücher, die mit den Befehlen **VERIFY CURRENT DATA FILE** und **VERIFY DATA FILE** oder über das Maintenance und Security Center (MSC) erstellt wurden. Werden im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde keine Überprüfung ausgeführt oder existiert kein Logbuch, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.
Web request log file	Lange Ganzzahl	8	Mit dem Befehl **WEB SET OPTION**(Web log recording) erstelltes Logbuch. Wird im Ordner Logs neben der Strukturdatei der Anwendung gespeichert. Wurde kein Logbuch angelegt oder existiert es nicht, wird ein leerer Pfad zurückgegeben. Es erscheint keine Fehlermeldung.

4D Write Pro

Weitere Informationen dazu finden Sie im Handbuch **4D Write Pro Programmiersprache**.

Konstante	Typ	Wert	Kommentar
wk 4wp	Lange Ganzzahl	4	Das 4D Write Pro Dokument wird in einem nativen Archivformat gesichert (gezipptes HTML und Bilder in einem eigenen Ordner abgelegt). 4D spezifische Tags sind enthalten, 4D Ausdrücke werden nicht berechnet. Dieses Format ist besonders geeignet, um 4D Write Pro Dokumente ohne Verluste auf der Festplatte zu sichern und archivieren.
wk armenian	Lange Ganzzahl	19	
wk author	Zeichenkette	author	
wk auto	Lange Ganzzahl	0	
wk background clip	Zeichenkette	backgroundClip	
wk background color	Zeichenkette	backgroundColor	
wk background image	Zeichenkette	backgroundImage	
wk background origin	Zeichenkette	backgroundOrigin	
wk background position h	Zeichenkette	backgroundPositionH	
wk background position v	Zeichenkette	backgroundPositionV	
wk background repeat	Zeichenkette	backgroundRepeat	
wk background size h	Zeichenkette	backgroundSizeH	
wk background size v	Zeichenkette	backgroundSizeV	
wk bar	Lange Ganzzahl	4	
wk baseline	Lange Ganzzahl	4	
wk border box	Lange Ganzzahl	0	
wk border color	Zeichenkette	borderColor	
wk border color bottom	Zeichenkette	borderColorBottom	
wk border color left	Zeichenkette	borderColorLeft	
wk border color right	Zeichenkette	borderColorRight	
wk border color top	Zeichenkette	borderColorTop	
wk border radius	Zeichenkette	borderRadius	
wk border style	Zeichenkette	borderStyle	
wk border style bottom	Zeichenkette	borderStyleBottom	
wk border style left	Zeichenkette	borderStyleLeft	
wk border style right	Zeichenkette	borderStyleRight	
wk border style top	Zeichenkette	borderStyleTop	
wk border width	Zeichenkette	borderWidth	
wk border width bottom	Zeichenkette	borderWidthBottom	
wk border width left	Zeichenkette	borderWidthLeft	

Konstante	Typ	Wert	Kommentar
wk border width right	Zeichenkette	borderWidthRight	
wk border width top	Zeichenkette	borderWidthTop	
wk bottom	Lange Ganzzahl	1	
wk capitalize	Lange Ganzzahl	1	
wk center	Lange Ganzzahl	2	
wk circle	Lange Ganzzahl	11	
wk cjk ideographic	Lange Ganzzahl	24	
wk club	Lange Ganzzahl	27	
wk company	Zeichenkette	company	
wk contain	Lange Ganzzahl	-1	
wk content box	Lange Ganzzahl	2	
wk cover	Lange Ganzzahl	-2	
wk custom	Lange Ganzzahl	29	
wk dashed	Lange Ganzzahl	3	
wk date creation	Zeichenkette	dateCreation	
wk date modified	Zeichenkette	dateModified	
wk decimal	Lange Ganzzahl	3	
wk decimal greek	Lange Ganzzahl	28	
wk decimal leading zero	Lange Ganzzahl	13	
wk default	Lange Ganzzahl	-1	
wk diamond	Lange Ganzzahl	26	
wk direction	Zeichenkette	direction	
wk disc	Lange Ganzzahl	10	
wk dotted	Lange Ganzzahl	2	
wk double	Lange Ganzzahl	4	
wk dpi	Zeichenkette	dpi	
wk end text	Lange Ganzzahl	0	
wk false	Lange Ganzzahl	0	
wk font	Zeichenkette	font	
wk font bold	Zeichenkette	fontBold	
wk font family	Zeichenkette	fontFamily	
wk font italic	Zeichenkette	fontItalic	
wk font size	Zeichenkette	fontSize	
wk georgian	Lange Ganzzahl	20	
wk groove	Lange Ganzzahl	6	
wk hebrew	Lange Ganzzahl	21	
wk height	Zeichenkette	height	

Konstante	Typ	Wert	Kommentar
wk hidden	Lange Ganzzahl	5	
wk hiragana	Lange Ganzzahl	22	
wk hollow square	Lange Ganzzahl	25	
wk html debug	Lange Ganzzahl	1	Formatierter HTML Code ("pretty print"), leichter zu debuggen
wk image	Zeichenkette	image	
wk image alternative text	Zeichenkette	imageAltText	
wk inset	Lange Ganzzahl	8	
wk inside	Zeichenkette	Inside	
wk justify	Lange Ganzzahl	5	
wk katakana	Lange Ganzzahl	23	
wk layout unit	Zeichenkette	userUnit	
wk left	Lange Ganzzahl	0	
wk left to right	Lange Ganzzahl	0	
wk line height	Zeichenkette	lineHeight	
wk linethrough	Lange Ganzzahl	2	
wk list auto	Lange Ganzzahl	2147483647	
wk list font	Zeichenkette	listFont	
wk list font family	Zeichenkette	listFontFamily	
wk list start number	Zeichenkette	listStartNumber	
wk list string format LTR	Zeichenkette	listStringFormatLtr	
wk list string format RTL	Zeichenkette	listStringFormatRtl	
wk list style image	Zeichenkette	listStyleImage	
wk list style image height	Zeichenkette	listStyleImageHeight	
wk list style type	Zeichenkette	listStyleType	
wk lower greek	Lange Ganzzahl	18	
wk lower latin	Lange Ganzzahl	14	
wk lower roman	Lange Ganzzahl	15	
wk lowercase	Lange Ganzzahl	2	
wk margin	Zeichenkette	margin	
wk margin bottom	Zeichenkette	marginBottom	
wk margin left	Zeichenkette	marginLeft	
wk margin right	Zeichenkette	marginRight	
wk margin top	Zeichenkette	marginTop	
wk middle	Lange Ganzzahl	2	
wk mime html	Lange Ganzzahl	1	Das 4D Write Pro Dokument wird als standard MIME HTML mit html Dokumenten und Bildern, eingebunden als MIME Teile (codiert in base64), gesichert. Ausdrücke werden berechnet und 4D spezifische Tags werden entfernt. Dieses Format eignet sich besonders, um E-Mails in HTML mit dem Befehl **SMTP_QuickSend** zu senden.

Konstante	Typ	Wert	Kommentar
wk min height	Zeichenkette	minHeight	
wk min width	Zeichenkette	minWidth	
wk mixed	Lange Ganzzahl	-2147483648	
wk new line style sheet	Zeichenkette	newLineStyleSheet	
wk no repeat	Lange Ganzzahl	3	
wk none	Lange Ganzzahl	0	
wk normal	Lange Ganzzahl	0	Standard HTML Code
wk notes	Zeichenkette	notes	
wk outset	Lange Ganzzahl	9	
wk outside	Zeichenkette	Outside	
wk padding	Zeichenkette	padding	
wk padding bottom	Zeichenkette	paddingBottom	
wk padding box	Lange Ganzzahl	1	
wk padding left	Zeichenkette	paddingLeft	
wk padding right	Zeichenkette	paddingRight	
wk padding top	Zeichenkette	paddingTop	
wk range end	Zeichenkette	rangeEnd	
wk range owner	Zeichenkette	rangeOwner	
wk range start	Zeichenkette	rangeStart	
wk repeat	Lange Ganzzahl	0	
wk repeat x	Lange Ganzzahl	1	
wk repeat y	Lange Ganzzahl	2	
wk ridge	Lange Ganzzahl	7	
wk right	Lange Ganzzahl	1	
wk right to left	Lange Ganzzahl	1	
wk semi transparent	Lange Ganzzahl	5	
wk small uppercase	Lange Ganzzahl	4	
wk solid	Lange Ganzzahl	1	
wk square	Lange Ganzzahl	12	
wk start text	Lange Ganzzahl	1	
wk style sheet	Zeichenkette	styleSheet	
wk subject	Zeichenkette	subject	
wk subscript	Lange Ganzzahl	6	
wk superscript	Lange Ganzzahl	5	
wk tab stop offsets	Zeichenkette	tabStopOffsets	
wk tab stop types	Zeichenkette	tabStopTypes	
wk text align	Zeichenkette	textAlign	
wk text color	Zeichenkette	color	

Konstante	Typ	Wert	Kommentar
wk text indent	Zeichenkette	textIndent	
wk text linethrough color	Zeichenkette	textLinethroughColor	
wk text linethrough style	Zeichenkette	textLinethroughStyle	
wk text shadow color	Zeichenkette	textShadowColor	
wk text shadow offset	Zeichenkette	textShadowOffset	
wk text transform	Zeichenkette	textTransform	
wk text underline color	Zeichenkette	textUnderlineColor	
wk text underline style	Zeichenkette	textUnderlineStyle	
wk title	Zeichenkette	title	
wk top	Lange Ganzzahl	0	
wk transparent	Lange Ganzzahl	-1	
wk true	Lange Ganzzahl	1	
wk underline	Lange Ganzzahl	1	
wk unit cm	Zeichenkette	cm	
wk unit inch	Zeichenkette	in	
wk unit mm	Zeichenkette	mm	
wk unit percent	Zeichenkette	%	
wk unit pt	Zeichenkette	pt	
wk unit px	Zeichenkette	px	
wk upper latin	Lange Ganzzahl	16	
wk upper roman	Lange Ganzzahl	17	
wk uppercase	Lange Ganzzahl	3	
wk value unit not percentage	Lange Ganzzahl	-100000	
wk value unit percentage	Lange Ganzzahl	-100001	
wk version	Zeichenkette	version	
wk vertical align	Zeichenkette	verticalAlign	
wk web page complete	Lange Ganzzahl	2	Endung .htm oder .html. Das Dokument wird als Standard HTML gesichert, seine Resssourcen werden getrennt gesichert. 4D Tags werden entfernt und Ausdrücke berechnet. Dieses Format ist besonders geeignet, um ein 4D Write Pro Dokument in einem Web Browser anzuzeigen.
wk web page html 4D	Lange Ganzzahl	3	Das 4D Write Pro Dokument wird als HTML mit spezifischen 4D Tags gesichert; jeder Ausdruck wird als nicht-umgebrochener Bereich eingefügt. Da dieses Format verlustlos ist, eignet es sich zum Speichern in einem Textfeld.
wk width	Zeichenkette	width	
wk word	Lange Ganzzahl	6	

Konstante	Typ	Wert	Kommentar

ASCII Codes

Konstante	Typ	Wert	Kommentar
ACK ASCII code	Lange Ganzzahl	6	
At sign	Lange Ganzzahl	64	
Backspace	Lange Ganzzahl	8	
BEL ASCII code	Lange Ganzzahl	7	
BS ASCII code	Lange Ganzzahl	8	
CAN ASCII code	Lange Ganzzahl	24	
Carriage return	Lange Ganzzahl	13	
CR ASCII code	Lange Ganzzahl	13	
DC1 ASCII code	Lange Ganzzahl	17	
DC2 ASCII code	Lange Ganzzahl	18	
DC3 ASCII code	Lange Ganzzahl	19	
DC4 ASCII code	Lange Ganzzahl	20	
DEL ASCII code	Lange Ganzzahl	127	
DLE ASCII code	Lange Ganzzahl	16	
Double quote	Lange Ganzzahl	34	
EM ASCII code	Lange Ganzzahl	25	
ENQ ASCII code	Lange Ganzzahl	5	
Enter	Lange Ganzzahl	3	
EOT ASCII code	Lange Ganzzahl	4	
ESC ASCII code	Lange Ganzzahl	27	
Escape	Lange Ganzzahl	27	
ETB ASCII code	Lange Ganzzahl	23	
ETX ASCII code	Lange Ganzzahl	3	
FF ASCII code	Lange Ganzzahl	12	
FS ASCII code	Lange Ganzzahl	28	
GS ASCII code	Lange Ganzzahl	29	
HT ASCII code	Lange Ganzzahl	9	
LF ASCII code	Lange Ganzzahl	10	
Line feed	Lange Ganzzahl	10	
NAK ASCII code	Lange Ganzzahl	21	
NBSP	Lange Ganzzahl	202	
NUL ASCII code	Lange Ganzzahl	0	
Period	Lange Ganzzahl	46	
Quote	Lange Ganzzahl	39	
RS ASCII code	Lange Ganzzahl	30	
SI ASCII code	Lange Ganzzahl	15	
SO ASCII code	Lange Ganzzahl	14	
SOH ASCII code	Lange Ganzzahl	1	
SP ASCII code	Lange Ganzzahl	32	
Space	Lange Ganzzahl	32	
STX ASCII code	Lange Ganzzahl	2	
SUB ASCII code	Lange Ganzzahl	26	
SYN ASCII code	Lange Ganzzahl	22	
Tab	Lange Ganzzahl	9	
US ASCII code	Lange Ganzzahl	31	
VT ASCII code	Lange Ganzzahl	11	

Backup und Wiederherstellen

Konstante	Typ	Wert	Kommentar
Auto repair mode	Lange Ganzzahl	1	Verwendet flexiblen Modus mit auto-repair Aktionen und füllt den Parameter *errObjekt* (falls vorhanden).
Field attribute with name	Lange Ganzzahl	2	Felder werden über ihren Namen identifiziert. Beispiel: {"Nachname":"Jones"}
Field attribute with number	Lange Ganzzahl	1	Felder werden über ihre Nummer identifiziert (Standard, wenn weggelassen). Beispiel: {"5":"Jones"}.
Last backup date	Lange Ganzzahl	0	
Last backup status	Lange Ganzzahl	2	
Last restore date	Lange Ganzzahl	0	
Last restore status	Lange Ganzzahl	2	
Next backup date	Lange Ganzzahl	4	
Strict mode	Lange Ganzzahl	0	Verwendet strikten Integrationsmodus (Standard).

Bild Anzeigeformate

Konstante	Typ	Wert	Kommentar
On background	Lange Ganzzahl	3	
Replicated	Lange Ganzzahl	7	
Scaled to fit	Lange Ganzzahl	2	
Scaled to fit prop centered	Lange Ganzzahl	6	
Scaled to fit proportional	Lange Ganzzahl	5	
Truncated centered	Lange Ganzzahl	1	
Truncated non centered	Lange Ganzzahl	4	

Bild Metadaten Namen

Konstante	Typ	Wert	Kommentar
EXIF aperture value	Zeichenkette	EXIF/ApertureValue	Mögliche Werte: Zahl (APEX Wert)
EXIF brightness value	Zeichenkette	EXIF/BrightnessValue	Mögliche Werte: Zahl (APEX Wert)
EXIF color space	Zeichenkette	EXIF/ColorSpace	Mögliche Werte: EXIF Adobe RGB, EXIF s RGB, EXIF Uncalibrated. Nur-Lesen Modus
EXIF components configuration	Zeichenkette	EXIF/ComponentsConfiguration	Mögliche Werte: EXIF B, EXIF Cb, EXIF Cr, EXIF G, EXIF R, EXIF Unused, EXIF Y
EXIF compressed bits per pixel	Zeichenkette	EXIF/CompressedBitsPerPixel	Mögliche Werte: Zahl
EXIF contrast	Zeichenkette	EXIF/Contrast	Mögliche Werte: EXIF High, EXIF Low, EXIF Normal
EXIF custom rendered	Zeichenkette	EXIF/CustomRendered	Mögliche Werte: EXIF Normal, EXIF Custom
EXIF date time digitized	Zeichenkette	EXIF/DateTimeDigitized	Mögliche Werte: Datum oder Text (XML Datumzeit)
EXIF date time original	Zeichenkette	EXIF/DateTimeOriginal	Mögliche Werte: Datum oder Text (XML Datumzeit)
EXIF digital zoom ratio	Zeichenkette	EXIF/DigitalZoomRatio	Mögliche Werte: Zahl
EXIF EXIF version	Zeichenkette	EXIF/ExifVersion	Mögliche Werte: Ganzzahl Array (4 Werte)
EXIF exposure bias value	Zeichenkette	EXIF/ExposureBiasValue	Mögliche Werte: Zahl
EXIF exposure index	Zeichenkette	EXIF/ExposureIndex	Mögliche Werte: Zahl
EXIF exposure mode	Zeichenkette	EXIF/ExposureModus	Mögliche Werte: EXIF Auto, EXIF Auto Bracket, EXIF Manual
EXIF exposure program	Zeichenkette	EXIF/ExposureProgram	Mögliche Werte: 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	Zeichenkette	EXIF/ExposureTime	Mögliche Werte: Zahl
EXIF F number	Zeichenkette	EXIF/FNumber	Mögliche Werte: Zahl
EXIF file source	Zeichenkette	EXIF/FileSource	Mögliche Werte: EXIF Digital Camera, EXIF Film Scanner, EXIF Reflection Print Scanner
EXIF flash	Zeichenkette	EXIF/Flash	Mögliche Werte: 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	Zeichenkette	EXIF/FlashEnergy	Mögliche Werte: Zahl
EXIF flash fired	Zeichenkette	EXIF/Flash/Fired	Mögliche Werte: Boolean
EXIF flash function present	Zeichenkette	EXIF/Flash/FunctionPresent	Mögliche Werte: Boolean
EXIF flash mode	Zeichenkette	EXIF/Flash/Mode	Mögliche Werte: EXIF Auto Mode, EXIF Compulsory Flash Firing, EXIF Compulsory Flash Suppression, EXIF Unknown
EXIF flash pix version	Zeichenkette	EXIF/FlashPixVersion	Mögliche Werte: Ganzzahl Array (4 Werte)
EXIF flash red eye reduction	Zeichenkette	EXIF/Flash/RedEyeReduction	Mögliche Werte: Boolean
EXIF flash return light	Zeichenkette	EXIF/Flash/ReturnLight	Mögliche Werte: EXIF Detected, EXIF No Detection Function, EXIF Not Detected, EXIF Reserved
EXIF focal len in 35 mm film	Zeichenkette	EXIF/FocalLenIn35mmFilm	Mögliche Werte: Lange Ganzzahl
EXIF focal length	Zeichenkette	EXIF/FocalLength	Mögliche Werte: Zahl
EXIF focal plane resolution unit	Zeichenkette	EXIF/FocalPlaneResolutionUnit	Mögliche Werte: Lange Ganzzahl
EXIF focal plane X resolution	Zeichenkette	EXIF/FocalPlaneXResolution	Mögliche Werte: Zahl

Konstante	Typ	Wert	Kommentar
EXIF focal plane Y resolution	Zeichenkette	EXIF/FocalPlaneYResolution	Mögliche Werte: Zahl
EXIF gain control	Zeichenkette	EXIF/GainControl	Mögliche Werte: EXIF High Gain Down, EXIF High Gain Up, EXIF Low Gain Down, EXIF Low Gain Up, EXIF None
EXIF gamma	Zeichenkette	EXIF/Gamma	Mögliche Werte: Zahl
EXIF image unique ID	Zeichenkette	EXIF/ImageUniqueID	Mögliche Werte: Text
EXIF ISO speed ratings	Zeichenkette	EXIF/ISOSpeedRatings	Mögliche Werte: Lange Ganzzahl oder Lange Ganzzahl Array
EXIF light source	Zeichenkette	EXIF/LightSource	Mögliche Werte: 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	Zeichenkette	EXIF/MakerNote	Mögliche Werte: Text
EXIF max aperture value	Zeichenkette	EXIF/MaxApertureValue	Mögliche Werte: Zahl
EXIF metering mode	Zeichenkette	EXIF/MeteringMode	Mögliche Werte: EXIF Other, EXIF Average, EXIF Center Weighted Average, EXIF Multi Segment, EXIF Multi Spot, EXIF Partial, EXIF Spot
EXIF pixel X dimension	Zeichenkette	EXIF/PixelXDimension	Mögliche Werte: Lange Ganzzahl, Nur-Lesen Modus
EXIF pixel Y dimension	Zeichenkette	EXIF/PixelYDimension	Mögliche Werte: Lange Ganzzahl, Nur-Lesen Modus
EXIF related sound file	Zeichenkette	EXIF/RelatedSoundFile	Mögliche Werte: Text
EXIF saturation	Zeichenkette	EXIF/Saturation	Mögliche Werte: EXIF High, EXIF Low, EXIF Normal
EXIF scene capture type	Zeichenkette	EXIF/SceneCaptureType	Mögliche Werte: EXIF Scene Landscape, EXIF Night, EXIF Scene Portrait, EXIF Standard
EXIF scene type	Zeichenkette	EXIF/SceneType	Mögliche Werte: Lange Ganzzahl
EXIF sensing method	Zeichenkette	EXIF/SensingMethod	Mögliche Werte: 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	Zeichenkette	EXIF/Sharpness	Mögliche Werte: EXIF High, EXIF Low, EXIF Normal
EXIF shutter speed value	Zeichenkette	EXIF/ShutterSpeedValue	Mögliche Werte: Zahl
EXIF spectral sensitivity	Zeichenkette	EXIF/SpectralSensitivity	Mögliche Werte: Text
EXIF subject area	Zeichenkette	EXIF/SubjectArea	Mögliche Werte: Lange Ganzzahl Array (2, 3 oder 4 Werte)
EXIF subject dist range	Zeichenkette	EXIF/SubjectDistRange	Mögliche Werte: EXIF Unknown, EXIF Close, EXIF Distant, EXIF Macro
EXIF subject distance	Zeichenkette	EXIF/SubjectDistance	Mögliche Werte: Zahl
EXIF subject location	Zeichenkette	EXIF/SubjectLocation	Mögliche Werte: Lange Ganzzahl Array (2 Werte)
EXIF user comment	Zeichenkette	EXIF/UserComment	Mögliche Werte: Text
EXIF white balance	Zeichenkette	EXIF/WhiteBalance	Mögliche Werte: EXIF Auto, EXIF Manual
GPS altitude	Zeichenkette	GPS/Altitude	Mögliche Werte: GPS Above Sea Level, GPS Below Sea Level
GPS altitude ref	Zeichenkette	GPS/AltitudeRef	Mögliche Werte: GPS Above Sea Level, GPS Below Sea Level
GPS area information	Zeichenkette	GPS/AreaInformation	Mögliche Werte: Text
GPS date time	Zeichenkette	GPS/DateTime	Mögliche Werte: Datum oder Text (Datumzeit XML)
GPS dest bearing	Zeichenkette	GPS/DestBearing	Mögliche Werte: Text (1 Zeichen)
GPS dest bearing ref	Zeichenkette	GPS/DestBearingRef	Mögliche Werte: Text (1 Zeichen)
GPS dest distance	Zeichenkette	GPS/DestDistance	Mögliche Werte: Text (1 Zeichen)

Konstante	Typ	Wert	Kommentar
GPS dest distance ref	Zeichenkette	GPS/DestDistanceRef	Mögliche Werte: Text (1 Zeichen)
GPS dest latitude	Zeichenkette	GPS/DestLatitude	Mögliche Werte: Text
GPS dest latitude deg	Zeichenkette	GPS/DestLatitude/Deg	Mögliche Werte: Zahl
GPS dest latitude dir	Zeichenkette	GPS/DestLatitude/Dir	Mögliche Werte: Text (1 Zeichen)
GPS dest latitude min	Zeichenkette	GPS/DestLatitude/Min	Mögliche Werte: Zahl
GPS dest latitude sec	Zeichenkette	GPS/DestLatitude/Sec	Mögliche Werte: Zahl
GPS dest longitude	Zeichenkette	GPS/DestLongitude	Mögliche Werte: Text
GPS dest longitude deg	Zeichenkette	GPS/DestLongitude/Deg	Mögliche Werte: Zahl
GPS dest longitude dir	Zeichenkette	GPS/DestLongitude/Dir	Mögliche Werte: Text (1 Zeichen)
GPS dest longitude min	Zeichenkette	GPS/DestLongitude/Min	Mögliche Werte: Zahl
GPS dest longitude sec	Zeichenkette	GPS/DestLongitude/Sec	Mögliche Werte: Zahl
GPS differential	Zeichenkette	GPS/Differential	Mögliche Werte: GPS Correction Applied, GPS Correction Not Applied
GPS DOP	Zeichenkette	GPS/DOP	Mögliche Werte: Zahl
GPS img direction	Zeichenkette	GPS/ImgDirection	Mögliche Werte: GPS Magnetic north, GPS True north
GPS img direction ref	Zeichenkette	GPS/ImgDirectionRef	Mögliche Werte: GPS Magnetic north, GPS True north
GPS latitude	Zeichenkette	GPS/Latitude	Mögliche Werte: GPS North, GPS South
GPS latitude deg	Zeichenkette	GPS/Latitude/Deg	Mögliche Werte: Zahl
GPS latitude dir	Zeichenkette	GPS/Latitude/Dir	Mögliche Werte: GPS North, GPS South
GPS latitude min	Zeichenkette	GPS/Latitude/Min	Mögliche Werte: Zahl
GPS latitude sec	Zeichenkette	GPS/Latitude/Sec	Mögliche Werte: Zahl
GPS longitude	Zeichenkette	GPS/Longitude	Mögliche Werte: GPS West, GPS East
GPS longitude deg	Zeichenkette	GPS/Longitude/Deg	Mögliche Werte: Zahl
GPS longitude dir	Zeichenkette	GPS/Longitude/Dir	Mögliche Werte: GPS West, GPS East
GPS longitude min	Zeichenkette	GPS/Longitude/Min	Mögliche Werte: Zahl
GPS longitude sec	Zeichenkette	GPS/Longitude/Sec	Mögliche Werte: Zahl
GPS map date	Zeichenkette	GPS/MapDate	Mögliche Werte: Text
GPS measure mode	Zeichenkette	GPS/MeasureMode	Mögliche Werte: GPS 2D, GPS 3D
GPS processing method	Zeichenkette	GPS/ProcessingMethod	Mögliche Werte: Text
GPS satellites	Zeichenkette	GPS/Satellites	Mögliche Werte: Text
GPS speed	Zeichenkette	GPS/Speed	Mögliche Werte: GPS km h, GPS miles h, GPS knots h
GPS speed ref	Zeichenkette	GPS/SpeedRef	Mögliche Werte: GPS km h, GPS miles h, GPS knots h
GPS status	Zeichenkette	GPS/Status	Mögliche Werte: GPS Measurement in progress, GPS Measurement Interoperability
GPS track	Zeichenkette	GPS/Track	Mögliche Werte: Zahl (0.00..359.99)
GPS track ref	Zeichenkette	GPS/TrackRef	Mögliche Werte: Text (1 Zeichen)
GPS version ID	Zeichenkette	GPS/VersionID	Mögliche Werte: Lange Ganzzahl Array (4 Zeichen)
IPTC byline	Zeichenkette	IPTC/Byline	Mögliche Werte: Text oder Text Array
IPTC byline title	Zeichenkette	IPTC/BylineTitle	Mögliche Werte: Text oder Array
IPTC caption abstract	Zeichenkette	IPTC/CaptionAbstract	Mögliche Werte: Text
IPTC category	Zeichenkette	IPTC/Category	Mögliche Werte: Text
IPTC city	Zeichenkette	IPTC/City	Mögliche Werte: Text

Konstante	Typ	Wert	Kommentar
IPTC contact	Zeichenkette	IPTC/Contact	Mögliche Werte: Text oder Text Array
IPTC content location code	Zeichenkette	IPTC/ContentLocationCode	Mögliche Werte: Text oder Text Array
IPTC content location name	Zeichenkette	IPTC/ContentLocationName	Mögliche Werte: Text oder Text Array
IPTC copyright notice	Zeichenkette	IPTC/CopyrightNotice	Mögliche Werte: Text
IPTC country primary location code	Zeichenkette	IPTC/CountryPrimaryLocationCode	Mögliche Werte: Text
IPTC country primary location name	Zeichenkette	IPTC/CountryPrimaryLocationName	Mögliche Werte: Text
IPTC credit	Zeichenkette	IPTC/Credit	Mögliche Werte: Text
IPTC date time created	Zeichenkette	IPTC/DateTimeCreated	Mögliche Werte: Datum oder Text (XML Datumzeit)
IPTC digital creation date time	Zeichenkette	IPTC/DigitalCreationDateTime	Mögliche Werte: Datum oder Text (XML Datumzeit)
IPTC edit status	Zeichenkette	IPTC/EditStatus	Mögliche Werte: Text
IPTC expiration date time	Zeichenkette	IPTC/ExpirationDateTime	Mögliche Werte: Datum oder Text (XML Datumzeit)
IPTC fixture identifier	Zeichenkette	IPTC/FixtureIdentifier	Mögliche Werte: Text
IPTC headline	Zeichenkette	IPTC/Headline	Mögliche Werte: Text
IPTC image orientation	Zeichenkette	IPTC/ImageOrientation	Mögliche Werte: Text
IPTC image type	Zeichenkette	IPTC/ImageType	Mögliche Werte: Text
IPTC keywords	Zeichenkette	IPTC/Keywords	Mögliche Werte: Text oder Text Array
IPTC language identifier	Zeichenkette	IPTC/LanguageIdentifier	Mögliche Werte: Text
IPTC object attribute reference	Zeichenkette	IPTC/ObjectAttributeReference	Mögliche Werte: Text
IPTC object cycle	Zeichenkette	IPTC/ObjectCycle	Mögliche Werte: Text
IPTC object name	Zeichenkette	IPTC/ObjektName	Mögliche Werte: Text
IPTC original transmission reference	Zeichenkette	IPTC/OriginalTransmissionReference	Mögliche Werte: Text
IPTC originating program	Zeichenkette	IPTC/OriginatingProgram	Mögliche Werte: Text
IPTC program version	Zeichenkette	IPTC/ProgramVersion	Mögliche Werte: Text
IPTC province state	Zeichenkette	IPTC/ProvinceState	Mögliche Werte: Text
IPTC release date time	Zeichenkette	IPTC/ReleaseDateTime	Mögliche Werte: Datum oder Text (XML Datumzeit)
IPTC scene	Zeichenkette	IPTC/Scene	Mögliche Werte: 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	Zeichenkette	IPTC/Source	Mögliche Werte: Text
IPTC special instructions	Zeichenkette	IPTC/SpecialInstructions	Mögliche Werte: Text
IPTC star rating	Zeichenkette	IPTC/StarRating	Mögliche Werte: Lange Ganzzahl
IPTC sub location	Zeichenkette	IPTC/SubLocation	Mögliche Werte: Text

Konstante	Typ	Wert	Kommentar
IPTC subject reference	Zeichenkette	IPTC/SubjectReference	Mögliche Werte: Lange Ganzzahl oder Lange Ganzzahl Array
IPTC supplemental category	Zeichenkette	IPTC/SupplementalCategory	Mögliche Werte: Text oder Array
IPTC urgency	Zeichenkette	IPTC/Urgency	Mögliche Werte: Ganzzahl
IPTC writer editor	Zeichenkette	IPTC/WriterEditor	Mögliche Werte: Text oder Text Array
TIFF artist	Zeichenkette	TIFF/Artist	Mögliche Werte: Text
TIFF compression	Zeichenkette	TIFF/Compression	Mögliche Werte: 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	Zeichenkette	TIFF/Copyright	Mögliche Werte: Text
TIFF date time	Zeichenkette	TIFF/DateTime	Mögliche Werte: Datum oder Text (XML Datumzeit)
TIFF document name	Zeichenkette	TIFF/DocumentName	Mögliche Werte: Text
TIFF host computer	Zeichenkette	TIFF/HostComputer	Mögliche Werte: Text
TIFF image description	Zeichenkette	TIFF/ImageDescription	Mögliche Werte: Text
TIFF make	Zeichenkette	TIFF/Make	Mögliche Werte: Text
TIFF model	Zeichenkette	TIFF/Model	Mögliche Werte: Text
TIFF orientation	Zeichenkette	TIFF/Orientation	Mögliche Werte: 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	Zeichenkette	TIFF/PhotometricInterpretation	Mögliche Werte: 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	Zeichenkette	TIFF/ResolutionUnit	Mögliche Werte: TIFF CM, TIFF Inches, TIFF MM, TIFF None, TIFF UM
TIFF software	Zeichenkette	TIFF/Software	Mögliche Werte: Text
TIFF xResolution	Zeichenkette	TIFF/XResolution	Mögliche Werte: Zahl, Nur-Lesen Modus
TIFF yResolution	Zeichenkette	TIFF/YResolution	Mögliche Werte: Zahl, Nur-Lesen Modus

Bild Metadaten Werte

Konstante	Typ	Wert	Kommentar
EXIF Action	Lange Ganzzahl	6	
EXIF Adobe RGB	Lange Ganzzahl	2	
EXIF Aperture Priority AE	Lange Ganzzahl	3	
EXIF Auto	Lange Ganzzahl	0	
EXIF Auto Bracket	Lange Ganzzahl	2	
EXIF Auto Mode	Lange Ganzzahl	3	
EXIF Average	Lange Ganzzahl	1	
EXIF B	Lange Ganzzahl	6	
EXIF Cb	Lange Ganzzahl	2	
EXIF Center Weighted Average	Lange Ganzzahl	2	
EXIF Close	Lange Ganzzahl	2	
EXIF Cloudy	Lange Ganzzahl	10	
EXIF Color Sequential Area	Lange Ganzzahl	5	
EXIF Color Sequential Linear	Lange Ganzzahl	8	
EXIF Compulsory Flash Firing	Lange Ganzzahl	1	
EXIF Compulsory Flash Suppression	Lange Ganzzahl	2	
EXIF Cool White Fluorescent	Lange Ganzzahl	14	
EXIF Cr	Lange Ganzzahl	3	
EXIF Creative	Lange Ganzzahl	5	
EXIF Custom	Lange Ganzzahl	1	
EXIF D50	Lange Ganzzahl	23	
EXIF D55	Lange Ganzzahl	20	
EXIF D65	Lange Ganzzahl	21	
EXIF D75	Lange Ganzzahl	22	
EXIF Day White Fluorescent	Lange Ganzzahl	13	
EXIF Daylight	Lange Ganzzahl	1	
EXIF Daylight Fluorescent	Lange Ganzzahl	12	
EXIF Detected	Lange Ganzzahl	3	
EXIF Digital Camera	Lange Ganzzahl	3	
EXIF Distant	Lange Ganzzahl	3	
EXIF Exposure Portrait	Lange Ganzzahl	7	
EXIF Film Scanner	Lange Ganzzahl	1	
EXIF Fine Weather	Lange Ganzzahl	9	
EXIF Flashlight	Lange Ganzzahl	4	
EXIF G	Lange Ganzzahl	5	
EXIF High	Lange Ganzzahl	2	
EXIF High Gain Down	Lange Ganzzahl	4	
EXIF High Gain Up	Lange Ganzzahl	2	
EXIF ISOStudio Tungsten	Lange Ganzzahl	24	
EXIF Landscape	Lange Ganzzahl	8	
EXIF Light Fluorescent	Lange Ganzzahl	2	
EXIF Low	Lange Ganzzahl	1	
EXIF Low Gain Down	Lange Ganzzahl	3	
EXIF Low Gain Up	Lange Ganzzahl	1	
EXIF Macro	Lange Ganzzahl	1	
EXIF Manual	Lange Ganzzahl	1	
EXIF Multi Segment	Lange Ganzzahl	5	
EXIF Multi Spot	Lange Ganzzahl	4	
EXIF Night	Lange Ganzzahl	3	
EXIF No Detection Function	Lange Ganzzahl	0	
EXIF None	Lange Ganzzahl	0	
EXIF Normal	Lange Ganzzahl	0	
EXIF Not Defined	Lange Ganzzahl	1	
EXIF Not Detected	Lange Ganzzahl	2	
EXIF One Chip Color Area	Lange Ganzzahl	2	
EXIF Other	Lange Ganzzahl	255	
EXIF Partial	Lange Ganzzahl	6	
EXIF Program AE	Lange Ganzzahl	2	
EXIF R	Lange Ganzzahl	4	
EXIF Reflection Print Scanner	Lange Ganzzahl	2	
EXIF Reserved	Lange Ganzzahl	1	
EXIF s RGB	Lange Ganzzahl	1	
EXIF Scene Landscape	Lange Ganzzahl	1	
EXIF Scene Portrait	Lange Ganzzahl	2	
EXIF Shade	Lange Ganzzahl	11	
EXIF Shutter Speed Priority AE	Lange Ganzzahl	4	
EXIF Spot	Lange Ganzzahl	3	

Konstante	Typ	Wert	Kommentar
EXIF Standard	Lange Ganzzahl	0	
EXIF Standard Light A	Lange Ganzzahl	17	
EXIF Standard Light B	Lange Ganzzahl	18	
EXIF Standard Light C	Lange Ganzzahl	19	
EXIF Three Chip Color Area	Lange Ganzzahl	4	
EXIF Trilinear	Lange Ganzzahl	7	
EXIF Tungsten	Lange Ganzzahl	3	
EXIF Two Chip Color Area	Lange Ganzzahl	3	
EXIF Uncalibrated	Lange Ganzzahl	-1	
EXIF Unknown	Lange Ganzzahl	0	
EXIF Unused	Lange Ganzzahl	0	
EXIF White Fluorescent	Lange Ganzzahl	15	
EXIF Y	Lange Ganzzahl	1	
GPS 2D	Lange Ganzzahl	2	
GPS 3D	Lange Ganzzahl	3	
GPS Above Sea Level	Lange Ganzzahl	0	
GPS Below Sea Level	Lange Ganzzahl	1	
GPS Correction Applied	Lange Ganzzahl	1	
GPS Correction Not Applied	Lange Ganzzahl	0	
GPS East	Zeichenkette	E	
GPS km h	Zeichenkette	K	
GPS knots h	Zeichenkette	K	
GPS Magnetic north	Zeichenkette	M	
GPS Measurement in progress	Zeichenkette	A	
GPS Measurement Interoperability	Zeichenkette	V	
GPS miles h	Zeichenkette	M	
GPS North	Zeichenkette	N	
GPS South	Zeichenkette	S	
GPS True north	Zeichenkette	T	
GPS West	Zeichenkette	W	
IPTC Action	Lange Ganzzahl	11900	
IPTC Aerial View	Lange Ganzzahl	11200	
IPTC Close Up	Lange Ganzzahl	11800	
IPTC Couple	Lange Ganzzahl	10700	
IPTC Exterior View	Lange Ganzzahl	11600	
IPTC Full Length	Lange Ganzzahl	10300	
IPTC General View	Lange Ganzzahl	11000	
IPTC Group	Lange Ganzzahl	10900	
IPTC Half Length	Lange Ganzzahl	10200	
IPTC Headshot	Lange Ganzzahl	10100	
IPTC Interior View	Lange Ganzzahl	11700	
IPTC Movie Scene	Lange Ganzzahl	12400	
IPTC Night Scene	Lange Ganzzahl	11400	
IPTC Off Beat	Lange Ganzzahl	12300	
IPTC Panoramic View	Lange Ganzzahl	11100	
IPTC Performing	Lange Ganzzahl	12000	
IPTC Posing	Lange Ganzzahl	12100	
IPTC Profile	Lange Ganzzahl	10400	
IPTC Rear View	Lange Ganzzahl	10500	
IPTC Satellite	Lange Ganzzahl	11500	
IPTC Single	Lange Ganzzahl	10600	
IPTC Symbolic	Lange Ganzzahl	12200	
IPTC Two	Lange Ganzzahl	10800	
IPTC Under Water	Lange Ganzzahl	11300	
TIFF Adobe Deflate	Lange Ganzzahl	8	
TIFF Black Is Zero	Lange Ganzzahl	1	
TIFF CCIRLEW	Lange Ganzzahl	32771	
TIFF CCITT1D	Lange Ganzzahl	2	
TIFF CIELab	Lange Ganzzahl	8	
TIFF CM	Lange Ganzzahl	3	
TIFF CMYK	Lange Ganzzahl	5	
TIFF Color Filter Array	Lange Ganzzahl	32803	
TIFF DCS	Lange Ganzzahl	32947	
TIFF Deflate	Lange Ganzzahl	32946	
TIFF Epson ERF	Lange Ganzzahl	32769	
TIFF Horizontal	Lange Ganzzahl	1	
TIFF ICCLab	Lange Ganzzahl	9	

Konstante	Typ	Wert	Kommentar
TIFF Inches	Lange Ganzzahl	2	
TIFF IT8BL	Lange Ganzzahl	32898	
TIFF IT8CTPAD	Lange Ganzzahl	32895	
TIFF IT8LW	Lange Ganzzahl	32896	
TIFF IT8MP	Lange Ganzzahl	32897	
TIFF ITULab	Lange Ganzzahl	10	
TIFF JBIG	Lange Ganzzahl	34661	
TIFF JBIGB&W	Lange Ganzzahl	9	
TIFF JBIGColor	Lange Ganzzahl	10	
TIFF JPEG	Lange Ganzzahl	7	
TIFF JPEG2000	Lange Ganzzahl	34712	
TIFF JPEGThumbs Only	Lange Ganzzahl	6	
TIFF Kodak DCR	Lange Ganzzahl	65000	
TIFF Kodak KDC	Lange Ganzzahl	32867	
TIFF Kodak262	Lange Ganzzahl	262	
TIFF Linear Raw	Lange Ganzzahl	34892	
TIFF LZW	Lange Ganzzahl	5	
TIFF MDIBinary Level Codec	Lange Ganzzahl	34718	
TIFF MDIProgressive Transform Codec	Lange Ganzzahl	34719	
TIFF MDIVector	Lange Ganzzahl	34720	
TIFF Mirror Horizontal	Lange Ganzzahl	2	
TIFF Mirror Horizontal And Rotate270CW	Lange Ganzzahl	5	
TIFF Mirror Horizontal And Rotate90CW	Lange Ganzzahl	7	
TIFF Mirror Vertical	Lange Ganzzahl	4	
TIFF MM	Lange Ganzzahl	4	
TIFF Next	Lange Ganzzahl	32766	
TIFF Nikon NEF	Lange Ganzzahl	34713	
TIFF None	Lange Ganzzahl	1	
TIFF Pack Bits	Lange Ganzzahl	32773	
TIFF Pentax PEF	Lange Ganzzahl	65535	
TIFF Pixar Film	Lange Ganzzahl	32908	
TIFF Pixar Log	Lange Ganzzahl	32909	
TIFF Pixar Log L	Lange Ganzzahl	32844	
TIFF Pixar Log Luv	Lange Ganzzahl	32845	
TIFF RGB	Lange Ganzzahl	2	
TIFF RGBPalette	Lange Ganzzahl	3	
TIFF Rotate180	Lange Ganzzahl	3	
TIFF Rotate270CW	Lange Ganzzahl	8	
TIFF Rotate90CW	Lange Ganzzahl	6	
TIFF SGILog	Lange Ganzzahl	34676	
TIFF SGILog24	Lange Ganzzahl	34677	
TIFF Sony ARW	Lange Ganzzahl	32767	
TIFF T4Group3Fax	Lange Ganzzahl	3	
TIFF T6Group4Fax	Lange Ganzzahl	4	
TIFF Thunderscan	Lange Ganzzahl	32809	
TIFF Transparency Mask	Lange Ganzzahl	4	
TIFF UM	Lange Ganzzahl	5	
TIFF Uncompressed	Lange Ganzzahl	1	
TIFF White Is Zero	Lange Ganzzahl	0	
TIFF YCb Cr	Lange Ganzzahl	6	

Bildkomprimierung

Konstante	Typ	Wert	Kommentar

Bildtransformation

Konstante	Typ	Wert	Kommentar
Crop	Lange Ganzzahl	100	
Fade to grey scale	Lange Ganzzahl	101	
Flip horizontally	Lange Ganzzahl	3	
Flip vertically	Lange Ganzzahl	4	
Horizontal concatenation	Lange Ganzzahl	1	
Reset	Lange Ganzzahl	0	
Scale	Lange Ganzzahl	1	
Superimposition	Lange Ganzzahl	3	
Translate	Lange Ganzzahl	2	
Transparency	Lange Ganzzahl	102	
Vertical concatenation	Lange Ganzzahl	2	

BLOB

Konstante	Typ	Wert	Kommentar
Compact compression mode	Lange Ganzzahl	1	So kompakt wie möglich komprimieren (Standardmodus). Verlangsamt die Komprimierungsgeschwindigkeit und die Operationen zur Entkomprimierung.
Extended real format	Lange Ganzzahl	1	
Fast compression mode	Lange Ganzzahl	2	So schnell wie möglich komprimieren (wird auch schnellstmöglich entkomprimiert), verringert die Komprimierungsrate, d.h. das komprimierte BLOB wird größer.
GZIP best compression mode	Lange Ganzzahl	-1	Kompakteste GZIP Komprimierung
GZIP fast compression mode	Lange Ganzzahl	-2	Schnellste GZIP Komprimierung
Is not compressed	Lange Ganzzahl	0	No compression
Mac C string	Lange Ganzzahl	0	
Mac Pascal string	Lange Ganzzahl	1	
Mac text with length	Lange Ganzzahl	2	
Mac text without length	Lange Ganzzahl	3	
Macintosh byte ordering	Lange Ganzzahl	1	
Macintosh double real format	Lange Ganzzahl	2	
Native byte ordering	Lange Ganzzahl	0	
Native real format	Lange Ganzzahl	0	
PC byte ordering	Lange Ganzzahl	2	
PC double real format	Lange Ganzzahl	3	
UTF8 C string	Lange Ganzzahl	4	
UTF8 text with length	Lange Ganzzahl	5	
UTF8 text without length	Lange Ganzzahl	6	

Cache Verwaltung

Konstante	Typ	Wert	Kommentar
Cache priority high	Lange Ganzzahl	1000	
Cache priority low	Lange Ganzzahl	-900	
Cache priority normal	Lange Ganzzahl	0	Setzt den Wert für Priorität im Cache auf den Standardwert
Cache priority very high	Lange Ganzzahl	30000	
Cache priority very low	Lange Ganzzahl	-1	

Datenbank Engine

Konstante	Typ	Wert	Kommentar
New record	Lange Ganzzahl	-3	
No current record	Lange Ganzzahl	-1	

Datenbankereignisse

Konstante	Typ	Wert	Kommentar
On after host database exit	Lange Ganzzahl	4	Die **Semaphore** der Host Datenbank wurde gerade beendet
On after host database startup	Lange Ganzzahl	2	Die **Datenbankmethode On Startup** der Host Datenbank wurde gerade beendet
On application background move	Lange Ganzzahl	1	Die 4D Anwendung geht in den Hintergrund
On application foreground move	Lange Ganzzahl	2	Die 4D Anwendung kommt in den Vordergrund
On before host database exit	Lange Ganzzahl	3	Die Host Datenbank schließt. Die **Semaphore** der Host Datenbank wurde noch nicht aufgerufen. Die **Semaphore** der Host Datenbank wird nicht aufgerufen, während die **Datenbankmethode On Host Database Event** der Komponente läuft
On before host database startup	Lange Ganzzahl	1	Die Host Datenbank wurde gerade gestartet. Die **Datenbankmethode On Startup** der Host Datenbank wurde noch nicht aufgerufen. Die Datenbankmethode On Startup wird nicht aufgerufen, solange die **Datenbankmethode On Host Database Event** der Komponente läuft.

Datenbankparameter

Konstante	Typ	Wert	Kommentar
_o_4D Local mode scheduler	Lange Ganzzahl	10	**** *Dieser Selector ist überholt und darf nicht mehr verwendet werden.*
_o_4D Remote mode scheduler	Lange Ganzzahl	12	**** *Dieser Selector ist überholt und darf nicht mehr verwendet werden.*
_o_4D Server scheduler	Lange Ganzzahl	11	**** *Dieser Selector ist überholt und darf nicht mehr verwendet werden.*
_o_Client IP address to listen	Lange Ganzzahl	23	**** *Selector ist deaktiviert, die Befehle **WEB SET OPTION** und **WEB GET OPTION** verwenden* ****
_o_Database cache size	Lange Ganzzahl	9	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** - **Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen, stattdessen die Funktion **Get cache size** zu verwenden.
_o_IP Address to listen	Lange Ganzzahl	16	**** *Selector ist deaktiviert, die Befehle **WEB SET OPTION** und **WEB GET OPTION** verwenden* ****
_o_Real display precision	Lange Ganzzahl	32	**** Selector deaktiviert ****
_o_Web conversion mode	Lange Ganzzahl	8	**** *Selector deaktiviert* ****
_o_Web Log recording	Lange Ganzzahl	29	**Reichweite:** 4D lokal, 4D Server **Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
4D Remote mode timeout	Lange Ganzzahl	14	**Reichweite:** 4D Anwendung bei positivem Wert **Wird zwischen 2 Sitzungen beibehalten:** Ja bei positivem Wert **Beschreibung:** Nur für ganz spezifische Fälle verwenden. Damit ändern Sie die Zeitspanne zum Abschalten des Rechners mit remote 4D vom 4D Server Rechner. Der Standardwert für das Timeout wird auf dem remote Rechner in den Einstellungen der Datenbank auf der Seite "Client/Server>Konfiguration" festgelegt. Der Selector 4D Remote Mode Timeout wird nur bei Verwenden der legacy Netzwerkschicht berücksichtigt. Er wird ignoriert, wenn *ServerNet* aktiviert ist: Diese Einstellung wird vom Selector 4D Server Timeout (13) verwaltet.
4D Server log recording	Lange Ganzzahl	28	**Reichweite:** 4D Server, remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 oder von 1 bis X (0 = nicht speichern, 1 bis X = sequentielle Nummer, an den Dateinamen angefügt. **Beschreibung:** Startet oder stoppt das Speichern von Standardanfragen, die von 4D Server empfangen werden (mit Ausnahme von Web Anfragen). Der Standardwert ist 0, d.h. Anfragen werden nicht gespeichert. 4D Server ermöglicht, jede Anfrage, die vom Server-Rechner empfangen wird, in einem Logbuch zu speichern. Ist das Speichern aktiviert, werden im Logbuch neben der Strukturdatei der Datenbank zwei Dateien mit den Namen *4DRequestsLog_X.txt* und *4DRequestsLog_ProcessInfo_X.txt* angelegt, wobei X die Sequenznummer des Logbuchs ist. Hat *4DRequestsLog_X.txt* die Größe von 10 MB, wird sie geschlossen und es wird ein neues Logbuch mit der nächsthöheren Sequenznummer angelegt. Ist bereits eine Datei mit diesem Namen vorhanden, wird sie direkt ersetzt. Mit dem Parameter *Wert* können Sie die Startnummer für die Sequenzen setzen. Diese Textdateien speichern verschiedene Informationen zu jeder Anfrage in einer einfachen Liste: Zeit, Prozessnummer und -dauer, Größe der Anfrage, etc. Weitere Informationen dazu finden Sie im **Anhang E: Beschreibung der Protokolldateien**.

Konstante	Typ	Wert	Kommentar
4D Server timeout	Lange Ganzzahl	13	**Reichweite:** 4D Anwendung bei positivem *Wert* **Wird zwischen 2 Sitzungen beibehalten:** Ja bei positivem *Wert* **Mögliche Werte:** 0 -> 32 767 **Beschreibung:** Damit ändern Sie den Timeout-Wert des 4D Server. Der Standardwert für das Timeout des 4D Server wird auf dem Server Rechner in den Datenbank-Eigenschaften auf der Seite **Client-Server>IP Konfiguration** festgelegt. Mit dem Selector 4D Server Timeout legen Sie im dazugehörigen Parameter *Wert* ein neues Timeout in Minuten fest. Das ist besonders hilfreich bei Operationen auf dem Client, die den Rechner blockieren oder viel Zeit beanspruchen, z.B. wenn eine große Anzahl an Seiten gedruckt wird. Das kann bei einem kleinen Standardwert zu einem unerwartetem Timeout führen. Es gibt zwei Optionen: • Ein positiver Wert im Parameter *Wert* setzt ein globales und permanentes Timeout: Der neue Wert gilt für alle Prozesse und wird in den Voreinstellungen der 4D Anwendung gespeichert. Das entspricht einer Änderung in den Einstellungen der Datenbank. • Ein negativer Wert im Parameter *Wert* setzt ein lokales und temporäres Timeout: Der neue Wert gilt nur für den aufgerufenen Prozess. Die anderen Prozesse behalten den Standardwert bei. Der temporäre Wert wird zurückgesetzt, sobald der Server vom Client ein Signal für Aktivität erhält – zum Beispiel, wenn die Operation beendet ist. Diese Option eignet sich für lange Operationen, die über 4D Plug-Ins gestartet werden. Um die Option "Kein Timeout" zu setzen, übergeben Sie 0 (Null) in Wert (siehe 1. Beispiel).
Auto synchro resources folder	Lange Ganzzahl	48	**Reichweite:** Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 (keine Synchronisation), 1 (auto Synchronisation) oder 2 (Fragen). **Beschreibung:** Dynamische Synchronisation des lokalen Ordners **Resources** auf dem Client-Rechner, der den Befehl mit dem Ordner **Resources** auf dem Server ausführt. Wurde der Inhalt des Ordners **Resources** auf dem Server geändert oder hat ein Benutzer die Synchronisation angefordert, z.B. über den Resourcen Explorer oder nach der Ausführung des Befehls **SET DATABASE LOCALIZATION**, benachrichtigt der Server die angemeldeten Client-Rechner. Auf der Client-Seite gibt es dann drei Möglichkeiten zur Synchronisation. Der Selektor Auto Synchro Resources Folder ermöglicht die Einstellung auf dem Client für die aktuelle Arbeitssitzung: • 0 (Standardwert) = keine dynamische Synchronisation (die Anfrage zur Synchronisation wird ignoriert) • 1 = automatische dynamische Synchronisation • 2 = Anzeige des Dialogfensters auf den Client-Rechnern, um die Synchronisation zu erlauben oder zu verweigern. Der Synchronisationsmodus lässt sich auch global in den Einstellungen der Anwendung definieren.
Cache flush periodicity	Lange Ganzzahl	95	**Reichweite**: 4D local, 4D Server **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: Lange Ganzzahl > 1 (Sekunden) **Beschreibung**: Erhält oder setzt das Zeitintervall zum Sichern des aktuellen Cache in Sekunden. Eine Änderung dieses Werts überschreibt die Option **Daten-Cache sichern alle X Sekunden** auf der Seite **Seite Datenbank/ Speicher** der Datenbank Eigenschaften für die Sitzung (wird nicht in den Datenbank Eigenschaften gespeichert).
Cache unload minimum size	Lange Ganzzahl	66	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Mögliche Werte**: Positive Lange Ganzzahl > 1. **Beschreibung**: Mindestgröße des Speichers, der vom Datenbank Cache freigegeben werden soll, wenn die Engine Platz schaffen muss, um dem Cache ein Objekt zuzuweisen (Wert in Bytes). Dieser Selektors dient dazu, für eine bessere Performance die Momente zu reduzieren, zu denen Daten aus dem Cache freigegeben werden. Sie können diese Einstellung je nach Größe des Cache bzw. nach Datenblock, der in Ihrer Datenbank bearbeitet wird, variieren. Ohne diesen Selektor entlädt 4D standardmäßig mindestens 10% des Cache, wenn Platz gebraucht wird.
Character set	Lange Ganzzahl	17	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
Circular log limitation	Lange Ganzzahl	90	**Reichweite**: 4D lokal, 4D Server. **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: Jeder Wert vom Typ Zahl, 0 = alle Journale behalten **Beschreibung**: Maximale Anzahl der beizubehaltenden Journale pro Typ. Standardmäßig werden alle Dateien beibehalten. Übergeben Sie einen Wert *X*, werden nur die *letzten X* Dateien behalten, wobei bei Erstellen eines neuen Journals das älteste automatisch entfernt wird. Diese Einstellung gilt jeweils pro Journal für folgende Typen: request logs (Selector 28 und 45), debug log (Selector 34), events log (Selector 79), sowie Web request logs und Web debug logs (Selector 29 und 84 des Befehls **WEB SET OPTION**).

Konstante	Typ	Wert	Kommentar
Client character set	Lange Ganzzahl	24	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 17 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client HTTPS port ID	Lange Ganzzahl	40	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 bis 65535 **Beschreibung:** TCP Port Nummer, welche die Web Server der Client-Rechner für sichere Verbindungen via SSL (HTTPS Protokoll) verwenden. Der Wert ist standardmäßig 443. Mit diesem Selector können Sie per Programmierung die TCP Portnummer ändern, die Web Server von Client-Rechnern für sichere Verbindungen via SSL (HTTPS protocol) verwenden. Dieser Selector funktioniert auf dieselbe Weise wie der Selector 39; er wird jedoch auf alle Client-Rechner angewandt, die als Web Server dienen. Wollen Sie nur den Wert eines bestimmten Client-Rechners verändern, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client log recording	Lange Ganzzahl	45	**Reichweite:** Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 oder von 1 bis X (0 = nicht speichern, 1 bis X= sequentielle Nummer, an den Dateinamen angehängt). **Beschreibung:** Startet oder stoppt das Speichern der Standardanfragen, die vom 4D Client Rechner ausgehen, der den Befehl ausführt (außer Web Anfragen). Der Wert ist standardmäßig 0 (Null), d.h. die Anfragen werden nicht gespeichert. Sie können in 4D alle vom Client Rechner ausgeführten Anfragen in einem Logbuch speichern. Ist das Speichern aktiviert, werden auf dem Client Rechner im lokalen Ordner der Datenbank im Unterordner *Logs* zwei Dateien angelegt: *4DRequestsLogX.txt* und *4DRequestsLog_ProcessInfo_X.txt*, wobei X die fortlaufende Nummer des Logbuchs ist. Hat *4DRequestsLogX.txt* die Größe von 10 MB erreicht, wird es geschlossen und ein neues Logbuch mit der nächsthöheren Nummer erstellt. Gibt es bereits eine Datei mit demselben Namen, wird sie ersetzt. Über den Parameter *Wert* können Sie die Anfangsnummer setzen. Diese Textdateien speichern die Informationen zu jeder Anfrage in einem einfachen Format mit Tabs: Zeit, Prozessnummer, Größe der Anfrage, Bearbeitungsdauer, etc. Weitere Informationen dazu finden Sie im Abschnitt **Anhang E: Beschreibung der Protokolldateien**.
Client max concurrent Web proc	Lange Ganzzahl	25	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 18 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client Max Web requests size	Lange Ganzzahl	21	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 27 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client maximum Web process	Lange Ganzzahl	20	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 17 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client minimum Web process	Lange Ganzzahl	19	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 6 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.
Client port ID	Lange Ganzzahl	22	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Siehe Selector 15 **Beschreibung:** Gibt die Parameter für alle Rechner mit remote 4D an, die als Web Server verwendet werden. Die über diesen Selector definierten Werte gelten für alle als Web Server verwendete Client-Rechner. Um Werte nur für bestimmte Client-Rechner festzulegen, wählen Sie das Dialogfenster Einstellungen von remote 4D.

Konstante	Typ	Wert	Kommentar
Client Server port ID	Lange Ganzzahl	35	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 bis 65535 **Beschreibung:** Mit diesem Parameter lässt sich per Programmierung die Nummer des TCP-Port bestimmen, in dem 4D Server die Datenbank veröffentlicht (Zugriff für 4D im lokalen Modus). Der Standardwert ist 19813. Durch individuelles Anpassen dieses Wertes können auf einem Rechner mit dem TCP Protokoll mehrere 4D Client/Server Anwendungen laufen. Dazu müssen Sie für jede Anwendung eine andere Port-Nummer angeben. Der Wert wird in der Strukturdatei der Datenbank gespeichert. Er lässt sich mit 4D im lokalen Modus setzen, wirkt sich jedoch nur im Client/Server-Betrieb aus. Verändern Sie diesen Wert, müssen Sie den Server-Rechner neu starten, damit er berücksichtigt wird.
Client Web log recording	Lange Ganzzahl	30	**Reichweite:** Alle Rechner mit remote 4D **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 = Nicht speichern (Standard), 1 = In CLF Format speichern, 2 = In DLF Format speichern, 3 = In ELF Format speichern, 4 = In WLF Format speichern. **Beschreibung:** Startet oder stoppt das Speichern von Web Anfragen, die vom Web Server aller Client-Rechner empfangen werden. Der Standardwert ist 0, d.h. Anfragen werden nicht gespeichert. Dieser Selector arbeitet genauso wie Selector 29; mit dem Unterschied, dass er für alle Client Rechner gilt, die als Web Server verwendet werden. Die Datei "logweb.txt" wird in diesem Fall automatisch in den Unterordner Logs des 4D Client Anwendungsordners (Cache Ordner) gelegt. Wollen Sie nur Werte für bestimmte Client-Rechner setzen, verwenden Sie die Voreinstellungen von remote 4D.
Date type	Lange Ganzzahl	2	Datumstyp-Selector für Dates inside objects
Dates inside objects	Lange Ganzzahl	85	**Reichweite**: Aktueller Prozess **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: String type without time zone (0), String type with time zone (1), Date type (2) (Standard) **Beschreibung**: Definiert wie Datumsangaben innerhalb von Objekten gespeichert werden und sie in JSON importiert/exportiert werden. Mit Date type (Standardwert für Anwendungen, die mit 4Dv17 oder höher erstellt wurden) werden 4D Datumsangaben mit dem Datumstyp im Objekt gespeichert unter Berücksichtigung der lokalen Datumseinstellung. Beim Konvertieren in das JSON Format werden Datumsattribute in Strings ohne Zeitangabe konvertiert. (**Hinweis:** Dies kann über die Option "Verwende Datumstyp statt ISO Datumsformat in Objekten" in den Datenbank-Einstellungen auf der Seite **Seite Kompatibilität** gesetzt werden) Mit String type with time zone werden 4D Datumsangaben in ISO Strings konvertiert und berücksichtigen die lokale Zeitzone. Zum Beispiel erhalten Sie beim Konvertieren des Datums !23.08.2013! in JSON Format 2013-08-22T22:00:00Z, wenn die Operation in Deutschland zu einer Sicherungszeit bei Tageslicht (GMT+2) ausgeführt wird. Dieses Prinzip entspricht der Standardoperation von JavaScript. Das kann zu Fehlern führen, wenn Sie Datumswerte in JSON an jemanden in einer anderen Zeitzone senden wollen. Das ist z.B. der Fall, wenn Sie eine Tabelle über **Selection to JSON** in Deutschland exportieren, die dann in USA über **JSON TO SELECTION** importiert wird. Da Datumsangaben in jeder Zeitzone erneut interpretiert werden, sind die in der Anwendung gespeicherten Werte unterschiedlich. In diesem Fall können Sie den Konvertierungsmodus für Datum ändern. Damit die Zeitzone nicht berücksichtigt wird, übergeben Sie in diesem Selektor String type without time zone. Dann ergibt die Konvertierung des Datums !23.08.2013! in allen Fällen 2013-08-23T00:00:00Z.

Konstante	Typ	Wert	Kommentar

Debug log recording	Lange Ganzzahl	34	**Reichweite**: 4D Anwendung
			Wird zwischen 2 Sitzungen beibehalten: Nein
			Mögliche Werte: 0 oder 1
			Beschreibung: Startet oder stoppt das sequentielle Speichern von Ereignissen auf 4D Programmierebene in der Datei *4DDebugLog*, die automatisch in den Unterordner Logs neben der Strukturdatei gelegt wird. Ab 4D v14 wird im Logbuch für Ereignisse "4DDebugLog[_n].txt" ein neues kompakteres Textformat verwendet (_n ist die Segmentnummer der Datei).
			Werte: Lange Ganzzahl mit einem Bit-Feld: Wert = bit1(1)+bit2(2)+bit3(4)+bit4(8)+…).
			- Bit 1 (Wert 1) aktiviert das Schreiben des Logbuchs. Beachten Sie, dass jeder andere Wert, der nicht Null ist, es ebenfalls aktiviert.
			- Bit 2 (Wert 2) nimmt Parameter von Methoden und Befehlen mit auf.
			- Bit 3 (Wert 4) aktiviert neue Formate mit Tabulatoren.
			- Bit 4 (Wert 8) deaktiviert sofortiges Schreiben jeder Operation auf die Festplatte (standardmäßig aktiviert). Sofortiges Schreiben ist langsamer, erlaubt aber, die Ursachen bei einem Absturz herauszufinden. Deaktivieren macht den Inhalt der Datei kompakter und generiert ihn schneller.
			- Bit 5 (Wert 16) deaktiviert das Protokollieren von Plug-In Aufrufen (standardmäßig aktiviert)
			Im älteren Format ohne Tabs werden Ausführungszeiten in Millisekunden angezeigt und der Wert "< ms" erscheint für Operationen, die weniger als eine Millisekunde dauern
			Im Format mit Tab werden Ausführungszeiten in Mikrosekunden angezeigt.
			Beispiele:
			SET DATABASE PARAMETER (34;1) // aktiviert das Logbuch im Modus v13 ohne Parameter und mit der Dauer
			SET DATABASE PARAMETER (34;2) // aktiviert das Logbuch im Modus v13 mit Parametern und Dauer
			SET DATABASE PARAMETER (34;2+4) // aktiviert das Logbuch im v14 Format, mit Parametern und Dauer
			SET DATABASE PARAMETER (34;0) // deaktiviert das Logbuch
			Damit eine Datei nicht zuviel Information protokolliert, können Sie mit dem Selektor 80 Log Command list die Liste der zu prüfenden 4D Befehle einschränken.
			Diese Option lässt sich für alle 4D Applikationen aktivieren (4D alle Modi, 4D Server, 4D Volume Desktop), im interpretierten oder kompiliertem Modus.
			Hinweis: Diese Option dient nur für Debugging-Zwecke und sollte nicht im laufenden Betrieb verwendet werden, da sie zu Einschränkungen der Performance und Überlastung der Festplatte führen kann. Weitere Informationen zu diesem Format und Einsatz der Datei *4DDebugLog[_n].txt* finden Sie im **Anhang E: Beschreibung der Protokolldateien**.

Diagnostic log recording	Lange Ganzzahl	79	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 oder 1 (0 = nicht speichern, 1 = speichern) **Beschreibung:** Startet oder stoppt das Protokollieren der 4D Diagnosedatei. Standardmäßig ist der Wert 0 (nicht protokollieren). 4D kann kontinuierlich eine Reihe Ereignisse, die bei der internen Operationsweise der Anwendung auftreten, in einer Diagnosedatei mitschreiben. Information in dieser Datei ist hilfreich beim Entwickeln von 4D Anwendungen und kann mit Hilfe des 4D Tech Support analysiert werden. Übergeben Sie 1 in diesem Selector, wird im Ordner **Logs** der Anwendung automatisch eine Diagnosedatei mit Namen *DatenbankName.txt* angelegt bzw. geöffnet. Bei einer Größe von 10 MB wird sie geschlossen und es wird eine neue Datei mit Namen *DatenbankName_N.txt* generiert mit der ansteigenden Nummer N. Über den Befehl **LOG EVENT** können Sie in dieser Datei eigene Informationen hinzufügen.
Direct2D disabled	Lange Ganzzahl	0	Siehe Selector 69 (Direct2D Status)

Direct2D get active status	Lange Ganzzahl	74	**Hinweis:** Sie können diesen Selector nur mit der Funktion **Get database parameter** verwenden und keinen Wert setzen.
			Beschreibung: Gibt die aktuelle Integration von Direct2D unter Windows zurück.
			Mögliche Werte: 0, 1, 2, 3, 4 oder 5 (siehe Werte des Selector 69). Der zurückgegebene Wert richtet sich nach der Verfügbarkeit von Direct2D, sowie Hardware und der vom Betriebssystem unterstützten Qualität von Direct2D.
			Führen Sie zum Beispiel folgendes aus:
			`SET DATABASE PARAMETER(Direct2D status;Direct2D Hardware)`
			`$mode:=Get database parameter(Direct2D get active status)`
			- Ab Windows 7 und höher wird $mode auf 1 gesetzt, wenn das System Hardware findet, die mit Direct2D kompatibel ist; andernfalls wird $mode auf 3 gesetzt (Software-Kontext).
			- Unter Windows Vista wird *$mode* auf 1 gesetzt, wenn das System Hardware findet, die mit Direct2D; kompatibel ist; andernfalls wird *$mode* auf 0 gesetzt, d.h. Direct2D wird deaktiviert.
			- Unter Windows XP wird *$mode* immer auf 0 gesetzt, da diese Version nicht mit Direct2D kompatibel ist.

Direct2D hardware	Lange Ganzzahl	1	Siehe Selector 69 (Direct2D Status)
Direct2D software	Lange Ganzzahl	3	Siehe Selector 69 (Direct2D Status)

Konstante	Typ	Wert	Kommentar
Direct2D status	Lange Ganzzahl	69	**Reichweite:** 4D Anwendung **Zwischen 2 Sitzungen beibehalten:** Nein **Beschreibung:** Aktivierungsmodus zum Integrieren von Direct2D unter Windows. **Mögliche Werte:** Eine der folgenden Konstanten (standardmäßig Modus 5): Direct2D Disabled (0): Direct2D Modus ist nicht aktiviert. Die Anwendung funktioniert wie im früheren Modus (GDI/GDIPlus). Direct2D Hardware (1): Direct2D als Grafik Hardware-Kontext für die gesamte 4D Anwendung verwenden. Falls nicht verfügbar, Direct2D Grafik Software-Kontext verwenden (außer unter Vista; hier wird zur besseren Performance der Modus GDI/GDIPlus verwendet. Direct2D Software (3):(Standardmodus) Ab Windows 7 den Software-Kontext Direct2D Grafik für die gesamte 4D Anwendung verwenden. Unter Vista wird zur besseren Performance der Modus GDI/GDIPlus verwendet. **Hinweis zur Kompatibilität:** Ab 4D v14 werden hybrid Modi deaktiviert und auf die verfügbaren Modi umgeleitet (der bisherige Modus 2 entspricht 1; die bisherigen Modi 4 und 5 entsprechen Modus 3).
HTTP compression level	Lange Ganzzahl	50	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
HTTP compression threshold	Lange Ganzzahl	51	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
HTTPS Port ID	Lange Ganzzahl	39	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
Idle connections timeout	Lange Ganzzahl	54	**Reichweite:** 4D Anwendung, außer Wert ist negativ **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** Ganzzahl für die Dauer in Sekunden. Der Wert kann positiv sein (neue Verbindungen) oder negativ (vorhandene Verbindungen). Standardmäßig ist der Wert 20. **Beschreibung:** Dieser Parameter ermöglicht auf der Server-Seite die maximale Zeitspanne an Inaktivität (timeout) für Verbindungen zur 4D Datenbank-Engine und der SQL Engine zu definieren. Erreicht eine pausierende Verbindung das Limit, wird sie automatisch auf standby gesetzt. Das friert die Client/Server Sitzung ein und schließt den Netzwerk Socket. Im Server Verwaltungsfenster lautet der Status des Benutzerprozesses "Zurückgestellt". Dieser Ablauf läuft für den Benutzer unsichtbar ab: Sobald bei der auf standby gesetzten Verbindung neue Aktivität auftritt, wird der Socket automatisch wieder geöffnet und die Client/Server Sitzung wiederhergestellt. Diese Einstellung lässt Sie einerseits Ressourcen auf dem Server einsparen: Verbindungen in standby schließen den Socket und geben einen Prozess auf dem Server frei. Andererseits können Sie so auch verhindern, dass Verbindungen verloren gehen, weil pausierende Sockets durch die Firewall geschlossen werden. Deshalb muss in diesem Fall der Wert für das Timeout von pausierenden Verbindungen niedriger als der für die Firewall sein. Übergeben Sie einen positiven Wert in *Wert*, gilt er für alle neuen Verbindungen in allen Prozessen. Übergeben Sie einen negativen Wert, gilt er für Verbindungen, die im aktuellen Prozess geöffnet sind. Übergeben Sie 0, unterliegen pausierende Verbindungen keinem Timeout. Dieser Parameter lässt sich auf Server und Client setzen. Geben Sie zwei unterschiedliche Zeiten an, wird der kleinere Wert berücksichtigt. Im Normalfall müssen Sie diesen Wert nicht ändern.
Invert objects	Lange Ganzzahl	37	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0, 1 oder 2 (0 = Modus inaktiv, 1 = automatischer Modus, 2 = Modus aktiviert). **Beschreibung:** Einstellen des Modus "Objekt-Inversion", um Formulare, Objekte, Menüleisten, etc. im Anwendungsmodus umzukehren, wenn die Datenbank unter Windows in einer rechts-nach-links laufenden Sprache angezeigt wird. Dieser Modus lässt sich auch auf der Seite **Oberfläche** in den Datenbank-Eigenschaften einstellen: • Wert 0 gibt an, dass der Modus nie aktiviert wird, unabhängig von der Systemkonfiguration (entspricht der Einstellung **Nie** auf der Seite **Oberfläche**). • Wert 1 gibt an, dass der Modus je nach Systemkonfiguration aktiviert oder deaktiviert ist (entspricht der Einstellung **Automatisch** auf der Seite **Oberfläche**). • Wert 2 gibt an, dass der Modus aktiviert ist, unabhängig von der Systemkonfiguration (entspricht der Einstellung **Immer** auf der Seite **Oberfläche**). Weitere Informationen dazu finden Sie im Handbuch *4D Designmodus*.
Log command list	Zeichenkette	80	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** String mit der Liste der 4D Befehlsnummern zum Protokollieren, getrennt durch Strichpunkte. Sie können "all" übergeben, um alle Befehle zu protokollieren oder "" (leerer String), um keine Befehle zu protokollieren. **Beschreibung***:* Liste der 4D Befehle, die in der Protokolldatei mitgeschrieben werden (siehe Selector 34, Debug Log Recording). Standardmäßig werden alle 4D Befehle protokolliert. Dieser Selector reduziert die Informationen, die in der Diagnosedatei gesichert wird, durch Einschränken der 4D Befehle, deren Ausführung Sie nicht protokollieren wollen. Sie können z.B. schreiben: `SET DATABASE PARAMETER(Log command list;"277;334") //Nur die Befehle QUERY und QUERY SELECTION protokollieren`

Konstante	Typ	Wert	Kommentar
Max concurrent Web processes	Lange Ganzzahl	18	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
Maximum temporary memory size	Lange Ganzzahl	61	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Werte:** Positive Lange Ganzzahl **Beschreibung:** Maximale Größe des temporären Speichers, den 4D jedem Prozess zuordnen kann, ausgedrückt in MB. Der Wert ist standardmäßig 0 (keine maximale Größe). 4D verwendet einen speziellen temporären Speicher zum Indizieren und Sortieren der Operationen. Ziel dieses Speichers ist, den "standard" Cache Speicher bei massiven Operationen beizubehalten. Er wird nur bei Bedarf aktiviert. Die Größe des temporären Speichers wird nur durch die verfügbaren Ressourcen begrenzt. Das richtet sich nach der Konfiguration des Systemspeichers. Diese Vorgehensweise passt für die meisten Programme. In einigen spezifischen Fällen, insbesondere wenn eine Client-/Server Anwendung simultan eine große Anzahl sequentieller Sortierungen durchführt, kann der temporäre Speicher eine kritische Größe erreichen, die das System instabil macht. Für solche Fälle können Sie für den temporären Speicher eine maximale Größe einrichten, damit die Anwendung weiterhin einwandfrei laufen kann. Das kann jedoch wiederum die Ausführungsgeschwindigkeit beeinträchtigen: Ist die maximale Größe für einen Prozess erreicht, verwendet 4D Dateien der Festplatte, was den Vorgang verlangsamen kann. Für spezifische Anforderungen, wie z.B. oben beschrieben, ist im allgemeinen eine maximale Größe von ca. 50 MB ein guter Kompromiss. Der ideale Wert muss jedoch anhand der Eigenheiten der Anwendung bestimmt werden und ergibt sich aus Volumentests in Echtzeit.
Maximum Web process	Lange Ganzzahl	7	**Reichweite:** 4D lokal, 4D Server **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 -> 32,767 **Beschreibung:** Höchstanzahl der zu unterhaltenden Web Prozesse im nicht kontextuellen Modus mit 4D im lokalen Modus und 4D Server. Standardmäßig ist der Wert 10. Um den Web Server reaktivierbar zu machen, verzögert 4D im nicht-kontextuellen Modus die Web Prozesse um 5 Sekunden und verwendet sie wieder, um mögliche spätere HTTP Anfragen auszuführen. In Bezug auf die Performance ist dies vorteilhafter als pro Anfrage einen neuen Prozess zu erstellen. Sobald ein Web Prozess wieder verwendet wird, wird er erneut um 5 Sekunden verzögert, außer die Höchstanzahl der Web Prozesse ist bereits erreicht. Dann wird er abgebrochen. Erhält ein Web Prozess nicht binnen 5 Sekunden eine Anfrage, wird er abgebrochen. Mit diesen Parametern können Sie je nach Anzahl der Anfragen, nach verfügbarem Speicher, etc. einstellen, wie Ihr Web Server arbeitet.
Maximum Web requests size	Lange Ganzzahl	27	**Beschreibung:** Konstante ist überholt und wird nur zur Wahrung der Kompatibilität beibehalten. Wir empfehlen zum Konfigurieren des HTTP Server die Befehle **WEB SET OPTION** und **WEB GET OPTION** zu verwenden.
Min TLS version	Lange Ganzzahl	105	**Reichweite**: 4D Server, 4D Web Server und 4D SQL Server **Wird zwischen 2 Sitzungen beibehalten**: Nein **Beschreibung**: Gibt die Ebene von Transport Layer Security (TLS) für die Datenverschlüsselung und Authentifizierung zwischen Anwendungen und Servern an. Die definierten Werte gelten global für die Netzwerkebene. Ein geänderter Wert wird erst nach Neustart des Servers verwendet. Das standardmäßige Mindestprotokoll ist TLSv1_2. **Mögliche Werte**: • TLSv1_0 - TLS 1.0, eingeführt in 1999 als Upgrade von SSL v3.0. TLS 1.0 und SSL 3.0 funktioniere n nicht zusammen. • TLSv1_1 - TLS 1.1, eingeführt in 2006 als Update von TLS 1.0. Verbesserungen sind u.a. bessere Sicherheit und Fehlerverwaltung. • TLSv1_2 - TLS 1.2, eingeführt in 2008 als Update von TLS 1.1. Verbesserungen sind u.a. erhöhte Flexibilität, zusätzliche Unterstützung für authentifizierte Verschlüsselungen. **Hinweis**: Das Plug-In 4D Internet Commands nutzt eine andere Netzwerkschicht. Von daher hat dieser Selector keine Auswirkung auf deren TLS Version.
Minimum Web process	Lange Ganzzahl	6	**Reichweite:** 4D lokal, 4D Server **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Werte: 0 -> 32 767 **Beschreibung:** Mindestanzahl der zu unterhaltenden Web Prozesse im nicht kontextuellen Modus mit 4D im lokalen Modus und 4D Server. Standardmäßig ist der Wert 0 (siehe unten).

Konstante	Typ	Wert	Kommentar
Number of formulas in cache	Lange Ganzzahl	92	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Mögliche Werte**: Positive Lange Ganzzahl **Standardwert**: 0 (kein Cache) **Beschreibung**: Setzt oder erhält die maximale Anzahl Formeln, die im vom Befehl [#cmd id="63"/] verwendeten Formel-Cache beibehalten werden sollen. Diese Begrenzung gilt für alle Prozesse, jedoch hat jeder Prozess seinen eigenen Formel-Cache. Formeln im Cache zu halten, beschleunigt die Ausführung von **EXECUTE FORMULA** im kompilierten Modus, da jede dieser Formeln nur einmal tokenisiert wird (weitere Info siehe [#title id="8567"/]). Ändern Sie den Cache Wert, wird der vorhandene Inhalt zurückgesetzt, selbst wenn die neue Größe die bisherige übersteigt. Ist die Anzahl Formeln im Cache erreicht, ersetzt die neu ausgeführte Formel die älteste im Cache (FIFO Modus). Dieser Parameter wird nur in kompilierten Anwendungen oder Komponenten berücksichtigt.
Order by formula on server	Lange Ganzzahl	47	**Reichweite:** Aktuelle Tabelle und Prozess **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 (Konfiguration der Datenbank verwenden), 1 (auf Client ausführen) oder 2 (auf Server ausführen) **Beschreibung:** Ausführungsort des Befehls **ORDER BY FORMULA** für die im Parameter übergebene Tabelle. Bei einer Datenbank im Client/Server-Betrieb lässt sich dieser Befehl entweder auf dem Server oder auf dem Client-Rechner ausführen. Über diesen Selector können Sie den Ausführungsort dieses Befehls angeben (Server oder Client). Sie können diesen Modus auch in dern Datenbank-Eigenschaften setzen. Weitere Informationen dazu finden Sie unter Selector 46, Query By Formula On Server. **Hinweis:** Wollen Sie auch die Möglichkeit haben, "SQL type" joins zu aktivieren (siehe Selector QUERY BY FORMULAR Joins, müssen Sie Formeln immer auf dem Server ausführen, damit diese auf die Datensätze zugreifen können. Bedenken Sie, dass die Formel in diesem Kontext keine Aufrufe einer Methode enthalten darf, da sie sonst automatisch auf den remote Rechner wechselt.
PHP interpreter IP address	Lange Ganzzahl	55	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: Formatierter String vom Typ IPv4 (zum Beispiel "127.0.0.1") oder IPv6 (zum Beispiel "2001:0db8:0000:0000:0000:ff00:0042:8329"). **Beschreibung**: von 4D lokal verwendete IP Adresse, um mit dem PHP Interpreter via FastCGI zu kommunizieren. Der Wert ist standardmäßig "127.0.0.1" (Adressen im IPv6 Format werden ab 4D v16R4 unterstützt). Diese Adresse muss dem Rechner entsprechen, auf dem 4D läuft. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner setzen. Weitere Informationen zum PHP Interpreter finden Sie im Handbuch *4D Designmodus*.
PHP interpreter port	Lange Ganzzahl	56	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: Positive Lange Ganzzahl. Standardmäßig ist der Wert 8002. **Beschreibung**: Nummer des TCP Port, den der PHP Interpreter von 4D verwendet. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner setzen. Weitere Informationen zum PHP Interpreter finden Sie im Handbuch *4D Designmodus*.
PHP max requests	Lange Ganzzahl	58	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: Positive Lange Ganzzahl. Standardmäßig ist der Wert 500. **Beschreibung**: Maximale Anzahl der Anfragen, die der PHP Interpreter akzeptiert. Ist die maximale Anzahl erreicht, gibt der Interpreter Fehler vom Typ "Server ist überlastet" zurück. Sie können diesen Wert aus Sicherheits- oder Performance-Gründen verändern. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner verändern. Weitere Informationen dazu finden Sie in der Dokumentation zu FastCGI-PHP. **Hinweis:** Auf 4D Seite werden diese Parameter dynamisch angewandt; d.h. 4D muss nicht beendet werden, damit sie berücksichtigt werden. Ist dagegen der PHP Interpreter bereits gestartet, muss beendet und neu gestartet werden, damit diese Änderungen berücksichtigt werden.
PHP number of children	Lange Ganzzahl	57	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: Positive Lange Ganzzahl. Standardmäßig ist der Wert 5. **Beschreibung**: Anzahl der Kindprozesse, die vom PHP Interpreter von 4D erstellt und lokal gewartet werden. Aus Optimierungsgründen erstellt und verwendet der PHP Interpreter zum Bearbeiten der Anfragen zur Skript-Ausführung einen Satz von Systemprozessen, genannt "Kindprozesse". Sie können die Anzahl der Kindprozesse je nach den Anforderungen Ihrer Anwendung variieren. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner setzen. Weitere Informationen zum PHP Interpreter finden Sie im Handbuch *4D Designmodus*. **Hinweis:** Auf Mac OS nutzen alle Kindprozesse den gleichen Port. Unter Windows verwendet jeder Kindprozess eine spezifische Port Nummer. Die erste Nummer ist die für den PHP Interpreter gesetzte; die anderen Kindprozesse erhöhen die erste Nummer jeweils. Ist zum Beispiel der Standardport 8002 und starten Sie 5 Kindprozesse, verwenden sie die Ports 8002 bis 8006.

Konstante	Typ	Wert	Kommentar
PHP use external interpreter	Lange Ganzzahl	60	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: 0 = Verwende internen Interpreter, 1 = Verwende externen Interpreter **Beschreibung**: Der Wert gibt an, ob PHP Anfragen in 4D an den von 4D intern gelieferten Interpreter oder an einen externen Interpreter gesendet werden. Standardmäßig ist der Wert 0, d.h. der interne Interpreter von 4D wird verwendet. Wollen Sie Ihren eigenen PHP Interpreter verwenden, um z.B. zusätzliche Module oder eine spezifische Konfiguration zu verwenden, übergeben Sie 1 in *Wert*. In diesem Fall startet 4D bei PHP Anfragen nicht den internen Interpreter. Eigene PHP Interpreter müssen in FastCGI kompiliert sein und auf demselben Rechner wie die 4D Engine liegen. Beachten Sie, dass Sie diese Interpreter komplett selbst verwalten müssen; sie werden von 4D weder gestartet noch gestoppt. Dieser Parameter lässt sich über die Einstellungen auch global für alle Rechner setzen.
Port ID	Lange Ganzzahl	15	**Reichweite**: 4D lokal, 4D Server **Wird zwischen zwei Sitzungen beibehalten**: Nein **Beschreibung**: ID des TCP Port, den der 4D Web Server mit 4D im lokalen Modus und 4D Server verwendet. Der Standardwert ist 80 und lässt sich in den Datenbank-Eigenschaften auf der Seite **Web/Konfiguration** setzen. Für den Parameter *Wert* können Sie eine Konstante unter dem Thema **TCP Port Nummern** verwenden. Der Selector Port ID ist hilfreich für 4D Web Server mit einkompilierter 4D Desktop, d.h. ohne Zugriff auf den Designmodus. Weitere Informationen dazu finden Sie im Abschnitt **Web Server, Einstellungen**.
Query by formula joins	Lange Ganzzahl	49	**Reichweite:** Aktueller Prozess **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 (Konfiguration der Datenbank verwenden), 1 (immer automatische Verknüpfungen) oder 2 (Wenn möglich, SQL joins verwenden). **Beschreibung:** Arbeitsweise der Befehle **QUERY BY FORMULA** und **QUERY SELECTION BY FORMULA** bei Verwendung von "SQL joins". In Datenbanken, die mit Version 11.2 von 4D v11 SQL erstellt wurden, führen diese Befehle Verknüpfungen nach dem SQL joins Modell aus. Auf diese Weise lässt sich die ausgewählte Tabelle verändern, wenn eine Suche in einer anderen Tabelle ausgeführt wird, auch wenn diese Tabellen nicht über eine automatische Verknüpfung miteinander verbunden sind. Diese automatische Verknüpfung war in bisherigen 4D Versionen erforderlich. Über den Selector QUERY BY FORMULA Joins legen Sie fest, wie die Suchbefehle im aktuellen Prozess arbeiten: • 0 (Standardwert) = Verwendet die aktuellen Einstellungen der Datenbank. In Datenbanken, die mit Version 11.2 von 4D v11 SQL erstellt wurden, sind für Suchen nach Formel immer "SQL joins" aktiviert. In konvertierten Datenbanken ist diese Arbeitsweise aus Kompatibilitätsgründen nicht aktiv, sie lässt sich aber über eine Voreinstellung einrichten. • 1 = Verwende immer automatische Verknüpfungen (= bisherige Funktionsweise in 4D). In diesem Modus ist eine Verknüpfung notwendig, damit bei Suche in einer anderen Tabelle die entsprechende Tabelle ausgewählt wird. 4D macht keine "SQL joins." • 2 = Verwende SQL joins, wenn möglich (= Standardverhalten in Datenbanken, die mit Version 11.2 von 4D v11 SQL und höher erstellt wurden). In diesem Modus erstellt 4D "SQL joins" für Suchen nach Formel, wenn die Formel dazu passt - bis auf zwei Ausnahmen. Weitere Informationen dazu finden Sie unter den Befehlen **QUERY BY FORMULA** oder **QUERY SELECTION BY FORMULA**. **Hinweis:** Mit 4D im remote Modus lassen sich "SQL joins" nur verwenden, wenn die Formeln auf dem Server ausgeführt werden, da diese auf die Datensätze zugreifen müssen. Wie Sie konfigurieren, wo Formeln ausgeführt werden, sehen Sie unter den Selectoren 46 und 47.

Konstante	Typ	Wert	Kommentar
Query by formula on server	Lange Ganzzahl	46	**Reichweite:** Aktuelle Tabelle und Prozess **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 (Konfiguration der Datenbank verwenden), 1 (auf Client ausführen) oder 2 (auf Server ausführen) **Beschreibung:** Ausführungsort der Befehle **QUERY BY FORMULA** und **QUERY SELECTION BY FORMULA** für die im Parameter übergebene Tabelle. Bei einer Datenbank im Client/Server-Betrieb können die Suchbefehle "nach Formel" entweder auf dem Server oder auf dem Client Rechner ausgeführt werden. • In einer mit 4D v11 SQL erstellten Datenbank werden die Befehle auf dem Server ausgeführt. • In konvertierten Datenbanken werden sie, wie in den bisherigen 4D Versionen, auf dem Client Rechner ausgeführt. • In konvertierten Datenbanken gibt es eine spezielle Voreinstellung (Seite Anwendung>Kompatibilität), um den Ausführungsort dieser Befehle global zu verändern. Der Ausführungsort beeinflusst nicht nur die Performance der Anwendung - die Ausführung auf dem Server ist in der Regel schneller - sondern auch die Programmierung. Der Wert der Komponenten der Formel, insbesondere bei Variablen, die über eine Methode aufgerufen werden, ändert sich je nach Ausführungsort. Über diesen Selektor können Sie die Arbeitsweise Ihrer Anwendung einstellen. Übergeben Sie 0 (Null) im Parameter Wert, richtet sich der Ausführungsort der Suchbefehle nach der Konfiguration der Datenbank: In mit 4D v11 SQL erstellten Datenbanken werden diese Befehle auf dem Server ausgeführt. In konvertierten Datenbanken werden sie, je nach den Datenbank-Eigenschaften, auf dem Client Rechner oder Server ausgeführt. Übergeben Sie 1 oder 2 in Wert, um die Ausführung dieser Befehle jeweils auf dem Client oder Server Rechner zu erzwingen. Siehe Beispiel 2. **Hinweis:** Wollen Sie auch die Möglichkeit haben, "SQL type" joins zu aktivieren (siehe Selector QUERY BY FORMULAR Joins, müssen Sie Formeln immer auf dem Server ausführen, damit diese auf die Datensätze zugreifen können. Bedenken Sie, dass die Formel in diesem Kontext keine Aufrufe einer Methode enthalten darf, da sie sonst automatisch auf den remote Rechner wechselt.
QuickTime support	Lange Ganzzahl	82	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Ja **Werte**: 0 (Standard) = QuickTime deaktiviert, 1 = QuickTime aktiviert **Beschreibung**: 4D Version 14 unterstützt QuickTime Codecs nicht mehr standardmäßig. Zur Wahrung der Kompatibilität können Sie die Unterstützung in Ihrer Anwendung über diesen Selector reaktivieren. Beachten Sie jedoch, dass die QuickTime Unterstützung in zukünftigen 4D Versionen komplett eingestellt wird.
Server base process stack size	Lange Ganzzahl	53	**Reichweite:** 4D Server **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** Positive Lange Ganzzahl **Beschreibung:** Größe des Stapelspeichers, der jedem preemptiven Systemprozess auf dem Server zugewiesen wird, ausgedrückt in Bytes. Die Standardgröße wird vom System bestimmt. Preemptive Systemprozesse, d.h. Prozesse vom Typ 4D Client base process werden geladen, um die Hauptprozesse von 4D Client zu steuern. Die Größe, die dem Stapelspeicher jedes preemptiven Prozesses standardmäßig zugewiesen ist, sorgt für eine gut laufende Ausführung, kann aber eine wichtige Rolle spielen, wenn eine umfangreiche Anzahl, also mehrere hundert Prozesse erstellt werden. Diese Größe lässt sich für Optimierungszwecke beträchtlich verringern, wenn die von der Datenbank ausgeführten Operationen dies zulassen. Das ist z.B. der Fall, wenn die Datenbank keine Sortierung auf eine große Anzahl Datensätze ausführt. Möglich sind Werte von 512 oder auch 256 KB. Beachten Sie, dass eine zu geringe Stapelgröße kritisch werden kann, da sie die Operationen von 4D Server beeinträchtigt. Setzen Sie diesen Parameter mit Bedacht und unter Berücksichtigung der Datenbankbedingungen, wie z.B. Anzahl der Datensätze, Art der Operationen, etc. Damit dieser Parameter berücksichtigt wird, muss er auf dem Server-Rechner ausgeführt werden, z.B. in der **Datenbankmethode On Server Startup**.
Spellchecker	Lange Ganzzahl	81	**Reichweite**: 4D Anwendung **Wird zwischen zwei Sitzungen beibehalten**: Nein **Werte**: 0 (Standard) = native OS X Rechtschreibprüfung (Hunspell deaktiviert), 1 = Hunspell Rechtschreibprüfung aktiviert. **Beschreibung**: Aktiviert die Hunspell Rechtschreibprüfung auf OS X. Auf dieser Plattform wird standardmäßig die native Rechtschreibprüfung aktiviert. Sie können jedoch die Hunspell Rechtschreibprüfung aktivieren, z.B. um die Oberfläche Ihrer plattformunabhängigen Anwendungen einheitlich zu gestalten (unter Windows ist nur die Hunspell Rechtschreibprüfung verfügbar). Weitere Informationen dazu finden Sie im Abschnitt **Rechtschreibprüfung**.
SQL Autocommit	Lange Ganzzahl	43	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 (deaktivieren) oder 1 (aktivieren) **Beschreibung:** Aktivierung oder Deaktivierung des Modus SQL auto-commit. Standardmäßig ist der Wert 0 (deaktiviert). Der Modus auto-commit sorgt für strengere referentielle Integrität der Datenbank. Ist er aktiv, werden alle Suchläufe mit **SELECT**, **INSERT**, **UPDATE** und **DELETE** (SIUD) automatisch in ad hoc Transaktionen gesetzt, wenn das noch nicht der Fall ist. Sie können diesen Modus auch in den Einstellungen der Datenbank setzen.

Konstante	Typ	Wert	Kommentar
SQL Engine case sensitivity	Lange Ganzzahl	44	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 oder 1 (0 = Groß-/Kleinschreibung nicht beachten, 1 = Groß-/Kleinschreibung beachten) **Beschreibung:** Aktiviert oder deaktiviert die Berücksichtigung von Groß-/Kleinschreibung beim Vergleichen von Strings mit der SQL Engine. Der Wert ist standardmäßig 1, d.h. die SQL Engine unterscheidet beim Vergleichen von Strings zwischen Groß- und Kleinschreibung (Sortieren und Suchen) und Akzenten. Beispiel: "ABC"= "ABC" aber "ABC" # "Abc und "abc" # "âbc". In bestimmten Fällen, z.B. um die Funktionsweise der SQL Engine an die 4D Engine anzupassen, sollen beim Vergleichen von Strings die Groß- und Kleinschreibung, sowie Akzente nicht berücksichtigt werden ("ABC"="Abc"="âbc"). Sie können diese Option auch in den Datenbank-Eigenschaften auf der **Seite SQL** einstellen.
SQL Server Port ID	Lange Ganzzahl	88	**Reichweite**: 4D im lokalen Modus und 4D Server. **Wird zwischen 2 Sitzungen beibehalten**: Ja **Beschreibung**:Setzt oder erhält die Nummer des TCP Port, den der in 4D integrierte SQL Server mit 4D im lokalen Modus oder 4D Server verwendet. Der Wert ist standardmäßig 19812. Die TCP Port Nummer lässt sich auch auf der Seite "SQL" in den Datenbank-Eigenschaften setzen. Mit diesem Selector wird die Datenbank-Eigenschaft entsprechend aktualisiert und bleibt auch nach Beenden und Neustart erhalten. **Mögliche Werte**: 0 bis 65535 **Standardwert**: 19812
SSL cipher list	Zeichenkette	64	**Reichweite**: 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: Folge von Strings, getrennt durch Doppelpunkt (zum Beispiel "RC4-MD5:RC4-64-MD5:....") **Beschreibung:** Cipher Liste, die 4D für das SSL Protokoll verwendet. Diese Liste verändert die Priorität der in 4D implementierten cipher Algorithmen. Sie können im Parameter *Wert* z.B. folgenden String übergeben. "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH". Eine ausführliche Beschreibung der Syntax für die Cipher Liste finden Sie auf der ciphers Seite der OpenSSL Site. Diese Einstellung gilt für die gesamte Anwendung (sie betrifft HTTP Server, SQL Server, Client/Server Verbindungen, sowie den HTTP Client und alle 4D Befehle, die das SSL Protokoll nutzen), aber sie ist temporär, d.h. sie bleibt zwischen Sitzungen nicht erhalten. Wurde die Cipher Liste geändert, müssen Sie den betroffenen Server neu starten, damit die neuen Einstellungen berücksichtigt werden. Um die Cipher Liste auf ihren Standardwert zurückzusetzen, der permanent in der SLI Datei gespeichert ist, rufen Sie den Befehl **SET DATABASE PARAMETER** auf und übergeben im Parameter *Wert* einen leeren String (""). **Hinweis:** Mit der Funktion **Get database parameter** wird die Cipher Liste im optionalen Parameter *StringWert* zurückgegeben und der Parameter ist immer 0.
String type with time zone	Lange Ganzzahl	1	Stringtyp mit Zeitzonen-Selector für Dates inside objects
String type without time zone	Lange Ganzzahl	0	Stringtyp ohne Zeitzonen-Selector für Dates inside objects
Table sequence number	Lange Ganzzahl	31	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** Jeder Wert vom Typ Lange Ganzzahl. **Beschreibung:** Dieser Selector dient zum Ändern oder Erhalten der aktuellen einmaligen Nummer für Datensätze der Tabelle, die als Parameter übergeben ist. "Aktuelle Nummer" bedeutet "zuletzt verwendete Nummer": Verändern Sie diesen Wert über den Befehl **SET DATABASE PARAMETER**, erhält der nächste Datensatz die Nummer übergebener Wert + 1. Diese neue Nummer wird von der Funktion **Sequence number** sowie in einem Feld der Tabelle zurückgegeben, der die Eigenschaft "Autoincrement" im Inspektorfenster oder via SQL zugeordnet wurde. Diese einmalige Nummer wird standardmäßig von 4D angelegt und entspricht der Reihenfolge, in der die Datensätze erstellt werden. Weitere Informationen dazu finden Sie in der Beschreibung zur Funktion **Sequence number**.
Times in milliseconds	Lange Ganzzahl	1	Times storage selector for Times inside objects
Times in seconds	Lange Ganzzahl	0	Times storage selector for Times inside objects
Times inside objects	Lange Ganzzahl	109	**Reichweite**: 4D lokal, 4D Server (alle Prozesse) **Wird zwischen 2 Sitzungen beibehalten**: Nein **Mögliche Werte**: Times in seconds (0) (Standard), Times in milliseconds (1) **Beschreibung**: Definert, wie Zeitangaben konvertiert und in Objekteigenschaften und Elementen in Collections gespeichert werden, und wie sie in JSON und in Web Areas importiert bzw. daraus exportiert werden. Ab 4D v17 werden Zeitangaben in Objekten standardmäßig in Anzahl von Sekunden konvertiert und gespeichert. In bisherigen Releases wurden Zeitwerte in diesem Kontext als Anzahl von Millisekunden konvertiert und gespeichert. Dieser Selector unterstützt Sie beim Migrieren Ihrer Anwendungen und geht bei Bedarf zurück auf die bisherige Einstellung. **Hinweis**: ORDA Methoden und SQL Engine ignorieren diese Einstellung, sie gehen immer davon aus, dass Zeitwerte die Anzahl von Sekunden sind.

Konstante	Typ	Wert	Kommentar
Tips delay	Lange Ganzzahl	102	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** Lange Ganzzahl >= 0 (Ticks) **Beschreibung:** Verzögerte Anzeige von Tipps, nachdem der Mauszeiger in Objekten mit zugewiesenen Hilfemeldungen gesetzt wurde. Wert wird in Ticks gerechnet (1/60stel Sekunde). Standardwert ist 45 Ticks (0.75 Sekunde).
Tips duration	Lange Ganzzahl	103	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** Lange Ganzzahl >= 60 (Ticks) **Beschreibung:** Maximale Anzeigedauer für einen Tipp. Wert wird in Ticks gerechnet (1/60stel Sekunde). Standardwert ist 720 Ticks (12 Sekunden).
Tips enabled	Lange Ganzzahl	101	**Reichweite:** 4D Anwendung **Wird zwischen 2 Sitzungen beibehalten:** Nein **Mögliche Werte:** 0 = Tipps deaktiviert, 1 = Tipps aktiviert (Standard) **Beschreibung:** Setzt oder erhält den aktuellen Status der Anzeige von Tipps für die 4D Anwendung. Tipps sind standardmäßig aktiviert. Beachten Sie, dass dieser Parameter alle 4D Tipps setzt, z.B. Hilfemeldungen für Formulare und Tipps im Editor des Designmodus.
TLSv1_0	Lange Ganzzahl	1	Siehe Selector 105 (Min TLS Version).
TLSv1_1	Lange Ganzzahl	2	Siehe Selector 105 (Min TLS Version).
TLSv1_2	Lange Ganzzahl	3	Siehe Selector 105 (Min TLS Version).
Unicode mode	Lange Ganzzahl	41	**Reichweite:** Datenbank **Wird zwischen 2 Sitzungen beibehalten:** Ja **Mögliche Werte:** 0 (Modus Kompatibilität) oder 1 (Modus Unicode) **Beschreibung:** Aktueller Ausführungsmodus der Datenbank in Bezug auf den Zeichensatz. 4D unterstützt den Unicode Zeichensatz, kann aber auch im Modus "Kompatibilität" arbeiten, der auf dem Zeichensatz Mac ASCII basiert. Standardmäßig werden konvertierte Datenbanken im Modus Kompatibilität (0), in Version 11 oder höher erstellte Datenbanken im Modus Unicode ausgeführt. Der Ausführungsmodus lässt sich über eine Option in den Voreinstellungen steuern. Er lässt sich über diesen Selector auch lesen oder für Testzwecke verändern. Sie müssen die Datenbank neu starten, damit die Änderung berücksichtigt wird. Beachten Sie, dass Sie diesen Wert innerhalb einer Komponente nicht verändern, sondern ihn nur lesen können.
Use legacy network layer	Lange Ganzzahl	87	**Reichweite**: 4D im lokalen Modus, 4D Server **Wird zwischen 2 Sitzungen beibehalten**: Ja **Beschreibung**: Setzt oder erhält den aktuellen Status der legacy Netzwerk-Schicht für Client/Server Verbindungen. Die legacy Netzwerk-Schicht ist ab 4D v14 R5 überholt und sollte in Ihren Anwendungen nach und nach durch die Netzwerk-Schicht *ServerNet* ersetzt werden. *ServerNet* ist in den nächsten 4D Releases erforderlich, um auch in Zukunft von Netzwerk Evolutionen zu profitieren. Zur Wahrung der Kompatibilität wird die bisherige Netzwerk-Schicht noch unterstützt, um den allmählichen Übergang für vorhandene Anwendungen zu ermöglichen. Sie wird in konvertierten Anwendungen aus einem Release vor v14 R5 standardmäßig genutzt. Übergeben Sie 1, um für Ihre Client/Server Verbindungen die bisherige Netzwerk-Schicht zu verwenden (und *ServerNet* zu deaktivieren); 0, um sie zu deaktivieren (und *ServerNet* zu verwenden). Diese Eigenschaft lässt sich auch über die Option "Verwende legacy Netzwerk Schicht" auf der **Seite Kompatibilität** der Datenbank-Eigenschaften setzen. Hier wird auch die Vorgehensweise zum Migrieren erläutert. Wir empfehlen, so bald wie möglich in all Ihren Anwendungen auf *ServerNet* umzustellen (siehe auch **Netzwerk und Client-Server Optionen**). Sie müssen die Anwendung neu starten, damit dieser Selektor berücksichtigt wird. **Mögliche Werte**: 0 oder 1 (0 = legacy Schicht nicht verwenden, 1 = legacy Schicht verwenden) **Standardwert:** 0 in Datenbanken, die mit 4D v14 R5 oder neuer erstellt wurden, 1 in Datenbanken, die aus 4D v14 R4 oder früher konvertiert wurden.

Datendatei Wartung

Konstante	Typ	Wert	Kommentar
Compact address table	Lange Ganzzahl	131072	Neuschreiben der Adresstabellen der Datensätze erzwingen (verlangsamt die Komprimierung). Beachten Sie, dass in diesem Fall die Nummern der Datensätze neu geschrieben werden. Übergeben Sie nur diese Option, aktiviert 4D automatisch die Option 'Datensätze aktualisieren'.
Create process	Lange Ganzzahl	32768	Ist diese Option übergeben, erfolgt das Komprimieren asynchron und Sie müssen die Ergebnisse über die Callback Methode verwalten. 4D zeigt keinen Ablaufbalken an (das ist bei Verwenden der Callback Methode möglich). Die Systemvariable OK wird auf 1 gesetzt, wenn der Prozess korrekt gestartet wurde, in allen anderen Fällen auf 0. Ist diese Option nicht übergeben, wird die Systemvariable OK auf 1 gesetzt, wenn die Komprimierung korrekt ausgeführt wurde, sonst auf 0.
Do not compact index	Lange Ganzzahl	2	
Do not create log file	Lange Ganzzahl	16384	Dieser Befehl erstellt generell ein Logbuch im XML Format. Mit dieser Option wird kein Logbuch angelegt.
Move to replaced files folder	Lange Ganzzahl	4	
New file	Lange Ganzzahl	0	
New file dialog	Lange Ganzzahl	1	
Renumber records	Lange Ganzzahl	1	
Timestamp log file name	Lange Ganzzahl	262144	Ist diese Option übergeben, zeigt der Name des generierten Logbuchs Datum und Uhrzeit seiner Erstellung und ersetzt dann folglich nicht das vorige Logbuch. Standardmäßig hat der Name eines Logbuchs keinen Zeitstempel und das vorige Logbuch wird überschrieben.
Update records	Lange Ganzzahl	65536	Neuschreiben aller Datensätze gemäß der aktuellen Definition der Felder in der Struktur erzwingen.
Use default folder	Lange Ganzzahl	1	Die Daten des Feldes, das als Parameter übergeben wird, werden im Standardordner mit Namen DatenbankName.ExternalData gesichert und neben die Datendatei gelegt. In diesem Modus verwaltet 4D externe Daten wie Daten innerhalb der Datendatei.
Use selected file	Lange Ganzzahl	2	
Use structure definition	Lange Ganzzahl	0	4D verwendet die Parameter, die in der Struktur zum Speichern des Felds angegeben sind (siehe Handbuch *4D Designmodus*). Wechseln Sie vom externen Speichern zum internen Speichern, wird die externe Datei nicht gelöscht.
Verify all	Lange Ganzzahl	16	
Verify indexes	Lange Ganzzahl	8	Diese Option prüft die physikalische Konsistenz der Indizes, ohne Verknüpfung zu den Daten. Sie zeigt ungültige Verknüpfungen an, kann jedoch keine doppelten Verknüpfungen ausfindig machen (zwei Indizes zeigen auf den gleichen Datensatz). Diese Art Fehler lässt sich nur mit der Option Verify all ausfindig machen.
Verify records	Lange Ganzzahl	4	

Datum Anzeigeformate

Konstante	Typ	Wert	Kommentar
Blank if null date	Lange Ganzzahl	100	"" statt 0
Date RFC 1123	Lange Ganzzahl	10	Fri, 10 Sep 2010 13:07:20 GMT
Internal date abbreviated	Lange Ganzzahl	6	29. Dez. 2006
Internal date long	Lange Ganzzahl	5	29. Dezember 2006
Internal date short	Lange Ganzzahl	7	29.12.2006
Internal date short special	Lange Ganzzahl	4	29.12.06 (aber 29.12.1896 oder 29.12.2096)
ISO Date	Lange Ganzzahl	8	2006-29-12T00:00:00 (überholt)
ISO Date GMT	Lange Ganzzahl	9	2010-09-13T16:11:53Z
System date abbreviated	Lange Ganzzahl	2	So, 29. Dez. 2006
System date long	Lange Ganzzahl	3	Sonntag, 29. Dezember 2006
System date short	Lange Ganzzahl	1	29.12.2006

Digest Typ

Konstante	Typ	Wert	Kommentar
4D digest	Lange Ganzzahl	2	Interner Algorithmus von 4D. Dient zum Verschlüsseln von Benutzerkennwörtern. Dieser Algorithmus ist besonders im Rahmen der **Datenbankmethode On 4D Mobile Authentication** hilfreich, wenn Sie Ihre eigene Benutzerliste verwenden wollen.
MD5 digest	Lange Ganzzahl	0	*Message Digest 5* Algorithmus. Das ist eine Serie von 128 bits, zurückgegeben als String mit 32 hexadezimalen Zeichen.
SHA1 digest	Lange Ganzzahl	1	*Secure Hash 1* Algorithmus. Das ist eine Serie von 160 bits, zurückgegeben als String mit 40 hexadezimalen Zeichen.
SHA256 digest	Lange Ganzzahl	3	(SHA-2 Familie) SHA-256 ist eine Serie von 256 bits, zurückgegeben als String mit 64 hexadezimalen Zeichen.
SHA512 digest	Lange Ganzzahl	4	(SHA-2 Familie) SHA-512 ist eine Serie von 512 bits, zurückgegeben als String mit 128 hexadezimalen Zeichen.

Druckoptionen

Konstante	Typ	Wert	Kommentar
Color option	Lange Ganzzahl	8	(nur Windows) nur *Wert1*: Code zum Verwalten der Farbe: 1=schwarz/weiß (monochrome), 2=Farbe. **64-bit Versionen**: Diese Option wird in 4D 64-bit Versionen nicht unterstützt (überholt)
Destination option	Lange Ganzzahl	9	*Wert1*: Code für Druckausgabe: 1=Drucker, 2=(PC)/PS File (Mac), 3=PDF Datei, 5=Bildschirm (OS X Treiber). Ist *Wert1* ungleich 1 oder 5, enthält *Wert2* den Pfadnamen des Ergebnisdokuments. Dieser Pfad wird benützt, bis ein anderer Pfad angegeben wird. Gibt es bereits eine gleichnamige Datei am Zielort, wird sie ersetzt. Ist der aktuelle Wert nicht in der vordefinierten Liste, enthält *Wert1* mit **GET PRINT OPTION** -1 und die Systemvariable OK wird auf 1 gesetzt. Tritt ein Fehler auf, werden *Wert1* und die Systemvariable OK auf 0 gesetzt. **Hinweis**: Unter Windows können Sie das Druckziel auf 3 (PDF-Datei) setzen, wenn der Treiber PDF Creator installiert ist. Werden die Werte (9;3;Pfad) übergeben, startet 4D automatisch ein "stilles" PDF Drucken, das alle übergebenen Codes für Option berücksichtigt. Übergeben Sie in *Wert2* einen leeren String oder lassen diesen Parameter weg, erscheint beim Drucken ein Sichern-Dialog. Nach dem Drucken wird wieder auf die aktuellen Einstellungen zurückgesetzt. Das vereinfacht das Steuern von Drucken als PDF für 4D und ermöglicht das Schreiben von Multiplattform Code. Sind die Werte (9;3;Pfad) nicht übergeben, wird das Drucken nicht durch 4D gesteuert und alle für PDF Creator übergebenen Optionen werden ignoriert.
Double sided option	Lange Ganzzahl	11	(nur Windows) *Wert1*: 0=Einseitig oder Standard), 1=Doppelseitig. Ist *Wert1*=1, enthält *Wert2* die Bindung: 0=Bindung links (Standard), 1=Bindung oben. **Hinweis:** Diese Option ist nur unter Windows verwendbar.
Generic PDF driver	Zeichenkette	_4d_pdf_printer	**Hinweis:** Diese Funktionalität ist in 32 bit Versionen von 4D nicht verfügbar - Setzt auf OS X den aktuellen Drucker auf den standardmäßigen Druckertreiber. Er ist nicht sichtbar und wird nicht von **PRINTERS LIST** zurückgegeben. Beachten Sie, dass über **SET PRINT OPTION** ein PDF Dokumentpfad gesetzt sein muss, andernfalls wird Fehler 3107 zurückgegeben. - Setzt unter Windows den aktuellen Drucker auf den Windows PDF Druckertreiber (genannt "Microsoft Print to PDF"). Diese Option ist nur unter Windows 10 mit installierter PDF Option verfügbar. In anderen Windows-Versionen oder wenn kein PDF Treiber verfügbar ist, führt der Befehl nichts aus und die Systemvariable OK wird auf 0 gesetzt.
Hide printing progress option	Lange Ganzzahl	14	(nur Mac) Nur *Wert1*: 1=Fenster mit dem Druckverlauf ausblenden, 0=Fenster mit dem Druckverlauf anzeigen (Standard). Diese Option ist besonders hilfreich beim Drucken von PDF auf OS X im Hintergrund. **Hinweis**: Es gibt bereits eine Option Druckverlauf im Dialogfenster Einstellungen (Seite **Anwendung/Optionen**). Diese gilt jedoch global für die Anwendung und blendet nicht alle Fenster unter Mac OS X aus.
Legacy printing layer option	Lange Ganzzahl	16	(nur 4D 64-bit Versionen für Windows) nur *Wert1*: 1=auf GDI basierende bisherige Druckebene für nachfolgende Druckaufträge auswählen. 0=die D2D Druckerebene verwenden (Standard) **64-bit Versionen**: Selector wird nur in 4D 64-bit Versionen für Windows (Einzelplatz) unterstützt: auf anderen Plattformen wird er ignoriert. Er dient hauptsächlich dazu, damit bisherige Plug-Ins in 4D Druckaufträgen in 4D 64-bit Anwendungen drucken können.
Mac spool file format option	Lange Ganzzahl	13	(nur Mac) nur *Wert1*: 0= Druckauftrag im PDF-Modus (Standardwert) 1=Druckauftrag im PostScript Modus **Hinweise:** - Diese Option hat keine Auswirkung unter Windows. - Auf Mac OS X wird standardmäßig im PDF-Modus gedruckt. Der PDF-Treiber unterstützt jedoch keine PICT-Bilder mit eingebundener PostScript Information – diese Bilder werden über vektororientierte Programme erstellt. Um dieses Problem zu vermeiden, können Sie über diese Option den Druckmodus für die aktuelle Sitzung auf Mac OS X verändern. Bedenken Sie, dass das Drucken im PostScript Modus unerwünschte Nebenwirkungen haben kann. **64-bit Versionen:** Diese Option wird nicht unterstützt; sie wird ersetzt durch die Option Generic PDF driver des Befehls **SET CURRENT PRINTER**.
Number of copies option	Lange Ganzzahl	4	nur *Wert1*: Anzahl der Kopien zum Drucken
Orientation option	Lange Ganzzahl	2	Nur *Wert1*: 1=Hochformat, 2=Querformat. Bei einer anderen Ausrichtung gibt **GET PRINT OPTION** 0 in *Wert1* zurück. **64-bit Versionen**: Diese Option lässt sich im Druckauftrag aufrufen, d.h. Sie können im gleichen Auftrag zwischen Hoch- und Querformat wechseln.
Page range option	Lange Ganzzahl	15	*Wert1*=erste Seite zum Drucken (Standardwert ist 1) und optional *Wert2*=letzte Seite zum Drucken (Standardwert -1 = Dokumentende)
Page setup dialog	Lange Ganzzahl	1	Dialogfenster Seite einrichten anzeigen
Paper option	Lange Ganzzahl	1	Verwenden Sie nur *Wert1*, enthält er den Namen des Papierformats. Verwenden Sie beide Parameter, enthält *Wert1* die Papierbreite und *Wert2* die Papierhöhe. Breite und Höhe werden in Pixel auf dem Bildschirm angegeben. Über den Befehl **PRINT OPTION VALUES** erhalten Sie Name, Höhe und Breite aller Papierformate, die der Drucker anbietet.

Konstante	Typ	Wert	Kommentar
Paper source option	Lange Ganzzahl	5	(nur Windows) nur *Wert1*: Nummer, die dem Index des zu verwendenden Papierschachts entspricht. Über den Befehl **PRINT OPTION VALUES** erhalten Sie das Array mit den Namen. Diese Option ist nur unter Windows verwendbar.
PDFCreator Printer name	Zeichenkette	PDFCreator	Anzeige des Druckernamens
Print dialog	Lange Ganzzahl	2	Dialogfenster Drucken anzeigen
Scale option	Lange Ganzzahl	3	nur *Wert1*: Skalierungswert in Prozent. Bedenken Sie jedoch, dass einige Drucker keine Skalierung zulassen. Übergeben Sie einen ungültigen Wert, wird die Eigenschaft beim Drucken auf 100% gesetzt.
Spooler document name option	Lange Ganzzahl	12	nur *Wert1*: Namen des aktuellen Druckdokuments, das in der Dokumentenliste des Druck-Servers erscheint. Der hier definierte Name wird für alle Druckdokumente der Sitzung verwendet, solange kein neuer Name oder ein leerer String übergeben wird. Um die Standardoperation wiederherzustellen (Methodenname statt Methode, Tabellenname für Datensatz, etc.), übergeben Sie in *Wert1* einen leeren String.

Ereignisse (Inhalt)

Konstante	Typ	Wert	Kommentar
Activate event	Lange Ganzzahl	8	
Auto key event	Lange Ganzzahl	5	
Disk event	Lange Ganzzahl	7	
Key down event	Lange Ganzzahl	3	
Key up event	Lange Ganzzahl	4	
Mouse down event	Lange Ganzzahl	1	
Mouse up event	Lange Ganzzahl	2	
Null event	Lange Ganzzahl	0	
Operating system event	Lange Ganzzahl	15	
Update event	Lange Ganzzahl	6	

Ereignisse (Zusatztasten)

Konstante	Typ	Wert	Kommentar
Activate window bit	Lange Ganzzahl	0	
Activate window mask	Lange Ganzzahl	1	
Caps lock key bit	Lange Ganzzahl	10	Windows und OS X
Caps lock key mask	Lange Ganzzahl	1024	Windows und OS X
Command key bit	Lange Ganzzahl	8	Ctrl Taste unter Windows, Befehlstaste auf OS X
Command key mask	Lange Ganzzahl	256	Strg-Taste unter Windows, Befehlstaste auf OS X
Control key bit	Lange Ganzzahl	12	Ctrl-Taste auf OS X, oder rechter Mausklick unter Windows und OS X
Control key mask	Lange Ganzzahl	4096	Ctrl-Taste auf OS X, oder rechter Mausklick unter Windows und OS X
Mouse button bit	Lange Ganzzahl	7	
Mouse button mask	Lange Ganzzahl	128	
Option key bit	Lange Ganzzahl	11	Alt Taste (Wahltaste unter OS X)
Option key mask	Lange Ganzzahl	2048	Windows = Alt-Taste, Mac OS = Wahltaste
Right control key bit	Lange Ganzzahl	15	
Right control key mask	Lange Ganzzahl	32768	
Right option key bit	Lange Ganzzahl	14	
Right option key mask	Lange Ganzzahl	16384	
Right shift key bit	Lange Ganzzahl	13	
Right shift key mask	Lange Ganzzahl	8192	
Shift key bit	Lange Ganzzahl	9	Windows und OS X
Shift key mask	Lange Ganzzahl	512	Windows und Mac OS

Euro Währungen

Konstante	Typ	Wert	Kommentar
Austrian Schilling	Zeichenkette	ATS	
Belgian Franc	Zeichenkette	BEF	
Deutsche Mark	Zeichenkette	DEM	
Euro	Zeichenkette	EUR	
Finnish Markka	Zeichenkette	FIM	
French Franc	Zeichenkette	FRF	
Greek Drachma	Zeichenkette	GRD	
Irish Pound	Zeichenkette	IEP	
Italian Lira	Zeichenkette	ITL	
Luxembourg Franc	Zeichenkette	LUF	
Netherlands Guilder	Zeichenkette	NLG	
Portuguese Escudo	Zeichenkette	PTE	
Spanish Peseta	Zeichenkette	ESP	

Farben

Konstante	Typ	Wert	Kommentar
Black	Lange Ganzzahl	15	
Blue	Lange Ganzzahl	6	
Brown	Lange Ganzzahl	13	
Dark blue	Lange Ganzzahl	5	
Dark brown	Lange Ganzzahl	10	
Dark green	Lange Ganzzahl	9	
Dark grey	Lange Ganzzahl	11	
Green	Lange Ganzzahl	8	
Grey	Lange Ganzzahl	14	
Light blue	Lange Ganzzahl	7	
Light grey	Lange Ganzzahl	12	
Orange	Lange Ganzzahl	2	
Purple	Lange Ganzzahl	4	
Red	Lange Ganzzahl	3	
White	Lange Ganzzahl	0	
Yellow	Lange Ganzzahl	1	

Feld und Variablentypen

Konstante	Typ	Wert	Kommentar
Array 2D	Lange Ganzzahl	13	
Blob array	Lange Ganzzahl	31	
Boolean array	Lange Ganzzahl	22	
Date array	Lange Ganzzahl	17	
Integer array	Lange Ganzzahl	15	
Is alpha field	Lange Ganzzahl	0	
Is BLOB	Lange Ganzzahl	30	
Is Boolean	Lange Ganzzahl	6	
Is collection	Lange Ganzzahl	42	
Is date	Lange Ganzzahl	4	
Is float	Lange Ganzzahl	35	
Is integer	Lange Ganzzahl	8	
Is integer 64 bits	Lange Ganzzahl	25	
Is longint	Lange Ganzzahl	9	
Is null	Lange Ganzzahl	255	
Is object	Lange Ganzzahl	38	
Is picture	Lange Ganzzahl	3	
Is pointer	Lange Ganzzahl	23	
Is real	Lange Ganzzahl	1	
Is string var	Lange Ganzzahl	24	
Is subtable	Lange Ganzzahl	7	
Is text	Lange Ganzzahl	2	
Is time	Lange Ganzzahl	11	
Is undefined	Lange Ganzzahl	5	
LongInt array	Lange Ganzzahl	16	
Object array	Lange Ganzzahl	39	
Picture array	Lange Ganzzahl	19	
Pointer array	Lange Ganzzahl	20	
Real array	Lange Ganzzahl	14	
String array	Lange Ganzzahl	21	
Text array	Lange Ganzzahl	18	
Time array	Lange Ganzzahl	32	

Fenster

Konstante	Typ	Wert	Kommentar
External window	Lange Ganzzahl	5	
Floating window	Lange Ganzzahl	14	
Modal dialog	Lange Ganzzahl	9	
Regular window	Lange Ganzzahl	8	
XY Current form	Lange Ganzzahl	1	Beginn ist linke Ecke oben des aktuellen Formulars
XY Current window	Lange Ganzzahl	2	Beginn ist linke Ecke oben des aktuellen Fensters
XY Main window	Lange Ganzzahl	4	Unter Windows: Beginn ist linke Ecke oben des Hauptfensters; auf OS X: genauso wie für XY Screen
XY Screen	Lange Ganzzahl	3	Beginn ist linke Ecke oben des Hauptbildschirms (genauso wie für den Befehl **SCREEN COORDINATES**)

Find window

Konstante	Typ	Wert	Kommentar
_o_In contents	Lange Ganzzahl	3	Plattform: Mac OS und Windows

Formular Objekttypen

Konstante	Typ	Wert	Kommentar
Object type 3D button	Lange Ganzzahl	16	
Object type 3D checkbox	Lange Ganzzahl	26	
Object type 3D radio button	Lange Ganzzahl	23	
Object type button grid	Lange Ganzzahl	20	
Object type checkbox	Lange Ganzzahl	25	
Object type combobox	Lange Ganzzahl	11	
Object type dial	Lange Ganzzahl	28	
Object type group	Lange Ganzzahl	21	
Object type groupbox	Lange Ganzzahl	30	
Object type hierarchical list	Lange Ganzzahl	6	
Object type hierarchical popup menu	Lange Ganzzahl	13	
Object type highlight button	Lange Ganzzahl	17	
Object type invisible button	Lange Ganzzahl	18	
Object type line	Lange Ganzzahl	32	
Object type listbox	Lange Ganzzahl	7	
Object type listbox column	Lange Ganzzahl	9	
Object type listbox footer	Lange Ganzzahl	10	
Object type listbox header	Lange Ganzzahl	8	
Object type matrix	Lange Ganzzahl	35	
Object type oval	Lange Ganzzahl	34	
Object type picture button	Lange Ganzzahl	19	
Object type picture input	Lange Ganzzahl	4	
Object type picture popup menu	Lange Ganzzahl	14	
Object type picture radio button	Lange Ganzzahl	24	
Object type plugin area	Lange Ganzzahl	38	
Object type popup dropdown list	Lange Ganzzahl	12	
Object type progress indicator	Lange Ganzzahl	27	
Object type push button	Lange Ganzzahl	15	
Object type radio button	Lange Ganzzahl	22	
Object type radio button field	Lange Ganzzahl	5	
Object type rectangle	Lange Ganzzahl	31	
Object type rounded rectangle	Lange Ganzzahl	33	
Object type ruler	Lange Ganzzahl	29	
Object type splitter	Lange Ganzzahl	36	
Object type static picture	Lange Ganzzahl	2	
Object type static text	Lange Ganzzahl	1	
Object type subform	Lange Ganzzahl	39	
Object type tab control	Lange Ganzzahl	37	
Object type text input	Lange Ganzzahl	3	
Object type unknown	Lange Ganzzahl	0	
Object type view pro area	Lange Ganzzahl	42	
Object type web area	Lange Ganzzahl	40	
Object type write pro area	Lange Ganzzahl	41	

Formularbereich

Konstante	Typ	Wert	Kommentar
Form break0	Lange Ganzzahl	300	
Form break1	Lange Ganzzahl	301	
Form break2	Lange Ganzzahl	302	
Form break3	Lange Ganzzahl	303	
Form break4	Lange Ganzzahl	304	
Form break5	Lange Ganzzahl	305	
Form break6	Lange Ganzzahl	306	
Form break7	Lange Ganzzahl	307	
Form break8	Lange Ganzzahl	308	
Form break9	Lange Ganzzahl	309	
Form detail	Lange Ganzzahl	0	
Form footer	Lange Ganzzahl	100	
Form header	Lange Ganzzahl	200	
Form header1	Lange Ganzzahl	201	
Form header10	Lange Ganzzahl	210	
Form header2	Lange Ganzzahl	202	
Form header3	Lange Ganzzahl	203	
Form header4	Lange Ganzzahl	204	
Form header5	Lange Ganzzahl	205	
Form header6	Lange Ganzzahl	206	
Form header7	Lange Ganzzahl	207	
Form header8	Lange Ganzzahl	208	
Form header9	Lange Ganzzahl	209	

Formularereignisse

Konstante	Typ	Wert	Kommentar
_o_On Mac toolbar button	Lange Ganzzahl	55	*** Konstante überholt ***
On Activate	Lange Ganzzahl	11	Das Formularfenster wird zum vordersten Fenster.
On After Edit	Lange Ganzzahl	45	Der Inhalt des eingebbaren Objekts mit Fokus wurde gerade geändert.
On After Keystroke	Lange Ganzzahl	28	Ein Zeichen wird gerade in das Objekt mit Fokus eingegeben. **Get edited text** gibt den Text im Objekt inkl. diesem Zeichen zurück
On After Sort	Lange Ganzzahl	30	(*nur Listbox*) In einer Spalte der Listbox wurde gerade eine Standard-Sortierung ausgeführt.
On Alternative Click	Lange Ganzzahl	38	• *3D Schaltflächen:* Der Pfeilbereich einer 3D Schaltfläche ist angeklickt • Listboxen: In einer Spalte eines Objekt Arrays ist eine Schaltfläche [...] angeklickt (Attribut "alternateButton")
On Before Data Entry	Lange Ganzzahl	41	(*nur Listbox*) Eine Zelle der Listbox wechselt gerade in den Editiermodus
On Before Keystroke	Lange Ganzzahl	17	Ein Zeichen wird gerade in das Objekt mit Fokus eingegeben. **Get edited text** gibt den Text im Objekt ohne dieses Zeichen zurück.
On Begin Drag Over	Lange Ganzzahl	46	Ein Objekt wird gerade bewegt (Drag)
On Begin URL Loading	Lange Ganzzahl	47	(*nur Web Areas*) Eine neue URL wird in den Webbereich geladen.
On bound variable change	Lange Ganzzahl	54	Die dem Unterformular zugewiesene Variable wird geändert.
On Clicked	Lange Ganzzahl	4	Das Objekt wurde angeklickt.
On Close Box	Lange Ganzzahl	22	Die Schließbox des Fensters wurde angeklickt.
On Close Detail	Lange Ganzzahl	26	Sie haben das Eingabeformular verlassen und gehen zurück zum Ausgabeformular.
On Collapse	Lange Ganzzahl	44	(*hierarchische Listen und hierarchische Listboxen*) Ein Element der hierarchischen Liste bzw. Listbox wurde über Mausklick oder Tastenanschlag zugeklappt.
On Column Moved	Lange Ganzzahl	32	(*nur Listbox*) Der Benutzer hat eine Spalte der Listbox per Drag and Drop bewegt.
On Column Resize	Lange Ganzzahl	33	(*nur Listbox*) Der Benutzer hat die Breite einer Spalte der Listbox mit der Maus geändert.
On Data Change	Lange Ganzzahl	20	Daten im Objekt wurden geändert.
On Deactivate	Lange Ganzzahl	12	Das Formularfenster ist nicht mehr das vorderste Fenster.
On Delete Action	Lange Ganzzahl	58	(*nur hierarchische Listen und Listboxen*) Ein Benutzer möchte ein Element löschen
On Display Detail	Lange Ganzzahl	8	Ein Datensatz wird gleich in einer Liste bzw. eine Zeile in einer Listbox angezeigt.
On Double Clicked	Lange Ganzzahl	13	Auf ein Objekt wurde ein Doppelklick ausgeführt.
On Drag Over	Lange Ganzzahl	21	Daten werden in ein Objekt gezogen.
On Drop	Lange Ganzzahl	16	Daten werden in ein Objekt gezogen.
On End URL Loading	Lange Ganzzahl	49	(*nur Web Areas*) Alle Ressourcen des URL wurden geladen.
On Expand	Lange Ganzzahl	43	(*hierarchische Listen und hierarchische Listboxen*) Ein Element der hierarchischen Liste bzw. Listbox wurde per Mausklick oder Tastenanschlag aufgeklappt.
On Footer Click	Lange Ganzzahl	57	(*nur Listboxen*) Der Fußteil einer Listbox oder einer Spalte der Listbox ist angeklickt
On Getting Focus	Lange Ganzzahl	15	Ein Formularobjekt erhält den Fokus.
On Header	Lange Ganzzahl	5	Der Kopfteil des Formulars wird gleich gedruckt oder angezeigt.
On Header Click	Lange Ganzzahl	42	(*nur Listbox*) Ein Spaltentitel der Listbox wird angeklickt.
On Load	Lange Ganzzahl	1	Das Formular wird gleich angezeigt oder gedruckt.
On Load Record	Lange Ganzzahl	40	Bei der Eingabe in die Liste, wird ein Datensatz während der Änderung geladen (Der Benutzer klickt auf eine Zeile im Datensatz und ein Feld wechselt in den Editiermodus).
On Long Click	Lange Ganzzahl	39	(*nur 3D buttons*) Eine 3D Schaltfläche wird angeklickt und die Maustaste bleibt für eine gewisse Zeit gedrückt.
On Losing Focus	Lange Ganzzahl	14	Ein Formularobjekt verliert den Fokus.

Konstante	Typ	Wert	Kommentar
On Menu Selected	Lange Ganzzahl	18	Ein Menüeintrag wurde ausgewählt.
On Mouse Enter	Lange Ganzzahl	35	Der Mauszeiger geht in den grafischen Bereich eines Objekts.
On Mouse Leave	Lange Ganzzahl	36	Der Mauszeiger verlässt den grafischen Bereich eines Objekts.
On Mouse Move	Lange Ganzzahl	37	Der Mauszeiger bewegt sich (mindestens 1 Pixel) ODER eine Modifier-Taste (Shift, Alt, Shift Lock) wurde gedrückt. Wurde das Ereignis nur für ein Objekt markiert, wird es nur generiert, wenn der Cursor im grafischen Bereich eines Objekts liegt.
On Mouse Up	Lange Ganzzahl	2	(*nur Bilder*) Der Benutzer hat gerade in einem Bildobjekt die linke Maustaste losgelassen
On Open Detail	Lange Ganzzahl	25	Ein dem Ausgabeformular oder der Listbox zugeordnetes Eingabeformular wird gerade geöffnet.
On Open External Link	Lange Ganzzahl	52	(*nur Web Areas*) Im Browser wurde eine externe URL geöffnet.
On Outside Call	Lange Ganzzahl	10	Das Formular hat einen Aufruf **POST OUTSIDE CALL** erhalten.
On Page Change	Lange Ganzzahl	56	Aktuelle Seite im Formular wird geändert
On Plug in Area	Lange Ganzzahl	19	Ein externes Objekt hat angefragt, seine Objektmethode auszuführen.
On Printing Break	Lange Ganzzahl	6	Ein Umbruchbereich im Formular wird gleich gedruckt.
On Printing Detail	Lange Ganzzahl	23	Der Detailbereich des Formulars wird gleich gedruckt.
On Printing Footer	Lange Ganzzahl	7	Der Fußteil des Formulars wird gleich gedruckt.
On Resize	Lange Ganzzahl	29	Das Formularfenster wird angepasst.
On Row Moved	Lange Ganzzahl	34	(*nur Listbox*) Der Benutzer hat eine Zeile der Listbox per Drag-and-Drop bewegt.
On Scroll	Lange Ganzzahl	59	Der Benutzer scrollt den Inhalt eines Feldes vom Typ Bild oder Variable mit der Maus oder Tastatur.
On Selection Change	Lange Ganzzahl	31	• *Listbox*: Die aktuelle Auswahl der Zeilen oder Spalten wurde geändert. • *Datensätze in Liste*: Der aktuelle Datensatz oder die aktuelle Auswahl der Zeilen in einem Listen- bzw. Unterformular wurde geändert. • *Hierarchische Liste*: Die Auswahl in der Liste wurde nach einem Mausklick oder Tastenanschlag geändert. • *Eingebbares Feld oder Variable*: Die Textauswahl oder Position des Cursors im Bereich wurde nach einem Mausklick oder Tastenanschlag geändert.
On Timer	Lange Ganzzahl	27	Die Anzahl der durch **SET TIMER** definierten Ticks wurde überschritten.
On Unload	Lange Ganzzahl	24	Das Formular wird gerade verlassen oder erneuert.
On URL Filtering	Lange Ganzzahl	51	(*nur Web Areas*) Der Web Bereich hat eine URL geblockt.
On URL Loading Error	Lange Ganzzahl	50	(*nur Web Areas*) Beim Laden der URL ist ein Fehler aufgetreten.
On URL Resource Loading	Lange Ganzzahl	48	(*nur Web Areas*) Eine neue Ressource wird in den Web Bereich geladen.
On Validate	Lange Ganzzahl	3	Die Eingabe in den Datensatz wurde bestätigt.
On VP Ready	Lange Ganzzahl	9	(*nur 4D View Pro Areas*) Laden des 4D View Pro Bereichs ist komplett
On Window Opening Denied	Lange Ganzzahl	53	(*nur Web Areas*) Ein PopUp-Fenster wurde blockiert.

Formularobjekte (Eigenschaften)

Konstante	Typ	Wert	Kommentar
Align bottom	Lange Ganzzahl	4	
Align center	Lange Ganzzahl	3	
Align default	Lange Ganzzahl	1	
Align left	Lange Ganzzahl	2	
Align right	Lange Ganzzahl	4	
Align top	Lange Ganzzahl	2	
Asynchronous progress bar	Lange Ganzzahl	3	Drehendes Rad
Barber shop	Lange Ganzzahl	2	Animierter Balken
Border Dotted	Lange Ganzzahl	2	Objekte werden mit einer gepunkteten Linie von einem Punkt umrahmt
Border Double	Lange Ganzzahl	5	Objekte werden mit Doppellinie umrahmt, z.B. zwei kontinuierliche Linien von 1 Punkt mit 1 Pixel Abstand
Border None	Lange Ganzzahl	0	Objekte erscheinen ohne Rahmen
Border Plain	Lange Ganzzahl	1	Objekte werden mit einer kontinuierlichen Linie von 1 Punkt umrahmt
Border Raised	Lange Ganzzahl	3	Objekte werden mit einem erhobenen 3D Effekt umrahmt
Border Sunken	Lange Ganzzahl	4	Objekte werden mit einem vertieften 3D Effekt umrahmt
Border System	Lange Ganzzahl	6	Die Rahmenlinie wird gemäß der grafischen Spezifikation des Systems gezeichnet
Choice list	Lange Ganzzahl	0	Liste zum Auswählen von Werten (Option "Auswahlliste" in der Eigenschaftenliste) (Standard)
Disable events others unchanged	Lange Ganzzahl	2	Alle im *arrEreignisse* aufgeführten Ereignisse werden deaktiviert; der Status anderer Ereignisse bleibt unverändert
Enable events disable others	Lange Ganzzahl	0	Alle im *arrEreignisse* aufgeführten Ereignisse werden aktiviert; alle anderen werden deaktiviert
Enable events others unchanged	Lange Ganzzahl	1	Alle im *arrEreignisse* aufgeführten Ereignisse werden aktiviert; der Status anderer Ereignisse bleibt unverändert
Excluded list	Lange Ganzzahl	2	Liste mit ausgeschlossenen Werten für die Eingabe (Option "Ausgenommen-Liste" in der Eigenschaftenliste)
Multiline Auto	Lange Ganzzahl	0	In einzeiligen Bereichen werden Wörter am Zeilenende abgeschnitten und keine Zeilenumbrüche gesetzt. In mehrzeiligen Bereichen führt 4D automatisch Zeilenumbrüche aus.
Multiline No	Lange Ganzzahl	2	Es gibt nie Zeilenumbrüche: Der Text erscheint immer als eine Zeile. Bei Alpha oder Textfeldern bzw. Variablen mit Zeilenumbrüchen wird der Text nach dem ersten Zeilenumbruch entfernt, sobald der Text geändert wird.
Multiline Yes	Lange Ganzzahl	1	In einzeiligen Bereichen erscheint der Text bis zum ersten Zeilenumbruch oder bis zum letzten Wort, das sich ganz anzeigen lässt. 4D fügt Zeilenumbrüche ein; über die Pfeiltaste nach unten lässt sich im Inhalt scrollen. In mehrzeiligen Bereichen führt 4D automatische Zeilenumbrüche aus.
Orientation 0°	Lange Ganzzahl	0	Keine Rotation (Standardwert)
Orientation 180°	Lange Ganzzahl	180	Text um 180° im Uhrzeigersinn drehen
Orientation 90° left	Lange Ganzzahl	270	Text um 90° gegen den Uhrzeigersinn drehen
Orientation 90° right	Lange Ganzzahl	90	Text um 90° im Uhrzeigersinn drehen
Print Frame fixed with multiple records	Lange Ganzzahl	2	Der Rahmen behält die gleiche Größe, 4D druckt das Formular jedoch mehrmals, um alle Datensätze einzuschließen.
Print Frame fixed with truncation	Lange Ganzzahl	1	4D druckt nur die Datensätze, die in den Bereich des Unterformulars passen. Das Formular wird nur einmal gedruckt und die nicht gedruckten Datensätze werden ignoriert.
Progress bar	Lange Ganzzahl	1	Standard Ablaufbalken
Required list	Lange Ganzzahl	1	Liste mit erforderlichen Werten für die Eingabe (Option "Erforderlich-Liste" in der Eigenschaftenliste)
Resize horizontal grow	Lange Ganzzahl	1	Wächst das Fenster um 50% in der Breite, wird das Objekt um 50% nach rechts verbreitert
Resize horizontal move	Lange Ganzzahl	2	Wächst das Fenster um 100 Pixel in der Breite, wird das Objekt um 100 Pixel nach rechts bewegt

Konstante	Typ	Wert	Kommentar
Resize horizontal none	Lange Ganzzahl	0	Wird das Fenster verbreitert, bleiben Breite und Position des Objekts unverändert
Resize vertical grow	Lange Ganzzahl	1	Wächst das Fenster um 50% in der Höhe, wird das Objekt um 50% nach unten verlängert
Resize vertical move	Lange Ganzzahl	2	Wächst das Fenster um 100 Pixel in der Höhe, wird das Objekt um 100 Pixel nach unten verlängert
Resize vertical none	Lange Ganzzahl	0	Wird das Fenster verlängert, bleiben Höhe und Position des Objekts unverändert

Formularobjekte (Zugriff)

Konstante	Typ	Wert	Kommentar
Form all pages	Lange Ganzzahl	2	gibt alle Objekte von allen Seiten ohne vererbte Objekte zurück
Form current page	Lange Ganzzahl	1	Gibt alle Objekte der aktuellen Seite zurück, einschließlich der Seite 0, aber ohne vererbte Objekte
Form inherited	Lange Ganzzahl	4	Gibt nur die vererbten Objekte zurück
Object current	Lange Ganzzahl	0	
Object first in entry order	Zeichenkette	;FirstObject	
Object named	Lange Ganzzahl	3	
Object subform container	Lange Ganzzahl	2	
Object with focus	Lange Ganzzahl	1	

Formularoptionen

Konstante	Typ	Wert	Kommentar
Multiple selection	Lange Ganzzahl	2	Der Benutzer kann mehrere Datensätze auf einmal auswählen. Für zusammenhängende Datensätze den ersten Datensatz anklicken und mit gedrückter Umschalttaste den letzten Datensatz für die Auswahl anklicken. Für nicht zusammenhängende Datensätze jeden Datensatz einzeln anklicken, unter Windows mit gedrückter **Strg-Taste**, auf Mac OS mit gedrückter **Befehlstaste**.
No selection	Lange Ganzzahl	0	In der Liste lässt sich kein Datensatz auswählen.
NonInverted objects	Lange Ganzzahl	0	
Single selection	Lange Ganzzahl	1	Nur ein Datensatz ist auf einmal auswählbar.

Formulas

Konstante	Typ	Wert	Kommentar
Formula in with virtual structure	Zeichenkette	1	Formel enthält eigene (virtuelle) Namen). Die zurückgegebene Formel enthält standardmäßig reale Namen.
Formula out with tokens	Zeichenkette	4	Die zurückgegebene Formel muss Text Tokens enthalten (z.B. :Cxx).
Formula out with virtual structure	Zeichenkette	2	Die zurückgegebene Formel muss eigene (virtuelle) Namen enthalten.

Funktionstasten

Konstante	Typ	Wert	Kommentar
Backspace key	Lange Ganzzahl	8	
Down arrow key	Lange Ganzzahl	31	
End key	Lange Ganzzahl	4	
Enter key	Lange Ganzzahl	3	
Escape key	Lange Ganzzahl	27	
F1 key	Lange Ganzzahl	-122	
F10 key	Lange Ganzzahl	-109	
F11 key	Lange Ganzzahl	-103	
F12 key	Lange Ganzzahl	-111	
F13 key	Lange Ganzzahl	-105	
F14 key	Lange Ganzzahl	-107	
F15 key	Lange Ganzzahl	-113	
F2 key	Lange Ganzzahl	-120	
F3 key	Lange Ganzzahl	-99	
F4 key	Lange Ganzzahl	-118	
F5 key	Lange Ganzzahl	-96	
F6 key	Lange Ganzzahl	-97	
F7 key	Lange Ganzzahl	-98	
F8 key	Lange Ganzzahl	-100	
F9 key	Lange Ganzzahl	-101	
Help key	Lange Ganzzahl	5	
Home key	Lange Ganzzahl	1	
Left arrow key	Lange Ganzzahl	28	
Page down key	Lange Ganzzahl	12	
Page up key	Lange Ganzzahl	11	
Return key	Lange Ganzzahl	13	
Right arrow key	Lange Ganzzahl	29	
Tab key	Lange Ganzzahl	9	
Up arrow key	Lange Ganzzahl	30	

Graph Parameter

Konstante	Typ	Wert	Kommentar
Graph background color	Zeichenkette	graphBackgroundColor	**Mögliche Werte:** Zu SVG passende Farbangabe (Text), z.B. "#7F8E00", "Pink", oder "#0a1414"
Graph background opacity	Zeichenkette	graphBackgroundOpacity	**Mögliche Werte:** Ganzzahl, Bereich 0-100 **Standardwert:** 100
Graph background shadow color	Zeichenkette	graphBackgroundShadowColor	**Mögliche Werte**: Zu SVG passende Farbangabe (Text), z.B. "#7F8E00", "Pink", oder "#0a1414"
Graph bottom margin	Zeichenkette	bottomMargin	**Mögliche Werte**: Zahl **Standardwert**: 12
Graph colors	Zeichenkette	colors	**Mögliche Werte**: Text Array. Farben für jede Diagrammreihe. **Standardwerte**: Blaugrün (#19BAC9), Gelb (#FFC338), Purpur (#573E82), Grün (#4FA839), Orange (#D95700), Blau (#1D9DF2), Gelbgrün (#B5CF32), Rot (#D43A26)
Graph column gap	Zeichenkette	columnGap	**Mögliche Werte**: Lange Ganzzahl **Standardwert**: 12 Setzt den Abstand zwischen den Balken Nur Typ 1, 2, 3
Graph column width max	Zeichenkette	columnWidthMax	**Mögliche Werte**: Zahl **Standardwert**: 200 Nur Typ 1, 2, 3
Graph column width min	Zeichenkette	columnWidthMin	**Mögliche Werte**: Zahl **Standardwert**: 10 Nur Typ 1, 2, 3
Graph default height	Zeichenkette	defaultHeight	**Mögliche Werte**: Zahl **Standardwert:** 400, bei Diagrammtyp=7 (Kreis) ist der Standardwert 600
Graph default width	Zeichenkette	defaultWidth	**Mögliche Werte**: Zahl **Standardwert:** 600. Bei Diagrammtyp=7 (Kreis) ist der Standardwert 800
Graph display legend	Zeichenkette	displayLegend	**Mögliche Werte**: Boolean **Standardwert:** wahr
Graph document background color	Zeichenkette	documentBackgroundColor	**Mögliche Werte**: Zu SVG passende Farbangabe (Text), z.B. "#7F8E00", "Pink" oder "#0a1414". Wird ein als SVG Bild gesichertes Diagramm anderswo geöffnet, erscheint die Hintergrundfarbe des Dokuments nur, wenn die SVG Rendering Engine die *SVG Norm tiny 1.2* unterstützt (auf IE, Firefox, aber nicht auf Chrome).
Graph document background opacity	Zeichenkette	documentBackgroundOpacity	**Mögliche Werte**: Ganzzahl, Bereich 0-100 (Standardwert: 100). Wird ein als SVG Bild gesichertes Diagramm anderswo geöffnet, erscheint die Hintergrunddichte des Dokuments nur, wenn die SVG Rendering Engine die *SVG Norm tiny 1.2* unterstützt (auf IE, Firefox, aber nicht auf Chrome).
Graph font color	Zeichenkette	fontColor	**Mögliche Werte:** Zu SVG passende Farbangabe (Text), z.B. "#7F8E00", "Pink" oder "#0a1414"
Graph font size	Zeichenkette	fontSize	**Mögliche Werte**: Lange Ganzzahl **Standardwert**: 12. Ist Diagrammtyp=7 (Kreis), siehe Graph pie font size
Graph left margin	Zeichenkette	leftMargin	**Mögliche Werte**: Zahl **Standardwert**: 12
Graph legend font color	Zeichenkette	legendFontColor	**Mögliche Werte**: Zu SVG passende Farbangabe (Text), z.B. "#7F8E00", "Pink" oder "#0a1414"
Graph legend icon gap	Zeichenkette	legendIconGap	**Mögliche Werte**: Zahl **Standardwert**: Graph legend icon height/2
Graph legend icon height	Zeichenkette	legendIconHeight	**Mögliche Werte**: Zahl **Standardwert**: 20
Graph legend icon width	Zeichenkette	legendIconWidth	**Mögliche Werte**: Zahl **Standardwert**: 20
Graph legend labels	Zeichenkette	legendLabels	**Mögliche Werte**: Text Array. Ohne Angabe zeigt 4D Icons ohne Text.
Graph line width	Zeichenkette	lineWidth	**Mögliche Werte**: Zahl **Standardwert**: 2 Nur Typ 4
Graph pie direction	Zeichenkette	pieDirection	**Mögliche Werte**: 1 oder -1 **Standardwert**: 1 1 gbit die Richtung im Uhrzeigersinn an, -1 die Richtung gegen den Uhrzeigersinn

Konstante	Typ	Wert	Kommentar
Graph pie font size	Zeichenkette	pieFontSize	**Mögliche Werte**: Zahl **Standardwert**: 16 Nur Typ 7
Graph pie shift	Zeichenkette	pieShift	**Mögliche Werte**: Zahl **Standardwert**: 8 Nur Typ 7
Graph pie start angle	Zeichenkette	pieStartAngle	**Mögliche Werte**: Zahl (positiv oder negativ) **Standardwert**: 0 ist ein Startwinkel von 0° (nach oben zeigende Position) Ein positiver Wert ist ein Winkel in Bezug auf die aktuelle Richtung des Tortendiagramms. Ein negativer Wert ist ein Winkel in Bezug auf die Gegenrichtung des Tortendiagramms.
Graph plot height	Zeichenkette	plotHeight	**Mögliche Werte**: Zahl **Standardwert**: 12 Nur Typ 4
Graph plot radius	Zeichenkette	plotRadius	**Mögliche Werte**: Zahl **Standardwert**: 12 Nur Typ 6
Graph plot width	Zeichenkette	plotWidth	**Mögliche Werte**: Zahl **Standardwert**: 12 Nur Typ 4
Graph right margin	Zeichenkette	rightMargin	**Mögliche Werte**: Zahl **Standardwert**: 2
Graph top margin	Zeichenkette	topMargin	**Mögliche Werte**: Zahl **Standardwert**: 2
Graph type	Zeichenkette	graphType	**Mögliche Werte**: Lange Ganzzahl [1 bis 8]: 1 = Säulen, 2 = proportional, 3 = gestapelt, 4 =Linien, 5 = Flächen, 6 = Punkte, 7 = Kreis, 8 = Bilder. **Standardwert**: 1 Bei Null wird das Diagramm nicht gezeichnet und es erscheint **keine** Fehlermeldung. Ist der Diagrammtyp außerhalb des Bereichs, wird das Diagramm nicht gezeichnet und es erscheint eine Fehlermeldung. Für Diagramme vom Typ Bild (Wert= 8) müssen die verwendeten Bilder in folgendem Ordner liegen: 4D/Resources/GraphTemplates/Graph_8_Pictures/. Es werden lokale Bilddateien anstelle von 4D Dateien verwendet. Es gibt kein Muster für Bildnamen; 4D sortiert die im Ordner enthaltenen Dateien und weist die erste Datei dem ersten Diagramm zu. Die Dateien können vom Typ SVG oder Bild sein.
Graph xGrid	Zeichenkette	xGrid	**Mögliche Werte:** Boolean **Standardwert**: Wahr Für alle Typen außer 7 verwendbar
Graph xMax	Zeichenkette	xMax	**Mögliche Werte**: Zahl, Datum, Zeit (gleicher Typ wie Parameter *xBeschriftung*). Nur Werte kleiner als xMax erscheinen im Diagramm. xMax wird nur für die Diagrammtypen 4, 5 oder 6 verwendet, wenn xProp=wahr und *xBeschriftung* vom Typ Zahl, Datum oder Zeit ist. Ohne Angabe oder wenn xMin>xMax berechnet 4D automatisch den Wert xMax.
Graph xMin	Zeichenkette	xMin	**Mögliche Werte**: Zahl, Datum, Zeit (gleicher Typ wie Parameter *xBeschriftung*). Nur Werte höher als xMin werden im Diagramm angezeigt. xMin wird nur für die Diagrammtypen 4, 5 oder 6 verwendet, wenn xProp=wahr und *xBeschriftung* vom Typ Zahl, Datum oder Zeit. Ohne Angabe oder wenn xMin>xMax berechnet 4D automatisch den Wert xMin.
Graph xProp	Zeichenkette	xProp	**Mögliche Werte**: Boolean **Standardwert**: Falsch Wahr für proportionale x-Achse; Falsch für normale x-Achse. xProp wird nur für die Diagrammtypen 4, 5 oder 6 verwendet.
Graph yGrid	Zeichenkette	yGrid	**Mögliche Werte**: Boolean **Standardwert**: Wahr Für alle Typen außer 7
Graph yMax	Zeichenkette	yMax	**Mögliche Werte**: Zahlen Ohne Angabe berechnet 4D automatisch den Wert yMax. Für alle Typen außer 7
Graph yMin	Zeichenkette	yMin	**Mögliche Werte**: Zahlen Ohne Angabe berechnet 4D automatisch den Wert yMin. Für alle Typen außer 7

Hierarchische Listen

Konstante	Typ	Wert	Kommentar
_o_Macintosh node	Lange Ganzzahl	860	
_o_Use PICT resource	Lange Ganzzahl	65536	
_o_Windows node	Lange Ganzzahl	138	
Additional text	Zeichenkette	4D_additional_text	Diese Konstante fügt auf der rechten Seite von *EintragRef* Text hinzu. Dieser Zusatztext erscheint immer im rechten Teil der Liste, selbst wenn der Benutzer den horizontal scrollenden Cursor bewegt. In *Wert* übergeben Sie den anzuzeigenden Text für diese Konstante.
Associated standard action	Zeichenkette	4D_standard_action_name	Diese Konstante weist *EintragRef* eine Standardaktion zu. In diesem Fall müssen Sie im Parameter *Wert* den Namen einer Standardaktion mit einem Parameter übergeben, z.B. "fontSize?value=10pt". Weitere Informationen dazu finden Sie im Abschnitt **Standardaktionen** des Handbuchs *4D Designmodus*.
Use PicRef	Lange Ganzzahl	131072	

HTTP Client

Konstante	Typ	Wert	Kommentar
HTTP basic	Lange Ganzzahl	1	Die Authentifizierungsmethode BASIC verwenden
HTTP compression	Lange Ganzzahl	6	Wert = 0 (nicht komprimieren) oder 1 (komprimieren), Standardwert: 0 Diese Option aktiviert oder deaktiviert das Komprimieren, um den Austausch für Anfragen zwischen Client und Server zu beschleunigen. Ist es aktiviert, verwendet der HTTP Client je nach der Server Antwort die Deflate oder gzip Komprimierung.
HTTP DELETE method	Zeichenkette	DELETE	Siehe unter RFC 2616
HTTP digest	Lange Ganzzahl	2	Die Authentifizierungsmethode DIGEST verwenden
HTTP display auth dial	Lange Ganzzahl	4	Wert = 0 (Dialogfenster nicht anzeigen) oder 1 (Dialogfenster anzeigen). Standardwert: 0 Diese Option zeigt den Authentifizierungsdialog beim Ausführen der Funktion **HTTP Get** oder **HTTP Request** an. Standardmäßig wird dieses Dialogfenster nie angezeigt und Sie müssen den Befehl **HTTP AUTHENTICATE** verwenden. Soll der Authentifizierungsdialog jedoch erscheinen, damit Benutzer ihre Daten zur Identifizierung eingeben können, übergeben Sie 1 in Wert. Das Dialogfenster erscheint nur, wenn für die Anfrage eine Authentifizierung erforderlich ist
HTTP follow redirect	Lange Ganzzahl	2	Wert = 0 (keine Redirektion zulassen) oder 1 (Redirektion zulassen). Standardwert = 1
HTTP GET method	Zeichenkette	GET	Siehe unter RFC 2616. Entspricht der Verwendung der Funktion **HTTP Get**.
HTTP HEAD method	Zeichenkette	HEAD	Siehe unter RFC 2616
HTTP max redirect	Lange Ganzzahl	3	Wert = Maximale Anzahl der zugelassenen Redirektionen. Standardwert = 2
HTTP OPTIONS method	Zeichenkette	OPTIONS	Siehe unter RFC 2616
HTTP POST method	Zeichenkette	POST	Siehe unter RFC 2616
HTTP PUT method	Zeichenkette	PUT	Siehe unter RFC 2616
HTTP reset auth settings	Lange Ganzzahl	5	Wert = 0 (Information nicht löschen) oder 1 (Information löschen). Standardwert: 0 Diese Option weist 4D an, die Authentifizierungsdaten des Benutzers (Benutzername, Kennwort, Methode, etc.) nach jeder Ausführung der Funktion **HTTP Get** oder **HTTP Request** im gleichen Prozess wiederherzustellen. Diese Information wird standardmäßig beibehalten und für jede Anfrage wiederverwendet. Übergeben Sie 1 in *Wert,* um diese Information nach jedem Aufruf zu löschen. Beachten Sie, dass die Information unabhängig von der Einstellung gelöscht wird, wenn der Prozess gestoppt wird.
HTTP timeout	Lange Ganzzahl	1	Wert = Timeout der Client Anfrage in Sekunden. Dieses Timeout bestimmt, wie lange der HTTP Client auf eine Antwort des Server wartet. Ist die Zeit überschritten, schließt der Client die Sitzung und die Anfrage geht verloren. Standardmäßig sind 120 Sekunden eingestellt. Das lässt sich bei bestimmten Bedingungen, wie Netzwerkstatus, Merkmale der Anfrage, etc. ändern.
HTTP TRACE method	Zeichenkette	TRACE	Siehe unter RFC 2616

Indextyp

Konstante	Typ	Wert	Kommentar
Cluster BTree index	Lange Ganzzahl	3	Index vom Typ B-Baum mit Clustern. Dieser Typ ist optimal für Indizes mit wenigen Schlagwörtern, z.B. wenn dieselben Werte in den Daten sich häufig wiederholen.
Default index type	Lange Ganzzahl	0	4D definiert den Indextyp (ausgenommen Index nach Schlüsselwörtern), der gemäß dem Feldinhalt am besten passt.
Keywords index	Lange Ganzzahl	-1	Volltext-Index, der die Indizierung des Feldinhalts Wort für Wort ermöglicht. Dieser Indextyp lässt sich nur für Datenfelder vom Typ Text, alphanumerisch oder Bild verwenden. **Achtung:** Volltext-Indizes können nicht zusammengesetzt sein.
Standard BTree index	Lange Ganzzahl	1	Index vom Typ Standard B-Baum. Dieser vielfältige Indextyp wird in den bisherigen Versionen von 4D verwendet.

Is license available

Konstante	Typ	Wert	Kommentar
4D Client SOAP license	Lange Ganzzahl	808465465	
4D Client Web license	Lange Ganzzahl	808465209	
4D for OCI license	Lange Ganzzahl	808465208	
4D Mobile license	Lange Ganzzahl	808464439	
4D Mobile Test license	Lange Ganzzahl	808465719	
4D ODBC Pro license	Lange Ganzzahl	808464946	
4D SOAP license	Lange Ganzzahl	808465464	
4D SOAP local license	Lange Ganzzahl	808531000	
4D SOAP one connection license	Lange Ganzzahl	825242680	
4D SQL Server license	Lange Ganzzahl	808464949	
4D SQL Server local license	Lange Ganzzahl	808530485	
4D SQL Server one conn license	Lange Ganzzahl	825242165	
4D View license	Lange Ganzzahl	808465207	
4D Web license	Lange Ganzzahl	808464945	
4D Web local license	Lange Ganzzahl	808530481	
4D Web one connection license	Lange Ganzzahl	825242161	
4D Write license	Lange Ganzzahl	808464697	

ISO Latin Zeichensatz

Konstante	Typ	Wert	Kommentar
ISO L1 a acute	Zeichenkette	á	
ISO L1 a circumflex	Zeichenkette	â	
ISO L1 a grave	Zeichenkette	à	
ISO L1 a ring	Zeichenkette	å	
ISO L1 a tilde	Zeichenkette	ã	
ISO L1 a umlaut	Zeichenkette	ä	
ISO L1 ae ligature	Zeichenkette	æ	
ISO L1 Ampersand	Zeichenkette	&	
ISO L1 c cedilla	Zeichenkette	ç	
ISO L1 Cap A acute	Zeichenkette	Á	
ISO L1 Cap A circumflex	Zeichenkette	Â	
ISO L1 Cap A grave	Zeichenkette	À	
ISO L1 Cap A ring	Zeichenkette	Å	
ISO L1 Cap A tilde	Zeichenkette	Ã	
ISO L1 Cap A umlaut	Zeichenkette	Ä	
ISO L1 Cap AE ligature	Zeichenkette	&AELig;	
ISO L1 Cap C cedilla	Zeichenkette	Ç	
ISO L1 Cap E acute	Zeichenkette	É	
ISO L1 Cap E circumflex	Zeichenkette	Ê	
ISO L1 Cap E grave	Zeichenkette	È	
ISO L1 Cap E umlaut	Zeichenkette	Ë	
ISO L1 Cap Eth Icelandic	Zeichenkette	Ð	
ISO L1 Cap I acute	Zeichenkette	Í	
ISO L1 Cap I circumflex	Zeichenkette	Î	
ISO L1 Cap I grave	Zeichenkette	Ì	
ISO L1 Cap I umlaut	Zeichenkette	Ï	
ISO L1 Cap N tilde	Zeichenkette	Ñ	
ISO L1 Cap O acute	Zeichenkette	Ó	
ISO L1 Cap O circumflex	Zeichenkette	Ô	
ISO L1 Cap O grave	Zeichenkette	Ò	
ISO L1 Cap O slash	Zeichenkette	Ø	
ISO L1 Cap O tilde	Zeichenkette	Õ	
ISO L1 Cap O umlaut	Zeichenkette	Ö	
ISO L1 Cap THORN Icelandic	Zeichenkette	Þ	
ISO L1 Cap U acute	Zeichenkette	Ú	
ISO L1 Cap U circumflex	Zeichenkette	Û	
ISO L1 Cap U grave	Zeichenkette	Ù	
ISO L1 Cap U umlaut	Zeichenkette	Ü	
ISO L1 Cap Y acute	Zeichenkette	Ý	
ISO L1 Copyright	Zeichenkette	©	
ISO L1 e acute	Zeichenkette	é	
ISO L1 e circumflex	Zeichenkette	ê	
ISO L1 e grave	Zeichenkette	è	
ISO L1 e umlaut	Zeichenkette	ë	
ISO L1 eth Icelandic	Zeichenkette	ð	
ISO L1 Greater than	Zeichenkette	>	
ISO L1 i acute	Zeichenkette	í	
ISO L1 i circumflex	Zeichenkette	î	
ISO L1 i grave	Zeichenkette	ì	
ISO L1 i umlaut	Zeichenkette	ï	
ISO L1 Less than	Zeichenkette	<	
ISO L1 n tilde	Zeichenkette	ñ	
ISO L1 o acute	Zeichenkette	ó	
ISO L1 o circumflex	Zeichenkette	ô	
ISO L1 o grave	Zeichenkette	ò	
ISO L1 o slash	Zeichenkette	ø	
ISO L1 o tilde	Zeichenkette	õ	
ISO L1 o umlaut	Zeichenkette	ö	
ISO L1 Quotation mark	Zeichenkette	"	
ISO L1 Registered	Zeichenkette	®	
ISO L1 sharp s German	Zeichenkette	ß	
ISO L1 thorn Icelandic	Zeichenkette	þ	
ISO L1 u acute	Zeichenkette	ú	
ISO L1 u circumflex	Zeichenkette	û	
ISO L1 u grave	Zeichenkette	ù	
ISO L1 u umlaut	Zeichenkette	ü	
ISO L1 y acute	Zeichenkette	ý	

Konstante	Typ	Wert	Kommentar
ISO L1 y umlaut	Zeichenkette	ÿ	

Kommunikation

Konstante	Typ	Wert	Kommentar
Data bits 5	Lange Ganzzahl	0	
Data bits 6	Lange Ganzzahl	2048	
Data bits 7	Lange Ganzzahl	1024	
Data bits 8	Lange Ganzzahl	3072	
MacOS printer port	Lange Ganzzahl	0	
MacOS serial port	Lange Ganzzahl	1	
Parity even	Lange Ganzzahl	12288	
Parity none	Lange Ganzzahl	0	
Parity odd	Lange Ganzzahl	4096	
Protocol DTR	Lange Ganzzahl	30	
Protocol none	Lange Ganzzahl	0	
Protocol XONXOFF	Lange Ganzzahl	20	
Speed 115200	Lange Ganzzahl	1022	
Speed 1200	Lange Ganzzahl	94	
Speed 1800	Lange Ganzzahl	62	
Speed 19200	Lange Ganzzahl	4	
Speed 230400	Lange Ganzzahl	1021	
Speed 2400	Lange Ganzzahl	46	
Speed 300	Lange Ganzzahl	380	
Speed 3600	Lange Ganzzahl	30	
Speed 4800	Lange Ganzzahl	22	
Speed 57600	Lange Ganzzahl	0	
Speed 600	Lange Ganzzahl	189	
Speed 7200	Lange Ganzzahl	14	
Speed 9600	Lange Ganzzahl	10	
Stop bits one	Lange Ganzzahl	16384	
Stop bits one and a half	Lange Ganzzahl	-32768	
Stop bits two	Lange Ganzzahl	-16384	

LDAP

Konstante	Typ	Wert	Kommentar
LDAP all levels	Zeichenkette	sub	Suche nur in der Root Einstiegsebene, definiert durch *dnRootEinstieg* (Standard, wenn weggelassen)
LDAP password MD5	Lange Ganzzahl	0	In MD5 verschlüsseltes Kennwort senden
LDAP password plain text	Lange Ganzzahl	1	Kennwort ohne Verschlüsselung senden (TLS Verbindung empfohlen)
LDAP root and next	Zeichenkette	one	Suche in der Root Einstiegsebene, definiert durch *dnRootEinstieg* und in den direkt nachfolgenden Einstiegen auf einer Ebene
LDAP root only	Zeichenkette	base	Suche nur in der Root Einstiegsebene, definiert durch *dnRootEinstieg* (Standard, wenn weggelassen)

Listbox

Konstante	Typ	Wert	Kommentar
_o_lk display hor scrollbar	Lange Ganzzahl	2	***Konstante ist überholt*** Befehl **OBJECT GET SCROLLBAR** verwenden
_o_lk display ver scrollbar	Lange Ganzzahl	4	***Konstante ist überholt*** Befehl **OBJECT GET SCROLLBAR** verwenden.
_o_lk footer height	Lange Ganzzahl	9	***Konstante ist überholt*** Befehl **LISTBOX Get footers height** verwenden
_o_lk header height	Lange Ganzzahl	1	***Konstante ist überholt*** Befehl **LISTBOX Get headers height** verwenden
_o_lk hor scrollbar position	Lange Ganzzahl	6	***Konstante ist überholt*** Befehl **OBJECT GET SCROLL POSITION** verwenden
_o_lk ver scrollbar position	Lange Ganzzahl	7	***Konstante ist überholt*** Befehl **OBJECT GET SCROLL POSITION** verwenden
lk add to selection	Lange Ganzzahl	1	Die ausgewählte Zeile wird der vorhandenen Auswahl hinzugefügt. Gehört die angegebene Zeile bereits zur Auswahl, führt die Konstante nichts aus.
lk all	Lange Ganzzahl	0	Der Befehl betrifft alle Unterebenen (Standardwert, wird ohne diesen Parameter verwendet)
lk allow wordwrap	Lange Ganzzahl	14	Eigenschaft **Zeilenumbruch** Gilt für: Spalte* Mögliche Werte: • lk no: (0) • lk yes: (1)
lk auto row height	Lange Ganzzahl	31	Eigenschaft **Automatische Zeilenhöhe** Gilt für: Listbox oder Spalte Mögliche Werte: • lk yes • lk no **Nur 4D View Pro:** Für dieses Feature wird eine 4D View Pro Lizenz benötigt. Weitere Informationen dazu finden Sie im **4D View Pro Handbuch**.
lk automatic	Lange Ganzzahl	2	Spalten werden automatisch zusammen mit der Listbox in der Größe angepasst (Eigenschaft **Spaltenbreite Automatisch** ist markiert).
lk background color	Lange Ganzzahl	1	
lk background color array	Lange Ganzzahl	1	
lk background color expression	Zeichenkette	22	Eigenschaft **Hintergrundfarbe Ausdruck** für Listbox vom Typ Auswahl Datensatz, Collection oder Entity-Selection Gilt für: Listbox oder Spalte
lk break row	Lange Ganzzahl	2	Der Befehl gilt für die Unterebene, zu der die Zelle, definiert mit den Parametern *Zeile* und *Spalte*, gehört. Beachten Sie, dass diese Parameter die Zeilen- und Spaltennummern in der Listbox im Standardmodus darstellen und nicht in der hierarchischen Form. In diesem Fall führt der Befehl nichts aus, wenn die Parameter *Zeile* und *Spalte* weggelassen werden.
lk column max width	Lange Ganzzahl	26	Eigenschaft **Maximale Breite** Gilt für: Spalte*
lk column min width	Lange Ganzzahl	25	Eigenschaft **Minimale Breite** Gilt für: Spalte*
lk column resizable	Lange Ganzzahl	15	Eigenschaft **Vergrößerbar** Gilt für: Spalte * Mögliche Werte: • lk no (0): • lk yes (1):
lk control array	Lange Ganzzahl	3	
lk detail form name	Zeichenkette	19	Eigenschaft **Name Detailformular** für Listbox vom Typ Auswahl. Gilt für: Listbox
lk display	Lange Ganzzahl	0	Zeigt am Ende der Listbox extra Leerzeilen an.

Konstante	Typ	Wert	Kommentar
lk display footer	Lange Ganzzahl	8	Eigenschaft **Fußteil anzeigen** Gilt für: Listbox Mögliche Werte: • lk no (0): Ausgeblendet • lk yes (1): Eingeblendet
lk display header	Lange Ganzzahl	0	Eigenschaft **Kopfteil anzeigen** Gilt für: Listbox Mögliche Werte: • lk no (0): Ausgeblendet • lk yes (1): Eingeblendet
lk display record	Lange Ganzzahl	2	Doppelklick auf eine Zeile zeigt den entsprechenden Datensatz im für die Listbox definierten Detailformular. Der Datensatz wird im Nur-Lesen Modus geöffnet, d.h. er lässt sich nicht verändern.
lk display type	Lange Ganzzahl	21	**Eigenschaft Typanzeige** für Spalten der Listbox vom Typ Zahl Gilt für: Spalte* Mögliche Werte: • lk numeric format: (0) Zeigt Werte im Zahlenformat an • lk three states checkbox: (1) Zeigt Werte als Kontrollkästchen mit drei Zuständen an
lk do nothing	Lange Ganzzahl	0	Doppelklick auf eine Zeile der Listbox löst keine automatische Aktion aus.
lk double click on row	Lange Ganzzahl	18	Eigenschaft **Doppelklick auf Zeile** für Listbox vom Typ Auswahl Gilt für: Listbox Mögliche Werte: • lk do nothing (0): Löst keine automatische Aktion aus • lk edit record (1): Zeigt den entsprechenden Datensatz im Lese-/Schreibmodus an • lk display record (2): Zeigt den entsprechenden Datensatz im Nur-Lesen Modus an
lk edit record	Lange Ganzzahl	1	Doppelklick in eine Zeile zeigt den entsprechenden Datensatz im für die Listbox definierten Detailformular. Der Datensatz wird im Schreibmodus geöffnet, so dass er änderbar ist.
lk extra rows	Lange Ganzzahl	13	Eigenschaft **Zusätzliche Leerzeilen ausblenden** Gilt für für Listbox Mögliche Werte: • lk display: (0) • lk hide: (1)
lk font color	Lange Ganzzahl	0	
lk font color array	Lange Ganzzahl	0	
lk font color expression	Zeichenkette	23	Eigenschaft **Schriftfarbe Ausdruck** für Listbox vom Typ Auswahl Datensatz, Collection oder Entity-Selection Gilt für: Listbox oder Spalte
lk font style expression	Zeichenkette	24	Eigenschaft **Stilausdruck** für Listbox vom Typ Auswahl Datensatz, Collection und Entity-Selection Gilt für: Listbox oder Spalte
lk hide	Lange Ganzzahl	1	Blendet zusätzliche Leerzeilen am Ende der Listbox aus.
lk hide selection highlight	Lange Ganzzahl	16	Eigenschaft **Markierung Auswahl ausblenden** Gilt für: Listbox Mögliche Werte: • lk no: (0) • lk yes*:* (1)
lk highlight set	Zeichenkette	27	Eigenschaft **Markierung Menge** für Listbox vom Typ Auswahl Gilt für: Listbox
lk hor scrollbar height	Lange Ganzzahl	3	Höhe in Pixel
lk inherited	Lange Ganzzahl	-255	
lk last printed row number	Lange Ganzzahl	0	Gibt in *Info* die Nummer der zuletzt gedruckten Zeile an. Damit finden Sie die Nummer der nächsten zu druckenden Zeile heraus. Die zurückgegebene Nummer kann größer als die Anzahl der aktuell gedruckten Zeilen sein, wenn die Listbox unsichtbare Zeichen enthält oder wenn der Befehl **OBJECT SET SCROLL POSITION** aufgerufen wurde. Wurden z.B. die Zeilen 1, 18 und 20 gedruckt, gibt *Info* 20 zurück.
lk level	Lange Ganzzahl	3	Dieser Befehl betrifft alle Umbruchzeilen für die Spalte *Ebene*. Dieser Parameter bezeichnet eine Spaltennummer in der Listbox im Standardmodus und nicht in seiner hierarchischen Darstellung. In diesem Fall führt der Befehl nichts aus, wenn der Parameter *Ebene* weggelassen wird.

Konstante	Typ	Wert	Kommentar
lk lines	Lange Ganzzahl	1	Höhe ist eine Anzahl Zeilen. 4D berechnet die Zeilenhöhe nach dem Schrifttyp.
lk manual	Lange Ganzzahl	0	Spalten werden nicht automatisch zusammen mit der Listbox vergrößert (Eigenschaft **Spaltenbreite Automatisch** ist nicht markiert).
lk mulltiple	Lange Ganzzahl	2	Mehrere Listboxzeilen lassen sich auf einmal auswählen.
lk multi style	Lange Ganzzahl	30	Eigenschaft **Mehrfachstil** Gilt für: Spalte * Mögliche Werte: • lk no (0) • lk yes (1)
lk named selection	Zeichenkette	28	Eigenschaft **temporäre Auswahl** Gilt für: Listbox
lk no	Lange Ganzzahl	0	
lk none	Lange Ganzzahl	0	
lk numeric format	Lange Ganzzahl	0	Zeigt numerische Werte in Spalten der Listbox als Zahl an. Wert für Eigenschaft lk display type.
lk pixels	Lange Ganzzahl	0	Höhe ist eine Anzahl Pixel (Standard)
lk printed height	Lange Ganzzahl	3	Gibt in *Info* die Höhe in Pixel des aktuell gedruckten Objekts zurück (inkl. Kopfzeilen, Zeilen, etc.). Beachten Sie, dass falls die Anzahl der zu druckenden Zeilen kleiner als die "Kapazität" der Listbox ist, seine Höhe automatisch verringert wird.
lk printed rows	Lange Ganzzahl	1	Gibt in *Info* die Anzahl der aktuell gedruckten Zeilen während dem letzten Aufruf der Funktion **Print object** zurück. Im Falle einer hierarchischen Liste beinhaltet diese Zahl auch alle hinzugefügten Umbruchzeilen. Zum Beispiel ist *Info* 10, bei einer Listbox mit 20 Zeilen, wenn die ungeraden Zeilen ausgeblendet sind.
lk printing is over	Lange Ganzzahl	2	Gibt in *Info* ein Boolean zurück ,der angibt, ob die letzte (sichtbare) Zeile der Listbox gerade gedruckt wurde. Wahr = Zeile wurde gedruckt; Andernfalls falsch.
lk remove from selection	Lange Ganzzahl	2	Die ausgewählte Zeile wird aus der vorhandenen Auswahl entfernt. Gehört die angegebene Zeile nicht zur Auswahl, führt die Konstante nichts aus.
lk replace selection	Lange Ganzzahl	0	Die gewählte Zeile wird zur neuen Auswahl und ersetzt die vorhandene Auswahl. Die Konstante hat dieselbe Wirkung wie Anklicken der Zeile durch den Benutzer (Das Ereignis On Clicked wird dagegen nicht generiert). Dies ist die Standardaktion, d.h. wenn der Parameter *Aktion* nicht verwendet wird.
lk resizing mode	Lange Ganzzahl	11	Eigenschaft **Spaltenbreite Automatisch** Gilt für: Listbox Mögliche Werte: • lk manual: (0) • lk automatic: (2)
lk row height array	Lange Ganzzahl	4	(4D View Pro Lizenz ist erforderlich)
lk row height unit	Lange Ganzzahl	17	Eigenschaft **Einheit für Zeilenhöhe** Gilt für: Listbox Mögliche Werte: • lk lines (1) • lk pixels (0)
lk row is disabled	Lange Ganzzahl	2	Die entsprechende Zeile ist deaktiviert. Text und Kontrollkästchen sind gedimmt oder in Grauschrift. Eingebbare Textbereiche sind nicht mehr eingebbar. Standardwert: Aktiviert
lk row is hidden	Lange Ganzzahl	1	Die entsprechende Zeile ist ausgeblendet. Das betrifft nur die Darstellung der Listbox. Ausgeblendete Zeilen sind weiterhin in den Arrays vorhanden und lassen sich per Programmierung steuern. Die Befehle der Programmiersprache, insbesondere **LISTBOX Get number of rows** oder **LISTBOX GET CELL POSITION**, ignorieren den Status angezeigt/ausgeblendet von Zeilen. Zum Beispiel gibt in einer Listbox mit 10 Zeilen, von denen 9 ausgeblendet sind, der Befehl **LISTBOX Get number of rows** 10 zurück. Auf Benutzerseite ist nicht ersichtlich, ob es ausgeblendete Zeilen in der Listbox gibt. Nur sichtbare Zeilen lassen sich auswählen, z.B. über den Befehl *All*e auswählen. Standardwert: Sichtbar
lk row is not selectable	Lange Ganzzahl	4	Die entsprechende Zeile ist nicht auswählbar (sie lässt sich nicht markieren). Eingebbare Textbereiche sind nicht länger eingebbar, außer die Option "Einzelklick Editieren" ist aktiviert. Kontrollen wie Optionsfelder und PopUp-Menüs funktionieren jedoch weiterhin. Diese Einstellung wird ignoriert, wenn als Auswahlmodus für Listbox "Nichts" markiert ist. Standard: Auswählbar
lk row max height	Lange Ganzzahl	33	
lk row min height	Lange Ganzzahl	32	

Konstante	Typ	Wert	Kommentar
lk selection	Lange Ganzzahl	1	Befehl gilt für ausgewählte Unterebenen.
lk selection mode	Lange Ganzzahl	10	Eigenschaft **Auswahlmodus** Gilt für: Listbox Mögliche Werte: • lk none (0) • lk single (1) • lk multiple (2)
lk single	Lange Ganzzahl	1	Nur eine Listboxzeile ist gleichzeitig auswählbar.
lk single click edit	Lange Ganzzahl	29	Eigenschaft **Einzelklick editieren** Gilt für: Listbox Mögliche Werte: • lk no (0) • lk yes (1)
lk sortable	Lange Ganzzahl	20	Eigenschaft **Sortierbar** Gilt für: Listbox Mögliche Werte: • lk no: (0) • lk yes: (1)
lk style array	Lange Ganzzahl	2	
lk three states checkbox	Lange Ganzzahl	1	Spalten mit numerischen Werten werden als Kontrollkästchen mit drei Zuständen angezeigt. Wert für Eigenschaft lk display type.
lk truncate	Lange Ganzzahl	12	Eigenschaft **Abkürzen mit Auslassungspunkten** Gilt für: Listbox oder Spalte Mögliche Werte: • lk without ellipsis: (0) • lk with ellipsis: (1)
lk ver scrollbar width	Lange Ganzzahl	5	Breite in Pixel
lk with ellipsis	Lange Ganzzahl	1	Es erscheinen Auslassungspunkte, wenn der Inhalt in der Zelle der Listbox die Spaltenbreite übersteigt.
lk without ellipsis	Lange Ganzzahl	0	Es erscheinen keine Auslassungspunkte, wenn der Inhalt in der Zelle der Listbox die Spaltenbreite übersteigt.
lk yes	Lange Ganzzahl	1	

Listbox Fußteil Berechnung

Konstante	Typ	Wert	Kommentar
Listbox footer std deviation	Lange Ganzzahl	7	Für Spalten vom Typ Zahl oder Zeit verwenden (nur für Arrays vom Typ Listboxen). Standardtyp des Berechnungsergebnis: Zahl
lk footer average	Lange Ganzzahl	6	Für Spalten vom Typ Zahl oder Zeit verwenden. Standardtyp des Ergebnisses: Zahl.
lk footer count	Lange Ganzzahl	5	Für Spalten vom Typ Zahl, Text, Datum, Zeit, Boolean oder Bild verwenden. Standardtyp des Berechnungsergebnis: Lange Ganzzahl
lk footer custom	Lange Ganzzahl	1	4D führt keine Berechnung durch. Die Variable im Fußteil muss per Programmierung berechnet werden. Standardtyp des Berechnungsergebnis: Typ der Variable
lk footer max	Lange Ganzzahl	3	Für Spalten vom Typ Zahl, Datum, Zeit oder Boolean verwenden. Standardtyp des Berechnungsergebnis: Spalte Array oder Feldtyp
lk footer min	Lange Ganzzahl	2	Für Spalten vom Typ Zahl, Datum, Zeit oder Boolean verwenden. Standardtyp des Berechnungsergebnis: Spalte Array oder Feldtyp
lk footer sum	Lange Ganzzahl	4	Für Spalten vom Typ Zahl, Zeit oder Boolean verwenden. Standardtyp des Berechnungsergebnis: Spalte Array oder Feldtyp.
lk footer sum squares	Lange Ganzzahl	9	Für Spalten vom Typ Zahl oder Zeit verwenden (nur für Arrays vom Typ Listboxen). Standardtyp des Berechnungsergebnis: Zahl
lk footer variance	Lange Ganzzahl	8	Für Spalten vom Typ Zahl oder Zeit verwenden (nur für Arrays vom Typ Listboxen). Standardtyp des Berechnungsergebnis: Zahl

Log Ereignisse

Konstante	Typ	Wert	Kommentar
Error message	Lange Ganzzahl	2	
Information message	Lange Ganzzahl	0	
Into 4D commands log	Lange Ganzzahl	3	Dieser Wert weist 4D an, *Meldung* im Logbuch der 4D Befehle zu speichern, wenn diese Datei aktiviert wurde. Das Logbuch der 4D Befehle lässt sich über den Befehl **SET DATABASE PARAMETER** (Selector 34) aktivieren. **Hinweis:** 4D Log Dateien werden im Ordner **Logs** gesammelt, der neben der Strukturdatei der Datenbank liegt.
Into 4D debug message	Lange Ganzzahl	1	Dieser Wert weist 4D an, *Meldung* an die Debugging Umgebung des Systems zu senden. Das Ergebnis richtet sich nach der jeweiligen Plattform: - Mac OS: Der Befehl sendet *Meldung* an die Konsole. - Windows: Der Befehl sendet *Meldung* als eine Debug Meldung. Zum Lesen dieser Meldung benötigen Sie Microsoft Visual Studio oder das Hilfsprogramm DebugView für Windows (http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx).
Into 4D diagnostic log	Lange Ganzzahl	5	Dieser Wert weist 4D an, *Meldung* in der Protokolldatei von 4D einzutragen, wenn diese Datei aktiviert wurde. Diese Datei lässt sich über den Befehl **SET DATABASE PARAMETER** (Selector 79) aktivieren.
Into 4D request log	Lange Ganzzahl	2	Dieser Wert weist 4D an, *Meldung* im Logbuch 4D Anfragen zu speichern, wenn diese Datei aktiviert wurde.
Into Windows log events	Lange Ganzzahl	0	Dieser Wert weist 4D an, *Meldung* die "Log Ereignisse" des Fensters zu senden. Dieses Logbuch empfängt und speichert Meldungen von den laufenden Anwendungen. Dann können Sie *Meldung* über den optionalen Parameter *Wichtigkeit* eine Wichtigkeitsstufe zuordnen (siehe oben). **Hinweise:** • Diese Funktionalität ist nur verfügbar, wenn der Service Windows Log Events läuft. • Auf Mac OS führt der Befehl mit diesem Wert in *AusgabeTyp* nichts aus.
Warning message	Lange Ganzzahl	1	

Mathematik

Konstante	Typ	Wert	Kommentar
Degree	Zahl	0.0174532925199432958	
e number	Zahl	2.71828182845904524	
Pi	Zahl	3.141592653589793239	
Radian	Zahl	57.29577951308232088	

Mehrfachstil Text

Konstante	Typ	Wert	Kommentar
ST 4D Expressions as sources	Lange Ganzzahl	2	Der ursprüngliche String der Referenzen auf 4D Ausdrücke wird zurückgegeben
ST 4D Expressions as values	Lange Ganzzahl	1	Referenzen auf 4D Ausdrücke werden in interpretierter Form zurückgegeben (standardmäßige Funktionsweise in Formularen)
ST End highlight	Lange Ganzzahl	-1001	Bestimmt das letzte Zeichen der aktuellen Textauswahl in Objekt (*)
ST End text	Lange Ganzzahl	0	Bestimmt das letzte Zeichen des Textes in Objekt
ST Expression type	Lange Ganzzahl	2	Auswahl enthält nur eine Referenz für Ausdruck
ST Expressions display mode	Lange Ganzzahl	1	Der Parameter *Wert* kann ST Values oder ST References enthalten
ST Mixed type	Lange Ganzzahl	3	Auswahl enthält mindestens zwei unterschiedliche Typen im Inhalt
ST Picture type	Lange Ganzzahl	6	Auswahl enthält nur ein Bild (nur 4D Write Pro Areas)
ST Plain type	Lange Ganzzahl	0	Auswahl enthält Text und keine Referenzen
ST References	Lange Ganzzahl	1	Zeigt Quelle der Strings für Ausdrücke
ST References as spaces	Lange Ganzzahl	0	Jede Referenz wird in Form geschützter Leerzeichen zurückgegeben (Standardoperation, wird von den anderen Befehlen verwendet)
ST Start highlight	Lange Ganzzahl	-1000	Bestimmt das erste Zeichen der aktuellen Textauswahl in Objekt (*)
ST Start text	Lange Ganzzahl	1	Bestimmt das erste Zeichen des Textes in Objekt
ST Tags as plain text	Lange Ganzzahl	64	Die Bezeichnung des Tag wird in Plain Text zurückgegeben. Beispiel: Für das Tag '<img src="test.jpg" alt="picture">my picture</img>', ist Plain Text "my picture" (standardmäßige Funktionsweise in Formularen)
ST Tags as XML code	Lange Ganzzahl	128	Der XML Code des Tag wird in Plain Text zurückgegeben. Beispiel: Für das Tag '<img src="test.jpg" alt="picture">my picture</img>', ist der Plain Text '<img src="test.jpg" alt="picture">my picture</img>
ST Text displayed with 4D Expression sources	Lange Ganzzahl	86	Gibt den Text so zurück, wie er im Formular angezeigt wird, mit den ursprünglichen Strings der 4D Ausdrücke. Entspricht einer vordefinierten Kombination der Konstanten 2+4+16+64
ST Text displayed with 4D Expression values	Lange Ganzzahl	85	Gibt den Text so zurück, wie er im Formular angezeigt wird, mit interpretierten 4D Ausdrücken. Entspricht einer vordefinierten Kombination der Konstanten 1+4+16+64
ST Unknown tag type	Lange Ganzzahl	4	Auswahl enthält nur einen unbekannten Tag Typ
ST URL as labels	Lange Ganzzahl	4	Die sichtbare Bezeichnung von URLs wird zurückgegeben, z.B. "Besuchen Sie unsere Web Site" (standardmäßige Funktionsweise in Formularen)
ST URL as links	Lange Ganzzahl	8	Der Link wird zurückgegeben, z.B. "http://www.4d.com"
ST URL type	Lange Ganzzahl	1	Auswahl enthält nur eine Referenz für URL
ST User links as labels	Lange Ganzzahl	16	Die sichtbare Bezeichnung des Benutzer-Links wird zurückgegeben (standardmäßige Funktionsweise in Formularen)
ST User links as links	Lange Ganzzahl	32	Der Inhalt des Benutzer-Links wird zurückgegeben
ST User type	Lange Ganzzahl	5	Auswahl enthält nur eine eigene Referenz
ST Values	Lange Ganzzahl	0	Zeigt berechnete Werte von Ausdrücken

Mehrfachstil Textattribute

Konstante	Typ	Wert	Kommentar
Attribute background color	Lange Ganzzahl	8	(nur Windows) Hexadezimale Werte oder HTML Farbnamen
Attribute bold style	Lange Ganzzahl	1	*attrWert*=0: Attribut fett aus Auswahl entfernen *attrWert*=1: Attribut fett auf Auswahl anwenden
Attribute font name	Lange Ganzzahl	5	*attrWert*=Schriftfamilienname (String)
Attribute italic style	Lange Ganzzahl	2	*attrWert*=0: Attribut kursiv aus Auswahl entfernen *attrWert*=1: Attribut kursiv auf Auswahl anwenden
Attribute strikethrough style	Lange Ganzzahl	3	*attrWert*=0: Attribut durchgestrichen aus Auswahl entfernen *attrWert*=1: Attribut durchgestrichen auf Auswahl anwenden
Attribute text color	Lange Ganzzahl	7	Hexadezimale Werte oder Konstanten mit hexadezimalen Werten
Attribute text size	Lange Ganzzahl	6	*attrWert*=Anzahl Punkte (Zahl)
Attribute underline style	Lange Ganzzahl	4	*attrWert*=0: Attribut unterstrichen aus Auswahl entfernen *attrWert*=1: Attribut unterstrichen auf Auswahl anwenden

Menüzeilen Eigenschaften

Konstante	Typ	Wert	Kommentar
Access privileges	Zeichenkette	4D_access_group	Dem Befehl eine Zugriffsgruppe zuweisen 0 = Alle Gruppen >0 = Gruppennummer
Associated standard action	Zeichenkette	4D_standard_action	Einem Menüeintrag eine Standardaktion zuweisen Siehe Konstanten unter dem Thema **Standardaktion**.
Start a new process	Zeichenkette	4D_start_new_process	Dem Befehl eine Zugriffsgruppe zuordnen 0 = Ohne Einschränkung >0 = Gruppennummer

Objekte und Collections

Konstante	Typ	Wert	Kommentar
ck ascending	Lange Ganzzahl	0	Elemente in aufsteigender Reihenfolge sortieren (Standard)
ck descending	Lange Ganzzahl	1	Elemente in absteigender Reihenfolge sortieren
ck diacritical	Lange Ganzzahl	8	Diakritische Bewertung durchführen
ck disable wildchar	Lange Ganzzahl	16	
ck ignore null or empty	Lange Ganzzahl	1	Null Werte und leere Strings im Ergebnis ignorieren
ck keep empty strings	Lange Ganzzahl	2	
ck keep null	Lange Ganzzahl	1	Eigenschaften *null* oder *undefiniert* im Ergebnis beibehalten
ck resolve pointers	Lange Ganzzahl	1	Zeiger in der kopierten Collection auflösen
dk auto merge	Lange Ganzzahl	32	Aktiviert den Modus automatisches Mischen (Option für die Methode **entity.save( )**)
dk diacritical	Lange Ganzzahl	8	Diakritische Bewertung durchführen
dk distinct values	Lange Ganzzahl	1	Berücksichtigt nur Entity Attribute mit nicht-wiederholten Werten (Option für die Methode **entitySelection.count( )**)
dk force drop if stamp changed	Lange Ganzzahl	2	Erzwingt Löschen, auch wenn sich der Stempel geändert hat (Option für die Methode **entity.drop( )**)
dk keep ordered	Lange Ganzzahl	1	Option zum Beibehalten der Reihenfolge der ursprünglichen Collection in der neuen
dk key as string	Lange Ganzzahl	1	Die Methode **entity.getKey( )** gibt den Wert des Primärschlüssels als String zurück
dk non ordered	Lange Ganzzahl	0	Option zum Erstellen einer unsortierten neuen Collection
dk reload if stamp changed	Lange Ganzzahl	1	Hat sich der Stempel der Entity geändert, diese vor Durchführen des Sperrverfahrens erneut laden (Option für die Methode **entity.lock( )**)
dk status automerge failed	Lange Ganzzahl	6	(Nur bei Verwenden der Option dk auto merge) Der Modus automatisches Mischen beim Sichern der Entity ist fehlgeschlagen. **Zugewiesener Statustext**: "Auto merge failed"
dk status entity does not exist anymore	Lange Ganzzahl	5	Die Entity existiert nicht mehr in den Daten. Dieser Fehler kann in folgenden Fällen auftreten: • Die Entity wurde gelöscht (der Stempel hat sich geändert und der Speicherplatz ist jetzt frei) • Die Entity wurde gelöscht und durch eine andere mit einem anderen Primärschlüssel ersetzt (der Stempel hat sich geändert und eine neue Entity verwendet jetzt den Speicherplatz). Mit **entity.drop( )** wird dieser Fehler beim Verwenden der Option dk force drop if stamp changed zurückgegeben. Mit **entity.lock( )** wird dieser Fehler beim Verwenden der Option dk reload if stamp changed zurückgegeben. **Zugewiesener Statustext**: "Entity does not exist anymore"
dk status locked	Lange Ganzzahl	3	Die Entity wird durch pessimistisches Sperrverfahren gesperrt. **Zugewiesener Statustext**: "Already locked"
dk status serious error	Lange Ganzzahl	4	Ein ernsthafter Fehler ist ein low-level Fehler in der Anwendung, wie z.B. duplizierter Schlüssel, Hardware Fehler, etc. **Zugewiesener Statustext**: "Other error"
dk status stamp has changed	Lange Ganzzahl	2	Der interne Stempelwert der Entity passt nicht zum Wert der in den Daten gespeicherten Entity (optimistisches Sperrverfahren). • mit **entity.save( )**: nur Fehler, wenn die Option dk auto merge nicht verwendet wird • mit **entity.drop( )**: nur Fehler, wenn die Option dk force drop if stamp changed nicht verwendet wird • mit **entity.lock( )**: nur Fehler, wenn die Option dk reload if stamp changed nicht verwendet wird **Zugewiesener Statustext**: "Stamp has changed"
dk stop dropping on first error	Lange Ganzzahl	8	Der Löschvorgang stoppt bei der ersten nicht-löschbaren Entity (Option für die Methode **entitySelection.drop( )**)
dk with primary key	Lange Ganzzahl	1	Primärschlüssel in extrahierter Collection bzw. Objekt hinzufügen (Option für die Methoden **entitySelection.toCollection( )** und **entity.toObject( )**)
dk with stamp	Lange Ganzzahl	2	Stempel in extrahierter Collection bzw. Objekt hinzufügen (Option für die Methoden **entitySelection.toCollection( )** und **entity.toObject( )**)

Open form window

Konstante	Typ	Wert	Kommentar
_o_Compositing mode form	Lange Ganzzahl	4096	*** Obsolete constant ***
_o_Has toolbar button Mac OS	Lange Ganzzahl	8192	*** Obsolete constant ***
At the bottom	Lange Ganzzahl	393216	
At the top	Lange Ganzzahl	327680	
Controller form window	Lange Ganzzahl	133056	
Form has full screen mode Mac	Lange Ganzzahl	65536	
Form has no menu bar	Lange Ganzzahl	2048	
Horizontally centered	Lange Ganzzahl	65536	
Modal form dialog box	Lange Ganzzahl	1	
Movable form dialog box	Lange Ganzzahl	5	
On the left	Lange Ganzzahl	131072	
On the right	Lange Ganzzahl	196608	
Palette form window	Lange Ganzzahl	1984	
Plain form window	Lange Ganzzahl	8	
Pop up form window	Lange Ganzzahl	32	
Sheet form window	Lange Ganzzahl	33	
Toolbar form window	Lange Ganzzahl	35	
Vertically centered	Lange Ganzzahl	262144	

Open Window

Konstante	Typ	Wert	Kommentar
_o_Compositing Mode	Lange Ganzzahl	4096	*** Konstante überholt ***
_o_Has toolbar button Mac	Lange Ganzzahl	8192	*** Obsolete constant ***
Alternate dialog box	Lange Ganzzahl	3	Kann Palettenfenster sein
Has full screen mode Mac	Lange Ganzzahl	65536	
Has grow box	Lange Ganzzahl	4	
Has highlight	Lange Ganzzahl	1	
Has window title	Lange Ganzzahl	2	
Has zoom box	Lange Ganzzahl	8	
Modal dialog box	Lange Ganzzahl	1	
Movable dialog box	Lange Ganzzahl	5	Kann Palettenfenster sein
Palette window	Lange Ganzzahl	1984	Kann Palettenfenster sein
Plain dialog box	Lange Ganzzahl	2	Kann Palettenfenster sein
Plain fixed size window	Lange Ganzzahl	4	
Plain no zoom box window	Lange Ganzzahl	0	
Plain window	Lange Ganzzahl	8	
Pop up window	Lange Ganzzahl	32	
Resizable sheet window	Lange Ganzzahl	34	
Round corner window	Lange Ganzzahl	16	
Sheet window	Lange Ganzzahl	33	
Texture appearance	Lange Ganzzahl	2048	

Pasteboard

Konstante	Typ	Wert	Kommentar
No such data in pasteboard	Lange Ganzzahl	-102	
Picture data	Zeichenkette	PICT	
Text data	Zeichenkette	TEXT	

PHP

Konstante	Typ	Wert	Kommentar
PHP privileges	Lange Ganzzahl	1	Definition spezifischer Benutzerprivilegien zur Ausführung von Skripts. **Mögliche Werte**: String in Form von "User:Password". Zum Beispiel: "root:jd51254d"
PHP raw result	Lange Ganzzahl	2	Definition des Bearbeitungsmodus für HTTP Header, die von PHP im Ausführungsergebnis zurückgegeben werden, wenn das Ergebnis vom Typ Text ist (bei einem Ergebnis vom Typ BLOB werden Header immer beibehalten). **Mögliche Werte**: Boolean. Falsch (Standardwert = HTTP Header aus Ergebnis entfernen. Wahr = HTTP Header beibehalten.

Plattformeigenschaften

Constants prefixed with _o_ are obsolete and cannot be returned by the command. They are only kept for compatibility.

Konstante	Typ	Wert	Kommentar
_o_INTEL 386	Lange Ganzzahl	386	
_o_INTEL 486	Lange Ganzzahl	486	
_o_Macintosh 68K	Lange Ganzzahl	1	
_o_PowerPC 601	Lange Ganzzahl	601	
_o_PowerPC 603	Lange Ganzzahl	603	
_o_PowerPC 604	Lange Ganzzahl	604	
_o_PowerPC G3	Lange Ganzzahl	510	
Intel compatible	Lange Ganzzahl	586	
Mac OS	Lange Ganzzahl	2	
Power PC	Lange Ganzzahl	406	
Windows	Lange Ganzzahl	3	

Plattformoberfläche

Konstante	Typ	Wert	Kommentar

Prozesstatus

Konstante	Typ	Wert	Kommentar
Aborted	Lange Ganzzahl	-1	
Delayed	Lange Ganzzahl	1	
Does not exist	Lange Ganzzahl	-100	
Executing	Lange Ganzzahl	0	
Hidden modal dialog	Lange Ganzzahl	6	
Paused	Lange Ganzzahl	5	
Waiting for input output	Lange Ganzzahl	3	
Waiting for internal flag	Lange Ganzzahl	4	
Waiting for user event	Lange Ganzzahl	2	

Prozesstypen

Konstante	Typ	Wert	Kommentar
_o_Web process with context	Lange Ganzzahl	-11	
Apple event manager	Lange Ganzzahl	-7	
Backup process	Lange Ganzzahl	-19	
Cache manager	Lange Ganzzahl	-4	
Client manager process	Lange Ganzzahl	-31	
Compiler process	Lange Ganzzahl	-29	
Created from execution dialog	Lange Ganzzahl	3	
Created from menu command	Lange Ganzzahl	2	
DB4D Cron	Lange Ganzzahl	-49	
DB4D Flush cache	Lange Ganzzahl	-46	
DB4D Garbage collector	Lange Ganzzahl	-47	
DB4D Index builder	Lange Ganzzahl	-45	
DB4D Listener	Lange Ganzzahl	-51	
DB4D Mirror	Lange Ganzzahl	-50	
DB4D Worker pool user	Lange Ganzzahl	-48	
Design process	Lange Ganzzahl	-2	
Event manager	Lange Ganzzahl	-8	
Execute on client process	Lange Ganzzahl	-14	
Execute on server process	Lange Ganzzahl	1	
External task	Lange Ganzzahl	-9	
HTTP Listener	Lange Ganzzahl	-56	
HTTP Log flusher	Lange Ganzzahl	-58	
HTTP Worker pool server	Lange Ganzzahl	-55	
Indexing process	Lange Ganzzahl	-5	
Internal 4D server process	Lange Ganzzahl	-18	
Internal timer process	Lange Ganzzahl	-25	
Log file process	Lange Ganzzahl	-20	
Logger process	Lange Ganzzahl	-57	
Main 4D process	Lange Ganzzahl	-39	
Main process	Lange Ganzzahl	-1	
Method editor macro process	Lange Ganzzahl	-17	
Monitor process	Lange Ganzzahl	-26	
MSC process	Lange Ganzzahl	-22	
None	Lange Ganzzahl	0	
On exit process	Lange Ganzzahl	-16	
Other 4D process	Lange Ganzzahl	-10	
Other internal process	Lange Ganzzahl	-40	
Other user process	Lange Ganzzahl	4	
Restore Process	Lange Ganzzahl	-21	
Serial Port Manager	Lange Ganzzahl	-6	
Server interface process	Lange Ganzzahl	-15	
ServerNet Listener	Lange Ganzzahl	-43	
ServerNet Session manager	Lange Ganzzahl	-44	
SQL Listener	Lange Ganzzahl	-54	
SQL Method execution process	Lange Ganzzahl	-24	
SQL Net Session manager	Lange Ganzzahl	-53	
SQL Worker pool server	Lange Ganzzahl	-52	
Web process on 4D remote	Lange Ganzzahl	-12	
Web process with no context	Lange Ganzzahl	-3	
Web server process	Lange Ganzzahl	-13	
Worker pool in use	Lange Ganzzahl	-41	
Worker pool spare	Lange Ganzzahl	-42	
Worker process	Lange Ganzzahl	5	Worker Prozess, vom Benutzer gestartet

QR Ausgabeart

Konstante	Typ	Wert	Kommentar
_o_qr 4D Chart area	Lange Ganzzahl	4	***Konstante ist überholt***
qr 4D View area	Lange Ganzzahl	3	*Detail*: N.A.
qr HTML file	Lange Ganzzahl	5	*Detail*: Pfadname zur Datei
qr printer	Lange Ganzzahl	1	*Details*: "*" um kein Dialogfenster beim Drucken anzuzeigen
qr text file	Lange Ganzzahl	2	*Detail*: Pfadname zur Datei

QR Befehle

Konstante	Typ	Wert	Kommentar
qr cmd 4D View destination	Lange Ganzzahl	2503	
qr cmd add column	Lange Ganzzahl	2608	
qr cmd alt back color palette	Lange Ganzzahl	1004	
qr cmd automatic width	Lange Ganzzahl	2605	
qr cmd average	Lange Ganzzahl	507	
qr cmd back color palette	Lange Ganzzahl	1003	
qr cmd back colors toolbar	Lange Ganzzahl	2052	
qr cmd bold	Lange Ganzzahl	500	
qr cmd borders	Lange Ganzzahl	2609	
qr cmd center justified	Lange Ganzzahl	504	
qr cmd columns toolbar	Lange Ganzzahl	2054	
qr cmd count	Lange Ganzzahl	510	
qr cmd default justified	Lange Ganzzahl	512	
qr cmd delete column	Lange Ganzzahl	2601	
qr cmd disk file destination	Lange Ganzzahl	2501	
qr cmd edit column	Lange Ganzzahl	2603	
qr cmd font color palette	Lange Ganzzahl	1002	
qr cmd font dropdown	Lange Ganzzahl	1000	
qr cmd format	Lange Ganzzahl	2606	
qr cmd generate	Lange Ganzzahl	2008	kompatibel mit 64-bit Editor (Befehl **QR RUN** wird empfohlen)
qr cmd graph destination	Lange Ganzzahl	2502	
qr cmd header and footer	Lange Ganzzahl	2005	
qr cmd hide column	Lange Ganzzahl	2602	
qr cmd hide line	Lange Ganzzahl	2607	
qr cmd HTML file destination	Lange Ganzzahl	2504	
qr cmd insert column	Lange Ganzzahl	2600	
qr cmd italic	Lange Ganzzahl	501	
qr cmd left justified	Lange Ganzzahl	503	
qr cmd max	Lange Ganzzahl	509	
qr cmd min	Lange Ganzzahl	508	
qr cmd move left	Lange Ganzzahl	3002	
qr cmd move right	Lange Ganzzahl	3003	
qr cmd new	Lange Ganzzahl	2000	
qr cmd open	Lange Ganzzahl	2001	
qr cmd operators toolbar	Lange Ganzzahl	2051	
qr cmd page setup	Lange Ganzzahl	2006	kompatibel mit 64-bit Editor
qr cmd plain	Lange Ganzzahl	511	
qr cmd presentation	Lange Ganzzahl	2611	
qr cmd print preview	Lange Ganzzahl	2007	kompatibel mit 64-bit Editor
qr cmd printer destination	Lange Ganzzahl	2500	
qr cmd repeated values	Lange Ganzzahl	2604	
qr cmd revert to save	Lange Ganzzahl	2004	
qr cmd right justified	Lange Ganzzahl	505	
qr cmd save	Lange Ganzzahl	2002	
qr cmd save as	Lange Ganzzahl	2003	
qr cmd standard deviation	Lange Ganzzahl	513	
qr cmd standard toolbar	Lange Ganzzahl	2053	
qr cmd style toolbar	Lange Ganzzahl	2050	
qr cmd sum	Lange Ganzzahl	506	
qr cmd totals spacing	Lange Ganzzahl	2610	
qr cmd underline	Lange Ganzzahl	502	

QR Bereichseigenschaften

Konstante	Typ	Wert	Kommentar
qr view color toolbar	Lange Ganzzahl	5	Zeige Status der Werkzeugleiste für Farben (Angezeigt=1, Ausgeblendet=0)
qr view column toolbar	Lange Ganzzahl	6	Zeige Status der Werkzeugleiste für Spalten (Angezeigt=1, Ausgeblendet=0)
qr view contextual menus	Lange Ganzzahl	7	Zeige Status des Kontextmenüs (Angezeigt=1, Ausgeblendet=0)
qr view menubar	Lange Ganzzahl	1	Zeige Status der Menüleiste (Angezeigt=1, Ausgeblendet=0)
qr view operators toolbar	Lange Ganzzahl	4	Zeige Status der Werkzeugleiste für Operatoren (Angezeigt=1, Ausgeblendet=0)
qr view standard toolbar	Lange Ganzzahl	2	Zeige Status der Standard Werkzeugleiste (Angezeigt=1, Ausgeblendet=0)
qr view style toolbar	Lange Ganzzahl	3	Zeige Status der Werkzeugleiste für Stil (Angezeigt=1, Ausgeblendet=0)

QR Berichtstypen

Konstante	Typ	Wert	Kommentar
qr cross report	Lange Ganzzahl	2	
qr list report	Lange Ganzzahl	1	

QR Dokumenteigenschaften

Konstante	Typ	Wert	Kommentar
qr printing dialog	Lange Ganzzahl	1	Druckdialog anzeigen: - Bei Wert = 0 wird der Druckdialog nicht vor dem Drucken angezeigt. - Bei Wert = 1 wird der Druckdialog vor dem Drucken angezeigt (Standardwert).
qr unit	Lange Ganzzahl	2	Maßeinheit für das Dokument: - Bei Wert = 0 ist die Einheit Punkte - Bei Wert = 1 ist die Einheit Zentimeter. - Bei Wert = 2 ist die Einheit Inches.

QR Operatoren

Konstante	Typ	Wert	Kommentar
qr average	Lange Ganzzahl	2	
qr count	Lange Ganzzahl	16	
qr max	Lange Ganzzahl	8	
qr min	Lange Ganzzahl	4	
qr standard deviation	Lange Ganzzahl	32	
qr sum	Lange Ganzzahl	1	

QR Rahmen

Konstante	Typ	Wert	Kommentar
qr bottom border	Lange Ganzzahl	8	Unterer Rand
qr inside horizontal border	Lange Ganzzahl	32	Innerer horizontaler Rand
qr inside vertical border	Lange Ganzzahl	16	Innerer vertikaler Rand
qr left border	Lange Ganzzahl	1	Linker Rand
qr right border	Lange Ganzzahl	4	Rechter Rand
qr top border	Lange Ganzzahl	2	Oberer Rand

QR Texteigenschaften

Konstante	Typ	Wert	Kommentar
_o_qr font	Lange Ganzzahl	1	Ab 4D v14 R3 überholt (qr font name verwenden)
qr alternate background color	Lange Ganzzahl	9	Nummer für wechselnde Hintergrundfarbe
qr background color	Lange Ganzzahl	8	Nummer für Hintergrundfarbe
qr bold	Lange Ganzzahl	3	Stilattribut für Fett (0 oder 1)
qr font name	Lange Ganzzahl	10	Schriftname, wie er z.B. vom Befehl **FONT LIST** zurückgegeben wird
qr font size	Lange Ganzzahl	2	Schriftgröße ausgedrückt in Punkten (9 bis 255)
qr italic	Lange Ganzzahl	4	Stilattribut für Kursiv (0 oder 1)
qr justification	Lange Ganzzahl	7	Attribut für Ausrichtung (0 = Standard, 1 = linksbündig, 2 = zentriert, 3 = rechtsbündig)
qr text color	Lange Ganzzahl	6	Attribut für Farbnummer (Lange Ganzzahl)
qr underline	Lange Ganzzahl	5	Stilattribut für Unterstrichen

QR Zeilen für Eigenschaften

Konstante	Typ	Wert	Kommentar
qr detail	Lange Ganzzahl	-2	Detailbereich des Berichts
qr footer	Lange Ganzzahl	-5	Seite Fußteil
qr grand total	Lange Ganzzahl	-3	Bereich Gesamtsumme
qr header	Lange Ganzzahl	-4	Seite Kopfteil
qr title	Lange Ganzzahl	-1	Titel des Berichts

Ressourcen Eigenschaften

Konstante	Typ	Wert	Kommentar
Changed resource bit	Lange Ganzzahl	1	
Changed resource mask	Lange Ganzzahl	2	
Locked resource bit	Lange Ganzzahl	4	
Locked resource mask	Lange Ganzzahl	16	
Preloaded resource bit	Lange Ganzzahl	2	
Preloaded resource mask	Lange Ganzzahl	4	
Protected resource bit	Lange Ganzzahl	3	
Protected resource mask	Lange Ganzzahl	8	
Purgeable resource bit	Lange Ganzzahl	5	
Purgeable resource mask	Lange Ganzzahl	32	
System heap resource bit	Lange Ganzzahl	6	
System heap resource mask	Lange Ganzzahl	64	

Schriften Typliste

Konstante	Typ	Wert	Kommentar
Favorite fonts	Lange Ganzzahl	1	*Schriften* enthält die Liste der bevorzugten Schriften. - Unter Windows: Liste mit den Namen der aktiven Schriftfamilien - Unter OS X: Liste mit Namen der Schriftfamilien aus der Sammlung, in Englisch genannt "Favorites", in Französisch "Favoris", in Deutsch "Favoriten", etc. Hat der Benutzer keine bevorzugten Schriften hinzugefügt, ist die Sammlung leer.
Recent fonts	Lange Ganzzahl	2	Schriften enthält die Liste der zuletzt benutzten Schriften, d.h. die während der 4D Sitzung verwendet werden. Diese Liste wird insbesondere für Textbereiche mit Mehrfachstil eingesetzt.
System fonts	Lange Ganzzahl	0	Schriften enthält die Liste aller Systemschriften. Standardoption, wenn *ListeTyp* weggelassen wird.

Schriftstile

Konstanten, die mit _O_ beginnen, sind veraltet und dürfen nicht mehr verwendet werden.

Konstante	Typ	Wert	Kommentar
_o_Condensed	Lange Ganzzahl	32	
_o_Extended	Lange Ganzzahl	64	
_o_Outline	Lange Ganzzahl	8	
_o_Shadow	Lange Ganzzahl	16	
Automatic style sheet	Zeichenkette	__automatic__	Wird standardmäßig für alle Objekte verwendet
Automatic style sheet_additional	Zeichenkette	__automatic_additional_text__	Gilt nur für statischen Text, Felder und Variablen für Zusatztext in Dialogfenstern.
Automatic style sheet_main	Zeichenkette	__automatic_main_text__	Gilt nur für statischen Text, Felder und Variablen für Haupttext in Dialogfenstern.
Bold	Lange Ganzzahl	1	
Bold and Italic	Lange Ganzzahl	3	
Bold and Underline	Lange Ganzzahl	5	
Italic	Lange Ganzzahl	2	
Italic and Underline	Lange Ganzzahl	6	
Plain	Lange Ganzzahl	0	
Underline	Lange Ganzzahl	4	

SCREEN DEPTH

Konstante	Typ	Wert	Kommentar
Black and white	Lange Ganzzahl	0	
Four colors	Lange Ganzzahl	2	
Is color	Lange Ganzzahl	1	
Is gray scale	Lange Ganzzahl	0	
Millions of colors 24 bit	Lange Ganzzahl	24	
Millions of colors 32 bit	Lange Ganzzahl	32	
Sixteen colors	Lange Ganzzahl	4	
Thousands of colors	Lange Ganzzahl	16	
Two fifty six colors	Lange Ganzzahl	8	

SET RGB COLORS

Konstante	Typ	Wert	Kommentar
Background color	Lange Ganzzahl	-2	
Background color none	Lange Ganzzahl	-16	Diese Konstante lässt sich in den Parametern *Hintergrundfarbe* und *altHintergrFarbe* verwenden.
Dark shadow color	Lange Ganzzahl	-3	
Disable highlight item color	Lange Ganzzahl	-11	
Foreground color	Lange Ganzzahl	-1	
Highlight menu background color	Lange Ganzzahl	-9	
Highlight menu text color	Lange Ganzzahl	-10	
Highlight text background color	Lange Ganzzahl	-7	
Highlight text color	Lange Ganzzahl	-8	
Light shadow color	Lange Ganzzahl	-4	

SQL

Konstante	Typ	Wert	Kommentar
SQL all records	Lange Ganzzahl	-1	
SQL asynchronous	Lange Ganzzahl	1	0 = Synchrone Verbindung (Standardwert), 1 (oder anderer Wert als 0) = Asynchrone Verbindung
SQL charset	Lange Ganzzahl	100	Textcodierung für Anfragen, die an externe Quellen gesendet werden (via SQL pass-through). Die Änderung wird für den aktuellen Prozess und die aktuelle Verbindung ausgeführt. Mögliche Werte: MIBEnum Identifier (siehe Hinweis 2) oder Wert -2 (siehe Hinweis 3) Standardwert: 106 (UTF-8)
SQL connection timeout	Lange Ganzzahl	5	Maximale Wartezeit für Antwort beim Ausführen des Befehls **SQL LOGIN**. Dieser Wert wird nur berücksichtigt, wenn er vor dem Öffnen der Verbindung gesetzt wurde. Mögliche Werte: Zeit in Sekunden Standardwert: Kein Timeout
SQL max data length	Lange Ganzzahl	3	Maximale Länge der zurückgegebenen Daten
SQL max rows	Lange Ganzzahl	2	Maximale Anzahl Zeilen in Ergebnisgruppe (verwendet für Vorschau)
SQL On error abort	Lange Ganzzahl	1	Bei einem Fehler stoppt 4D sofort die Ausführung des Skript.
SQL On error confirm	Lange Ganzzahl	2	Bei einem Fehler zeigt 4D ein Dialogfenster, das den Fehler beschreibt und dem Benutzer ermöglicht, die Skript-Ausführung abzubrechen oder weiter auszuführen.
SQL On error continue	Lange Ganzzahl	3	Bei einem Fehler ignoriert 4D diesen und fährt mit der Skript-Ausführung fort.
SQL param in	Lange Ganzzahl	1	
SQL param in out	Lange Ganzzahl	2	Ist nur im Rahmen einer SQL Serverprozedur verwendbar (in-out Parameter definiert in Serverprozedur)
SQL param out	Lange Ganzzahl	4	Ist nur im Rahmen einer SQL Serverprozedur verwendbar (in-out Parameter definiert in Serverprozedur)
SQL query timeout	Lange Ganzzahl	4	Maximale Wartezeit für Antwort beim Ausführen des Befehls **SQL EXECUTE**. Werte: Zeit in Sekunden Standardwert: kein Timeout
SQL use access rights	Zeichenkette	SQL_Use_Access_Rights	Dient zur Einschränkung der Zugriffsrechte, die während der Ausführung von SQL Befehlen des Skript gelten sollen. Verwenden Sie dieses Attribut, müssen Sie in *attrWert* 0 oder 1 verwenden. • *attrWert* = 1: 4D verwendet die Zugriffsrechte des aktuellen 4D Benutzers. • *attrWert* = 0 (oder Attribut nicht angegeben): 4D schränkt den Zugriff nicht ein, es gelten die Designer-Rechte.
SQL_INTERNAL	Zeichenkette	;DB4D_SQL_LOCAL;	
System data source	Lange Ganzzahl	2	
User data source	Lange Ganzzahl	1	

Standard System Signaturen

Die Standard System Signaturen sind 4-stellige Strings für standardmäßige Dateitypen, Ressourcentypen, standardmäßige Datentypen, gespeichert in der Zwischenablage, usw.

Konstante	Typ	Wert	Kommentar
Picture document	Zeichenkette	PICT	
Text document	Zeichenkette	TEXT	
Windows MIDI document	Zeichenkette	MID	
Windows sound document	Zeichenkette	WAV	
Windows video document	Zeichenkette	AVI	

Standardaktion

Konstante	Typ	Wert	Kommentar
_o_Object Accept action	Zeichenkette	2	
_o_Object Add subrecord action	Zeichenkette	14	
_o_Object Automatic splitter action	Zeichenkette	16	
_o_Object Cancel action	Zeichenkette	1	
_o_Object Clear action	Zeichenkette	21	
_o_Object Copy action	Zeichenkette	19	
_o_Object Cut action	Zeichenkette	18	
_o_Object Database Settings action	Zeichenkette	32	
_o_Object Delete record action	Zeichenkette	7	
_o_Object Delete subrecord action	Zeichenkette	13	
_o_Object Edit subrecord action	Zeichenkette	12	
_o_Object First page action	Zeichenkette	10	
_o_Object First record action	Zeichenkette	5	
_o_Object Goto page action	Zeichenkette	15	
_o_Object Last page action	Zeichenkette	11	
_o_Object Last record action	Zeichenkette	6	
_o_Object MSC action	Zeichenkette	36	
_o_Object Next page action	Zeichenkette	8	
_o_Object Next record action	Zeichenkette	3	
_o_Object No standard action	Zeichenkette	0	
_o_Object Open back URL action	Zeichenkette	37	
_o_Object Open next URL action	Zeichenkette	38	
_o_Object Paste action	Zeichenkette	20	
_o_Object Previous page action	Zeichenkette	9	
_o_Object Previous record action	Zeichenkette	4	
_o_Object Quit action	Zeichenkette	27	
_o_Object Redo action	Zeichenkette	31	
_o_Object Refresh current URL action	Zeichenkette	39	
_o_Object Return to Design mode action	Zeichenkette	35	
_o_Object Select all action	Zeichenkette	22	
_o_Object Show Clipboard action	Zeichenkette	23	
_o_Object Stop loading URL action	Zeichenkette	40	
_o_Object Test Application action	Zeichenkette	26	

Konstante	Typ	Wert	Kommentar
_o_Object Undo action	Zeichenkette	17	
ak accept	Zeichenkette	accept	
ak add subrecord	Zeichenkette	addSubrecord	
ak automatic splitter	Zeichenkette	automaticSplitter	
ak background color	Zeichenkette	backgroundColor	Zeigt das Untermenü für Standard Hintergrundfarbe
ak background color dialog	Zeichenkette	backgroundColor/showDialog	Öffnet das Dialogfenster für Hintergrundfarbe für Schrift
ak cancel	Zeichenkette	cancel	
ak clear	Zeichenkette	clear	Ziel dieser Aktion ist immer das Objekt mit Fokus über die Tastatur
ak compute expressions	Zeichenkette	computeExpressions	Aktualisiert alle dynamischen Ausdrücke im Bereich
ak copy	Zeichenkette	copy	Ziel dieser Aktion ist immer das Objekt mit Fokus über die Tastatur
ak current form	Lange Ganzzahl	1	Aktuelles Formular ist das Formular, wo die Aktion aufgerufen wurde. Das kann das Hauptformular oder ein Palettenfenster vor dem Hauptformular des aktuellen Prozesses sein.
ak cut	Zeichenkette	cut	Ziel dieser Aktion ist immer das Objekt mit Fokus über die Tastatur
ak database settings	Zeichenkette	databaseSettings	Zeigt das Standard Dialogfenster Datenbank-Eigenschaften
ak delete record	Zeichenkette	deleteRecord	
ak delete subrecord	Zeichenkette	deleteSubrecord	
ak display subrecord	Zeichenkette	displaySubrecord	
ak edit subrecord	Zeichenkette	editSubrecord	
ak first page	Zeichenkette	firstPage	
ak first record	Zeichenkette	firstRecord	
ak font bold	Zeichenkette	fontBold	Schaltet um auf das Attribut Fettschrift
ak font color	Zeichenkette	color	Attribut Schriftfarbe
ak font color dialog	Zeichenkette	color/showDialog	Zeigt das Dialogfenster des Systems für Schriftfarbe
ak font italic	Zeichenkette	fontItalic	Schaltet um auf das Attribut Kursivschrift
ak font linethrough	Zeichenkette	fontLinethrough	Schaltet um auf das Attribut Schrift Durchgestrichen
ak font show dialog	Zeichenkette	font/showDialog	Zeigt das Dialogfenster für Systemschriften
ak font size	Zeichenkette	fontSize	Attribut Schriftgröße
ak font style	Zeichenkette	fontStyle	Zeigt das Untermenü für Standard Schriftstil
ak font underline	Zeichenkette	fontUnderline	Übergibt das Attribut Schrift Unterstrichen
ak freeze expressions	Zeichenkette	freezeExpressions	Friert alle dynamischen Ausdrücke im Bereich ein
ak goto page	Zeichenkette	gotoPage	Parameter: "?value=Seitennummer"
ak last page	Zeichenkette	lastPage	
ak last record	Zeichenkette	lastRecord	
ak main form	Lange Ganzzahl	2	Hauptformular ist das vorderste Dokument oder Dialogfenster des Prozesses, ohne Paletten- oder PopUp-Fenster.
ak msc	Zeichenkette	msc	Zeigt das Fenster **Maintenance und Security Center**
ak next page	Zeichenkette	nextPage	
ak next record	Zeichenkette	nextRecord	
ak none	Zeichenkette	""	
ak open back url	Zeichenkette	openBackURL	Öffnet die vorherige URL in der Browser Sequenz, die vom Benutzer im Web Bereich ausgeführt wurde.
ak open forward url	Zeichenkette	openForwardURL	Öffnet die nächste URL in der Browser Sequenz, die vom Benutzer im Web Bereich ausgeführt wurde
ak paste	Zeichenkette	paste	Ziel dieser Aktion ist immer das Objekt mit Fokus über die Tastatur
ak previous page	Zeichenkette	previousPage	
ak previous record	Zeichenkette	previousRecord	
ak quit	Zeichenkette	quit	Zeigt eine Meldung "Sind Sie sicher?" und beendet die 4D Anwendung, wenn diese bestätigt wird.
ak redo	Zeichenkette	redo	Ziel dieser Aktion ist immer das Objekt mit Fokus über die Tastatur
ak refresh current url	Zeichenkette	refreshCurrentURL	Lädt erneut den aktuellen Inhalt des Web Bereichs

Konstante	Typ	Wert	Kommentar
ak return to design mode	Zeichenkette	design	Bringt die Fenster und Menüleisten der 4D Designumgebung nach vorne.
ak select all	Zeichenkette	selectAll	Ziel dieser Aktion ist immer das Objekt mit Fokus über die Tastatur
ak show clipboard	Zeichenkette	showClipboard	
ak show reference	Zeichenkette	visibleReferences	Zeigt alle dynamischen Ausdrücke im Bereich als Referenzen an
ak standard action title	Zeichenkette	4D_action_title	Standardmäßig lokalisierter Titel der Standardaktion. Enthält lokalisierten Aktionsnamen und u.U. weitere Informationen, z.B. "Rückgängig <vorige Aktion>".
ak stop loading url	Zeichenkette	stopLoadingURL	Stoppt das Laden der Seite bzw. Objekte der aktuellen URL im Web Bereich
ak undo	Zeichenkette	undo	Ziel dieser Aktion ist immer das Objekt mit Fokus über die Tastatur

Strings

Konstante	Typ	Wert	Kommentar
sk ignore empty strings	Lange Ganzzahl	1	Leere Strings aus der resultierenden Collection entfernen (sie werden ignoriert)
sk trim spaces	Lange Ganzzahl	2	Leerzeichen am Anfang und Ende von Unterstrings wegnehmen

Suchen

Konstante	Typ	Wert	Kommentar
Description in text format	Lange Ganzzahl	0	
Description in XML format	Lange Ganzzahl	1	
Into current selection	Lange Ganzzahl	0	
Into named selection	Lange Ganzzahl	2	
Into set	Lange Ganzzahl	1	
Into variable	Lange Ganzzahl	3	

Systemdokumente

Konstante	Typ	Wert	Kommentar
Absolute path	Lange Ganzzahl	2	Das Array Dokumente enthält absolute Pfadnamen
Alias selection	Lange Ganzzahl	8	Ermöglicht unter Windows eine Verknüpfung, auf Mac OS ein Alias als Dokument aufzurufen. Wird diese Konstante beim Auswählen einer Verknüpfung bzw. eines Alias nicht verwendet, gibt die Funktion den Pfad des Zielelements zurück (Standardeinstellung). Übergeben Sie die Konstante, gibt die Funktion den Pfad der Verknüpfung bzw. des Alias direkt zurück.
Delete only if empty	Lange Ganzzahl	0	Löscht den Ordner nur, wenn er leer ist
Delete with contents	Lange Ganzzahl	1	Löscht den Ordner mitsamt seinem Inhalt
Document unchanged	Lange Ganzzahl	0	Keine Bearbeitung
Document with CR	Lange Ganzzahl	3	Zeilenumbrüche werden in das Mac OS Format konvertiert: CR (carriage return)
Document with CRLF	Lange Ganzzahl	2	Zeilenumbrüche werden in das Windows Format konvertiert: CRLF (carriage return + line feed)
Document with LF	Lange Ganzzahl	4	Zeilenumbrüche werden in das Unix Format konvertiert: LF (line feed)
Document with native format	Lange Ganzzahl	1	(Standard) Zeilenumbrüche werden in das native Format des Betriebssystems konvertiert: CR (carriage return auf Mac OS), CRLF (carriage return + line feed unter Windows)
File name entry	Lange Ganzzahl	32	Erlaubt dem Benutzer, einen Dateinamen im Dialogfenster "Speichern unter" einzugeben. Es wird keine Datei gesichert, der Entwickler muss als Reaktion auf diese Aktion selbst eine Datei erstellen (die Systemvariable *Document* wird aktualisiert). In diesem Kontext kann der Parameter *Verzeichnis* den Pfad zu einer Datei anstatt zu einem Verzeichnis enthalten. Der Dateiname wird im Textbereich "Speichern unter" als Name vorgeschlagen. Das dazugehörige Verzeichnis wird als Standardpfad verwendet.
Folder separator	Zeichenkette	Windows="\" Mac OS=":"	Diese Konstante hat einen anderen Wert, abhängig vom Betriebssystem, das sie aufruft. Damit können Sie gültige Pfadnamen erstellen, ohne das Betriebssystem zu berücksichtigen. **Hinweis:** Diese Konstante gilt nur für 4D Anwendungen im Unicode Modus. Im nicht-Unicode Kompatibilitätsmodus wird sie nicht unterstützt
Get pathname	Lange Ganzzahl	3	
Ignore invisible	Lange Ganzzahl	8	Unsichtbare Dokumente werden nicht aufgelistet
Is a document	Lange Ganzzahl	1	
Is a folder	Lange Ganzzahl	0	
Multiple files	Lange Ganzzahl	1	Ermöglicht, mehrere Dateien auszuwählen; für eine zusammenhängende Auswahl über die Kombination **Umschalttaste + Klick**, für eine unterbrochene Auswahl unter Windows über die **Strg-Taste + Klick**, auf Mac OS die **Befehlstaste + Klick**. In diesem Fall enthält der übergebene Parameter die Liste mit allen gewählten Dateien. Wird diese Konstante nicht verwendet, ermöglicht die Funktion standardmäßig nicht, mehrerer Dateien auszuwählen.
Package open	Lange Ganzzahl	2	(nur Mac OS) ermöglicht, „Packages" als Ordner zu öffnen und so ihren Inhalt anzusehen bzw. auszuwählen. Wird diese Konstante nicht verwendet, ermöglicht die Funktion standardmäßig kein Öffnen von Packages.
Package selection	Lange Ganzzahl	4	(nur Mac OS) ermöglicht, „Packages" als Einheit auszuwählen. Wird diese Konstante nicht verwendet, ermöglicht die Funktion standardmäßig keine Auswahl von Software-Paketen.
Path is POSIX	Lange Ganzzahl	1	Der Pfad wird in Posix Syntax angegeben.
Path is system	Lange Ganzzahl	0	(Standard) Der Pfad wird in der Syntax des aktuellen Systems angegeben (Windows oder macOS).
Posix path	Lange Ganzzahl	4	Das Array Dokumente enthält Pfadnamen im Posix Format
Read and write	Lange Ganzzahl	0	Standardmodus
Read mode	Lange Ganzzahl	2	
Recursive parsing	Lange Ganzzahl	1	Das Array Dokumente enthält alle Dateien und Unterordner des angegebenen Ordners
Use sheet window	Lange Ganzzahl	16	(nur Mac OS): Zeigt das Auswahlfenster in Form eines Sheet Fensters. Diese Option wird unter Windows ignoriert. Sheet Fenster sind spezifisch für die Mac OS X Oberfläche, da diese grafische Animation zulässt. Weitere Informationen dazu finden Sie im Abschnitt **Fenstertypen (Kompatibilität)**. Wird diese Konstante nicht verwendet, zeigt die Funktion das Standard Dialogfenster.
Write mode	Lange Ganzzahl	1	

Systemformat

Konstante	Typ	Wert	Kommentar
Currency symbol	Lange Ganzzahl	2	Währungszeichen, z.B."€"
Date separator	Lange Ganzzahl	13	Trenner für Datumsformate, z.B. “.”
Decimal separator	Lange Ganzzahl	0	Dezimaltrenner, z.B. “,”
Short date day position	Lange Ganzzahl	15	Position des Tages für kurzes Datumsformat: “1” = links, “2” = mittig, “3” = rechts
Short date month position	Lange Ganzzahl	16	Position des Monats für kurzes Datumsformat: “1” = links, “2” = mittig, “3” = rechts
Short date year position	Lange Ganzzahl	17	Position des Jahres für kurzes Datumsformat: “1” = links, “2” = mittig, “3” = rechts
System date long pattern	Lange Ganzzahl	8	Anzeigeformat für langes Datumsformat in Form von “dddd MMMM yyyy”
System date medium pattern	Lange Ganzzahl	7	Anzeige für mittleres Datumsformat in Form von “dddd MMMM yyyy”
System date short pattern	Lange Ganzzahl	6	Anzeige für kurzes Datumsformat in Form von “dddd MMMM yyyy”
System time AM label	Lange Ganzzahl	18	Zusatzbezeichnung für Zeit vor Mittag in 12-Stunden Formaten, z.B. “Morgen”
System time long pattern	Lange Ganzzahl	5	Anzeige für langes Zeitformat in Form von “HH:MM:SS”
System time medium pattern	Lange Ganzzahl	4	Anzeige für mittleres Zeitformat in Form von “HH:MM:SS”
System time PM label	Lange Ganzzahl	19	Zusatzbezeichnung für Zeit nach Mittag in 12-Stunden Formaten, z.B. “Abend”
System time short pattern	Lange Ganzzahl	3	Anzeige für kurzes Zeitformat in Form von “HH:MM:SS”
Thousand separator	Lange Ganzzahl	1	Trenner für Tausend, z.B “.”
Time separator	Lange Ganzzahl	14	Trenner für Zeitformate, z.B “:”

Systemordner

Konstante	Typ	Wert	Kommentar
_o_Mac control panels	Lange Ganzzahl	11	
_o_Mac extensions	Lange Ganzzahl	10	
_o_Mac shutdown items_all	Lange Ganzzahl	6	
_o_Mac shutdown items_user	Lange Ganzzahl	7	
Applications or program files	Lange Ganzzahl	16	
Desktop	Lange Ganzzahl	15	
Documents folder	Lange Ganzzahl	17	Benutzerordner "Documents"
Favorites Win	Lange Ganzzahl	14	
Fonts	Lange Ganzzahl	1	
Start menu Win_all	Lange Ganzzahl	8	
Start menu Win_user	Lange Ganzzahl	9	
Startup Win_all	Lange Ganzzahl	4	
Startup Win_user	Lange Ganzzahl	5	
System	Lange Ganzzahl	0	
System Win	Lange Ganzzahl	12	
System32 Win	Lange Ganzzahl	13	
User preferences_all	Lange Ganzzahl	2	
User preferences_user	Lange Ganzzahl	3	

Tage und Monate

Konstante	Typ	Wert	Kommentar
April	Lange Ganzzahl	4	
August	Lange Ganzzahl	8	
December	Lange Ganzzahl	12	
February	Lange Ganzzahl	2	
Friday	Lange Ganzzahl	6	
January	Lange Ganzzahl	1	
July	Lange Ganzzahl	7	
June	Lange Ganzzahl	6	
March	Lange Ganzzahl	3	
May	Lange Ganzzahl	5	
Monday	Lange Ganzzahl	2	
November	Lange Ganzzahl	11	
October	Lange Ganzzahl	10	
Saturday	Lange Ganzzahl	7	
September	Lange Ganzzahl	9	
Sunday	Lange Ganzzahl	1	
Thursday	Lange Ganzzahl	5	
Tuesday	Lange Ganzzahl	3	
Wednesday	Lange Ganzzahl	4	

Tastaturabkürzungen

Konstante	Typ	Wert	Kommentar
Shortcut with Backspace	Zeichenkette	[backspace]	
Shortcut with Carriage Return	Zeichenkette	[return]	
Shortcut with Delete	Zeichenkette	[del]	
Shortcut with Down arrow	Zeichenkette	[down arrow]	
Shortcut with End	Zeichenkette	[end]	
Shortcut with Enter	Zeichenkette	[enter]	
Shortcut with Escape	Zeichenkette	[esc]	
Shortcut with F1	Zeichenkette	[F1]	
Shortcut with F10	Zeichenkette	[F10]	
Shortcut with F11	Zeichenkette	[F11]	
Shortcut with F12	Zeichenkette	[F12]	
Shortcut with F13	Zeichenkette	[F13]	
Shortcut with F14	Zeichenkette	[F14]	
Shortcut with F15	Zeichenkette	[F15]	
Shortcut with F2	Zeichenkette	[F2]	
Shortcut with F3	Zeichenkette	[F3]	
Shortcut with F4	Zeichenkette	[F4]	
Shortcut with F5	Zeichenkette	[F5]	
Shortcut with F6	Zeichenkette	[F6]	
Shortcut with F7	Zeichenkette	[F7]	
Shortcut with F8	Zeichenkette	[F8]	
Shortcut with F9	Zeichenkette	[F9]	
Shortcut with Help	Zeichenkette	[help]	
Shortcut with Home	Zeichenkette	[home]	
Shortcut with Left arrow	Zeichenkette	[left arrow]	
Shortcut with Page down	Zeichenkette	[page down]	
Shortcut with Page up	Zeichenkette	[page up]	
Shortcut with Right arrow	Zeichenkette	[right arrow]	
Shortcut with Tabulation	Zeichenkette	[tab]	
Shortcut with Up arrow	Zeichenkette	[up arrow]	

TCP Port Nummern

Konstante	Typ	Wert	Kommentar
TCP Authentication	Lange Ganzzahl	113	
TCP DNS	Lange Ganzzahl	53	
TCP Finger	Lange Ganzzahl	79	
TCP FTP Control	Lange Ganzzahl	21	
TCP FTP Data	Lange Ganzzahl	20	
TCP Gopher	Lange Ganzzahl	70	
TCP HTTP WWW	Lange Ganzzahl	80	
TCP IMAP3	Lange Ganzzahl	220	
TCP Kerberos	Lange Ganzzahl	88	
TCP KLogin	Lange Ganzzahl	543	
TCP Nickname	Lange Ganzzahl	43	
TCP NNTP	Lange Ganzzahl	119	
TCP NTalk	Lange Ganzzahl	518	
TCP NTP	Lange Ganzzahl	123	
TCP PMCP	Lange Ganzzahl	1643	
TCP PMD	Lange Ganzzahl	1642	
TCP POP3	Lange Ganzzahl	110	
TCP Printer	Lange Ganzzahl	515	
TCP RADACCT	Lange Ganzzahl	1646	
TCP RADIUS	Lange Ganzzahl	1645	
TCP Remote Cmd	Lange Ganzzahl	514	
TCP Remote Exec	Lange Ganzzahl	512	
TCP Remote Login	Lange Ganzzahl	513	
TCP Router	Lange Ganzzahl	520	
TCP SMTP	Lange Ganzzahl	25	
TCP SNMP	Lange Ganzzahl	161	
TCP SNMPTRAP	Lange Ganzzahl	162	
TCP SUN RPC	Lange Ganzzahl	111	
TCP Talk	Lange Ganzzahl	517	
TCP Telnet	Lange Ganzzahl	23	
TCP TFTP	Lange Ganzzahl	69	
TCP UUCP	Lange Ganzzahl	540	
TCP UUCP RLOGIN	Lange Ganzzahl	541	

Trigger Ereignisse

Konstante	Typ	Wert	Kommentar
_o_On Loading Record Event	Lange Ganzzahl	4	
On Deleting Record Event	Lange Ganzzahl	3	
On Saving Existing Record Event	Lange Ganzzahl	2	
On Saving New Record Event	Lange Ganzzahl	1	

Verknüpfungen

Konstante	Typ	Wert	Kommentar
Automatic	Lange Ganzzahl	3	
Do not modify	Lange Ganzzahl	0	
Manual	Lange Ganzzahl	2	
No relation	Lange Ganzzahl	0	
Structure configuration	Lange Ganzzahl	1	

Web Area

Konstante	Typ	Wert	Kommentar
WA enable contextual menu	Lange Ganzzahl	4	Erlaubt die Anzeige eines Standard Kontextmenü im Web Bereich
WA enable Java applets	Lange Ganzzahl	1	Erlaubt die Ausführung von Java applets im Web Bereich.
WA enable JavaScript	Lange Ganzzahl	2	Erlaubt die Ausführung von JavaScript Code im Web Bereich
WA enable plugins	Lange Ganzzahl	3	Erlaubt die Installation von Plug-Ins im Web Bereich
WA enable URL drop	Lange Ganzzahl	101	Erlaubt Ziehen einer URL oder Datei in Web Area (Standard = **False**)
WA enable Web inspector	Lange Ganzzahl	100	Erlaubt die Anzeige des Web Inspektors im Web Bereich
WA next URLs	Lange Ganzzahl	1	
WA previous URLs	Lange Ganzzahl	0	

Web Server

Konstante	Typ	Wert	Kommentar
wdl disable	Lange Ganzzahl	0	Web HTTP Fehlerprotokoll ist deaktiviert
wdl enable with all body parts	Lange Ganzzahl	7	Web HTTP Fehlerprotokoll ist aktiviert mit Body Bereichen der Anfrage und der Antwort
wdl enable with request body	Lange Ganzzahl	5	Web HTTP Fehlerprotokoll ist aktiviert nur mit Body Bereichen der Anfrage
wdl enable with response body	Lange Ganzzahl	3	Web HTTP Fehlerprotokoll ist aktiviert nur mit Body Bereichen der Antwort
wdl enable without body	Lange Ganzzahl	1	Web HTTP Fehlerprotokoll ist aktiviert ohne Body Bereiche (in diesem Fall wird die Body Größe angezeigt)
Web character set	Lange Ganzzahl	17	**Reichweite:** 4D lokal, 4D Server **Beschreibung:** Damit können Sie sofort den Zeichensatz ändern, den der 4D Web Server mit 4D im lokalen Modus und 4D Server für die Kommunikation mit Browsern verwenden soll, die sich an die Datenbank anmelden. Die aktuelle Standardeinstellung richtet sich nach der Sprache des Betriebssystems. Dieser Parameter wird in den Einstellungen der Datenbank festgelegt. **Mögliche Werte:** Der Wert richtet sich nach dem jeweiligen Ausführungsmodus der Datenbank • **Unicode Modus:** Wird die Anwendung im Modus Unicode ausgeführt, müssen jetzt für diesen Parameter Zeichensatz Identifier übergeben werden. Das sind Identifier vom Typ MIBEnum Lange Ganzzahl oder Name String, definiert IANA (siehe unter http://www.iana.org/assignments/character-sets) Nachfolgend sehen Sie die Liste der Identifier, der dem von 4D Web Server unterstützten Zeichensatz entspricht: 4 = ISO-8859-1 12 = ISO-8859-9 13 = ISO-8859-10 17 = Shift-JIS 2026 = Big5 38 = euc-kr 106 = UTF-8 2024 = Windows-31J 2250 = Windows-1250 2251 = Windows-1251 2253 = Windows-1253 2255 = Windows-1255 2256 = Windows 1256 • **ASCII Kompatibilitätsmodus** 0: Western European 1: Japanisch 2: Chinesisch 3: Koreanisch 4: Benutzerdefiniert 5: Reserviert 6: Central European 7: Kyrillisch 8: Arabisch 9: Griechisch 10: Hebräisch 11: Türkisch 12: Baltisch
Web Client IP address to listen	Lange Ganzzahl	23	**Reichweite**: Alle 4D remote Rechner **Mögliche Werte**: Siehe Selector 16 **Beschreibung**: Dient zur Angabe dieses Parameters für alle Rechner mit remote 4D, die als Web Server verwendet werden (gilt nur für den remote Web Server).
Web debug log	Lange Ganzzahl	84	**Reichweite:** Lokaler Web Server **Hinweis:** Nach Neustart des HTTP Server wird ein neues Protokoll verwendet. **Beschreibung:** Ermöglicht, den Status des HTTP Anfrageprotokolls des 4D Web Server zu erhalten oder zu setzen. Ist er aktiviert, wird diese Datei mit Namen "**HTTPDebugLog_*nn*.txt**", im Ordner "Logs" der Anwendung gespeichert (*nn* ist die Dateinummer). Das ist hilfreich zum Debuggen bei Problemen mit dem Web Server. Die Datei protokolliert jede Anfrage und Antwort im Rohmodus. Die Anfragen werden mit Kopfteilen protokolliert; optional lassen sich auch Body-Bereiche mitprotokollieren. Weitere Informationen dazu finden Sie im **Anhang E: Beschreibung der Protokolldateien**. **Werte**: Eine der Konstanten mit der Vorsilbe "wdl" (siehe Beschreibung dieser Konstanten unter diesem Thema). **Standardwert**: 0 (nicht aktiviert)

Konstante	Typ	Wert	Kommentar
Web HSTS enabled	Lange Ganzzahl	86	**Reichweite**: 4D lokal, 4D Server. **Beschreibung**: HTTP Strict Transport Security (HSTS) Status. Über HSTS kann der 4D Web Server festlegen, dass Browser nur über sichere HTTPS Verbindungen mit ihm kommunizieren. Ist HSTS aktiviert, fügt der 4D Web Server automatisch in allen Antwort-Kopfteilen HSTS-bezogene Information hinzu. Browser speichern diese Information, wenn sie zum ersten Mal eine Antwort vom 4D Web Server empfangen. Alle nachfolgenden HTTP Anfragen werden dann automatisch in HTTPS Anfragen umgewandelt. Mit dem Selektor Web HSTS max age lässt sich angeben, wie lange diese Information vom Browser gespeichert wird. Für HSTS muss HTTPS auf dem Server aktiviert werden. HTTP muss ebenfalls aktiviert werden, um das erste Starten von Client Verbindungen zuzulassen. **Mögliche Werte**: 0 (deaktiviert, Standard) oder 1 (aktiviert) **Hinweis:** Die Einstellung wird erst nach Neustart des 4D Web Server übernommen.
Web HSTS max age	Lange Ganzzahl	87	**Reichweite**: 4D lokal, 4D Server **Beschreibung**: Gibt die maximale Zeitspanne an (in Sekunden), die HSTS für jede neue Client-Verbindung aktiv bleibt. Diese Angabe wird auf Client-Seite für die angegebene Dauer gespeichert. **Mögliche Werte**: Lange Ganzzahl (Sekunden) **Standardwert**: 63072000 (2 Jahre) ***Warnung:*** *Ist HSTS aktiviert, verwenden Client Verbindungen diesen Mechanismus für die angegebene Dauer. Beim Testen Ihrer Anwendung empfehlen wir, einer kurze Dauer zu setzen, damit Sie bei Bedarf zwischen sicheren und nicht-sicheren Verbindungen wechseln können.*
Web HTTP compression level	Lange Ganzzahl	50	**Reichweite:** Lokaler Web Server **Mögliche Werte:** 1 bis 9 (1 = schneller, 9 = stärker komprimiert), -1 = beste Kombination **Beschreibung:** Setzt die Komprimierungsebene für jeden komprimierten HTTP Austausch für den 4D HTTP Server (Client Anfragen oder Server Antworten, Web und Web Service). Mit diesem Selektor können Sie den Austausch entweder über die Ausführungsgeschwindigkeit (weniger Komprimierung) oder die Komprimierungsmenge (weniger Geschwindigkeit) optimieren. Die Auswahl des passenden Wertes richtet sich nach der Größe und der Art der ausgetauschten Daten. Im Parameter *Wert* können Sie einen Wert von 1 bis 9 übergeben, wobei 1 die schnellste und 9 die höchste Komprimierung ist. Für einen Kompromiss zwischen Geschwindigkeit und Komprimierungsrate übergeben Sie -1. Standardmäßig ist als Komprimierungsebene 1 eingestellt (schnellere Komprimierung).
Web HTTP compression threshold	Lange Ganzzahl	51	**Reichweite:** Lokaler HTTP Server **Mögliche Werte:** Jeder Wert vom Typ Lange Ganzzahl **Beschreibung:** Setzt den Schwellwert, bis zu dem der Austausch von Daten im Rahmen von 4D Web Services nicht komprimiert werden soll. Diese Einstellung ist hilfreich, um beim Austausch geringer Datenmengen zu verhindern, dass Rechenzeit für die Komprimierung beansprucht wird. In *Wert* übergeben Sie eine Größe in Bytes. Standardmäßig ist als Schwellwert für Komprimierung 1024 Bytes eingestellt.
Web HTTP enabled	Lange Ganzzahl	88	**Reichweite**: 4D lokal, 4D Server **Beschreibung**: Status für Kommunikation über HTTP **Mögliche Werte**: 0 (deaktiviert) oder 1 (aktiviert)
Web HTTP TRACE	Lange Ganzzahl	85	**Reichweite**: Lokaler Web Server **Wird zwischen 2 Sitzungen beibehalten**: Nein **Beschreibung**: Ermöglicht, die Methode HTTP TRACE im 4D Web Server zu aktivieren/deaktivieren. Ab 4D v15 R2 weist der 4D Web Server aus Sicherheitsgründen HTTP TRACE Anfragen standardmäßig mit dem Fehler 405 ab (HTTP TRACE ist deaktiviert). Bei Bedarf können Sie die Methode HTTP TRACE für die Sitzung aktivieren, indem Sie für diese Konstante den Wert 1 übergeben. Dann sendet der 4D Web Server bei HTTP TRACE Anfragen die Anfragezeile, Kopfteil und Hauptteil. **Mögliche Werte**: 0 (deaktiviert) oder 1 (aktiviert) **Standardwert**: 0 (deaktiviert)
Web HTTPS enabled	Lange Ganzzahl	89	**Reichweite**: 4D lokal, 4D Server **Beschreibung**: Status für Kommunikation über HTTPS. **Mögliche Werte**: 0 (deaktiviert) oder 1 (aktiviert)
Web HTTPS port ID	Lange Ganzzahl	39	**Reichweite:** 4D lokal, 4D Server **Mögliche Werte:** 0 bis 65535 **Beschreibung:** Mit diesem Selector können Sie per Programmierung die TCP Portnummer ändern, die der Web Server von 4D im lokalen Modus und von 4D Server für sichere Verbindungen via TLS (HTTPS protocol) verwendet. Die HTTPS Portnummer wird in den Datenbank-Eigenschaften auf der Seite **Web>Konfiguration** gesetzt. Der Wert ist standardmäßig 443. Für den Parameter *Wert* können Sie Konstanten unter dem Thema **TCP Port Nummern** einsetzen.
Web inactive process timeout	Lange Ganzzahl	78	**Reichweite:** Lokaler Web Server **Mögliche Werte:** Lange Ganzzahl (Minuten) **Beschreibung:** Ändert die Lebensdauer des inaktiven Prozesses, der Sessions zugeordnet ist. Am Ende des Timeout wird der Prozess auf dem Server gestoppt, die **Datenbankmethode On Web Close Process** wird aufgerufen, dann wird der Session-Kontext gelöscht. **Standardwert:** 480 Minuten (zum Wiederherstellen des Standardwerts 0 übergeben)
Web inactive session timeout	Lange Ganzzahl	72	**Reichweite:** Lokaler Web Server **Beschreibung:** Ändert die Lebensdauer inaktiver Sessions (Dauer wird in Cookie gesetzt). Endet diese Periode, läuft das Cookie der Session ab und wird nicht mehr vom HTTP Client gesendet. **Mögliche Werte:** Lange Ganzzahl (Minuten) **Standardwert:** 480 Minuten (zum Wiederherstellen des Standardwerts 0 übergeben)

Konstante	Typ	Wert	Kommentar
Web IP address to listen	Lange Ganzzahl	16	**Reichweite:** 4D lokal, 4D Server **Beschreibung:** IP Adresse, unter der der 4D Web Server HTTP Anfragen mit 4D im lokalen Modus und 4D Server empfängt. Standardmäßig ist keine bestimmte Adresse definiert (Wert = 0). Dieser Parameter lässt sich in den Datenbank-Eigenschaften auf der Seite **Web>Konfiguration** festlegen. Dieser Selector ist hilfreich für 4D Web Server mit einkompilierter 4D Volume Desktop, die keinen Zugriff auf den Designmodus zulassen. **Mögliche Werte**: IP Adresse String. Beide Formate werden unterstützt: IPv6, z.B. "2001:0db8:0000:0000:0000:ff00:0042:8329") und IPv4 (z.B. "123.45.67.89"). **Hinweis:** Zur Wahrung der Kompatibilität werden überholte IPv4 Adressen in Form von hexadezimalen Langen Ganzzahlen noch unterstützt.
Web keep session	Lange Ganzzahl	70	**Reichweite:** Lokaler Web Server **Beschreibung:** Aktiviert oder deaktiviert den neuen Modus zum Verwalten der Sessions. Weitere Informationen dazu finden Sie im Abschnitt **Web Sessions verwalten** **Mögliche Werte:** 1 (Modus aktivieren) oder 0 (Modus deaktivieren) **Standardwert:** 1 für mit v13 erstellte Datenbanken, 0 für konvertierte Datenbanken. Beachten Sie, dass dieser Modus auch den Mechanismus zum Wiederverwenden temporärer Kontexte im remote Modus ermöglicht. Weitere Informationen dazu finden Sie im Abschnitt **Web Server, Einstellungen**
Web log recording	Lange Ganzzahl	29	**Reichweite:** 4D lokal, 4D Server **Beschreibung:** Startet oder stoppt das Speichern von Web Anfragen, die vom Web Server von 4D im lokalen Modus oder 4D Server empfangen werden. Der Standardwert ist 0, d.h. Anfragen werden nicht gespeichert. Das Logbuch von Web Anfragen wird als Textdatei mit Namen “logweb.txt” gespeichert, die automatisch in den Ordner Logs neben der Strukturdatei der Anwendung gesetzt wird. Das Format dieser Datei richtet sich nach dem übergebenen Wert. Weitere Informationen zu Formaten für Web Logfiles finden Sie im Abschnitt **Information über die Web Site** . Diese Datei lässt sich auch in den Datenbank-Eigenschaften auf der Seite **Log (Typ)** aktivieren. **Mögliche Werte:** 0 = Nicht speichern (Standard), 1 = In CLF Format speichern, 2 = In DLF Format speichern, 3 = In ELF Format speichern, 4 = In WLF Format speichern. **Warnung:** Format 3 und 4 sind individuell anpassbare Formate, d.h. Sie müssen den Typ zuvor in den Datenbank-Eigenschaften auf der Seite **Log (Typ)** definieren. Verwenden Sie diese Formate, ohne zuvor den Typ festzulegen, wird kein Logbuch angelegt.
Web max concurrent processes	Lange Ganzzahl	18	**Reichweite:** 4D lokal, 4D Server **Werte:** Sie können jeden Wert zwischen 10 und 32 000 übergeben. Der Standardwert ist 100. **Beschreibung:** Damit setzen Sie die maximale Anzahl aller gleichzeitig laufenden Web Prozesse (kontextuell, nicht kontextuell oder die zum Pool der Prozesse gehören), die der 4D Web Server mit 4D im lokalen Modus und 4D Server unterstützt. Ist die maximale Anzahl erreicht, erstellt 4D keinen weiteren Prozess und gibt den HTTP Status 503 zurück - Dienst für weitere neue Anfragen nicht verfügbar. Dieser Parameter verhindert die Übersättigung des 4D Web Server. Sie kann eintreten, wenn gleichzeitig eine zu große Anzahl an Anfragen gesendet wird oder zu viele Kontext-Erstellungen angefordert werden. Theoretisch ist die max. Anzahl der Web Prozesse das Ergebnis der folgenden Formel: Verfügbarer Speicher/Stapelgröße der Web Prozesse. Sie können sich auch die Information über die Web Prozesse im Runtime Explorer ansehen: Er zeigt die aktuelle Anzahl der Web Prozesse und die max. erreichte Anzahl seit dem Hochfahren des Web Server an.
Web max sessions	Lange Ganzzahl	71	**Reichweite:** Lokaler Web Server **Beschreibung:** Begrenzt die Anzahl gleichzeitiger Sessions. Bei Erreichen des Limits wird die älteste Session geschlossen (die **Datenbankmethode On Web Close Process** wird aufgerufen), wenn der Web Server eine neue erstellen muss. **Mögliche Werte:** Lange Ganzzahl. Die Anzahl gleichzeitiger Sessions kann nicht größer sein als die Gesamtzahl der Web Prozesse (Option Web Max Concurrent Processes, standardmäßig 100). **Standardwert:** 100 (zum Wiederherstellen des Standardwerts 0 übergeben)
Web maximum requests size	Lange Ganzzahl	27	**Reichweite:** 4D lokal, 4D Server **Mögliche Werte:** 500 000 bis 2 147 483 648 **Beschreibung:** Maximale Größe (in Bytes) hereinkommender HTTP Anfragen (POST), die der Web Server akzeptiert. Standardmäßig ist der Wert 2 000 000 vorgegeben, z.B. etwas unter 2 MB. Übergeben Sie den maximalen Wert (2 147 483 648), wird praktisch keine Grenze gesetzt. Die Begrenzung sorgt dafür, dass der Web Server nicht überlastet wird durch zu große eingehende Anfragen. Erreicht eine Anfrage den Grenzwert, weist der 4D Web Server diese zurück.
Web port ID	Lange Ganzzahl	15	**Reichweite:** 4D im lokalen Modus und 4D Server **Beschreibung:** Setzt oder erhält die Nummer des TCP Port, den 4D Web Server mit 4D im lokalen Modus und mit 4D Server (Lange Ganzzahl) verwendet. Standardmäßig ist der Wert 80. Die TCP Port Nummer wird in den Einstellungen der Datenbank auf der Seite "Web/Konfiguration" gesetzt. Für den Parameter *Wert* können Sie eine Konstante unter dem Thema **TCP Port Nummern** verwenden. Dieser Selector ist im Rahmen von 4D Web Servern mit einkompiliertem 4D Desktop hilfreich (kein Zugriff auf die Design-Umgebung). **Mögliche Werte:** Weitere Informationen über die TCP Port Nummer finden Sie im Abschnitt **Web Server, Einstellungen**. **Standardwert:** 80
Web session cookie domain	Lange Ganzzahl	81	**Reichweite:** lokaler Web Server **Mögliche Werte:** Text **Beschreibung:** Setzt oder erhält den Wert des Feldes "domain" des Session Cookie (Text). Dieser Selektor, sowie Selektor 82 sind hilfreich zum Überprüfen der Reichweite von Session Cookies: Haben Sie z.B. für diesen Selektor den Wert "/*.4d.fr" gesetzt, sendet der Client ein Cookie nur, wenn die Anfrage an die Domaine ".4d.fr" gerichtet ist. Das schließt Server aus, die externe statische Daten hosten.

Konstante	Typ	Wert	Kommentar
Web session cookie name	Lange Ganzzahl	73	**Reichweite:** Lokaler Web Server **Beschreibung:** Setzt den Namen des Cookie zum Sichern der Session ID **Mögliche Werte:** Text **Standardwert:** "4DSID" (zum Wiederherstellen des Standardwerts leeren String übergeben)
Web session cookie path	Lange Ganzzahl	82	**Reichweite:** lokaler Web Server **Mögliche Werte:** Text **Beschreibung:** Setzt oder erhält den Wert des Feldes "path" des Session Cookie (Text). Dieser Selektor, sowie Selektor 81 sind hilfreich zum Überprüfen der Reichweite von Session Cookies: Haben Sie z.B. für diesen Selector den Wert "/4DACTION" gesetzt, sendet der Client ein Cookie nur für dynamische Anfragen, die mit 4DACTION beginnen und nicht für Bilder, statische Seiten, etc.
Web session enable IP address validation	Lange Ganzzahl	83	**Reichweite**: Lokaler Web Server **Beschreibung**: Damit können Sie die Überprüfung der IP Adresse für Session Cookies aktivieren oder deaktivieren. Aus Sicherheitsgründen prüft der 4D Web Server standardmäßig die IP Adresse einer Anfrage mit einem Session Cookie und weist sie ab, wenn sie nicht zur IP Adresse passt, über die das Cookie erstellt wurde. Sie können diese Überprüfung bei bestimmten Applikationen deaktivieren und das Session Cookie akzeptieren, auch wenn die IP Adresse nicht dazu passt. Wechseln z.B. mobile Geräte zwischen Wifi und 3G/4G Netzwerken, ändert sich die IP Adresse. In diesem Fall übergeben Sie den Wert 0, damit Clients weiterhin ihre Web Sessions verwenden können, auch wenn sie die IP Adresse wechseln. Beachten Sie, dass diese Einstellung die Sicherheitstufe Ihrer Applikation herabsetzt. Die Einstellung ist sofort wirksam, d.h. Sie müssen den HTTP Server nicht neu starten. **Mögliche Werte**: 0 (deaktiviert) oder 1 (aktiviert) **Standardwert**: 1 (IP Adressen werden geprüft)

Web Services (Client)

Konstante	Typ	Wert	Kommentar
Web Service compression	Lange Ganzzahl	1	
Web Service detailed message	Lange Ganzzahl	1	Detaillierte Meldung mit Fehlerbeschreibung. Die Art der Meldung unterscheidet sich je nach der Art des Hauptfehlers. - Ist der Hauptfehler = 9910 (Soap Fehler), wird der Grund des SOAP Fehlers zurückgegeben (z.B.: "the remote method does not exist"). - Ist der Hauptfehler = 9911 (Parser Fehler), wird die Stelle des Fehlers im XML Dokument zurückgegeben. - Ist der Hauptfehler = 9912 (HTTP Fehler): - Liegt der HTTP Fehler im Bereich [300-400] (Probleme mit der Stelle des angefragten Dokuments), wird die neue Stelle der angefragten URL zurückgegeben. - bei allen anderen HTTP Fehlercodes wird <body> zurückgegeben. - Ist der Hauptfehler = 9913 (Netzwerk Fehler), wird der Grund des Netzwerkfehlers zurückgegeben (z.B.: "ServerAddress: DNS lookup failure") - Ist der Hauptfehler = 9914 (interner Fehler), wird der Grund des internen Fehlers zurückgegeben.
Web Service display auth dialog	Lange Ganzzahl	4	*Wert* = 0 (Dialogfenster nicht anzeigen) oder 1 (Dialogfenster anzeigen) Diese Option verwaltet die Anzeige des Authentifizierungsdialogs beim Ausführen des Befehls **WEB SERVICE CALL**. Dieser Befehl zeigt standardmäßig nie den Dialog an; normalerweise müssen Sie dafür den Befehl **WEB SERVICE AUTHENTICATE** verwenden. Verwenden Sie diese Option, wenn der Authentifizierungsdialog erscheinen soll, damit Benutzer ihre Identifier eintragen können: Übergeben Sie 1 in Wert, um das Dialogfenster anzuzeigen, sonst 0. Das Dialogfenster erscheint nur, wenn der Web Service eine Authentifizierung benötigt.
Web Service dynamic	Lange Ganzzahl	0	
Web Service error code	Lange Ganzzahl	0	Hauptfehler Code (von 4D definiert). Dieser Code wird auch in der Systemvariable Error zurückgegeben. Hier die Liste möglicher Fehler: 9910: Soap Fehler (siehe auch Web Service Fault Actor) 9911: Parser Fehler 9912: HTTP Fehler (siehe auch Web Service HTTP Error code) 9913: Netzwerkfehler 9914: Interner Fehler
Web Service fault actor	Lange Ganzzahl	3	Grund des Fehlers (vom SOAP Protokoll zurückgegeben — zu verwenden im Fall von Hauptfehler 9910). - Version passt nicht - Verstehen ist erforderlich (der Server konnte einen als zwingend definierten Parameter nicht interpretieren) - Sender Fehler - Receiver Fehler - Codierung unbekannt
Web Service HTTP compression	Lange Ganzzahl	6	*value* = Web Service Compression Damit lässt sich ein interner Kompressionsmechanismus für SOAP Anfragen aktivieren, um den Austausch zwischen 4D Anwendungen zu beschleunigen. Führen Sie die Anweisung **WEB SERVICE SET OPTION**(Web Service HTTP Compression; Web Service Compression) auf dem 4D Client des Web Service aus, werden die Daten der nächsten vom Client gesendeten SOAP Anfrage vor dem Senden an den 4D SOAP Server über einen standard HTTP Mechanismus komprimiert ("gzip" oder "deflate", je nach Inhalt der Anfrage). Der Server entkomprimiert und analysiert die Anfrage und antwortet dann automatisch über denselben Mechanismus. Das gilt nur für die Anfrage, die auf den Aufruf des Befehls **WEB SERVICE SET OPTION** folgt. Sie müssen deshalb diesen Befehl vor jedem Komprimieren erneut aufrufen. 4D komprimiert standardmäßig keine Web Service HTTP Anfragen. **Hinweis:** Dieser Mechanismus lässt sich nicht für Anfragen verwenden, die an einen 4D SOAP Server älter als Version 11.3 gesendet werden. Zur weiteren Optimierung dieser Funktionsweise können Sie zusätzliche Optionen wie Durchfluß- und Komprimierungsrate konfigurieren. Sie sind im Befehl **SET DATABASE PARAMETER** verfügbar.
Web Service HTTP error code	Lange Ganzzahl	2	HTTP Fehler-Code (bei Hauptfehler 9912 verwenden)
Web Service HTTP timeout	Lange Ganzzahl	1	*Wert* = Timeout des Client Teils, angegeben in Sekunden. Timeout ist die Zeitspanne, die der Web Service Client wartet, wenn der Server nicht antwortet. Ist sie abgelaufen, schließt der Client die Sitzung und die Anfrage geht verloren. Das Timeout beträgt standardmäßig 180 Sekunden. Es kann für spezifische Fälle geändert werden (Netzwerkstatus, Web Service Eigenheiten, etc.).
Web Service manual	Lange Ganzzahl	3	
Web Service manual in	Lange Ganzzahl	1	
Web Service manual out	Lange Ganzzahl	2	
Web Service reset auth settings	Lange Ganzzahl	5	*Wert* = 0 (Information nicht entfernen) oder 1 (Information entfernen) Damit können Sie für 4D angeben, ob die Information zur Authentifizierung des Benutzers (Benutzername, Kennwort, Methode, etc.) gespeichert werden soll, um sie in Folge erneut zu verwenden. Diese Information wird standardmäßig nach jeder Ausführung des Befehls **WEB SERVICE CALL** entfernt. Übergeben Sie 0 in *Wert*, um die Information zu speichern, 1 um sie zu entfernen. Beachten Sie, dass bei 0 die Information während der Sitzung beibehalten, aber nicht gespeichert wird.

Konstante	Typ	Wert	Kommentar
Web Service SOAP header	Lange Ganzzahl	2	*Wert* = XML Root Element Referenz, die als Header in der SOAP Anfrage eingegeben werden soll. Damit können Sie einen Header in einer SOAP Anfrage eingeben, die über den Befehl **WEB SERVICE CALL** erstellt wurde. SOAP Anfragen enthalten standardmäßig keinen spezifischen Header. Bestimmte Web Services benötigen jedoch einen Header, z.B. zur Verwaltung von Parametern zur Identifizierung.
Web Service SOAP version	Lange Ganzzahl	3	*Wert* = Web Service SOAP_1_1 oder Web Service SOAP_1_2 Über diese Option können Sie die Version des in der Anfrage verwendeten SOAP Protokolls angeben. Übergeben Sie Web Service SOAP_1_1 für Version 1.1, Web Service SOAP_1_2 für Version 1.2.
Web Service SOAP_1_1	Lange Ganzzahl	0	
Web Service SOAP_1_2	Lange Ganzzahl	1	

Web Services (Server)

Konstante	Typ	Wert	Kommentar
Is DOM reference	Lange Ganzzahl	37	
Is XML	Lange Ganzzahl	36	
SOAP client fault	Lange Ganzzahl	1	
SOAP input	Lange Ganzzahl	1	
SOAP method name	Lange Ganzzahl	1	Name der auszuführenden Web Service Methode
SOAP output	Lange Ganzzahl	2	
SOAP server fault	Lange Ganzzahl	2	
SOAP service name	Lange Ganzzahl	2	Name des Web Service, zu dem die Methode gehört.

Wertebereiche

Konstante	Typ	Wert	Kommentar
MAXINT	Lange Ganzzahl	32767	
MAXLONG	Lange Ganzzahl	2147483647	
MAXTEXTLENBEFOREV11	Lange Ganzzahl	32000	

Wörterbücher

Hinweis zur Kompatibilität: Diese Konstanten entsprechen den Nummern für "Cordial" Wörterbücher, die in 4D ab Version 14 nicht mehr unterstützt werden. Sie werden zur Wahrung der Kompatibilität beibehalten (intern wird ein entsprechendes Hunspell Wörterbuch verwendet).

Konstante	Typ	Wert	Kommentar

XML

Konstante	Typ	Wert	Kommentar
Copy XML data source	Lange Ganzzahl	1	4D behält eine Kopie des DOM Baumes mit dem Bild bei, d.h. das Bild lässt sich in einem Bildfeld der Datenbank sichern und dann jederzeit erneut anzeigen oder exportieren.
DOCTYPE Name	Lange Ganzzahl	3	Name des Root Elements, wie im DOCTYPE Marker definiert
Document URI	Lange Ganzzahl	6	URI von der DTD
Encoding	Lange Ganzzahl	4	Verwendete Codierung (UTF-8, ISO...)
Get XML data source	Lange Ganzzahl	0	4D liest die XML Datenquelle nur; sie wird nicht mit dem Bild beibehalten. Das steigert die Ausführungsgeschwindigkeit des Befehls signifikant; da jedoch der DOM Baum nicht beibehalten wird, lässt sich das Bild weder speichern noch exportieren.
Own XML data source	Lange Ganzzahl	2	4D exportiert den DOM Baum mit Bild. Es lässt sich speichern oder exportieren und die Ausführung des Befehls ist schnell. Dann ist jedoch die XML Referenz *ElementRef* für andere 4D Befehle nicht mehr verwendbar. Dies ist der Standardmodus für Exportieren, wenn der Parameter *ExportTyp* nicht angegeben ist.
PUBLIC ID	Lange Ganzzahl	1	Öffentlicher Identifier (FPI) der DTD, zu dem das Dokument passt (sofern das tag DOCTYPE xxx PUBLIC vorhanden ist).
SYSTEM ID	Lange Ganzzahl	2	System Identifier
Version	Lange Ganzzahl	5	Zugelassene XML Version
XML Base64	Lange Ganzzahl	1	
XML binary encoding	Lange Ganzzahl	5	Gibt an, wie binäre Daten konvertiert werden. **Mögliche Werte:** • XML Base64 (Standardwert): binäre Daten werden einfach in Base64 konvertiert. • XML Data URI scheme: binäre Daten werden in Base64 konvertiert und der Header "data:;base64" wird hinzugefügt. Dieses Format ermöglicht hauptsächlich einem Browser, ein Bild automatisch zu decodieren. Es ist auch zum Einfügen von SVG Bildern erforderlich. Weitere Information dazu siehe http://en.wikipedia.org/wiki/Data_URI_scheme.
XML case insensitive	Lange Ganzzahl	2	
XML case sensitive	Lange Ganzzahl	1	
XML CDATA	Lange Ganzzahl	7	
XML comment	Lange Ganzzahl	2	
XML convert to PNG	Lange Ganzzahl	1	
XML DATA	Lange Ganzzahl	6	
XML data URI scheme	Lange Ganzzahl	2	
XML date encoding	Lange Ganzzahl	2	Gibt an, wie 4D das Datum konvertiert, z.B. !01/01/2003! in der Paris Zeitzone. **Mögliche Werte:** • XML ISO (Standardwert): verwendet das Format xs:datetime ohne Angabe der Zeitzone. Ergebnis: "2003-01-01". Ist im 4D Wert (via SQL) ein Zeitteil enthalten, geht er verloren. • XML Local: verwendet das Format xs:date mit Angabe der Zeitzone. Ergebnis: "2003-01-01 +01:00". Ist im 4D Wert (via SQL) ein Zeitteil enthalten, geht er verloren. • XML Datetime local: verwendet das Format xs:dateTime (ISO 8601) mit Angabe der Zeitzone. Ist im 4D Wert (via SQL) ein Zeitteil enthalten, wird er beibehalten. Ergebnis: "<Date>2003-01-01T00:00:00 +01:00</Date>".. • XML UTC: verwendet das Format xs:date. Ergebnis: "2003-01-01Z". Ist im 4D Wert (via SQL) ein Zeitteil enthalten, geht er verloren. • XML Datetime UTC: verwendet das Format xs:dateTime (ISO 8601). Ist im 4D Wert (via SQL) ein Zeitteil enthalten, wird er beibehalten. Ergebnis: "<Date>2003-01-01T00:00:00Z</Date>"
XML datetime local	Lange Ganzzahl	3	
XML datetime local absolute	Lange Ganzzahl	1	
XML datetime UTC	Lange Ganzzahl	5	
XML disabled	Lange Ganzzahl	2	

Konstante	Typ	Wert	Kommentar
XML DOCTYPE	Lange Ganzzahl	10	
XML DOM case sensitivity	Lange Ganzzahl	8	Definiert, ob Groß- und Kleinschreibung in Elementnamen für die Befehle **DOM Get XML element** und **DOM Count XML elements** berücksichtigt wird. **Mögliche Werte:** • XML case sensitive (Standardwert): Befehle unterscheiden zwischen Groß- und Kleinschreibung. • XML case insensitive: Befehle unterscheiden nicht zwischen Groß- und Kleinschreibung.
XML duration	Lange Ganzzahl	2	
XML ELEMENT	Lange Ganzzahl	11	
XML enabled	Lange Ganzzahl	1	
XML end document	Lange Ganzzahl	9	
XML end element	Lange Ganzzahl	5	
XML entity	Lange Ganzzahl	8	
XML external entity resolution	Lange Ganzzahl	7	Steuert, ob externe Einheiten in XML Dokumente aufgelöst werden. Aus Sicherheitsgründen erlauben die DOM und SAX 4D XML Parser standardmäßig keine Auflösung externer Einheiten. **Mögliche Werte:** • XML enabled: Erlaubt die Auflösung externer Einheiten in XML Dokumente • XML disabled (Standardwert): Verweigert die Auflösung externer Einheiten (Die Deklaration generiert einen Parser/Analysefehler)
XML indentation	Lange Ganzzahl	4	Gibt die Einrückung des XML Dokuments an. **Mögliche Werte:** • XML with indentation (Standardwert): das Dokument ist eingerückt. • XML no indentation: Das Dokument ist nicht eingerückt; sein Inhalt wird in eine einzige Zeile gesetzt.
XML ISO	Lange Ganzzahl	1	
XML local	Lange Ganzzahl	2	
XML native codec	Lange Ganzzahl	2	
XML no indentation	Lange Ganzzahl	2	
XML picture encoding	Lange Ganzzahl	6	Gibt an, wie Bilder konvertiert werden müssen (vor Codierung in Base64). **Mögliche Werte:** • XML Convert to PNG (Standardwert): Bilder werden vor der Codierung in Base64 in PNG konvertiert. • XML Native codec: Bilder werden vor Codierung in Base64 in ihr erstes native Speicher CODEC konvertiert. Sie müssen diese Optionen zum Codieren in SVG Bilder verwenden (siehe Beispiel zum Befehl **XML SET OPTIONS**.
XML processing instruction	Lange Ganzzahl	3	
XML raw data	Lange Ganzzahl	2	
XML seconds	Lange Ganzzahl	4	
XML start document	Lange Ganzzahl	1	
XML start element	Lange Ganzzahl	4	
XML string encoding	Lange Ganzzahl	1	Gibt an, wie 4D Strings in Elementwerte konvertiert werden. Das gilt nicht für die Konvertierung in Attribute, für die XML die Verwendung von Escape-Zeichen verlangt. **Mögliche Werte:** • XML With escaping (Standardwert): Konvertierung von 4D Strings in XML Elementwerte mit Ersetzen der Zeichen. Daten vom Typ Text werden automatisch analysiert, so dass verbotene Zeichen (<&>') durch XML Einheiten ersetzt werden (&&It;>'"). • XML Raw data: 4D Strings werden als reine Daten gesendet; 4D führt weder eine Codierung noch eine Analyse durch. 4D Werte werden nach Möglichkeit in XML Fragmente konvertiert und als Unterelement des Zielelements eingefügt. Kann ein Wert nicht als ein XML Fragment bewertet werden, wird es in Form von Rohdaten in einen neuen Knoten CDATA eingefügt.

Konstante	Typ	Wert	Kommentar
XML time encoding	Lange Ganzzahl	3	Gibt an, wie 4D die Zeit konvertiert, z.B. ?02/00/46? (Paris Zeit). Die Codierung ist unterschiedlich, je nachdem ob Sie eine Uhrzeit oder eine Zeitspanne ausdrücken wollen. **Mögliche Werte für Uhrzeit:** • XML Datetime UTC: Zeit ausgedrückt in UTC (Universal Time Coordinated). Beachten Sie, dass die Konvertierung in UTC automatisch ist. Ergebnis: "<Duration>0000-00-00T01:00:46Z</Duration>". • XML Datetime local: Zeit ausgedrückt mit der Zeitdifferenz des Rechners mit der 4D Engine. Ergebnis: "<Duration>0000-00-00T02:00:46+01:00</Duration>". • XML Datetime local absolute (Standardwert): Zeit ausgedrückt ohne Angabe der Zeitzone. Der Wert wird nicht geändert. Ergebnis: "<Duration>0000-00-00T02:00:46</Duration>". **Mögliche Werte für Zeitspanne:** • XML Seconds: Anzahl Sekunden ab Mitternacht; der Wert wird nicht geändert, da er eine Dauer ausdrückt. Ergebnis: "<Duration>7246</Duration>". • XML Duration: Dauer ausgedrückt in Übereinstimmung mit dem XML Schema Teil 2: Datentypen Zweite Edition. der Wert wird nicht geändert, da er eine Dauer ausdrückt. Ergebnis: "<Duration>PT02H00M46S</Duration>".
XML UTC	Lange Ganzzahl	4	
XML with escaping	Lange Ganzzahl	1	
XML with indentation	Lange Ganzzahl	1	

Zeit Anzeigeformate

Konstante	Typ	Wert	Kommentar
Blank if null time	Lange Ganzzahl	100	"" statt 0
HH MM	Lange Ganzzahl	2	01:02
HH MM AM PM	Lange Ganzzahl	5	1:02 AM
HH MM SS	Lange Ganzzahl	1	01:02:03
Hour min	Lange Ganzzahl	4	1 Stunde 2 Minuten
Hour min sec	Lange Ganzzahl	3	1 Stunde 2 Minuten 3 Sekunden
ISO time	Lange Ganzzahl	8	0000-00-00T01:02:03
Min sec	Lange Ganzzahl	7	62 Minuten 3 Sekunden
MM SS	Lange Ganzzahl	6	62:03
System time long	Lange Ganzzahl	11	1:02:03 AM HNEC (nur Mac OS)
System time long abbreviated	Lange Ganzzahl	10	1•02•03 AM (nur Mac OS)
System time short	Lange Ganzzahl	9	01:02:03

Zugewiesene Standardaktion

Alle Konstanten unter diesem Thema sind ab 4D v16 R3 überholt. Verwenden Sie die Textkonstanten unter dem Thema **Standardaktion**.

Konstante	Typ	Wert	Kommentar
_o_Accept action	Lange Ganzzahl	2	
_o_Add subrecord action	Lange Ganzzahl	14	
_o_Cancel action	Lange Ganzzahl	1	
_o_Clear action	Lange Ganzzahl	21	
_o_Copy action	Lange Ganzzahl	19	
_o_Cut action	Lange Ganzzahl	18	
_o_Database settings action	Lange Ganzzahl	32	
_o_Delete record action	Lange Ganzzahl	7	
_o_Delete subrecord action	Lange Ganzzahl	13	
_o_Edit subrecord action	Lange Ganzzahl	12	
_o_First page action	Lange Ganzzahl	10	
_o_First record action	Lange Ganzzahl	5	
_o_Last page action	Lange Ganzzahl	11	
_o_Last record action	Lange Ganzzahl	6	
_o_MSC action	Lange Ganzzahl	36	
_o_Next page action	Lange Ganzzahl	8	
_o_Next record action	Lange Ganzzahl	3	
_o_No action	Lange Ganzzahl	0	
_o_Paste action	Lange Ganzzahl	20	
_o_Previous page action	Lange Ganzzahl	9	
_o_Previous record action	Lange Ganzzahl	4	
_o_Quit action	Lange Ganzzahl	27	
_o_Redo action	Lange Ganzzahl	31	
_o_Return to Design mode	Lange Ganzzahl	35	
_o_Select all action	Lange Ganzzahl	22	
_o_Show clipboard action	Lange Ganzzahl	23	
_o_Test application action	Lange Ganzzahl	26	
_o_Undo action	Lange Ganzzahl	17	

Zugriff Designobjekte

Konstante	Typ	Wert	Kommentar
Attribute executed on server	Lange Ganzzahl	8	Entspricht der Option "Auf Server ausführen"
Attribute folder name	Lange Ganzzahl	1024	Name des Ordners für die Methode (Attribut "Ordner"). Übergeben Sie diese Konstante, müssen Sie in *attrWert* einen Ordnernamen übergeben: • Entspricht dieser Name einem gültigen Ordner, wird die Methode in diesen Elternordner gesetzt • Existiert der Ordner nicht, ändert der Befehl nichts auf der Ebene des Elternordners • Übergeben Sie einen leeren String, wird die Methode auf die Root-Ebene gesetzt
Attribute invisible	Lange Ganzzahl	1	Entspricht der Option "Unsichtbar"
Attribute published SOAP	Lange Ganzzahl	3	Entspricht der Option "Zugang per Web Service"
Attribute published SQL	Lange Ganzzahl	7	Entspricht der Option "Zugang per SQL"
Attribute published Web	Lange Ganzzahl	2	Entspricht der Option "Zugang per 4D HTML Tags und URLs (4DACTION...)"
Attribute published WSDL	Lange Ganzzahl	4	Entspricht der Option "Anbieten per WSDL"
Attribute shared	Lange Ganzzahl	5	Entspricht der Option "Gemeinsam von Komponenten und Host benutzt"
Code with tokens	Lange Ganzzahl	1	Tokens in exportiertem Code einfügen
On object locked abort	Lange Ganzzahl	0	Laden des Objekts wird abgebrochen (Standard Funktionsweise)
On object locked confirm	Lange Ganzzahl	2	4D zeigt ein Dialogfenster, über das Sie die Operation erneut ausführen oder abbrechen können. Das wird im remote Modus nicht unterstützt, d.h. Laden wird abgebrochen
On object locked retry	Lange Ganzzahl	1	4D versucht das Objekt weiter zu laden, bis es freigegeben ist.
Path all objects	Lange Ganzzahl	31	Der Befehl gibt die Pfade aller kombinierten Methoden zurück.
Path database method	Lange Ganzzahl	2	Der Befehl gibt den Pfad der spezifizierten Datenbankmethoden (englische Namen) zurück. Liste dieser Methoden: [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	Lange Ganzzahl	4	Pfad der Projektformularmethoden und aller dazugehörigen Objektmethoden. Beispiele: [projectForm]/myForm/{formMethod} [projectForm]/myForm/button1 [projectForm]/myForm/my%2list [projectForm]/myForm/button1
Path project method	Lange Ganzzahl	1	Name der Methode Beispiel: MyProjectMethod
Path table form	Lange Ganzzahl	16	Pfad der Tabellenformularmethoden und aller dazugehörigen Objektmethoden. Beispiele: [tableForm]/table_1/Form1/{formMethod} [tableForm]/table_1/Form1/button1 [tableForm]/table_1/Form1/my%2list [tableForm]/table_2/Form1/my%2list
Path trigger	Lange Ganzzahl	8	Pfad der Datenbank-Trigger. Beispiele: [trigger]/table_1 [trigger]/table_2

Fehlermeldungen

Syntaxfehler (1 -> 81)

Folgende Tabelle zeigt die Fehlermeldungen in der Syntax während der Code Ausführung in der Design- oder Anwendungsumgebung. Einige der Fehler treten nur im interpretierten Modus bzw. im kompilierten Modus auf, einige in beiden Modi. Mit einer Methode Unterbrechung bei Fehler mit dem Befehl **ON ERR CALL** können Sie diese Fehler ausfindig machen.

Code	Beschreibung
1	Eine "(" -Klammer wird erwartet.
2	Ein Datenfeldname wird erwartet.
3	Dieser Befehl kann nur auf ein Datenfeld einer Untertabelle angewandt werden.
4	Die Parameter der Liste müssen alle vom gleichen Typ sein.
5	Tabelle fehlt, auf die dieser Befehl angewandt werden kann.
6	Dieser Befehl kann nur in einem Datenfeld einer Untertabelle angewandt werden.
7	Ein numerischer Parameter wird erwartet.
8	Ein alphanumerischer Parameter wird erwartet.
9	Das Ergebnis einer Bedingung wird erwartet.
10	Dieser Befehl ist auf diesen Datentyp nicht anwendbar.
11	Dieser Befehl ist zwischen zwei Bedingungen nicht anwendbar.
12	Dieser Befehl ist zwischen zwei numerischen Argumenten nicht anwendbar.
13	Dieser Befehl ist zwischen zwei alphanumerischen Argumenten nicht anwendbar.
14	Dieser Befehl ist zwischen zwei Argumenten vom Typ Datum nicht anwendbar.
15	Diese Operation ist mit den zwei Argumenten nicht kompatibel
16	Dieses Datenfeld ist nicht verknüpft.
17	Ein Tabellenname wird erwartet.
18	Die Datenfeldtypen sind nicht kompatibel.
19	Das Datenfeld ist nicht indiziert.
20	Das Zeichen = wird erwartet.
21	Diese Methode ist nicht vorhanden.
22	Zum Sortieren oder graphisch darzustellen, müssen die Datenfelder der gleichen Tabelle oder Untertabelle angehören.
23	Das Zeichen "<" oder ">"wird erwartet.
24	Das Zeichen ";" wird erwartet.
25	Die Anzahl der zu sortierenden Datenfelder ist zu groß.
26	Datenfeld kann nicht vom Typ Text, Bild, Blob oder Untertabelle sein.
27	Der Tabellenname muss dem Datenfeldnamen vorangestellt werden.
28	Das Datenfeld muss numerisch sein.
29	Der Wert muss 1 oder 0 sein.
30	Eine Variable wird erwartet.
31	Es gibt keine Menüleiste mit dieser Nummer.
32	Ein Datum wird erwartet.
33	Der Befehl oder die Methode ist nicht vorhanden.
34	Buchhaltungsdateien sind nicht geöffnet.
35	Die Mengen sind aus verschiedenen Tabellen.
36	Ungültiger Tabellenname.
37	Das Zeichen := wird erwartet.
38	Dies ist eine Funktion, keine Methode.
39	Diese Menge ist nicht vorhanden.
40	Dies ist eine Methode, keine Funktion.
41	Eine Variable oder ein Feld einer Untertabelle wird erwartet.
42	Der Datensatz konnte nicht auf den Stapel gelegt werden.
43	Die Funktion wurde nicht gefunden.
44	Die Methode wurde nicht gefunden.
45	Datenfeld oder Variable wird erwartet.
46	Ein alphanumerisches oder numerisches Argument wird erwartet.
47	Das Datenfeld muss alphanumerisch sein.
48	Syntaxfehler.
49	Dieser Operator ist hier nicht anwendbar.
50	Diese Operatoren dürfen nicht zusammen verwendet werden.
51	Dieses Plug-In ist nicht implementiert.
52	Ein Array wird erwartet.
53	Angabe liegt außerhalb des Array-Bereichs.
54	Die Argumente sind nicht kompatibel.
55	Ein Argument vom Typ Boolean wird erwartet.
56	Datenfeld, Variable oder Tabelle wird erwartet.
57	Ein Operator wird erwartet.
58	Eine ")" -Klammer wird erwartet.
59	Dieser Argumententyp ist hier nicht anwendbar.
60	Parameter oder lokale Variable konnte in der kompilierten Datenbank im Befehl *EXECUTE* nicht benutzt werden.

61	Array konnte in der kompilierten Datenbank nicht geändert werden.
62	Dieser Befehl kann nicht auf eine Untertabelle angewendet werden.
63	Datenfeld ist nicht indiziert.
64	Datenfeld vom Typ Bild oder Variable wird erwartet.
65	Der Wert muss 4 Zeichen enthalten
66	Der Wert darf nur ein Zeichen enthalten
67	Dieser Befehl kann nicht auf dem 4D Server ausgeführt werden.
68	Eine Liste wird erwartet.
69	Eine Referenz auf ein externes Fenster wird erwartet.
70	Der Befehl kann nicht zwischen zwei Bildargumenten ausgeführt werden.
71	Der Befehl **SET PRINT MARKER** lässt sich nur im Kopfteil eines Formulars aufrufen, das gerade gedruckt wird.
72	Ein Array vom Typ Zeiger wird erwartet.
73	Ein Array vom Typ Zahl wird erwartet.
74	Die Größe des Array passt nicht.
75	Kein Zeiger auf lokale Arrays.
76	Falscher Array-Typ.
77	Falscher Variablenname.
78	Ungültiger Sortierparameter.
79	Dieser Befehl kann während der Erstellung einer Liste nicht ausgeführt werden.
80	Zuviele Suchargumente.
81	Das Formular wurde nicht gefunden.

Tipps

Einige dieser Fehlermeldungen zeigen einen Syntaxfehler an, wenn Sie sich vertippt haben. Sie erhalten zum Beispiel den Fehler #37, wenn Sie v:=0 ausführen wollen, jedoch v=0 getippt haben. Diesen Fehler beheben Sie im Methodeneditor der Designumgebung.

Einige dieser Fehlermeldungen beruhen auf einfachen Programmierfehlern. Sie erhalten zum Beispiel den Fehler #5, wenn Sie den Befehl **ALL RECORDS** ausführen, ohne zuvor mit dem Befehl **DEFAULT TABLE** die Standardtabelle gesetzt oder einen Parameter für die Tabelle übergeben zu haben. In diesem Fall gibt es keine Tabelle, auf die der Befehl angewendet werden kann. Prüfen Sie, ob Sie vergessen haben, für diesen Befehl die Standardtabelle zu setzen oder einen Parameter für die Tabelle zu übergeben.

Einige dieser Fehlermeldungen zeigen einen Fehler bei fehlerhaftem Design an. Sie erhalten zum Beispiel den Fehler #16, wenn Sie den Befehl **RELATE ONE** auf ein Datenfeld ohne Verknüpfung anwenden. Prüfen Sie in diesem Fall, ob Ihr Code falsch ist oder ob Sie lediglich vergessen haben, die Verknüpfung für dieses Datenfeld einzurichten.

Einige Fehler liegen, wenn sie auftreten, nicht genau da, wo Ihr Code abbricht. Erhalten Sie zum Beispiel in einer Unterroutine den Fehler #53 (Angabe liegt außerhalb des Array-Bereichs.) für die Zeile vpFld:=Field($1;$2), beruht der Fehler auf einer falschen Tabellen- und/oder Datenfeldnummer, die der Unterroutine als Parameter übergeben wurde. Der Fehler liegt also in der aufrufenden Methode und nicht dort, wo der Fehler auftritt. Durchlaufen Sie in diesem Fall Ihren Code im Fenster Debugger im Schrittmodus, um herauszufinden, welche Programmierzeile dafür verantwortlich ist. Beheben Sie dann den Fehler im Methodeneditor der Designumgebung.

Objektnotation Analyse Fehler (-10718 -> -10701)

Folgende Tabelle zeigt die Fehlernummern und -meldungen, die bei Analysieren von 4D Code mit Objektnotation auftreten können:

Fehlernr	Beschreibung
-10718	Der Wert 'null' wird bei diesem Ausdruck nicht unterstützt.
-10717	Es kann kein Zeiger auf eine Objekteigenschaft erzeugt werden.
-10716	Es wird ein Objekt oder eine Collection erwartet.
-10715	Die Eigenschaft für das Objekt oder die Collection ist unbekannt.
-10714	Der Index der Collection oder die Objekteigenschaft ist ungültig.
-10713	Der Index der Collection ist außerhalb des Bereichs.
-10712	Der Index der Collection muss eine positive endliche Zahl sein.
-10711	Die Eigenschaft dieser Collection kann nicht verändert werden.
-10710	Die Eigenschaft dieser Collection kann nicht entfernt werden.
-10709	Dieser Collection kann keine Eigenschaft hinzugefügt werden.
-10708	Diese Collection ist nicht definiert.
-10707	Der Typ der Eigenschaft ist ungültig.
-10706	Die Eigenschaft kann nicht numerisch sein.
-10705	Die Eigenschaft kann dem Objekt nicht hinzuzufügt werden, da es nicht erweiterbar ist.
-10704	Die Objekteigenschaft kann nicht entfernt werden, da sie nicht konfigurierbar ist.
-10703	Der Typ der Objekteigenschaft kann nicht verändert werden, da sie nicht konfigurierbar ist.
-10702	Die Objekteigenschaft kann nicht gesetzt werden (read-only).
-10701	Das Objekt ist nicht definiert

Fehler der Datenbank (-10602 -> 4004)

Diese Tabelle zeigt die Fehlermeldungen der Datenbank-Engine von 4D. Diese Fehler treten auf einer niederen Ebene der Datenbank-Engine auf, wie Benutzerunterbrechung, Zugriffsrechte und beschädigte Objekte.

Code	Beschreibung
-10602	Der angegebene Befehl kann nicht ausgeführt werden, da es keinen offenen Druckauftrag gibt.
-10601	Der Befehl **OBJECT DUPLICATE** kann nicht in Druckvorgängen verwendet werden.
-10600	Dieses BLOB ist nicht lesbar. Es ist möglicherweise beschädigt.
-10532	Die Fehlerverwaltungsmethode 'Methodenname' kann nicht aufgerufen werden.
-10526	Die Anwendung {database_name} kann nicht erzeugt werden.
-10525	Die bestehende Anwendung {database_name} kann nicht gelöscht werden.
-10524	Anwendung {database_name} kann nicht mehr geöffnet werden: Datendatei nicht mehr in '{datafile_path}' vorhanden.
-10523	Anwendung {database_name} kann nicht geöffnet werden: Keine Datendatei festgelegt.
-10522	Die Komponente {database_name} kann nicht geladen werden, da sie nicht im Unicode-Modus ist.
-10521	Die Anwendung {database_name} kann nicht geöffnet werden, da sie nicht im Unicode-Modus ist.
-10520	Nicht berechtigter Anwender: Nur der Administrator oder Designer kann diesen Befehl ausführen.
-10519	Falsche Url: {url}
-10518	Assert-Fehler: {assertion}
-10517	Fehler beim Synchronisieren des Ordners {Ordner_name}.
-10516	Der Server führt eine Wartungsoperation durch, bitte versuchen Sie es später noch einmal.
-10515	Ihr Versuch, sich am 4D Server anzumelden, wurde abgewiesen.
-10514	Die maximale Anzahl gleichzeitiger Anwender Ihrer Lizenz wurde erreicht.
-10513	Der Befehl {command_name} kann nicht von einer Remote-4D Developer aufgerufen werden.
-10512	Diese Codierung wird nicht unterstützt.
-10511	Der Befehl "{command_name}" kann von einer Komponente nicht aufgerufen werden.
-10510	Die Komponente "{component_name}" lässt sich nicht laden.
-10509	Die Anwendung "{database_name}" lässt sich nicht öffnen.
-10508	Die Projektmethode wurde nicht gefunden
-10507	Dieses Produkt kann keine kompilierte Anwendung öffnen
-10506	Limit der Standardedition
-10505	Client und Server haben nicht zueinander passende Versionsnummern
-10504	Falsche Indexseitennummer (Index muss repariert werden).
-10503	Datensatznummer nicht vorhanden (bei **GOTO RECORD** oder Datendatei muss repariert werden) (siehe Hinweis 3)
-10502	Defekte Datensatzstruktur (Datendatei muss repariert werden).
-10501	Defekte Indexseite (Datenbank muss repariert werden).
-10500	Defekte Datensatzadresse (Datenbank muss repariert werden).
-9999	Nicht genügend Speicher, um den Datensatz zu sichern (siehe Hinweis 4)
-9998	Dieser einmalige Datensatz ist bereits vorhanden.
-9997	Maximale Anzahl der Datensätze ist erreicht.
-9996	Stapelspeicher ist voll. (Zuviele Rekursionsstufen oder verschachtelte Aufrufe).
-9995	Grenze der Demo-Version ist erreicht.
-9994	Serielle Kommunikation vom Anwender unterbrochen durch gedrückte **Ctrl- + Alt- + Umschalttaste** (Windows) bzw. **Befehl- + Wahl- + Umschalttaste** (Mac OS).
-9993	Fehler in der Menüleiste (Datenbank muss repariert werden).
-9992	Falsches Kennwort.
-9991	Keine Zugriffsrechte.
-9990	Zeitfehler (Time-out).
-9989	Ungültige Datenbankstruktur (Datenbank muss repariert werden).
-9988	Formular kann nicht geladen werden, ist evtl. beschädigt.
-9987	Andere Datensätze sind bereits mit diesem Datensatz verknüpft.
-9986	Datensatz beim automatischen Löschen gesperrt.
-9985	Löschvorgang rekursiv.
-9984	Transaktion wurde unterbrochen, da ein einmaliges Feld zweimal vorhanden war.
-9983	Dasselbe Paket externer Routinen wurde zweimal installiert.
-9982	Datensatz wurde nicht geladen, da er nicht in der Auswahl auf der Arbeitsstation enthalten ist.
-9981	Tabelle wurde mit ungültigen Feldnamen/Feldnummern von Arbeitsstation gesendet.
-9980	Tabelle kann nicht erzeugt werden, da Struktur gesperrt ist.
-9979	Unbekannter Benutzer.
-9978	Falsches Benutzerkennwort.
-9977	Die temporäre Auswahl existiert nicht.
-9976	Befehl kann nicht ausgeführt werden, da Backup läuft.
-9975	Teil der Transaktion (Indexseiten) konnte nicht geladen werden.
-9974	Datensatz ist bereits gelöscht.
-9973	Sortierressourcen stimmen nicht überein.
-9972	Von Arbeitsstation angeforderte Dateinummer existiert nicht.
-9971	Von Arbeitsstation angeforderte Feldnummer existiert nicht.
-9970	Datenfeld ist nicht indiziert.
-9969	Defekter Datenfeldtyp von Arbeitsstation angefordert.
-9968	Ungültige Datensatznummer von Arbeitsstation angefordert.
-9967	Datensatz konnte nicht geladen werden. Daher nicht geändert.
-9966	Ungültiger Typ von Arbeitsstation angefordert.

-9965 Ungültige Suchtabelle von Arbeitsstation gesandt.
-9964 Ungültige Sortiertabelle von Arbeitsstation gesandt.
-9963 Ungültige Datensatznummer von Arbeitsstation angefordert.
-9962 Backup-Prozess nicht gestartet, da Server ausgeschaltet.
-9961 Backup-Prozess wird nicht korrekt ausgeführt.
-9960 4D Backup ist auf dem Server nicht installiert.
-9959 Backup-Prozess wurde bereits von anderem Anwender gestartet.
-9958 Prozess kann nicht gestartet werden.
-9957 Auswahlliste ist gesperrt.
-9956 Versionen von 4D Client und 4D Server sind unterschiedlich.
-9955 QuickTime ist nicht installiert
-9954 Kein aktueller Datensatz.
-9953 Kein Logbuch.
-9952 Ungültiger Header für Segment der Datendatei.
-9951 Datenfeld ist nicht verknüpft.
-9950 Ungültige Nummer für Segment der Datendatei.
-9949 Lizenz- oder Zugriffsfehler.
-9948 Modales Fenster ist aktiv
-9947 Kontrollkästchen „Nur 4D Client Verbindungen" wurde ausgewählt.
-9946 Temporäre Auswahl konnte nicht gelöscht werden, da nicht vorhanden.
-9945 Fehler in CD-ROM 4D Runtime, Änderungen können nicht gesichert werden.
-9944 Der Benutzer gehört nicht zur zugriffsberechtigten Gruppe für 4D Open.
-9943 Fehler in der Version der 4D Connectivitiy-Plug-Ins.
-9942 4D Client Lizenzierungsschema ist nicht kompatibel mit dieser Version von 4D Server.
-9941 Unbekannter Befehl EX_GESTALT in einer externen Routine.
-9940 Initialisierung einer externen Routine ist fehlerhaft.
-9939 Externe Routine wurde nicht gefunden.
-9938 Der aktuelle Datensatz wurde vom Trigger aus geändert.
-9937 Kennwortsystem ist durch einen anderen Benutzer gesperrt.
-9936 Der externe Kennwort-Code entspricht nicht dem der Datenbank.
-9935 Das XML-Dokument ist ungültig oder nicht richtig formatiert.
-9934 Das XML-Dokument ist nicht richtig formatiert.
-9933 Das XML-Dokument ist ungültig.
-9932 Das XML DLL ist nicht geladen.
-9931 Der Index für dieses Attribut ist ungültig.
-9930 Es gibt kein Attribut mit diesem Namen für dieses Element.
-9929 Der Index für dieses Element ist ungültig.
-9928 Der Name des Elements ist unbekannt.
-9927 Das angesprochene Element ist nicht das Grundelement (root).
-9926 Das angesprochene Element ist ungültig.
-9925 Das angesprochene Element ist leer.
-9924 Die Datei muss im Nur-Lesen Modus geöffnet werden.
-9923 Der Attributname ist ungültig.
-9922 Der Parameter-Wert für die Attribut-Definition fehlt.
-9921 Versuch, ein XML Protokoll in ein nicht-leeres Dokument zu schreiben
-9920 Der Typ des Knoten (node) ist ungültig.
-9919 Diese Kodierung wird nicht unterstützt.
-9918 Der Name des Elements ist ungültig.
-9917 Der Typ des übergebenen Array ist ungültig.
-9916 Das Element ist nicht geöffnet.
-9915 Die Dokumentreferenz ist ungültig.
-9914 Interner Fehler
-9913 Netzwerkfehler
-9912 HTTP Fehler
-9911 Parser Fehler
-9910 Soap Fehler
-9909 Kein Fenster für das Formular verfügbar
-9855 Ungültiger Parameter Nummer 5.
-9854 Ungültiger Parameter Nummer 4.
-9853 Ungültiger Parameter Nummer 3.
-9852 Ungültiger Parameter Nummer 2.
-9851 Ungültiger Parameter Nummer 1.
-9850 Einem externen Befehl wurde ein ungültiger Bereichsparameter übergeben.
-9803 Objekt ohne Namen im Formular "{form}" gefunden.
-9802 Objekt Pfad {Pfad} nicht einmalig. Anwendung mit MSC prüfen.
-9801 Methode {Pfad} lässt sich nicht öffnen.
-9800 Einer von mehreren Prozessen hat die Zugangsberechtigung geändert.
-9799 Fehler während Initialisierung der Vorschau.

-9778	Beim Laden des Wörterbuchs ist ein Fehler aufgetreten
-9777	Die Sprache der Methode passt nicht zur Lokalisierung der Anwendung: {Pfad}
-9776	Methode: {Pfad} lässt sich nicht erstellen
-9775	Code der Methode: {Pfad} lässt sich nicht schreiben
-9774	Code der Methode: {Pfad} lässt sich nicht lesen
-9773	Kommentare der Methode: {Pfad} lassen sich nicht schreiben
-9772	Kommentare der Methode: {Pfad} lassen sich nicht lesen
-9771	Eigenschaften der Methode: {Pfad} lassen sich nicht schreiben
-9770	Methodeneigenschaften können nicht gelesen werden: {path}
-9769	Ungültige Methodeneigenschaft: {Pfad}
-9768	Ungültiger Objektpfad: {Pfad}
-9767	Methoden können nicht upgedatet werden. Eine oder mehrere Ressourcen sind gesperrt.
-9766	Die Methode wird derzeit bearbeitet.
-9765	Ein oder mehrere Befehle werden gerade bearbeitet.
-9764	Der Befehl wird derzeit bearbeitet.
-9763	Der Befehl lässt sich nicht in einer Komponente ausführen.
-9762	Der Befehl lässt sich nicht in einer kompilierten Datenbank ausführen.
-9761	Ungültiger Objekttyp.
-9760	Es ist nicht möglich, den XML Node an die angegebene Position zu verschieben oder zu kopieren.
-9759	Objektbibliothek lässt sich nicht öffnen.
-9758	Benutzerformular existiert bereits.
-9757	Benutzerformular existiert nicht.
-9756	Es gibt keine Benutzer Strukturdatei.
-9755	Benutzerformular hat keinen Namen.
-9754	Dieser Befehl lässt sich nicht über ein Dialogfenster verwenden.
-9753	Quellformular ist nicht vorhanden
-9752	Benutzerformular lässt sich nicht erstellen.
-9751	Quellformular ist für Benutzer nicht zugänglich.
-9750	Quellformular ist nicht editierbar.
-1	Von einem Plug-In wurde eine ungültige Tabellennummer angesprochen
1001	Keine Spalte für den Import in Tabelle {TableName}
1002	Falsche Tabelle für den Import
1003	Keine Spalte für den Export für Tabelle {TableName}
1004	Falsche Tabelle für den Export
1005	Programm wurde vom Anwender unterbrochen—durch die Tastenkombination **Alt+Klick** (Windows) bzw. **Option+Klick** (Mac OS).
1006	Struktur für Anwendung {BaseName} kann nicht gesichert werden.
1006	Programm wurde vom Anwender unterbrochen—durch die Tastenkombination **Alt+Klick** (Windows) bzw. **Option+Klick** (Mac OS).
1007	Daten für Datenbank: {BaseName} können nicht erzeugt werden
1008	Falsches Datensegment für Datenbank {BaseName}
1009	Hauptspeicher reicht nicht aus
1010	Tabellendefinition für Datenbank {BaseName} kann nicht geladen werden
1011	Daten für Datenbank {BaseName} kann nicht geöffnet werden
1012	Datensegment für Datenbank: {BaseName} ist voll
1013	Datensegement für Datenbank {BaseName} kann nicht gespeichert werden
1014	Datensegment für Datenbank {BaseName} kann nicht gelesen werden
1015	Ungültiger Datenbank-Header in Datenbank {BaseName}
1016	Tabelle in Datenbank {BaseName} kann nicht erzeugt werden
1017	Indexliste in Datenbank {BaseName} kann nicht gelesen werden
1018	Indexliste in Datenbank {BaseName} kann nicht geschrieben werden
1019	Ungültige Tabellenreferenz in Datenbank {BaseName}
1020	Ungültige Feldreferenz in Datenbank {BaseName}
1021	Ungültiger Indextyp in Datenbank {BaseName}
1022	Ungültiger Feldername in Tabelle {TableName} in Datenbank {BaseName}
1023	Ungültiger Datenbankname
1024	Struktur für Datenbank {BaseName} kann nicht geöffnet werden
1025	Struktur für Datenbank {BaseName} kann nicht erzeugt werden
1026	Bit Selection für Datenbank {BaseName} kann nicht geladen werden
1027	Menge für Datenbank {BaseName} kann nicht geladen werden
1028	Menge für Datenbank {BaseName} kann nicht geändert werden
1029	Menge für Datenbank {BaseName} kann nicht gespeichert werden
1030	BLOB {BlobNum} in Tabelle {TableName} in Datenbank {BaseName} kann nicht gesichert werden
1031	BLOB {BlobNum} in Tabelle {TableName} in Datenbank {BaseName} kann nicht geladen werden
1032	Speicherplatz für BLOB {BlobNum} in Tabelle {TableName} in Datenbank {BaseName} kann nicht bereitgestellt werden
1033	Bit-Tabelle für Datenbank {BaseName} kann nicht geladen werden
1034	Bit-Tabelle für Datenbank {BaseName} kann nicht gespeichert werden
1035	Tabelle der Bit-Tabellen für Datenbank {BaseName} kann nicht geladen werden

1036	Tabelle der Bit-Tabellen für Datenbank {BaseName} kann nicht gespeichert werden
1037	Datensegment für Datenbank {BaseName} kann nicht geschlossen werden
1038	Datensegment für Datenbank {BaseName} kann nicht gelöscht werden
1039	Datensegment für Datenbank {BaseName} kann nicht geöffnet werden
1040	Datensegment für Datenbank {BaseName} kann nicht erzeugt werden
1041	Es kann kein Speicherplatz im Datensegment bereitgestellt werden
1042	Speicherplatz in Datensegment der Datenbank {BaseName} kann nicht freigegeben werden
1043	Datei ist schreibgeschützt
1044	Feld in Datensatz {RecNum} in Tabelle {TableName} in Datenbank {BaseName} kann nicht angesprochen werden
1045	Felddefinition fehlt: Datensatz {RecNum} in Tabelle {TableName} in Datenbank {BaseName}
1046	Datensatz {RecNum} in Tabelle {TableName} in Datenbank {BaseName} kann nicht gespeichert werden
1047	Datensatz {RecNum} in Tabelle {TableName} in Datenbank {BaseName} kann nicht geladen werden
1048	Speicherplatz für Datensatz {RecNum} in Tabelle {TableName} in Datenbank {BaseName} kann nicht bereitgestellt werden
1049	Datensatz {RecNum} in Tabelle {TableName} in Datenbank {BaseName} kann nicht geändert werden
1050	Ungültiger Header für Datensatz {RecNum} in Tabelle {TableName} in Datenbank {BaseName}
1051	Definition der Tabelle {TableName} in Datenbank {BaseName} kann nicht gespeichert werden
1052	Definition der Tabelle {TableName} in Datenbank {BaseName} kann nicht geändert werden
1053	Feldname existiert bereits
1055	Beim Speichern eines Datensatzes in {TableName} in Datenbank {BaseName} können die Indexwerte nicht aktualisiert werden
1056	Beim Speichern eines Datensatzes in {TableName} in Datenbank {BaseName} können die BLOBs nicht aktualisiert werden
1057	Beim Speichern oder Löschen eines Datensatzes in {TableName} in Datenbank {BaseName} können die BLOBs nicht gelöscht werden
1058	Feld für Tabelle {TableName} in Datenbank {BaseName} kann nicht hinzugefügt werden
1059	Speicherplatz für Tabelle kann nicht bereitgestellt werden
1061	Speicherplatz für Datensatz kann nicht bereitgestellt werden
1062	Tabelle {TableName} in Datenbank {BaseName} kann nicht vollständig gelöscht werden
1063	Datensatz {RecNum} in {TableName} in Datenbank {BaseName} kann nicht gesperrt werden
1064	Datensatz {RecNum} in {TableName} in Datenbank {BaseName} kann nicht freigegeben werden
1065	Datensatz {RecNum} in {TableName} in Datenbank {BaseName} kann nicht gelöscht werden
1066	Datensatz {RecNum} in {TableName} in Datenbank {BaseName} ist gesperrt
1067	Sequentielle Suche konnte für {TableName} in Datenbank {BaseName} nicht ausgeführt werden
1068	Header für Tabelle {TableName} in Datenbank {BaseName} konnte nicht gesichert werden
1069	Import für Tabelle {TableName} in Datenbank {BaseName} nicht möglich
1070	Indexheader für Datenbank {BaseName} kann nicht geladen werden
1071	Indexheader für Datenbank {BaseName} kann nicht gespeichert werden
1072	Seitenadressen für Index {IndexName} in Datenbank {BaseName} kann nicht geladen werden
1073	Seitenadressen für Index {IndexName} in Datenbank {BaseName} kann nicht gespeichert werden
1074	Index {IndexName} in Datenbank {BaseName} kann nicht gelöscht werden
1075	Sortierung auf Index {IndexName} in Datenbank {BaseName} nicht möglich
1076	Seite für Index {IndexName} in Datenbank {BaseName} kann nicht geladen werden
1077	Seite für Index {IndexName} in Datenbank {BaseName} kann nicht gespeichert werden
1078	Eintrag in Index {IndexName} in Datenbank {BaseName} kann nicht hinzugefügt werden
1079	Eintrag in index {IndexName} in Datenbank {BaseName} kann nicht gelöscht werden
1080	Vollständiges Löschen von index {IndexName} in Datenbank {BaseName} nicht möglich
1081	Vollständiger Scan des Index {IndexName} in Datenbank {BaseName} nicht möglich
1082	Vollständige Sortierung nach index {IndexName} in Datenbank {BaseName} nicht möglich
1083	Eintrag in Indexseite von {IndexName} in Datenbank {BaseName} kann nicht hinzugefügt werden
1084	Eintrag in Indexseite von {IndexName} in Datenbank {BaseName} kann nicht gelöscht werden
1085	Cluster Index in Datenbank {BaseName} kann nicht geladen werden
1086	Hinzufügen zu Indexcluster in Datenbank {BaseName} nicht möglich
1087	Hinzufügen zu Indexcluster in Datenbank {BaseName} nicht möglich
1088	Index {IndexName} ist ungültig
1089	SQL Syntax Error (Obsolete)
1090	SQL Token nicht gefunden (Obsolete)
1091	Nicht implementiert
1092	Code kann nicht registriert werden
1093	Operation von Anwender abgebrochen
1094	Transaktionskonflikt
1095	Ungültiger Tabellenname in Datenbank {BaseName}
1096	Tabelle {TableName} in Datenbank {BaseName} ist gesperrt
1097	Datenbank {BaseName} ist gesperrt
1098	Ungültige Datenadresse in Datenbank {BaseName}
1099	Datensatz ist leer
1100	Ungültiges Quellfeld
1101	Ungültiges Zielfeld

1102	Ungültiger Verknüpfungsname
1103	Feldtypen stimmen nicht überein
1104	Verknüpfung ist leer
1105	Verknüpfungsliste von Datenbank {BaseName} kann nicht geladen werden
1106	Verknüpfungsliste von Datenbank {BaseName} kann nicht gespeichert werden
1107	Suche und Sperre nicht erfolgreich, mindestens ein Datensatz ist bereits gesperrt
1108	Ungültiger Datensatz
1109	Ungültige Datensatz-ID
1110	Verknüpfung existiert bereits in Datenbank {BaseName}
1111	Index existiert bereits in Datenbank {BaseName}
1112	Ungültiger Vergleichsoperator
1113	Ende des Datenbuffers
1114	Ungültige DB4D Versionsnummer
1115	Doppelter Schlüssel (nicht einmalig)
1116	Zwingendes Feld ist NULL in Datensatz {RecNum} in Tabelle {TableName}
1117	Feld in Tabelle {TableName} kann nicht auf zwingend gesetzt werden
1118	Exklusiver Zugriff auf Tabelle {TableName} nicht möglich
1119	Referentielle Integritätsprüfung für Tabelle {TableName} nicht möglich
1120	Referentielle Integritätsprüfung: einige Fremdschlüssel entsprechen immer noch dem Primärschlüssel für Datensatz {RecNum} in Tabelle {TableName}
1121	Es können nicht alle ausgewählten Datensätze für Tabelle {TableName}gelöscht werden
1122	BLOB ist Null
1123	Ungültiger Datenbankkontext
1124	Ungültige Verknüpfungsreferenz
1125	Ungültiger Datensatzname in Tabelle {TableName}
1126	Ungültiger Feldtyp
1127	Zusätzliche Eigenschaften können nicht geladen werden
1128	Zusätzliche Eigenschaften können nicht gespeichert werden
1129	Unterdatensatz-ID ist außerhalb des gültigen Bereichs
1130	Bereits vorhandener Name für Index {IndexName} in Datenbank {BaseName}
1131	Ungültiger Name für Index {IndexName} in Datenbank {BaseName}
1132	Ungültiger Schlüsselwert für index {IndexName}
1133	Ungültiger Typ für Index {IndexName}
1134	Ungültiger accessor
1135	Accessor ist Nur Lesen
1136	Null-Wert nicht erlaubt
1137	THIS ist Null
1138	Auswahl ist Null
1139	Datenbank {BaseName} ist schreibgeschützt
1140	Datenbank {BaseName} wird geschlossen
1141	Ungültige Transaktion
1142	Arraybegrenzung wurde überschritten
1143	Erzeugung von Arraywerten fehlt
1144	Auswahl für Tabelle {TableName} kann nicht erzeugt werden
1145	Objekt wird benutzt
1146	Datendatei passt nicht zur Struktur
1147	Listenerprogramm kann nicht gestartet werden
1148	Server kann nicht gestartet werden
1149	Listenerprogramm existiert nicht
1150	Task wird beendet
1151	Ungültiger Anfrage-Tag
1152	Ungültige Kontext-ID
1153	Nicht genügend temporärer Speicherplatz auf Festplatte
1154	Datenset ist Null
1155	Kein primärer Schlüssel entspricht dem Fremdschlüssel
1156	Typ des Feldes {FieldName} in Tabelle {TableName} kann nicht einmalig sein
1157	Typ des Feldes {FieldName} in Tabelle {TableName} kann nicht auf NEVER NULL gesetzt werden
1158	Definition des Primärschlüssels für Tabelle {TableName} kann nicht geändert werden
1159	Maximale Anzahl Datensätze für Tabelle {TableName} erreicht
1160	Maximale Anzahl BLOBs für Tabelle {TableName} erreicht
1161	Index ist außerhalb Bereich
1162	Ungültige Suche
1163	Datensatz ist NULL
1164	Objekt ist NULL
1165	Ungültiger Eigentümer für das Objekt
1166	Objekt war nicht gesperrt
1167	Objekt wird von einem anderen Kontext gesperrt
1168	Interner Fehler bei Fernverbindung

1169 Ungültige Tabellennummer
1170 Ungültige Feldnummer
1171 Ungültige Datenbank-ID
1172 Ungültiger Parameter
1173 Sichern des Caches nicht erfolgreich
1174 Sichern der Daten nicht erfolgreich
1175 Sichern der Struktur nicht erfolgreich
1176 Ungültiges Logbuch für Datenbank {BaseName}
1177 Logbuch für Datenbank {BaseName} kann nicht gefunden werden
1178 Letzte Operation in Logbuch entspricht nicht der in Datenbank {BaseName}
1179 Falsches Logbuch für Datenbank {BaseName}
1180 Datentabelle wurde gelöscht
1181 Einträge in Index {IndexName} nicht einmalig
1182 Logbuch für Datenbank {DataBase} kann nicht angelegt werden
1183 In Logbuch für Datenbank {DataBase} kann nicht geschrieben werden
1184 Tabelle {TableName} in Datenbank {DataBase} kann nicht gelöscht werden
1185 Remote Datenbank kann nicht geöffnet werden
1186 Logbuch kann nicht in Datenbank {DataBase} integriert werden
1187 Interne Berechnung auf Menge kann nicht ausgeführt werden
1188 Array kann nicht gespeichert werden
1189 Array kann nicht geladen werden
1190 Header der Sequenznummern kann nicht geladen werden
1191 Datensatz kann nicht ausgewählt werden
1192 Datensatz kann nicht erzeugt werden
1193 Übertragung von Zusammenstellung in Daten für Tabelle {TableName} in Datenbank {BaseName} nicht erfolgreich
1194 Übertragung von Daten zu Zusammenstellung für Tabelle {TableName} in Datenbank {BaseName} nicht erfolgreich
1195 Sequentielle Sortierung kann nicht durchgeführt werden
1196 Auswahl kann nicht gesperrt werden
1197 Index Eintrag kann nicht geladen werden
1198 Index Eintrag kann nicht gespeichert werden
1199 Index Eintrag kann nicht erzeugt werden
1200 Suche kann nicht durchgeführt werden
1201 Suche kann nicht analysiert werden
1202 Formel kann auf dieser Spalte nicht ausgeführt werden
1203 Suche konnte nicht ausgeführt werden
1204 Suche konnte nicht analysiert werden
1205 Nicht alle eindeutigen Werte aus Tabelle {TableName} in Datenbank {BaseName} konnten ermittelt werden
1206 Array mit Werten für Tabelle {TableName} in Datenbank {BaseName} konnte nicht erzeugt werden
1207 Auswahl konnte nicht geladen werden
1208 Daten konnte nicht gesendet werden
1209 Antwort auf Anfrage konnte nicht empfangen werden
1210 Anfrage konnte nicht gesendet werden
1211 Verbindung konnte nicht hergestellt werden
1212 Index {IndexName} konnte nicht schnell in Datenbank {BaseName} erzeugt werden
1213 Eindeutige Index Werte konnten nicht erzeugt werden
1214 Auswahl für Tabelle {TableName} in Datenbank {BaseName} konnte nicht sortiert werden
1215 Adresstabelle konnte nicht geladen werden
1216 Adresstabelle konnte nicht geändert werden
1217 Neuer Eintrag in Adresstabelle konnte nicht erzeugt werden
1218 Leerer Eintrag in Adresstabelle konnte nicht erzeugt werden
1219 Transaktion temporärer Datensatz konnte nicht gesichert werden
1220 Transaktion temporäres Blob konnte nicht gesichert werden
1221 Transaktion temporärer Datensatz konnte nicht geladen werden
1222 Transaktion temporäres Blob konnte nicht geladen werden
1223 Transaktion konnte nicht gestartet werden
1224 Transaktion konnte nicht bestätigt werden
1225 Zusätzliche Eigenschaften konnten nicht geladen werden
1226 Zusätzliche Eigenschaften konnten nicht gespeichert werden
1227 Tabellenname wird bereits verwendet
1228 Liste der NULL Einträge von index {IndexName} konnte nicht geladen werden
1229 Liste der NULL Einträge von index {IndexName} konnte nicht geändert werden
1230 Ungültiger Eintrag für Index {IndexName}
1231 Logbuch kann nicht gesetzt werden
1232 Kontext ist NULL
1233 Datenbank {BaseName} kann nicht gesperrt werden
1234 Ungültige Feldreferenz # {RecNum} in Tabelle: {TableName} in Datenbank: {BaseName}
1235 Ungültige Feldreferenz in Tabelle: {TableName} in Datenbank: {BaseName}
1236 Daten können nicht aus temporärer Transaktionsdatei gelesen werden

1237	Cartesian Produkt fehlgeschlagen
1238	Auswahlen können nicht vereint werden
1239	Das Format der Datenbank {BaseName} kann im Nur-Lesen-Modus nicht aktualisiert werden
1240	Falscher Header
1241	Falsche Prüfsumme
1243	Datentabellen der Datenbank: {BaseName} können nicht geladen werden
1244	Die Liste der Fremdschlüssel-Constraints ist für Tabelle: {TableName} der Datenbank: {BaseName} nicht leer
1245	Adresseneintrag ist nicht leer
1246	Adresse nicht vor bereitgestellt werden
1247	Neuer Datensatz für Tabelle {TableName} der Datenbank {BaseName} kann nicht aktualisiert werden
1248	Neuer Datensatz für Tabelle {TableName} der Datenbank {BaseName} kann nicht gespeichert werden
1249	Unterdatensatz kann nicht gespeichert werden
1250	Datensatz kann nicht gespeichert werden
1251	Datenbank Struktur Objekt Definition kann nicht gesperrt werden
1252	Datenbank Struktur Objekt Definition kann nicht entsperrt werden
1253	Ungültige Relations-Nummer
1254	Zirkelreferenz in Datensatzadresstabelle für Tabelle {TableName} der Datenbank {BaseName}
1255	Zirkelreferenz in Blob-Adreßtabelle für Tabelle {TableName} der Datenbank {BaseName}
1256	Doppelter Schema Name in Datenbank {BaseName}
1257	Schema kann nicht in Datenbank {BaseName} gesichert werden
1258	Schema kann nicht aus Datenbank {BaseName} gelöscht werden
1259	Schema in Datenbank {BaseName} kann nicht umbenannt werden
1260	Das ausgewählte Logbuch ist für Datenbank {BaseName} zu neu
1261	Das ausgewählte Logbuch ist für Datenbank {BaseName} zu alt
1262	Einige Datentabellen passen nicht zu den Tabellendefinitionen der Struktur
1263	Der gegebene Stamp passt nicht zum aktuellen Datensatz # {RecNum} der Tabelle {TableName}
1264	Ein Primärschlüssel wird benötigt, fehlt aber für Tabelle {TableName}
1265	Ungültiges Feld
1266	Unbekannter Feldtyp (vermutlich ist diese Version von 4D zu alt)
1267	Stapelüberlauf
1268	Datentabelle konnte für Datenbank {BaseName} nicht wiederhergestellt werden
1269	Datentabelle kann nicht gefunden werden
1270	Fehlende Struktur konnte nicht neu erzeugt werden
1271	ungültiger REST Request Handler
1272	Es sind noch einige Datensätze in Tabelle {tableName} der Datenbank {BaseName} gesperrt
1273	Tabelle {TableName} der Datenbank {BaseName} kann nicht gelöscht werden
1274	Der Zugriff auf die Backup-Journaldatei der Datenbank {BaseName} ist nicht mehr möglich. Zur Sicherheit des Datenbestandes wurden alle Operationen angehalten. Bitte benachrichtigen Sie umgehend Ihren Administrator!
1275	"(" fehlt am Beginn der JavaScript Anweisung in der Suche.
1276	Datenstrom hat ungültigen Header.
1277	Die Integration der Backup-Journaldatei ist bei Eintrag # {p1} fehlgeschlagen.
1278	Javascript Code ist in der Suche nicht erlaubt.
1300	Ungültiger Auswahltyp
1301	Array ist zu groß
1302	Arraygröße stimmt nicht
1303	Ungültige Auswahl-ID
1304	Ungültiger Auswahlteil
4001	Falsche Tabellennummer von Plug-In angefragt.
4002	Falsche Datensatznummer von Plug-In angefragt.
4003	Falsche Datenfeldnummer von Plug-In angefragt.
4004	Aktueller Datensatz von Plug-In angefragt ist nicht vorhanden.

Hinweise

1. Einige der Fehlermeldungen weisen auf ernste Probleme hin, z.B. *-10502 Defekte Datensatzstruktur (Datendatei muss repariert werden)*, andere Fehlermeldungen können mit der Projektmethode **ON ERR CALL** verwaltet werden. Damit beheben Sie zum Beispiel den Fehler -9998 Dieser einmalige Datensatz ist bereits vorhanden, wenn Ihre Anwendung doppelte Werte in einer Tabelle zulässt, die ein indiziertes Datenfeld mit dem Attribut einmalig enthält.
2. Einige der Fehlermeldungen treten nie auf Programmierebene auf, sondern auf niederer Ebene. Sie lassen sich mit Routinen der Datenbank-Engine oder mit 4D Backup beheben.
3. Der Fehler *-10503 Datensatznummer nicht vorhanden* bedeutet in der Regel, dass Ihr Code (z.B. der Befehl **GOTO RECORD** versucht, auf einen Datensatz zuzugreifen, der nicht mehr existiert oder noch nie existiert hat. Der Fehler kann in manchen Fällen auch bedeuten, dass die Datenbank repariert werden muss.
4. Der Fehler *-9999 Nicht genügend Speicher, um den Datensatz zu sichern* tritt auf, wenn die Datendatei Ihrer Datenbank voll ist oder in einem vollen Volume liegen oder wenn die Datendatei gesperrt ist bzw. in einem gesperrten Volume liegt.

SQL Engine Fehlermeldungen (1001 - 3018)

Die SQL Engine von 4D gibt spezifische Fehler zurück, die nachfolgend aufgelistet sind. Diese Fehler lassen sich mit einer Fehlerverwaltungsmethode abfangen, die über den Befehl **ON ERR CALL** eingerichtet wurde und über den Befehl **GET LAST ERROR STACK** analysieren.

Generische Fehler

1001	INVALID ARGUMENT
1002	INVALID INTERNAL STATE
1003	SQL SERVER IS NOT RUNNING
1004	Zugriff untersagt
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

Semantische Fehler

1101 Tabelle '{key1}' existiert nicht in Datenbank
1102 Spalte '{key1}' existiert nicht
1103 Tabelle '{key1}' ist in FROM Klausel nicht definiert
1104 Spaltennamen-Referenz '{key1}' ist zweideutig
1105 Tabellen-Alias '{key1}' ist identisch zum Tabellennamen
1106 Doppelter Tabellen-Alias - '{key1}'
1107 Doppelte Tabelle in FROM Klausel
1108 Operation {key1} {key2} {key3} ist nicht Typ Sicher.
1109 Ungültiger ORDER BY index - {key1}.
1110 WRONG AMOUNT OF PARAMETERS
1111 INCOMPATIBLE PARAMETER Typ
1112 Unbekannte Funktion - {key1}.
1113 Division durch Null.
1114 Sortierung nach indiziertem Element der SELECT Liste ist nicht erlaubt
- ORDER BY Element {key1}.
1115 DISTINCT NOT ALLOWED
1116 Verschachtelte Aggregat-Funktionen sind nicht in einer Aggregat-Funktion {key1} erlaubt.
1117 Spaltenfunktion ist nicht erlaubt.
1118 Spalten und Scalar Operationen können nicht gemischt werden.
1119 Ungültiger GROUP BY index - {key1}.
1120 GROUP BY Index ist nicht erlaubt.
1121 GROUP BY Index ist nicht mit 'SELECT * FROM ...' erlaubt.
1122 HAVING ist kein Aggregat-Ausdruck.
1123 Spalte '{key1}' ist keine Gruppenspalte und kann nicht in der ORDER BY
Klausel verwendet werden.
1124 Mischung von {key1} und {key2} Typen in IN Prädikat nicht möglich.
1125 Escape Sequence '{key1}' in LIKE Prädikat zu lang. Es muss exakt ein Zeichen sein.
1126 Ungültiger Escape Character - '{key1}'.
1127 Unbekannte Escape Sequence - '{key1}'.
1128 Spaltenreferenzen von mehr als einer Suche in Aggregate Funktion {key1} sind nicht erlaubt.
1129 Scalar Eintrag in der SELECT Liste sind bei Verwendung einer GROUP BY
Klausel nicht erlaubt.
1130 Unterabfrage erzeugt mehr als eine Spalte.
1131 Eine Unterabfrage darf nur eine Zeile zurückgeben
1132 INSERT Werteanzahl {key1} entspricht nicht Spaltenanzahl {key2}.
1133 Doppelte Spaltenreferenz in INSERT Liste - '{key1}'.
1134 Spalte '{key1}' erlaubt keine NULL Werte.
1135 Doppelte Spaltenreferenz in UPDATE Liste - '{key1}'.
1136 Tabelle '{key1}' existiert bereits.
1137 Doppelte Spalte '{key1}' in CREATE TABLE Befehl
1138 DUPLICATE COLUMN IN COLUMN LIST
1139 Mehr als ein Primärschlüssel ist nicht erlaubt.
1140 Mehrdeutiger Fremdschlüssel Name - '{key1}'.
1141 Spaltenanzahl {key1} der Kindtabelle entspricht nicht der Spaltenanzahl
{key2} in der Eltern-Tabelle der Fremdschlüsseldefinition.
1142 Spaltentyp Unterschied in der Fremdschlüsseldefinition. Relation von {key1}
der Kindtabelle zu {key2} nicht möglich.
1143 Passende Spalte in Elterntabelle für '{key1}' Spalte der Kindtabelle nicht gefunden.
1144 UPDATE und DELETE Grenzen müssen identisch sein
1145 FOREIGN KEY DOES NOT EXIST
1146 Ungültiger Wert für LIMIT in SELECT Befehl - {key1}.
1147 Ungültiger Wert OFFSET in SELECT Befehl - {key1}.
1148 Primärer Schlüssel existiert nicht in Tabelle '{key1}'.
1149 FAILED TO CREATE FOREIGN KEY
1150 Spalte {key1} ist nicht Teil des primary key.
1151 FIELD IS NOT UPDATEABLE
1152 FOUND VIEW COLUMN
1153 Ungültige Datentyplänge '{key1}'.
1154 EXPECTED EXECUTE IMMEDIATE COMMAND
1155 INDEX ALREADY EXISTS
1156 Option Auto-increment ist nicht erlaubt für Spalte '{key1}' vom Typ {key2}.
1157 SCHEMA ALREADY EXISTS
1158 SCHEMA DOES NOT EXIST
1159 System Schema kann nicht gelöscht werden.
1160 CHARACTER ENCODING NOT ALLOWED

Implementation Fehler

1203 FUNCTIONALITY IS NOT IMPLEMENTED
1204 Datensatz {key1} kann nicht erzeugt werden.
1205 Feld '{key1}' kann nicht aktualisiert werden.
1206 Datensatz '{key1}' kann nicht gelöscht werden.
1207 NO MORE JOIN SEEDS POSSIBLE
1208 FAILED TO CREATE TABLE
1209 FAILED TO DROP TABLE
1210 CANT BUILD BTREE FOR ZERO RECORDS
1211 COMMANDE COUNT GREATER THAN ALLOWED
1212 FAILED TO CREATE DATABASE
1213 FAILED TO DROP COLUMN
1214 Wert IS OUT OF BOUNDS
1215 FAILED TO STOP SQL_SERVER
1216 FAILED TO LOCALIZE
1217 Tabelle kann nicht zum Lesen gesperrt werden.
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

Analysefehler

1301 PARSING FAILED

Zugriffsfehler auf Runtime language

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 Der Query Parameter ist entweder nicht erlaubt oder nicht vorhanden.
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

Datumsanalyse

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 Zeitstempel kann nicht geparst werden.
1513 Zeitdauer kann nicht geparst werden.
1551 FAILED TO PARSE DATE FORMAT

Lexer Fehler

1601 NULL INPUT STRING
1602 NON TERMINATED STRING
1603 NON TERMINATED COMMENT
1604 INVALID NUMBER
1605 UNKNOWN START OF TOKEN
1606 NON TERMINATED NAME
1607 NO VALID TOKENS

Bestätigungsfehler für error stack - state errors, die auf direkte Fehler folgen

1701 Tabelle kann nicht bestätigt werden '{key1}'.
1702 FROM Klausel kann nicht bestätigt werden.
1703 GROUP BY Klausel kann nicht bestätigt werden.
1704 SELECT Liste kann nicht bestätigt werden.
1705 WHERE Klausel kann nicht bestätigt werden.
1706 ORDER BY Klausel kann nicht bestätigt werden.
1707 HAVING Klausel kann nicht bestätigt werden.
1708 COMPARISON Prädikat kann nicht bestätigt werden.
1709 BETWEEN Prädikat kann nicht bestätigt werden.
1710 IN Prädikat kann nicht bestätigt werden.
1711 Failed to validate LIKE predicate.
1712 ALL ANY Prädikat kann nicht bestätigt werden.
1713 EXISTS Prädikat kann nicht bestätigt werden.
1714 IS NULL Prädikat kann nicht bestätigt werden.
1715 Subquery kann nicht bestätigt werden.
1716 SELECT Element {key1} kann nicht bestätigt werden.
1717 Spalte '{key1}' kann nicht bestätigt werden.
1718 Funktion '{key1}' kann nicht bestätigt werden.
1719 CASE Ausdruck kann nicht bestätigt werden.
1720 Befehlsparameter kann nicht bestätigt werden.
1721 Funktionsparameter {key1} kann nicht bestätigt werden.
1722 INSERT Element {key1} kann nicht bestätigt werden.
1723 UPDATE Element {key1} kann nicht bestätigt werden.
1724 Spaltenliste kann nicht bestätigt werden.
1725 Fremdschlüssel kann nicht bestätigt werden
1726 SELECT Befehl kann nicht bestätigt werden.
1727 INSERT Befehl kann nicht bestätigt werden.
1728 DELETE Befehl kann nicht bestätigt werden.
1729 UPDATE Befehl kann nicht bestätigt werden.
1730 CREATE TABLE Befehl kann nicht bestätigt werden.
1731 DROP TABLE Befehl kann nicht bestätigt werden.
1732 ALTER TABLE Befehl kann nicht bestätigt werden.
1733 CREATE INDEX Befehl kann nicht bestätigt werden.
1734 LOCK TABLE Befehl kann nicht bestätigt werden.
1735 Fehler beim Berechnen des LIKE Prädikat Musters.

Ausführungsfehler für error stack - state errors, die auf direkte Fehler folgen

1801 SELECT Befehl kann nicht ausgeführt werden.
1802 INSERT Befehl kann nicht ausgeführt werden.
1803 DELETE Befehl kann nicht ausgeführt werden.
1804 UPDATE Befehl kann nicht ausgeführt werden.
1805 CREATE TABLE Befehl kann nicht ausgeführt werden.
1806 DROP TABLE Befehl kann nicht ausgeführt werden.
1807 CREATE DATABASE Befehl kann nicht ausgeführt werden.
1808 ALTER TABLE Befehl kann nicht ausgeführt werden.
1809 CREATE INDEX Befehl kann nicht ausgeführt werden.
1810 DROP INDEX Befehl kann nicht ausgeführt werden.
1811 LOCK TABLE Befehl kann nicht ausgeführt werden.
1812 TRANSACTION Befehl kann nicht ausgeführt werden.
1813 WHERE Klausel kann nicht ausgeführt werden.
1814 GROUP BY Klausel kann nicht ausgeführt werden.
1815 HAVING Klausel kann nicht ausgeführt werden.
1816 Aggregat fehlgeschlagen.
1817 DISTINCT kann nicht ausgeführt werden.
1818 ORDER BY Klausel kann nicht ausgeführt werden.
1819 DB4D Query kann nicht erzeugt werden.
1820 Vergleichs-Prädikat kann nicht berechnet werden.
1821 Subquery kann nicht ausgeführt werden.
1822 Fehler beim Berechnen des BETWEEN Prädikat.
1823 Fehler beim Berechnen des IN Prädikat.
1824 Fehler beim Berechnen des ALL/ANY Prädikat.
1825 Fehler beim Berechnen des LIKE Prädikat.
1826 Fehler beim Berechnen des EXISTS Prädikat.
1827 Fehler beim Berechnen des NULL Prädikat.
1828 Fehler beim Ausführen einer arithmetischen Operation.
1829 Fehler beim Berechnen des Funktionsaufrufes '{key1}'.
1830 Fehler beim Berechnen des Case Ausdrucks.
1831 Fehler beim Berechnen des Funktionsparameters '{key1}'.
1832 Fehler beim Berechnen des 4D Funktionsaufrufes.
1833 Fehler beim Sortieren beim Ausführen der ORDER BY Klausel.
1834 Fehler beim Berechnen des Datensatz-Hash.
1835 Fehler beim Vergleichen von Datensätzen.
1836 Fehler beim Berechnen des INSERT Werts {key1}.
1837 DB4D QUERY FAILED
1838 FAILED TO EXECUTE ALTER SCHEMA COMMAND
1839 FAILED TO EXECUTE GRANT COMMAND

Zwischenspeicherung

2000 CACHEABLE NOT INITIALIZED
2001 VALUE ALREADY CACHED
2002 CACHED VALUE NOT FOUND
2003 CACHE IS FULL
2004 CACHING IS NOT POSSIBLE

Protokollfehler

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 Typ
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

Netzwerkfehler (-10051 -> 10001)

Folgende Tabelle beschreibt die Fehler, die mit einer Netzwerkverbindung auftreten können.

Code	Beschreibung
-10051	Falscher Wert in Get/SetOption.
-10050	Unbekannte Option in Get/SetOption.
-10033	Falsche Datengröße beim Lesevorgang.
-10031	Desynchronisation beim Lesevorgang.
-10030	Desynchronisation beim Schreibvorgang.
-10021	Kein Server gefunden.
-10020	Kein Server ausgewählt.
-10003	Falsche Parameter für die Verbindung.
-10002	Die Verbindung für diesen Prozess wurde unterbrochen oder konnte nicht hergestellt werden.
-10001	Die aktuelle Verbindung zur Datenbank wurde unterbrochen.
2	Benutzer hat beim Verwenden der Funktion OP Select 4D Server auf die Schaltfläche **Andere** geklickt.

Backup-Verwaltung Systemfehler (1401 -> 1421)

Folgende Tabelle zeigt die Fehlermeldungen, die bei Backup oder Wiederherstellen generiert werden.
Sie können diese Fehler mit einer Unterbrechungsmethode ausfindig machen, die über den Befehl **ON ERR CALL** installiert wird.

Code	Beschreibung
1401	Die maximale Anzahl der Backup-Versuche wurde erreicht; das automatische Backup ist zeitweise deaktiviert.
1403	Kein Logbuch.
1404	Eine Transaktion wird in diesem Prozess geöffnet.
1405	Das Timeout zum Beenden von Transaktionen in einem konkurrierenden Prozess wurde erreicht.
1406	Das Backup wurde vom Anwender abgebrochen.
1407	Der Zielordner ist ungültig.
1408	Ein Fehler trat beim Sichern des Logbuchs auf.
1409	Ein Fehler trat beim Backup auf.
1410	Die zu prüfende Backup-Datei kann nicht gefunden werden.
1411	Ein Fehler trat beim Prüfen der Backup-Datei auf.
1412	Die zu prüfende Logbuch-Datei kann nicht gefunden werden.
1413	Ein Fehler trat beim Prüfen der Logbuch-Datei auf.
1414	Dieser Befehl kann nur auf 4D Server ausgeführt werden.
1415	Backup des Logbuchs ist nicht möglich; es läuft gerade eine kritische Operation.
1416	Dieses Logbuch passt nicht zur geöffneten Datenbank.
1417	Ein Logbuch wird gerade integriert. Backup kann nicht gestartet werden.
1420	Integration abgebrochen, da gesperrte Datensätze gefunden wurden.
1421	Dieser Befehl lässt sich nicht in einer Client/Server-Umgebung verwenden.

- Die Fehler 1408 und 1409 stammen in der Regel von einem Fehler beim Lesen für Dateien, die gesichert werden sollen oder einem Fehler beim Schreiben während dem Backup der Datei.
- Die Fehler 1411 und 1413 treten während dem Prüfen von Archiven auf.
  Treten diese Fehler auf, sollten Sie zuerst den vorhandenden Speicherplatz auf der Platte und die Zugriffsrechte auf den Lese-/Schreibmodus überprüfen.

Client Update Fehler (-10650 -> -10655)

Nachfolgende Tabelle beschreibt die Fehler, die bei einem Client Update auftreten können:

Fehlernr.	Beschreibung
-10650	Client Update kann nicht geladen werden
-10651	Client Update kann nicht entpackt werden
-10652	Update Information ist nicht verfügbar ({path})
-10653	Update kann nicht heruntergeladen werden '{path}'
-10654	Update Information ungültig
-10655	Client Update ist nicht verfügbar

Updater Fehler (-10603 -> -10613)

Nachfolgende Tabelle beschreibt die möglichen Updater Fehler:

Fehlernr.	Beschreibung
-10603	Updater Installation fehlgeschlagen
-10604	Updater Paket kann nicht identifiziert werden {path}
-10605	Fehler beim Kopieren der Dateien des Updater Pakets
-10606	Aktueller Updater kann nicht deaktiviert werden
-10607	Der neue Updater Installlationsordner kann nicht erzeugt werden
-10608	Der Ordner mit dem Software Update kann nicht gefunden werden {path}
-10609	Der Software Updater kann nicht gestartet werden
-10611	Der Software Updater kann nicht gestartet werden (fehlerhafte Installation)
-10612	Die laufende Anwendung kann nicht mit sich selbst aktualisiert werden
-10613	Es können keine zwei Fenster vom Typ Toolbar erzeugt werden

OS Systemfehler (-124 -> -33)

Folgende Tabelle zeigt die Fehler im Datei-Manager des Betriebssystems (Operating System, abgekürzt OS). Diese Fehler treten insbesondere bei Befehlen im Kapitel **Systemdokumente** auf. “Volume” steht als Oberbegriff für Volume, Festplatte, Netzlaufwerk, etc.

Code	Beschreibung
-124	Es wurde ein Zugriff versucht auf ein nicht angeschlossenes gemeinsam nutzbares Volume.
-121	Dieser Zugriffspfad konnte nicht benutzt werden.
-120	Dateizugriff über Pfadname zu einem nicht existierenden Ordner.
-84	Volume-Fehler beim Überprüfen der Daten (Installation oder Formatierung ist falsch).
-64	Laufwerk nicht installiert oder ansprechbar (Installation oder Formatierung ist falsch).
-61	Lese-/Schreiberlaubnis lässt Schreiben nicht zu.
-60	Beschädigter Block im Hauptverzeichnis. Volume beschädigt.
-58	Fehler im externen Dateisystem.
-57	Dies ist kein Macintosh Volume.
-54	Es wurde versucht, ein gesperrtes Volume im Schreibmodus zu öffnen.
-53	Volume ist nicht "on-line" (ausgeworfen).
-52	Fehler beim Dateizugriff. Position des Dateizeigers ist ungültig.
-51	Kein Zugriff auf das Dokument möglich, da Referenznummer des Dokumentes ungültig.
-49	Datei ist bereits im Lese/Schreib-Modus geöffnet.
-48	Es wurde versucht, eine bereits vorhandene Datei zu erstellen.
-47	Die Datei ist schon offen oder der Ordner ist nicht leer.
-46	Volume ist durch Anwendung geschützt (Software).
-45	Datei ist gesperrt oder der Pfadname ist nicht korrekt.
-44	Volume ist geschützt (Hardware).
-43	Angegebene Datei kann nicht gefunden werden.
-42	Zu viele Dateien sind gleichzeitig geöffnet.
-41	Nicht genug Speicher, um neue Datei im Volume zu öffnen.
-40	Es wurde versucht, vor dem Anfang der Datei zu lesen bzw. zu schreiben.
-39	Es wurde versucht, nach dem Ende der Datei zu lesen.
-38	Es wurde versucht, in einer geschlossenen Datei zu lesen bzw. zu schreiben.
-37	Dateiname inkorrekt. Er ist zu lang bzw. enthält ein ungültiges Zeichen.
-36	Eingabe/Ausgabe Fehler, evtl. wegen beschädigtem Block im Volume
-35	Angegebenes Volume existiert nicht.
-34	Volume ist voll. Es ist kein Platz mehr vorhanden.
-33	Inhaltsverzeichnis des Volume ist voll. Sie können keine neuen Dateien anlegen.
-17	Der angesprochene Treiber reagiert nicht.

OS Speicherfehler (-117 -> -108)

Folgende Tabelle zeigt die Fehlermeldungen des Speichermanagers des Betriebssystems.

Code	Beschreibung
-117	Interner Speicherfehler. Der Inhalt des Hauptspeichers ist evtl. logisch verfälscht. Bitte sofort beenden und den Rechner neu starten.
-111	Interner Speicherfehler. Der Inhalt des Hauptspeichers ist evtl. logisch verfälscht. Bitte sofort beenden und den Rechner neu starten. (*)
-109	Interner Speicherfehler. Der Inhalt des Hauptspeichers ist evtl. logisch verfälscht. Bitte sofort beenden und den Rechner neustarten.
-108	Kein Speicherplatz verfügbar. Weisen Sie der 4D Anwendung mehr Speicher zu.

Tipp: Arbeiten Sie mit umfangreichen Arrays, BLOBs, Bildern oder Mengen (Objekte, die große Datenmengen verwalten können), verwenden Sie zum Überprüfen von Fehler -108 eine Projektmethode **ON ERR CALL**.

(*) Der Fehler -111 kann auch auftreten, wenn Sie versuchen, einen Wert in einem BLOB mit Offset außerhalb des gültigen Bereichs zu lesen. In diesem Fall ist es ein kleinerer Fehler, Sie müssen die Arbeitssitzung nicht beenden. Es reicht aus, wenn Sie das dem Befehl BLOB übergebene Offset korrigieren.

OS Druckerfehler (-8192 -> -1)

Nachfolgende Tabelle zeigt die Fehlermeldungen des Druckermanagers des Betriebssystems. Diese Fehler können während dem Druckvorgang auftreten.

Code	Beschreibung
-8192	LaserWriter Zeitfehler (time-out).
-8151	Drucker wurde mit anderer Druckertreiberversion initialisiert.
-8150	Kein LaserWriter ausgewählt.
-4101	Drucker arbeitet oder ist nicht angeschlossen.
-4100	Druckerverbindung wurde unterbrochen.
-193	Dokument nicht gefunden, wahrscheinlich ist der Druckertreiber oder das System defekt. Installieren Sie beides neu.
-128	Druckvorgang vom Benutzer unterbrochen.
-27	Problem beim Öffnen bzw. Schließen der Druckerverbindung.
-1	Problem beim Sichern der zu druckenden Datei.

Ressourcenfehler (-196 -> -1)

Nachfolgende Tabelle zeigt die Fehlermeldungen des Ressourcen-Managers des Betriebssystems.

Code	Beschreibung
-196	Ressource kann nicht gelöscht werden.
-194	Ressource kann nicht hinzugefügt werden.
-193	Ressourcentabelle beschädigt (Datei muss repariert werden).
-192	Ressource wurde nicht gefunden.
-1	Ressourcendatei kann nicht geöffnet werden.

OS Sound Fehler (-209 -> -203)

Folgende Tabelle zeigt die Fehlermeldungen des Sound-Managers des Betriebssystems.

Code	Beschreibung
-209	Soundkanal arbeitet.
-207	Nicht genügend Speicher zum Darstellen des Sounds vorhanden.
-206	Falsches Format der Sound-Ressource.
-205	Soundkanal ist logisch defekt.
-204	Soundressource konnte nicht geladen werden.
-203	Zuviele Soundbefehle

OS Serieller Port Fehler (-28)

Folgende Tabelle zeigt die Fehlermeldung des seriellen Port-Managers des Betriebssystems.

Code	Beschreibung
-28	Kein serieller Port geöffnet.

Mac OS Systemfehler (4 -> 28)

Folgende Tabelle zeigt einige Fehlermeldungen des Mac OS Systems. Im Normalfall können Sie diese Fehler nicht direkt beheben. Hier ist ein Neustart des Rechners erforderlich.

Code	Beschreibung
4	Division durch Null
15	Fehler beim Segmentladen: 4D scheitert beim Laden eines eigenen CodeSegments. Sie müssen 4D mehr Speicher zuweisen.
17 bis 24	Ein System Package fehlt. Prüfen Sie, ob Ihr Systemverzeichnis korrekt installiert wurde.
25	Speicher ist voll. Sie müssen 4D mehr Speicher zuweisen.
28	Stapelspeicher überlappt mit Heap der Anwendung. Sie müssen 4D mehr Speicher zuweisen.

Verschiedene Fehler (-10700)

Nachfolgende Tabelle beschreibt Fehler, die zu keiner spezifischen Gruppe gehören:

Fehlernummer	Beschreibung
-10700	Port muss im Bereich 0 bis 65535 liegen.

Zeichensätze

- Unicode Codes
- ASCII Codes
- Funktionstasten

Unicode Codes

In Datenbanken, die mit 4D Version 11 erstellt wurden, speichern und arbeiten die Programmiersprache und die Datenbank-Engine nativ mit Unicode Zeichen. Das erleichtert die internationale Verwendung von 4D Anwendungen. Unicode ist ein standardisierter einheitlicher Zeichensatz, der praktisch alle gängigen Sprachen auf der Welt bedienen kann. Ein Zeichensatz besteht aus einer Tabelle, in der jedem Zeichen eine Nummer zugeordnet ist, z.B. "a"->1, "b"->2, "5"->15, "oe"->662. Während im ASCII-Code die Basiswerte zwischen 1 und 127 liegen, liegt bei Unicode die Obergrenze über 65.000. Dies bedeutet, dass praktisch jedes Zeichen aller Sprachen darstellbar ist.

Es gibt verschiedene Arten, Zahlenwerte in Unicode zu codieren: UTF-16 codiert sie in Zahlen mit 16-bit, UTF-32 in Zahlen mit 32-bit und UTF-8 in Zahlen mit 8-bit.
4D arbeitet im allgemeinen mit UTF-16, wie Windows und Mac OS. In manchen Fällen, wie z.B. für spezitische Anforderungen für das Internet, verwendet 4D den Satz UTF-8, da er kompakter und für gängige Zeichen, d.h. a-z,0-9, leichter lesbar ist.

Hinweis: 4D verwendet intern UTF-16 Codierung, um Text in Feldern und Variablen zu speichern - selbst wenn die Anwendung im Nicht-Unicode Modus läuft. Wenn nicht anders angegeben, verweist ein über die Programmiersprache verwaltetes Zeichen, eine Position oder Länge immer auf Werte in UTF-16.

Weitere Informationen über den Unicode Standard finden Sie z.B. unter:
http://en.wikipedia.org/wiki/Unicode
Die Liste der Unicode Codes unter:
http://en.wikipedia.org/wiki/List_of_Unicode_characters#See_also

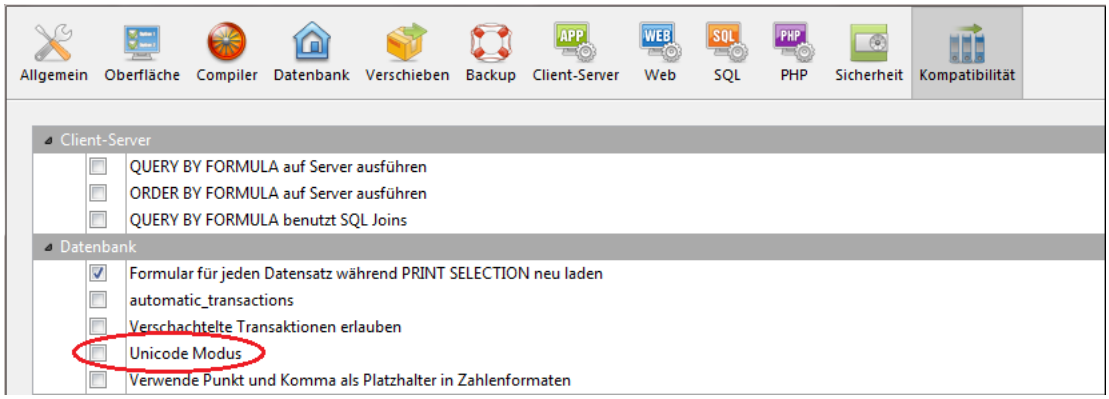
Warnung: Folgende Zeichencodes sind für Unicode reserviert und dürfen nie in einen Text eingefügt werden:
0
65534 (FFFE)
65535 (FFFF)

Hinweis zur Kompatibilität:
Anwendungen, die aus einer 4D Version vor Version 11 konvertiert wurden, können im ASCII Kompatibilitätsmodus laufen. Weitere Informationen dazu finden Sie im Abschnitt **EXPORT TEXT**.

ASCII Codes

Hinweis zur Kompatibilität: 4D kann mit zwei Zeichensätzen arbeiten: Unicode oder ASCII. Seit Version 11 von 4D wird standardmäßig der Unicode Modus verwendet. Der ASCII Modus wird aus Kompatibilitätsgründen für Anwendungen, die mit einer älteren 4D Version erstellt wurden, beibehalten. Dieser Modus heißt ASCII Kompatibilitätsmodus.

Sie können den Modus Unicode für konvertierte Datenbanken über den Selektor *Unicode Mode* der Routinen **Get database parameter** und **SET DATABASE PARAMETER** oder über die Option **Unicode-Modus** in den Datenbank-Eigenschaften auf der Seite **Kompatibilität** aktivieren:



In den meisten Fällen beeinträchtigt diese Einstellung nicht die eigentliche Funktionsweise der Anwendungen, da 4D alle notwendigen Zeichenumwandlungen intern regelt. Außerdem haben die gängigen Zeichen (a-z, 0-9) usw. in Unicode und ASCII (Windows und Mac OS) denselben Wert (von1 bis127).
Bestimmte Anweisungen der Programmiersprache, insbesondere bei Befehlen, die Zeichensätze enthalten, müssen angepasst werden. So gibt zum Beispiel die Anweisung **Char**(200) in Unicode einen anderen Wert als in ASCII zurück. Die unterschiedliche Funktionsweise im Unicode Modus und im ASCII Kompatibilitätsmodus wird bei allen davon betroffenen Befehlen beschrieben.

Hinweis: Dieser Modus gilt immer für eine Anwendung. So kann es z.B. eine Unicode-Anwendung mit Komponenten ohne Unicode geben, und umgekehrt.

ASCII Codes und 4D

Im ASCII-Kompatibilitätsmodus arbeiten das interne Datenbanksystem und die 4D Programmiersprache sowohl unter Windows als auch auf Macintosh mit dem erweiterten ASCII-Zeichensatz des Macintosh.
Beim Eingeben von Daten (z.B. Hinzufügen von Datensätzen, Bearbeiten von Methoden, etc.) wandelt 4D über das interne Altura ASCII Konvertierungsschema automatisch alles, was von der Tastatur kommt (im Zeichensatz von Windows ausgedrückt) in den Zeichensatz des Macintosh um. Für ein „é" beispielsweise wird ALT+0233 eingegeben. 4D speichert den ASCII Code 142 im Datensatz. Das ist für den Endbenutzer unsichtbar, da beim Erstellen einer Suche im Sucheditor der Typ des gesuchten Werts verwendet wird. Der Wert ALT+0233 wird in den ASCII Code 142 übersetzt und so auch gefunden.
Das funktioniert genauso, wenn Sie ALT+0233 in den Prozedureditor eingeben. Für die Suche des ASCII Code eines Zeichens verwenden Sie jedoch den ASCII Code des Macintosh.

Zum Beispiel:

```
QUERY(...;[MyFile]MyField="é")  ` é ist Alt+0233
```

ist dasselbe wie:

```
QUERY(...;[MyFile]MyField=Char(142))  ` é ist ASCII 142
```

ASCII Tabellen

- Der genormte ASCII-Satz reicht von 0 bis 127 und gilt für Windows und Macintosh gleichermaßen.
- Die ASCII Codes von 128 bis 255 sind unter Windows anders als auf Macintosh. Zur Wahrung der Plattformunbhängigkeit konvertiert die Windows Version von 4D automatisch die ASCII Codes von Windows in den ASCII-Satz des Macintosh, wenn Zeichen in der 4D Umgebung eingehen (Dateneingabe, Bearbeiten/Kopieren, Import, etc.) und von Macintosh in den ASCII-Satz unter Windows, wenn Zeichen die 4D Umgebung verlassen (Bearbeiten/Ausschneiden bzw. Kopieren, Export, etc.).

- ASCII Codes 0 bis 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	?

- ASCII Codes 64 bis 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

- ASCII Codes 128 bis 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 Codes 192 bis 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	∆	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	Â	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

Hinweis: Die Kästchen in grau zeigen an, dass die Zeichen unter Windows entweder nicht verfügbar oder anders als Macintosh Zeichen sind.

Funktionstasten

4D gibt den Wert einer gedrückten Funktionstaste in der Systemvariablen *KeyCode* zurück. Sie wird in Projektmethoden verwendet, die der Befehl **ON EVENT CALL** aufruft.

Die Werte für die Funktionstasten basieren nicht auf dem ASCII Codes

Funktionstaste	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

Die Systemvariable *KeyCode* ist nur in einer Projektmethode mit dem Befehl **ON EVENT CALL** gültig.

Folgende Liste zeigt die Werte, die in der Systemvariablen *KeyCode* zurückgeben werden, wenn Sie eine gängige Taste, wie **Zeilenschaltung** oder **Eingabetaste** drücken.

Taste	Code
Eingabetaste	3
Zeilenschaltung	13
Rückschritttaste	8
Tabulator	9
Escape-Taste	27
Delete-Taste	127
Hilfe	5
Pos1	1
Ende	4
Page Up	11
Page Down	12
Pfeil links	28
Pfeil rechts	29
Pfeil oben	30
Pfeil unten	31

4D Programmiersprache - Überholte Befehle

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*