




4D Server Reference


Introduction


 [4D Server in 10 minutes](#)

 [Using 4D Server](#)

 [4D Server Administration Window](#)


 [4D Server Database Methods](#)

 [Using a remote 4D](#)

 [4D Server and the 4D Language](#)

Introduction

 Overview

 4D Server Architecture

Overview

4D Server is the multi-user, cross-platform data and application server for 4D.

With 4D Server, you can create and use multi-user databases and custom applications in a client/server architecture. The platform-independent client/server architecture seamlessly manages databases for both Windows and Macintosh 4D clients. 4D Server includes professional-strength development tools, full scalability, data security, and connectivity options for enterprise systems.

4D Server provides a completely integrated architecture whereby both client and server use a single 4D application. 4D Server frees developers from the need to design separate front-end and back-end applications. In addition, 4D Server is a “zero admin” server. It is easy to install, use, and administer, and is extremely cost-effective.

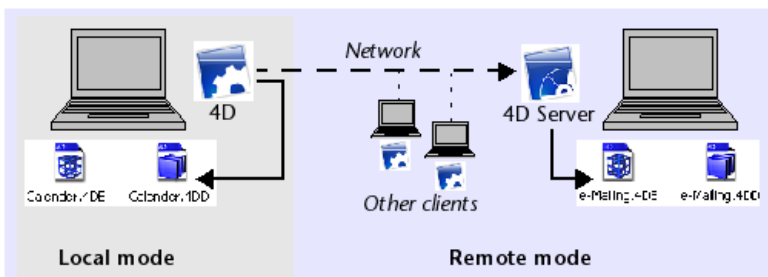
4D Server fills the gap between low-end file-sharing based systems and complex SQL-based RDBMS. Any 4D Server application smoothly interfaces with existing enterprise databases (such as Oracle, Sybase, or any ODBC compatible server). 4D Server addresses the needs of workgroups in all size businesses.

Integrated Back-end and Front-end Architecture

With 4D Server, the front-end and the back-end application are one and the same. The client software and the server application are two faces of the same product, 4D. The 4D Server application itself is divided into two parts—4D Server and 4D in remote mode—which correspond to the elements in client/server architecture.

The 4D Server portion resides on the server machine. It stores and manages the database on the server and allows end users to manipulate the database from their own machines (the clients).

The 4D application resides on each client machine. It can be used in local mode or in remote mode. In local mode, the users can work with a database or a 4D application stored locally on their machine. In remote mode, users access the database on the server and perform database operations such as adding data, generating reports, and modifying database design. Anything that can be done with 4D in local mode can be done using 4D Server and 4D in remote mode.



No additional middleware or development are required to operate in a client/server environment. 4D and 4D Server share the same interface tools, the same language and the same information management system.

Any local single-user application created for one platform (Windows or Macintosh) easily scales to a workgroup client/server solution. Conversely, an application created with 4D Server automatically scales down to a stand-alone application in local mode.

“Zero Admin” Data and Application Server

4D Server benefits from the user-focused heritage of 4D. As a result, 4D Server is a complete Plug and Play system (PNP).



4D Server Administration Window

- **Simple on-line graphical centralized information:** The 4D Server Administration Window automatically displays important information—total memory allocated to 4D Server, data cache, number and name of connected users, number of processes and process' status, monitoring of backups and requests, and all server activity.
- **Auto-configurability and scalability:** 4D Server is designed to support the addition of new protocols, new clients, and plug-ins without any reconfiguration or reengineering.
- **Automatic and dynamic updating and version controlling of client workstations:** All connected 4D machines are automatically and dynamically updated if the database is modified, or if a plug-in or a component is added, removed, or modified. Moreover, if you have built a custom client-server application, it is possible to automatically download new versions of executable 4D client applications when the 4D Server application is updated.
- **Automatic asynchronous connections using the standard TCP/IP protocol:** 4D Server and 4D communicate transparently using the TCP/IP protocol no matter what the platforms of the client and the server are. Since the TCP/IP protocol is integrated in all operating systems, using this protocol with 4D Server does not require any additional installation.
- **Simultaneous session and state management for 4D SQL and HTTP connections:** 4D Server automatically creates and maintains a current working environment for each table/process/user combination. This session-based architecture allows each process for each user to manipulate data independently and concurrently. The SQL server of 4D Server automatically handles internal or external SQL queries. The HTTP server of 4D Server replies to HTTP requests and to SOAP requests when so configured.
- **Automatic record locking:** 4D Server provides automatic record locking and release, avoiding common problems associated with modification of "in-use" records. Record locking also eliminates problems associated with page or file locking.
- **Integrated user-interface messaging system:** Having originated in the desktop arena, 4D Server provides all user interface aspects associated with modern integrated development environments. For example, 4D

Server can inform clients of administrative actions, such as scheduled disconnection and backups.

- **Automated start-up and disconnection methods:** 4D Server automatically invokes five server database methods in response to specific events: **On Server Startup**, **On Server Shutdown**, **On Server Open Connection**, **On Server Close Connection**, and **On Web Connection**. The **On Server Startup Database Method** can automatically initialize and load any objects that may be needed throughout the rest of the session.

An Unmatched Feature Set

In addition to 4D capabilities, 4D Server provides the following features:

- **Multi-user data management:** Multiple users can simultaneously perform database operations. Operations such as adding, modifying, deleting, searching, sorting, and printing records can be performed by several users on records from the same file or from different files. Data integrity is preserved through a built-in record locking system.
- **Multi-user development:** Multiple users can simultaneously develop and design a database. For example, several users can edit file definitions and create and modify layouts, scripts, and procedures at the same time. The integrity of your database design is preserved through a built-in object locking system.
- **Platform independent client /server architecture:** The architecture seamlessly manages database performance for both Macintosh and Windows 4D Clients. This includes concurrent multi-development across platforms, as well as a transparent interface to all the data entered and modified by 4D Client stations running in a heterogeneous hardware environment.
- **64-bit versions:** 4D Server on Windows (since version 12.1) and on Mac OS (since version 15.1) are available in 64-bit versions. 64-bit architecture allows your applications to address more RAM.
- **Windows and Mac OS-based 4D Plug-ins architecture:** The Windows and Mac OS versions of 4D Server let you install both Windows-based and Mac OS 4D plug-ins on the server machine. This architecture simplifies the distribution of platform-independent 4D plug-ins. Plug-ins are transparently handled by 4D Server and 4D, no matter what the platform of the client.
- **Built-in HTTP Server:** Like 4D in local mode, 4D Server and each 4D in remote mode has an HTTP engine that enables you to publish 4D databases on the Web. Your database can be directly published on the Web. You do not need to develop a database system, a Web site, nor a CGI interface between them. Your database is your Web site. You can also transform any remote 4D machine into a Web server. For more information about the built-in Web Server of 4D Server and 4D, see the section **[SORT ARRAY](#)** in the 4D Language Reference manual.
- **Connection Security:** You can encrypt 4D Server connections. In fact, the “classic” client/server architecture can make use of TLS/SSL encryption services. For more information, refer to section **[Encrypting Client/Server Connections](#)**.
- **Triggers:** A Trigger is a method attached to a table. It is a property of a table. You do not call triggers; they are automatically invoked by the 4D database engine each you manipulate table records (add, delete and modify). With 4D Server, triggers are executed on the server machine. Any client, whether it is a 4D or an application connected via ODBC, is subjected to the database rules enforced by the triggers. For more information about 4D triggers, see the **[ARRAY REAL](#)** section in the 4D Language Reference manual.
- **Stored Procedures:** You can create 4D methods to be executed locally on the server machine in their own separate process or on one or multiple specified clients. Using client/server industry terminology, this functionality is called “stored procedures.” Nevertheless, 4D Server provides an architecture that goes far beyond the regular concept of stored procedures. With 4D Server, a stored procedure is actually a custom server process (or a client process, see below) that runs your code asynchronously and independently from all the other processes running on the server or client machines. With regular client/server architecture, a stored procedure executes and returns a result (synchronously or asynchronously). With 4D Server, you can start a stored procedure that runs during a whole client/server session and replies, upon request, to the messages

sent by the clients. Concurrently, you can run a stored procedure that does not interact with any client, but instead synchronizes data with an SQL-based server or another 4D Server, using 4D Connectivity plug-ins or ODBC. There is no limit (except hardware and memory) to the number of stored procedures that you can run concurrently. A 4D Server stored procedure runs in its own process and therefore, like any other user process, can maintain its own private database context (i.e., current selections). In addition, the 4D language provides commands that enable client processes to read and write the process variables of any stored procedures (including BLOB variables), allowing sophisticated and flexible communication between clients and the stored procedures. In fact, using stored procedures, you can add new and custom services to 4D Server. For detailed information, see the section [Stored Procedures](#).

- **Stored Procedures executed on client:**

With 4D Server, you can execute, from the client or the server, stored procedures on one or several other clients. Consequently, the workload can be shared between the clients and the server or applications can be built using all the communication facilities between the clients. For further information, see the [Stored Procedures](#) section.

- **Server path:** The path to a server database can be saved with a user's password. This feature allows a user to connect to a database on the server by double-clicking a .4DLink document. For more information, please refer to the [Connecting to a 4D Server Database](#) section.
- **Registering as a service:** Under Windows, 4D Server can be launched as a Service.
- **Built-in backup system:** 4D Server includes a complete database back-up and restore module. This module lets you back up a database during operation, without having to exit the application. Back-ups can be launched manually or automatically, at regular intervals and without user intervention. In the event of an incident, restoring and/or restarting the database can also be initiated automatically.
- **Backup by logical mirror:** For critical applications, it is possible to set up a backup system by logical mirror, which permits an immediate restart if an incident occurs on the operational database.
- **Connectivity Plug-ins:** Using the 4D Connectivity plug-ins like 4D ODBC Pro, both 4D Server and 4D can directly access mainframe and minicomputer databases such as ORACLE or any other ODBC database server. Information can be shared interactively between these databases. In addition, 4D is offering a 4D Server ODBC driver that will allow any ODBC client to connect and work with 4D Server.

4D Server Architecture

Using client/server architecture, 4D Server not only stores and manages the database, it also provides services to the clients. These services are managed over a network through a system of requests and responses.

To search for a set of records, for instance, a client machine sends a query request to the server. Upon receiving the request, the server executes the query operation locally on the server machine and, when the query is completed, returns the result (the records found).

4D Server's architecture is based on the client/server model. For many years now, client/server architecture has surpassed its older counterpart, file sharing architecture, to become the most efficient model in multi-user databases.

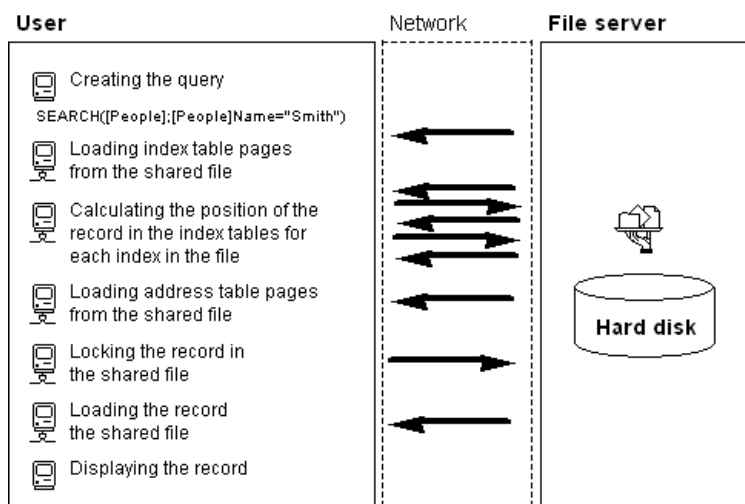
4D Server's implementation of client/server architecture is similar to that used in the world of minicomputers. However, 4D Server offers two significant innovations:

- A user-friendly, graphical interface at all levels of the database
- An integrated architecture that provides increased efficiency and speed

File Sharing Architecture

Before the introduction of client/server architecture, multi-user systems used the file sharing model of network architecture. In this model, all users share the same data, but data management is not controlled by a central database engine. Each client machine must store a copy of the database structure and engine, while the server maintains only the software needed to share files on the network.

Under the file sharing model, each workstation performs all data modification locally. This creates excessive network traffic as each request consists of numerous network passes. The following figure is an example of the traffic created over the network when a user searches the database for every person with the last name "Smith."



Another disadvantage of the file sharing model is the inability to use a memory cache to keep records in memory. If records were kept in memory, different users could have different versions of the same record stored in cache, leading to data inconsistency. As a result, each time a user accesses a record, it must be downloaded from the file server. This causes network traffic and increases the time necessary to access a record.

Heterogeneous Client/Server Architecture

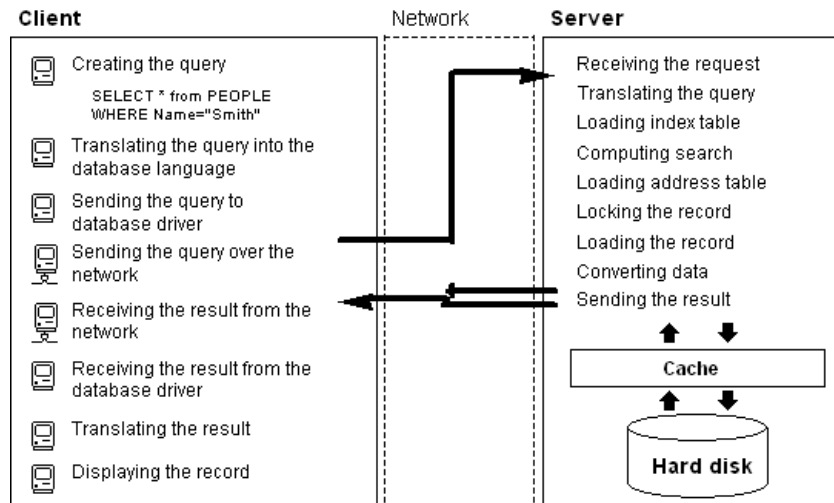
In the minicomputer world, client/server architecture is widely used for large database systems because of its efficiency and speed. In this architecture, work is divided between the server machine and the clients to improve performance.

The server contains the central database engine, which stores and manages the data. The database engine is the

only software accessing the data stored on disk. When a client sends a request to the server, the server sends the result. The result can be anything from a specific record that the client will modify to a sorted list of records.

In general, most client/server architectures are called heterogeneous architectures, because the front-end applications running on the client machines and the database engine running on the server machine are two different products. In this situation, a database driver is required to act as a translator between the clients and server.

To search for a record, for example, a client sends a query request to the server. Since the database is stored on the server, the server executes the command locally on the server machine and sends the result to the client. The following figure shows the traffic created over the network when a user requests the server to search for every person with the last name "Smith" and then display the first record found.



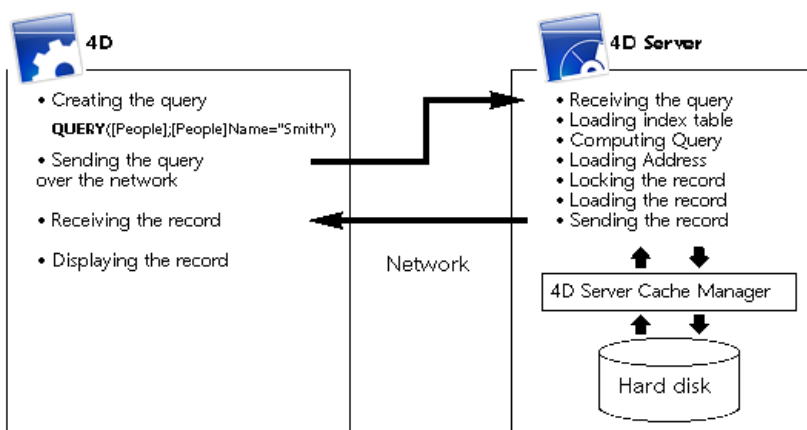
This example illustrates two important differences between file sharing architecture and client/server architecture:

- **Client/server architecture allows the use of cache:** Since the engine is the only software that physically accesses data, the server can maintain a cache that keeps modified records in memory until they are written to disk. Data is sent from one central location, so the clients are assured of always receiving the latest version of a record. In addition to the data integrity this assures, the use of a central cache mechanism accelerates database operations by replacing disk access with memory access. Under the file sharing model, all access is disk access.
- **Low-level database operations are performed on the server:** Client/server architecture offers dramatically increased speed because low-level database manipulations, such as browsing the index and address tables, are locally executed on the server machine at the speed of the machine. In file sharing architecture, the same operations are slowed by network transfers and the limitations of the client machine.

4D Server's Integrated Client/Server Architecture

In most client/server architectures, the client and server software consist of two separate products that require a communication layer to "speak" to one another. With 4D Server, the client/server architecture is fully integrated. 4D Server and 4D are two applications that share the same structure and communicate directly.

Since 4D Server and 4D speak the same language, the query language does not need to be translated. The division of labor between the client and the server is transparent and is managed automatically by 4D Server.



The division of labor is organized so that one request yields one response. As you can see in the diagram, the client is responsible for:

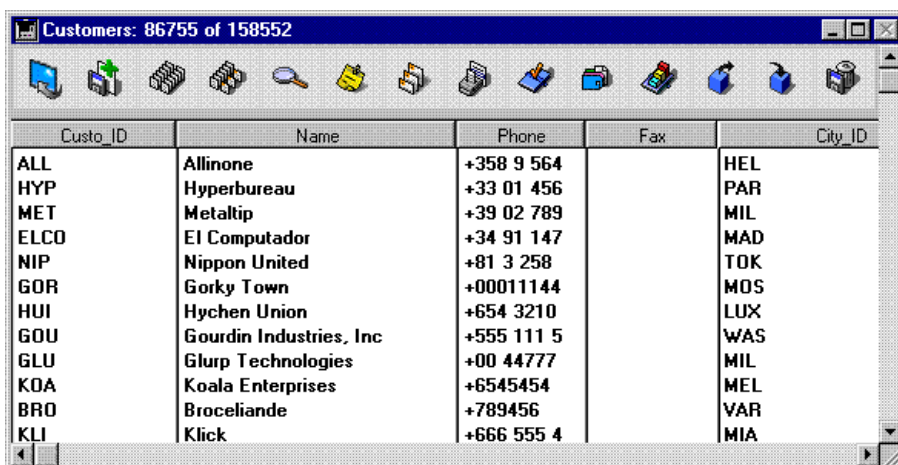
- **Requests:** The 4D client machine sends requests to 4D Server. These requests can be performed using the built-in editors, such as the Query and Order By editors, using the integrated 4D language or via SQL. 4D provides editors in which methods can be created and modified. It also manages method components such as variables and arrays.
- **Receiving responses:** The 4D client machine receives replies from 4D Server and updates the user through the user interface (different records are displayed in a form, etc.). For example, if the client requests all records with the last name "Smith," 4D receives the records from 4D Server and displays them in a form.

The server is responsible for the following:

- **Scheduling:** 4D Server uses multi-tasking architecture to schedule all simultaneous connections and processes created by clients.
- **Structure and data objects:** 4D Server stores and manages all data and structure objects, including fields, records, forms, methods, menus, and lists.
- **Cache:** 4D Server maintains a cache which contains records, as well as data objects specific to particular clients, such as selections and sets.
- **Low-level database operations:** 4D Server performs low-level database operations, such as queries and sorts, that involve using the index and address tables.

This division of labor is extremely efficient because of the unique integration of 4D Server and 4D. The integration of 4D Server's architecture is present at every level:








- **At the request level:** When 4D sends a request to 4D Server, such as a query or a sort, 4D sends a description of the query or the sort operation using the same internal structures found in 4D Server.
- **At the structure or data level:** When 4D and 4D Server exchange a data or structure object, both applications use the same internal format. For instance, when 4D needs a record, 4D Server sends the data exactly as it finds it in the disk or memory cache. In the same way, when 4D wants to update a record, it sends the data to 4D Server, which stores the data in cache exactly as it was received.
- **At the user interface level:** When 4D displays a list of records, the form used to display the records plays a role in the client/server architecture. For example, the following figure shows the result of a query in the [Customers] table.



Custo_ID	Name	Phone	Fax	City_ID
ALL	Allinone	+358 9 564		HEL
HYP	Hyperbureau	+33 01 456		PAR
MET	Metaltip	+39 02 789		MIL
ELCO	El Computador	+34 91 147		MAD
NIP	Nippon United	+81 3 258		TOK
GOR	Gorky Town	+00011144		MOS
HUI	Hychen Union	+654 3210		LUX
GOU	Gourdin Industries, Inc	+555 111 5		WAS
GLU	Glurp Technologies	+00 44777		MIL
KOA	Koala Enterprises	+6545454		MEL
BRO	Broceliande	+789456		VAR
KLI	Klick	+666 555 4		MIA

Since the window can display only twelve records and five fields at a time, 4D Server sends exactly twelve records. Instead of sending the entire set of records, 4D Server sends only the number of records and fields that can be displayed in the window. If the user were to scroll through the form, 4D Server would send the additional records or fields as needed. This optimization reduces network traffic by ensuring that records and fields are sent over the network only when necessary.

4D Server in 10 minutes

-  Checking Your Installation
-  Creating a Server Database
-  Connecting to the Server Database with a Remote 4D
-  Defining the Database Structure
-  Data Manipulation with 4D Server
-  Adding a Custom Menu Bar
-  Working Concurrently with 4D Server

Checking Your Installation

The Chapter **4D Server in 10 Minutes** is a quick tutorial that shows you how to:

- Create a server database
- Connect Clients to the created server database
- Create a database structure, including tables, fields, forms, menus and methods
- Connect with a second user and work concurrently

You will need at least two computers:

- One machine with 4D Server, 4D and a Web browser installed
- Another machine with 4D installed

Before working with 4D Server and 4D in local mode for the first time, it is a good idea to check your installation. To do so, read this section.

Installed elements

This paragraph specifies the location of the elements installed on your disk following a standard installation of 4D + 4D Server.

Windows

The elements have been installed in the **Program Files\4D\4D vXX** folder and appear in the **Start** menu.

- **4D Server**: This folder contains the 4D Server application as well as its associated files and folders. To launch 4D Server, simply double-click on the **4D Server.exe** file.
- **4D**: This folder contains the 4D application as well as its associated files and folders. To launch 4D, simply double-click on the **4D.exe** file.

Mac OS

The elements have been installed in the **Applications:4D:4D vXX** folder and appear in the applications.

- **4D Server**: 4D Server software package. To launch 4D Server, just double-click this package.
- **4D**: 4D software package. To launch 4D, just double-click this package.

For this particular exercise, you must install 4D on an additional machine as well.

Where To Go From Here?

Note that the TCP/IP protocol should be configured for your machines to communicate over the network.

If 4D Server and 4D are correctly installed, proceed with the **Creating a Server Database** section. Otherwise, if some of the files listed above are missing, refer to the 4D Installation Guide and proceed with the installation of these files.

Creating a Server Database

This section explains how to create a server database that can be accessed on the network using 4D in remote mode. Before working with 4D Server and 4D for the first time, it is a good idea to check your installation. To do so, read the section [Checking Your Installation](#).

Note: In this example, we assume that you have already activated your 4D Server license, as described in the Installation Guide. Using 4D in remote mode does not require a license on the client machine. Licenses are managed on the 4D Server machine. For more information, please refer to the Installation Guide.

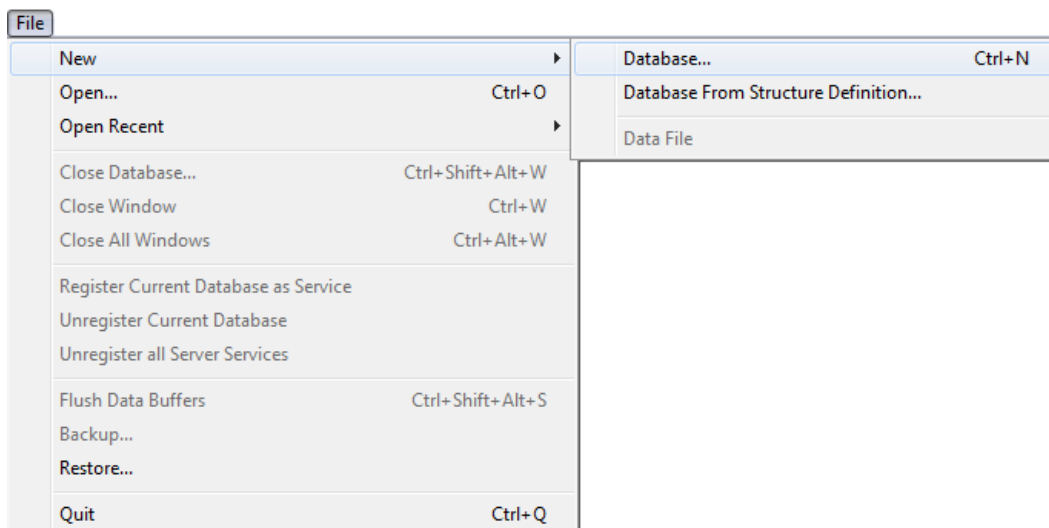
To create or open a server database, launch 4D Server.

1. Launch 4D Server by double-clicking on the 4D Server icon.



When the application is launched for the first time, the application activation dialog box is displayed. Afterwards, a blank window will be displayed at each startup. This functioning can be configured using the application Preferences. In this exercise, you will create a new database.

2. Choose the New Database... command from the File menu of 4D Server.

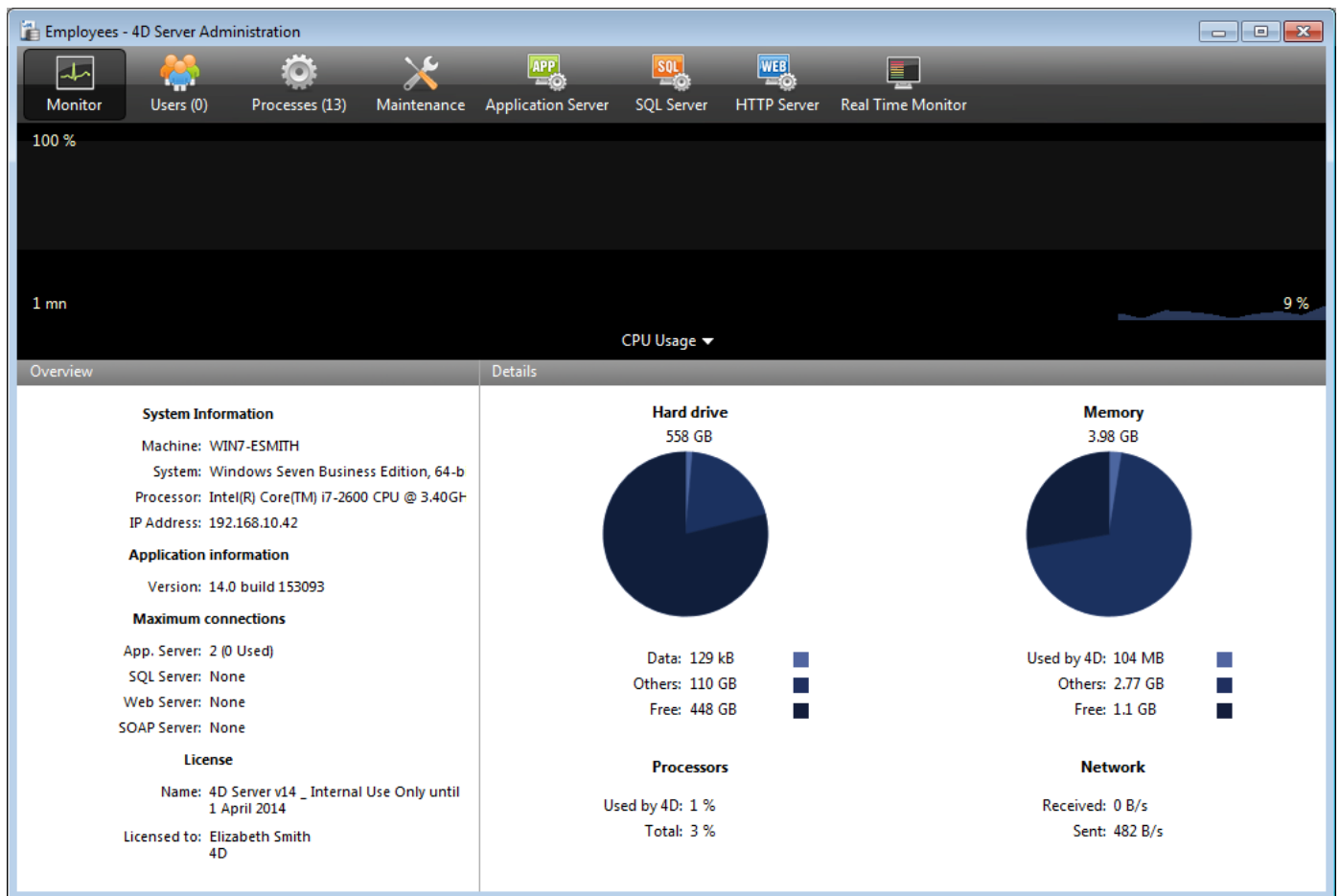


A standard save file dialog box appears, which lets you specify the name and location of the new database to be created.

3. Specify a location, then enter the name of the database.

Type **Employees**, then click **Save**.

4D Server automatically creates the files and folders required for the operation of the database, then the administration window appears:



The 4D Server **administration window** has several pages that can be accessed using tabs. The **Monitor** page displays dynamic information concerning database operation as well as information about the system and the 4D Server application.

Among the other pages available, note that the **Users** and **Processes** pages indicate, respectively, the number of users connected to the database and the number of processes currently running. Note that at this point, the number of users connected is zero. This means that you have not yet connected any clients to the database. However, several processes are running. These processes are created automatically by the database engine (kernel) and the built-in servers of 4D Server (application server, HTTP server, SQL server).

Where To Go From Here?

At this point, the database is available for Windows and/or Macintosh 4D remote connections over the network. However, the database is not yet ready for HTTP Connections, because these connections are not authorized by default.

Go to the section **Connecting to the Server Database with a Remote 4D**. In this tutorial, you will first connect using 4D in remote mode, define the structure of the database and add some records to the database.

Connecting to the Server Database with a Remote 4D

This section discusses:

- How to connect a remote 4D to the server database you created.
- Creating a database structure. This section includes a tutorial in which you create tables and fields in the database, enter new records, and modify existing records.
- Connecting a second user.
- Working concurrently with two remote clients.

Connecting to the Database

Although you created the database with 4D Server (see section [Creating a Server Database](#)), all modifications to the database design and the actual data are performed from the client machines. In this section, you will learn how to connect to the server and open the server database.

1. Double-click the remote 4D application icon.



Note: For the purposes of this tutorial, you can use a 4D application installed on the same machine as the 4D Server.

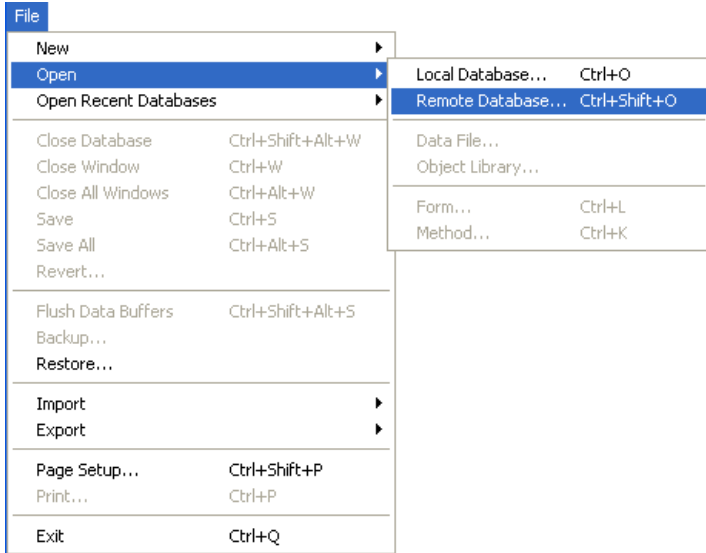
If this is the first time the 4D application has been launched or if you have not modified the startup parameters, the Welcome Wizard dialog box appears:



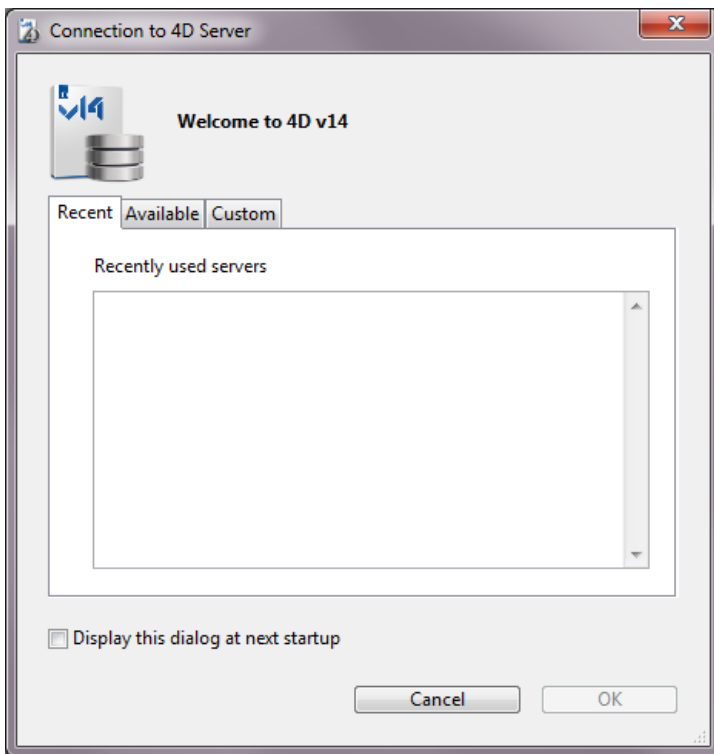
2. Click on "Connect to 4D Server".

OR:

If this dialog box does not appear, choose **Open>Remote Database...** in the File menu of 4D:

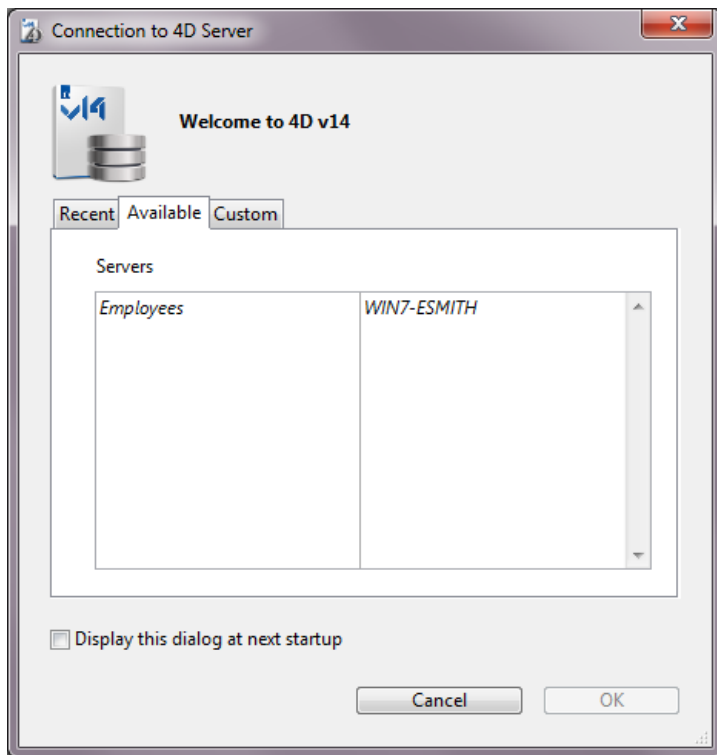


The Connection dialog box is displayed:



3. Click on the Available tab in order to display the list of 4D databases published on the network.

The Employees database should appear in the list:



4. Select Employees and click OK.

The database will be opened on the remote workstation. It opens in the Design environment, ready for you to create the structure of your database.

Trouble-shooting tips

If you do not see the name of the database you have just created with 4D Server, check the following points:

- Is 4D Server still running on the other machine?
- If you use a second machine, are both your machines connected to the network?
- Is the TCP/IP protocol correctly configured on both machines?
- If you are not sure about using the Connections dialog boxes, see the section [Creating a Server Database](#).

Server Activity

When you look at the 4D Server administration window, note that your network user name appears on the corresponding page and the number of connected users is one (1).

Employees - 4D Server Administration

Monitor Users (1) Processes (15) Maintenance Application Server SQL Server HTTP Server Real Time Monitor

User;Machine;Session...

4D User	Machine name	Session name	IP Address	Login date	CPU Time	Activity
Designer	WIN7-ESMITH	esmith	localhost	1/27/2014 15:51	00:00:00	0 %

Send message Watch Processes Drop user

New processes are running:

Employees - 4D Server Administration

Monitor Users (1) Processes (15) Maintenance Application Server SQL Server HTTP Server Real Time Monitor

Session;Process name...

Display processes by groups

Users processes (2) 4D Processes (11) Spare processes (2)

Process name	Session	Type	Num	State	CPU Time	Activity
Client Manager	-	Application server	3	Waiting for flag	00:00:00	0 %
DB4D CRON	-	DB4D Server	0	Running	00:00:00	0 %
DB4D Flush	-	DB4D Server	0	Running	00:00:01	0 %
DB4D Index builder	-	DB4D Server	0	Running	00:00:00	0 %
DB4D Server	-	DB4D Server	0	Running	00:00:00	0 %
Garbage Handler	-	DB4D Server	0	Running	00:00:00	0 %
Internal Timer Process	-	Application server	2	Executing	00:00:01	0 %
Task managers	-	SQL Server	0	Running	00:00:00	0 %
TCP connection listener	-	SQL Server	0	Running	00:00:00	0 %
User Interface	-	Application server	1	Waiting for event	00:00:06	11 %
Application process	esmith	4D Client Process	4	Waiting for I/O	00:00:00	0 %

Abort Process Pause Process Activate Process Debug Process Watch users

Note: By default, as in the above screen, spare processes are hidden. You can click on the **Spare processes** button to display them.

The first processes are for 4D Server itself and were created when 4D Server was launched. The new processes are for the first user connecting to the server:

- The Application process manages the record display window and the Application mode.
- The Design process manages the Design mode.

Each additional user will add several more processes to the list of processes.

You can filter the list of processes displayed using the **Users processes**, **4D Processes**, or **Spare processes** buttons, as well as the search/filtering area in the top right part of the administration window.

Where To Go From Here?

Now that you are connected, you can work with the database with the same feature set as that of 4D in local mode. First, you need to define the structure. Go to the section [Defining the Database Structure](#).

Defining the Database Structure

On the remote 4D machine, after you have connected to the server database (see the [Connecting to the Server Database with a Remote 4D](#) section), choose the **Database Structure** command in the **Design** menu.

The Structure window appears, empty by default. We are going to create a simple table.

Create the [Employees] Table — an Example

1. Choose **New>Table** in the File menu or in the 4D toolbar.

OR

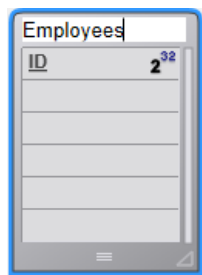
Right click in the Structure window and choose **Add Table** from the context menu.

OR

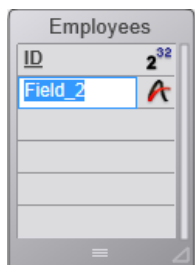
Click on the add button of the Structure window (in the form of a +) and choose **Table**.

The table is created.

2. Click in the title area and name the table **Employees**.



3. Double-click in the field area to create a new field.

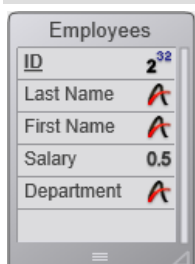


4. Rename the field **Last Name** and keep the type **Alpha (255)**.

You can double-click the field in order to display the Inspector palette.

5. Add the following fields to the [Employees] table in the same way:

Field Name	Field Type
First Name	Alphanumeric (255 characters)
Salary	Real
Department	Alphanumeric (255 characters)



Note: If other 4D remote applications were working simultaneously with the server database, the fields you have just created will appear on the other client machine in a few moments. The changes are implemented on the server in real time, but do not appear on other screens immediately, to avoid updating the screen too frequently.

Creating Forms for the [Employees] Table

After you have defined the *[Employees]* table, you need forms to add and work with its records. To do so, you could use the New Form Wizard and create forms at your convenience. However, 4D comes with a convenient shortcut for quickly creating default input and output forms.

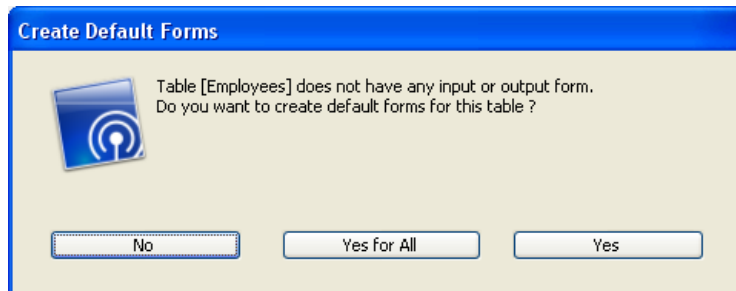
1. Click on the Tables button of the 4D tool bar.



OR

Choose the Show Current Table command from the Records menu.

The records window is then displayed. 4D detects that the table still has no forms and asks if you want to let the program create them for you.



2. Click Yes.

You have now an input form for adding or displaying records one by one, and an output form for displaying or entering multiple records in list mode.

Where To Go From Here?

Your server database is ready for data manipulation. Go to the section [Data Manipulation with 4D Server](#).

In the section **Defining the Database Structure**, you have created the *[Employees]* table and let 4D create the default forms for that table. You are now ready to enter records.

Entering Records

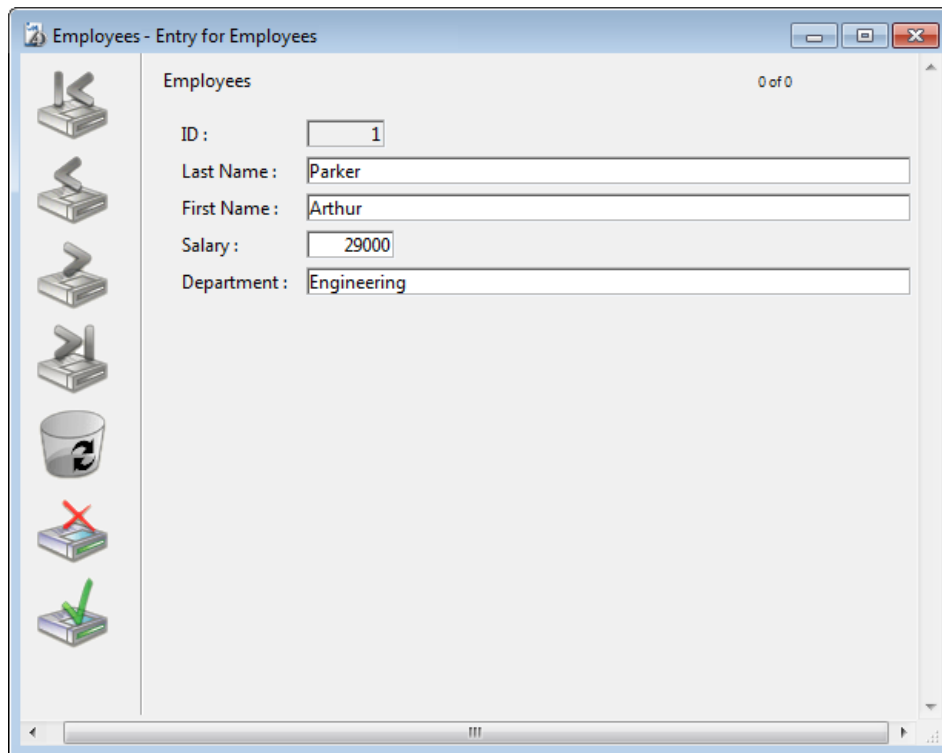
In Design mode, 4D provides you by default with tools and editors for entering, querying, printing or modifying records. Subsequently, you can set up your own tools for the Application mode.

1. Choose New Record from the Records menu.

The blank input form appears.

2. Enter your first record as shown.

Use the **Tab** key or the mouse to navigate between the fields.



The screenshot shows a window titled "Employees - Entry for Employees" with a "0 of 0" record indicator. The form has the following fields:

- ID: 1
- Last Name: Parker
- First Name: Arthur
- Salary: 29000
- Department: Engineering

The left toolbar includes icons for adding a record, deleting a record, and validating the data.

3. Click on the form validation button (the one at the bottom) to validate your data entry.

A blank input form appears so you can continue adding new records.

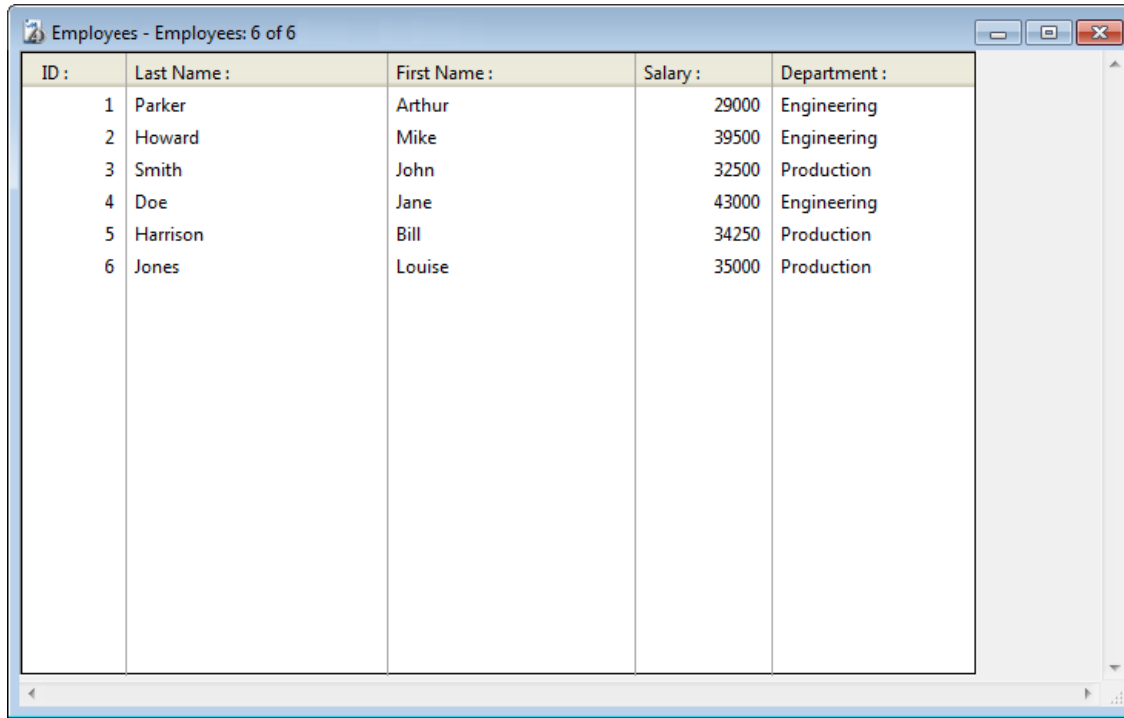
4. Enter five more records with the values listed here.

Last Name	First Name	Salary	Department
Howard	Mike	39500	Engineering
Smith	John	32500	Production
Doe	Jane	43000	Engineering
Harrison	Bill	34250	Production
Jones	Louise	35000	Production

After you have entered the last record, click the cancel icon (the one with a cross, above the validation icon) so that you cancel the new blank input form. You go back to the output form.

5. If all the six records are not shown, choose Show All from the Records menu and resize the window or columns if necessary.

You should have:



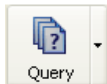
ID :	Last Name :	First Name :	Salary :	Department :
1	Parker	Arthur	29000	Engineering
2	Howard	Mike	39500	Engineering
3	Smith	John	32500	Production
4	Doe	Jane	43000	Engineering
5	Harrison	Bill	34250	Production
6	Jones	Louise	35000	Production

The records are now stored in the database on the server machine. If a second remote 4D were connected to the server machine, it could display the records you have just added. Conversely, if other clients were also entering records, you could choose **Show All** from the **Records** menu to display all the records, including the ones they had entered. Records stored on the server are accessible to all users.

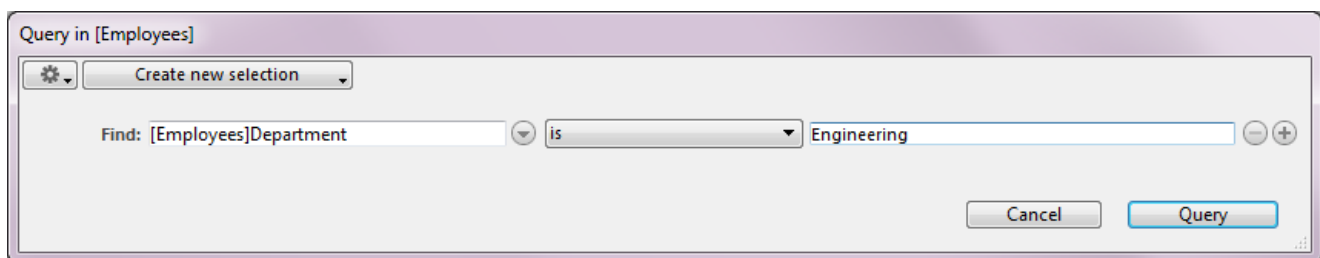
Querying Records

Once you have entered records in the *[Employees]* table, you can search, sort, print, and otherwise manipulate the records. For example, let's look for the employees of the Engineering department.

1. Click on the Query button in the toolbar.



The Query Editor appears:



Query in [Employees]

Create new selection

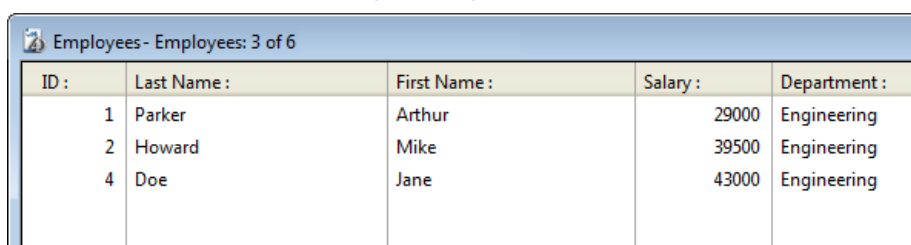
Find: [Employees]Department is Engineering

Cancel Query

2. Leave "[Employees]Department" as search criterion and "is" in the Comparisons list, then enter "Engineering".

3. Click Query.

The query is sent to 4D Server, then 4D Server replies to 4D. In the output form, you now have only the employees who work in the Engineering department:



ID :	Last Name :	First Name :	Salary :	Department :
1	Parker	Arthur	29000	Engineering
2	Howard	Mike	39500	Engineering
4	Doe	Jane	43000	Engineering

4. To show all the records again, choose Show All from the Records menu.

Where To Go From Here?

In just a few minutes, you have created a server database, defined a table, added records, and then made a query using the data entered in the database.

It is now time to add a custom menu bar to your database. Go to the [Adding a Custom Menu Bar](#) section.

Adding a Custom Menu Bar

In this section, you will design two methods and a custom menu bar. In short, you are going to create a custom 4D application.

Adding the Two Methods

1. Select **New > Method..** from the File menu.

The New Method dialog box appears.

2. Enter **"M_ADD_RECORDS"** in the New Method dialog box, then click **OK**.

A method editor window appears titled "Method:M_ADD_RECORDS".

3. Enter the code of the **M_ADD_RECORDS** method as shown:



```
Method: M_ADD_RECORDS
1 Repeat
2   ADD RECORD([Employees])
3 Until (OK=0)
```

4. Create a second method named **"M_LIST_RECORDS"** with the following code:



```
Method: M_LIST_RECORDS
1 QUERY([Employees])
2 If (OK=1)
3   MODIFY SELECTION([Employees])
4 End if
```

Now that the two methods have been created, you will create a custom menu bar and attach the methods to custom menu commands.

Adding a Custom Menu Bar

1. Select **Tool Box > Menus** from the Design menu.

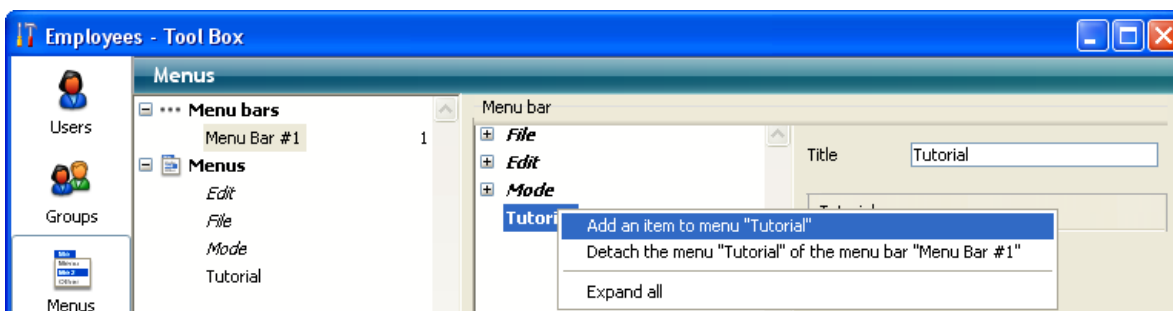
The Menu Bar Editor appears containing a default menu bar.

2. Select the title **"Menu Bar # 1"** and click the add button in the central part of the window.



3. Enter **"Tutorial"** as the menu title and press **Enter**.

4. Right click on the title **"Tutorial"** and choose **Add an item to menu "Tutorial"**:

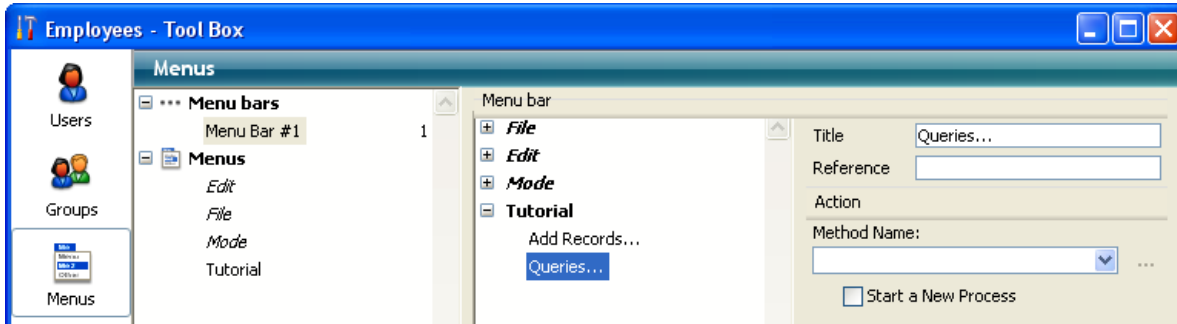


5. Enter **"Add Records..."** and press **Enter**.

6. Right click on the title "Tutorial" again to add a second menu command to the "Tutorial" menu.

7. Enter "Queries..." and press Enter.

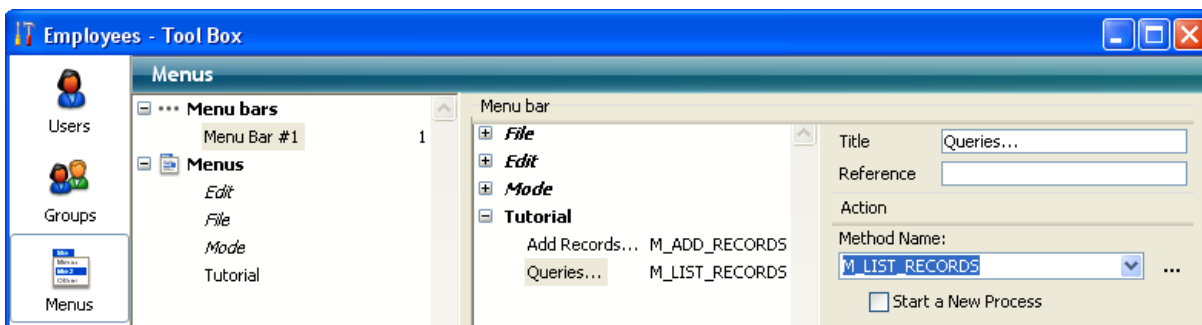
The menu bar #1 should look like this:



8. Click the "Add Records..." menu command and select "M_ADD_RECORDS" in the Method Name combo box.

9. Click the "Queries..." menu command and select "M_LIST_RECORDS" in the Method Name combo box.

The menu bar #1 should now look like this:

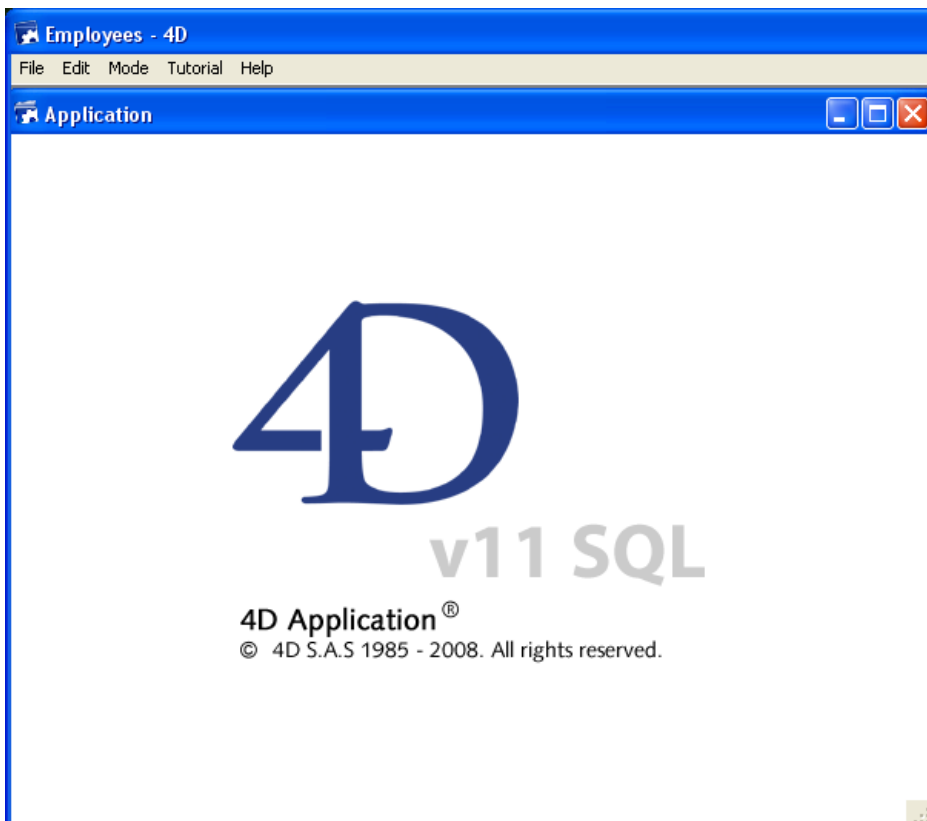


10. Close the Tool box window.

You're done!

11. Select Test Application from the Run menu.

You are now using your application with the menus you just designed:



For example, if you select **Queries...** from the **Tutorial** menu, the standard 4D Query editor appears. You can define your query, then display and modify the records found by the query.

The interesting point is that, without knowing it, you just developed two applications!

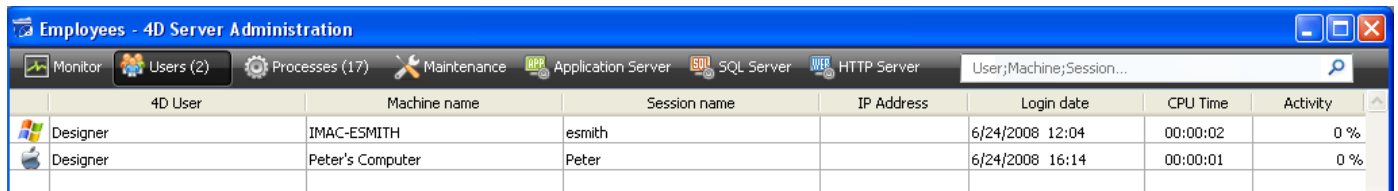
To see why, go to the section [Working Concurrently with 4D Server](#).

Working Concurrently with 4D Server

If you run this tutorial on Windows, you could use this server database “as is” on Macintosh. If you run this tutorial on Macintosh, you could use this server database “as is” on Windows.

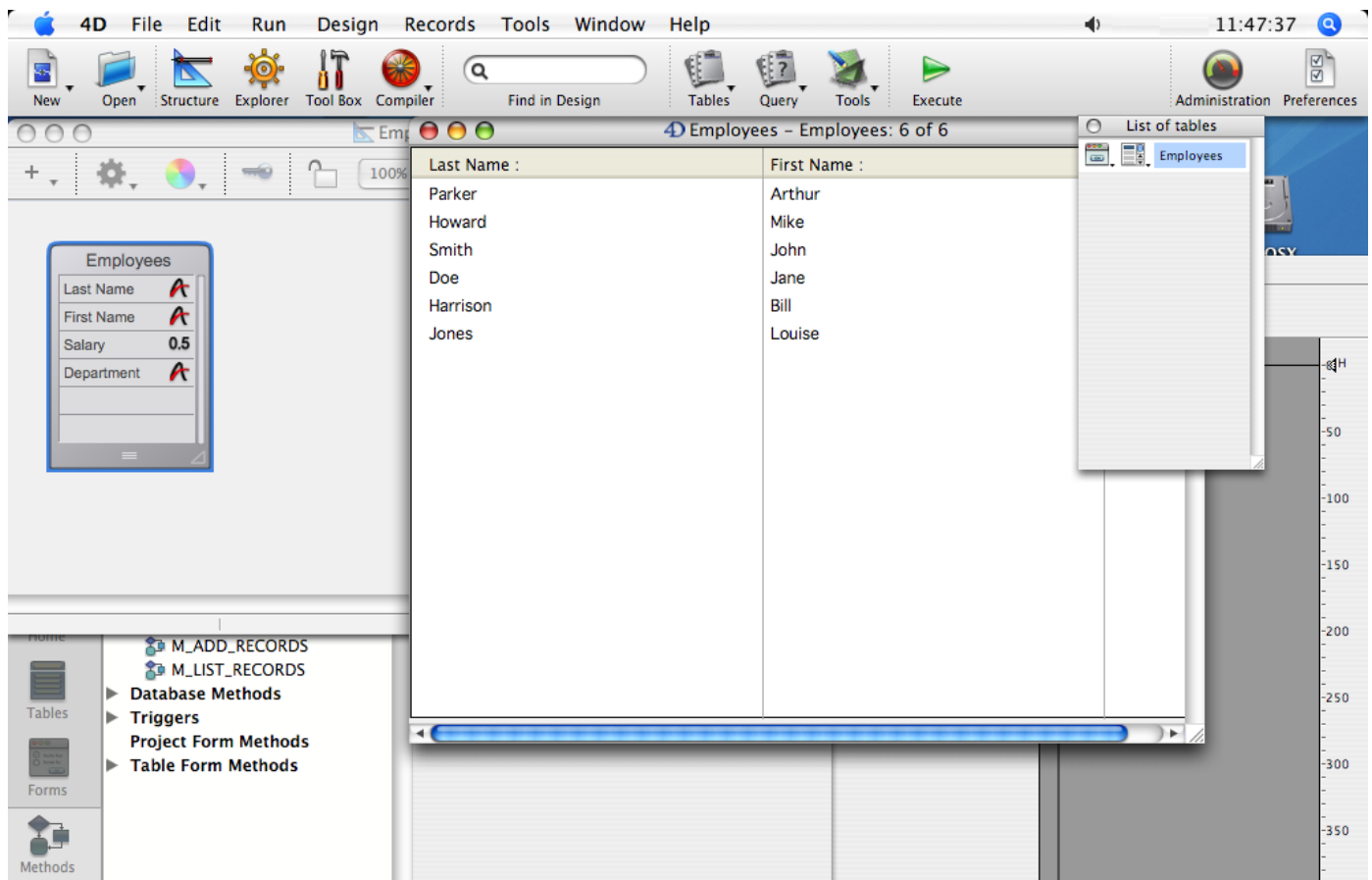
Connecting to the Server Database with a Second User

For this tutorial, we will connect to the server database with a remote 4D on Windows and a remote 4D on Mac OS. As soon as you are connected, you can see the second user entry in the 4D Server Administration window (the first column indicates the operating system of the remote machine):



4D User	Machine name	Session name	IP Address	Login date	CPU Time	Activity
Designer	IMAC-ESMITH	esmith		6/24/2008 12:04	00:00:02	0 %
Designer	Peter's Computer	Peter		6/24/2008 16:14	00:00:01	0 %

On each client machine, everything done on the other platform is instantaneously and transparently reusable. Here is the Design environment on a remote client machine on Mac OS:



Last Name :	First Name :
Parker	Arthur
Howard	Mike
Smith	John
Doe	Jane
Harrison	Bill
Jones	Louise

Your six records and your two methods are here!

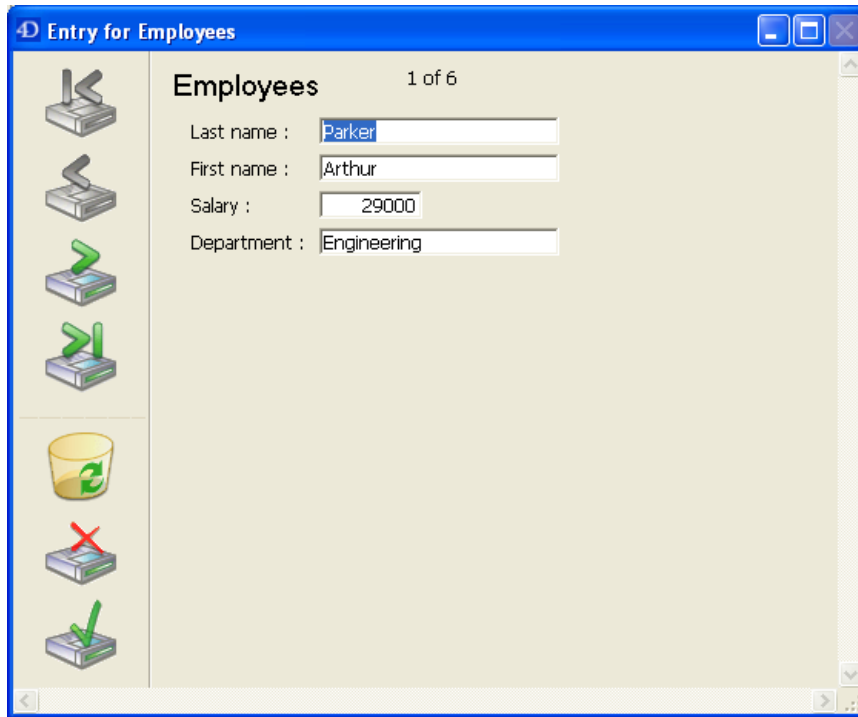
Working Concurrently with Records

1. On the first remote machine, in "Test Application" mode, choose Queries... from the Tutorial menu, and look for the records where “Department is equal to Engineering”.
2. Do the same thing on the second remote machine.

On both machines, you obtain a list composed of three records.

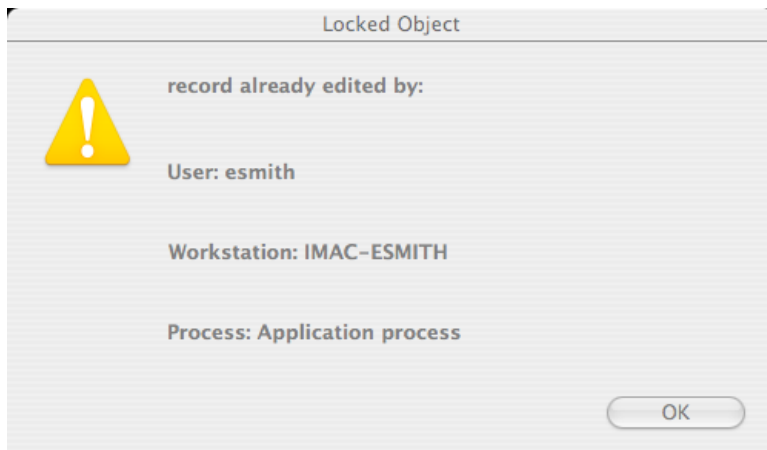
3. On the first machine, double-click on the record "Parker, Arthur".

Your screen should look like this:

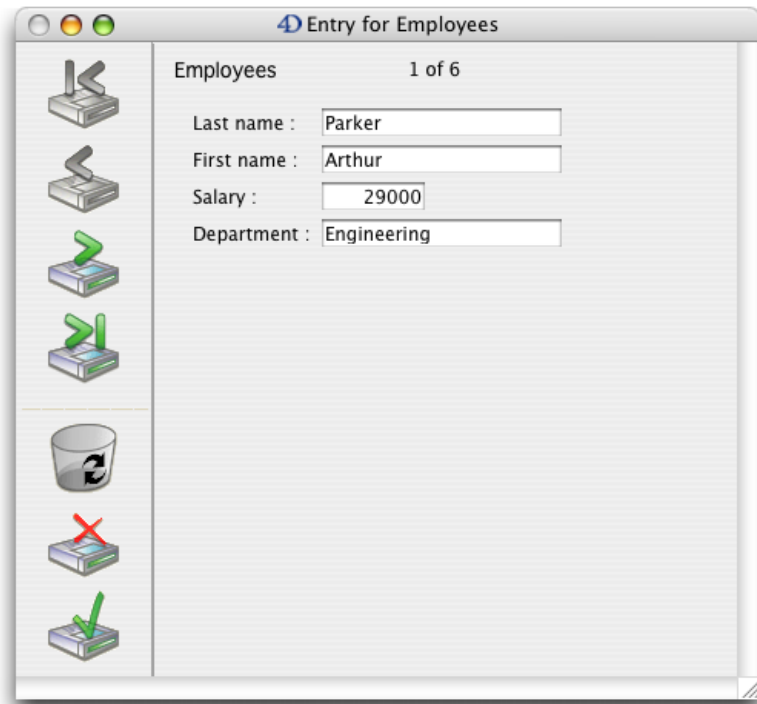


4. Do the same thing on the second machine.

4D Server has a built-in record locking mechanism and warns you that the record is already in use:



Nevertheless, you have access to the record in read-only (you can display it, but cannot modify it).



5. On the first machine, change the first name to "Michael" and validate your changes.

The list has been updated with the new first name.

The screenshot shows a table window titled "Employees: 6 of 6". The table has four columns: "Last name :", "First name :", "Salary :", and "Department :". The first row is highlighted in blue and shows the first name changed to "Michael".

Last name :	First name :	Salary :	Department :
Parker	Michael	29000	Engineering
Howard	Mike	39500	Engineering
Smith	John	32500	Production
Doe	Jane	43000	Engineering
Harrison	Bill	34250	Production
Jones	Louise	35000	Production

6. On the second machine, cancel the display of the record in the input form.

The list has been updated with the new first name too!

Last name :	First name :	Salary :	Department :
Parker	Michael	29000	Engineering
Howard	Mike	39500	Engineering
Smith	John	32500	Production
Doe	Jane	43000	Engineering
Harrison	Bill	34250	Production
Jones	Louise	35000	Production

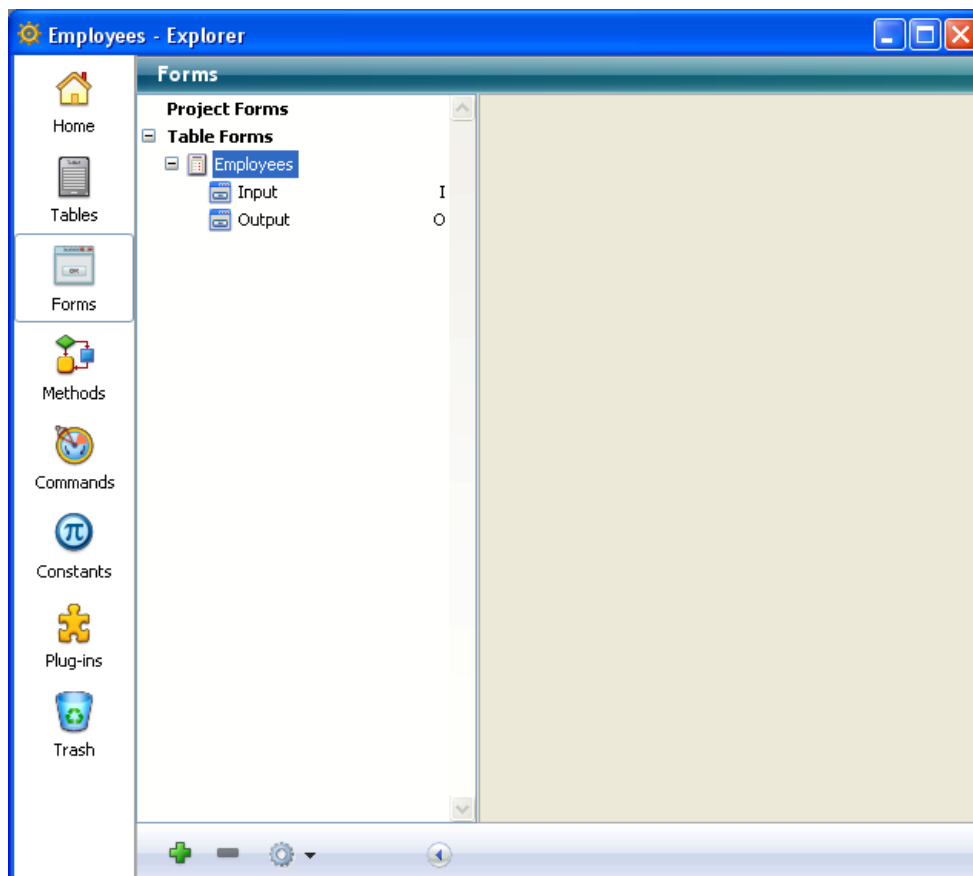
Working Concurrently with Design Objects

4D Server is both a data and an application server. Let's see what this means.

1. On the second machine, press the Esc. key, then choose the Return to Design mode command in the Mode menu.
2. Do the same thing on the first machine.
3. On the first machine, choose Explorer > Forms from the Design menu.

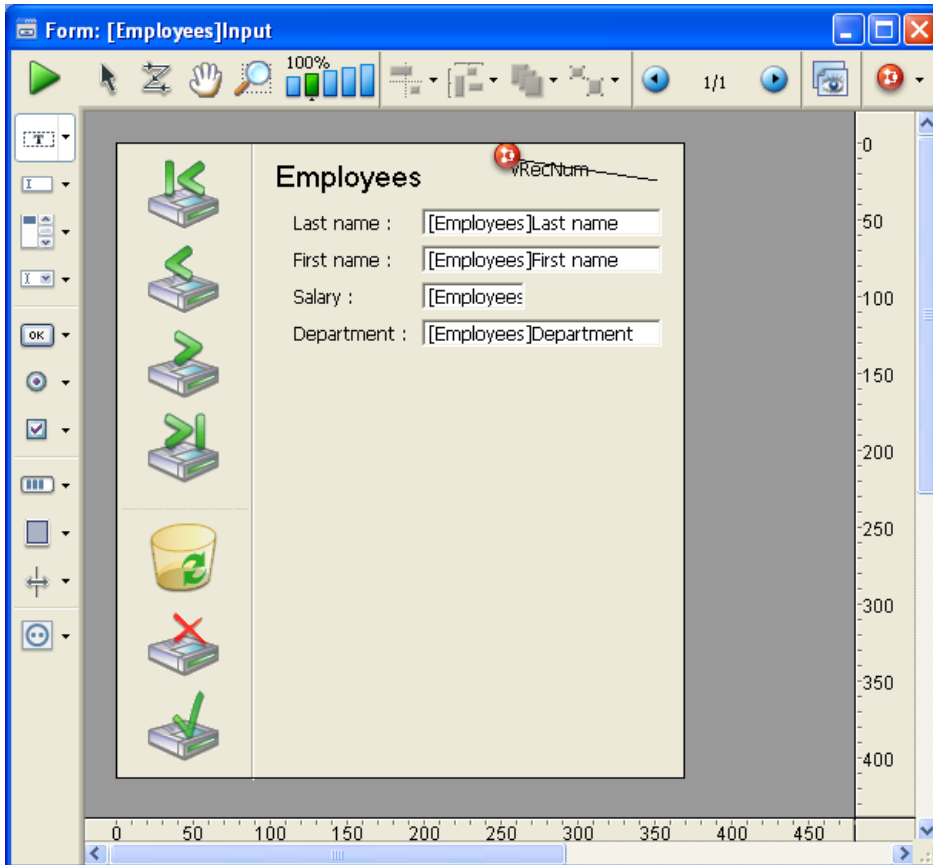
The Explorer window appears.

4. Choose the table form and expand the Employees table:



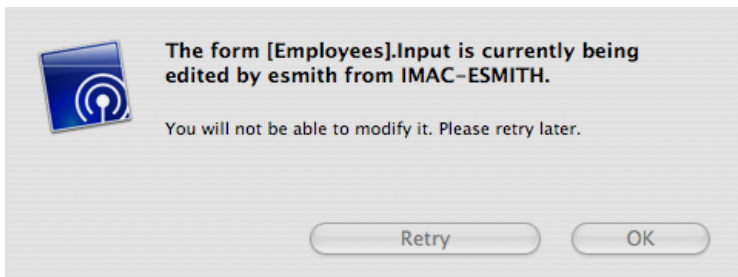
5. Double-click the Input form.

A Form Editor window is opened for the Input form:



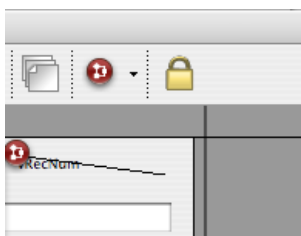
6. Do the same thing on the second machine.

Since the form is already in Edit mode on the other machine, the 4D Server built-in object locking mechanism informs you:



Nevertheless, you can open the form on the second machine in display mode. You can select objects and copy them to other forms, but you cannot modify the form itself.

Note the padlock icon in the upper right corner of the form. This icon reminds you that you cannot change the form.



7. On the first machine, select the legend "Last name" on the left of the [Employees]Last Name field.

8. Using the Object>Color hierarchical menu, set the color of this object to red.



9. Select Save Form: [Employees]Input from the File menu.

10. On the second machine, close and reopen the form in order to reload it.

The change made on the other machine is now available on this one.

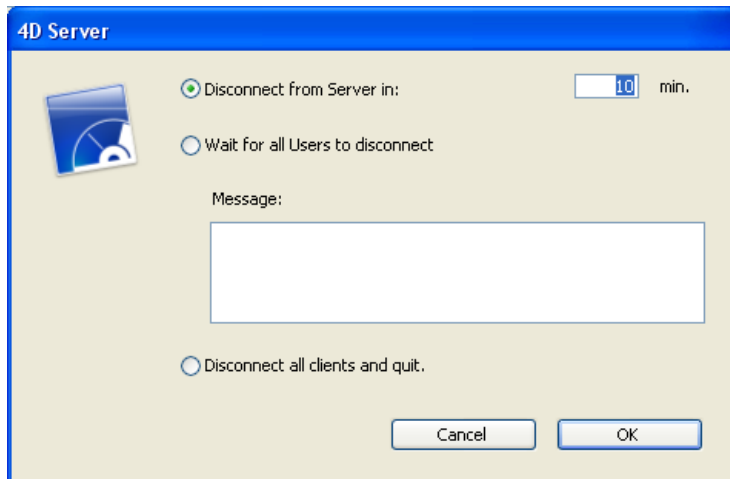
4D Server allows you to develop concurrently a database with other users!

Shutting Down the Server

In addition to informing 4D remote users during simultaneous access to the same records or objects, 4D Server includes a built-in shutdown warning message over the network.

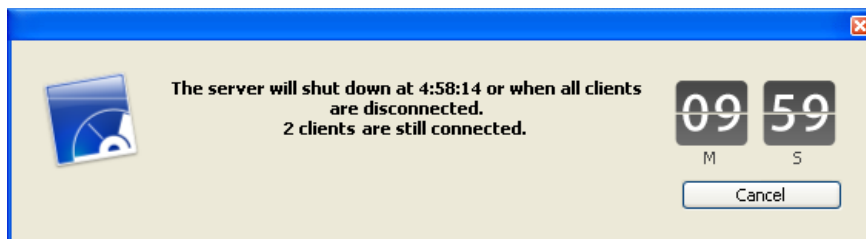
1. Keeping the two clients connected the server database, on the server machine, choose Quit from the File menu (Windows) or the 4D Server menu (Mac OS).

The Shutting Down dialog box appears:



2. Click OK.

Almost instantaneously, the two client machines are informed that the server is shutting down. For example, if a client were adding a record, the user would have enough time to finish and validate the data entry.



This warning dialog box is repeated regularly on each client machine.

Note: Alternatively, you can shut down the server using the "Wait for all users to disconnect" option (while possibly sending them a message asking them to disconnect as soon as possible) or to force immediate disconnection of the clients using the "Disconnect all clients and quit" option.

3. While the server is shutting down, quit 4D on the two remote machines.

Conclusion

With this tutorial (which may take more than 10 minutes, depending on various factors, such as time you took for a drink), you have discovered how easy it is to use 4D Server:

- You created a database from scratch.
- You defined a table and let 4D Server create the forms for you.
- You added and manipulated some records.
- You customized your application with your own menu bar.
- You used the server database concurrently on both Windows and Macintosh.
- You shut down the server and restarted it.

To conclude, you created two custom applications (Windows, Macintosh and Web) while actually developing only













one. In addition, if you need to use the database in local mode, you could open it directly, as is, with 4D.

To learn more about 4D Server, read the introductory sections of this manual, as well as the other sections that describe 4D Server in detail.

For a complete examination of the 4D environment, refer to:

- The Design Reference manual to learn about designing and using 4D applications and databases.
- The Language Reference manual to learn about the commands of the 4D language. For instance, to learn about the Web capability of 4D Server, read the section **Web Server, Overview** in this manual.

Using 4D Server

-  Creating a New 4D Server Database
-  Exiting 4D Server
-  Using 4D Server 64-bit version (Windows)
-  Using 4D Server 64-bit version (OS X)
-  4D Server Menus
-  Network and Client-Server options
-  IP Configuration
-  Encrypting Client/Server Connections
-  Single Sign On (SSO) on Windows
-  Managing the Resources folder
-  Registering a Database as a Service
-  Setting up a logical mirror

Creating a New 4D Server Database

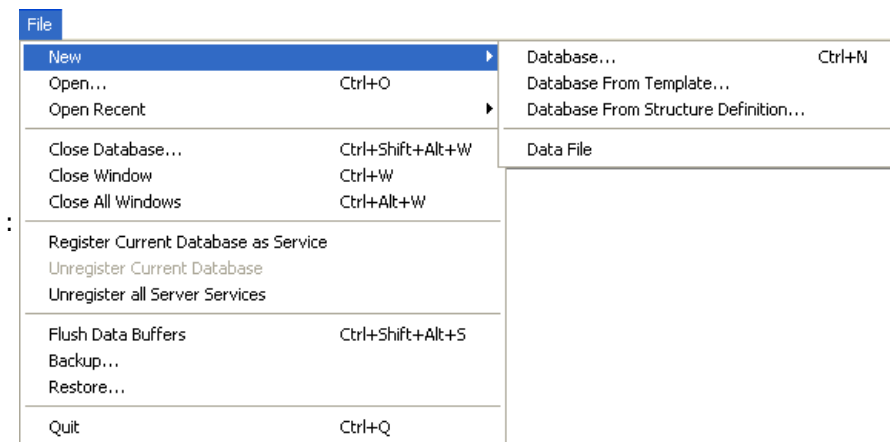
To create a server database or open an existing database, launch 4D Server by double-clicking the 4D Server application icon.



You can then create a new database or open an existing one using the **File** menu of 4D Server.

Creating a database

To create a new database, choose one of the commands from the **New>** submenu:



- **Database:** can be used to create a blank database, in other words, one without any tables, forms or predefined interface. When you choose this command, a standard save file dialog box appears so that you can set the name and location of the database.
- **Database From Template:** creates a database using a "ready to use" template that you can customize subsequently. In order to be able to use this function, the template databases must be placed in a "4D Templates" or "4D Modèles" folder at the same level as the 4D Server.exe (Windows) file or the 4D Server software package (Mac OS). When you choose this command, a database template selection dialog box appears.
- **Database From Structure File:** can be used to create a database based on a description of the structure in XML format. This definition can come from a structure exported from 4D or any design application. When you choose this command, a standard open file dialog box appears so that you can designate the XML file to use.

For more information about these options, please refer to the Design Reference manual.

Opening a database

To open an existing database, you can use a standard open document dialog box (**File>Open...** command) or select a previously-opened database directly (**File>Open Recent** command).

If a database was already open when you select an Open command, it will first be closed. If any client machines were connected, they are disconnected using the "Wait for all Users to disconnect" mode (see the **Exiting 4D Server** section).

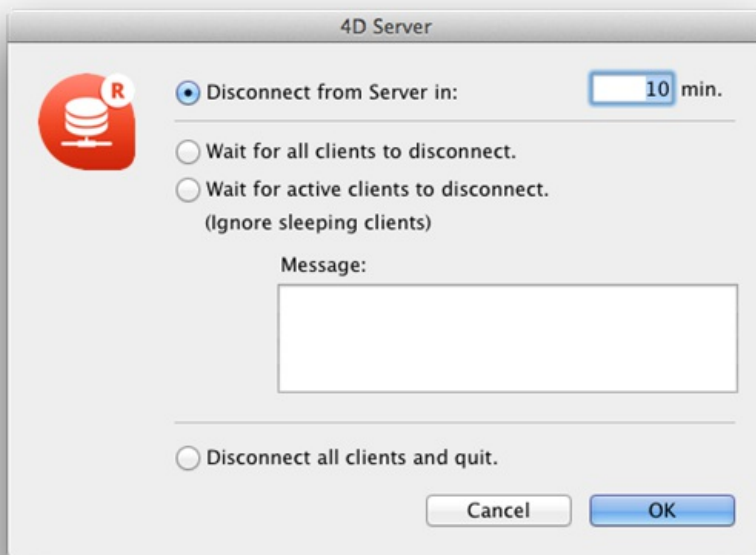
Note: You can also open an existing database directly by dragging and dropping an interpreted or compiled structure file (.4db ou .4dc) or a shortcut file (.4dlink) onto the 4D Server application icon.

Exiting 4D Server

To shut down the server:

1. Choose the Quit command from the File menu of 4D Server (Windows) or the 4D Server menu (OS X).

The following dialog box is displayed on the server machine:



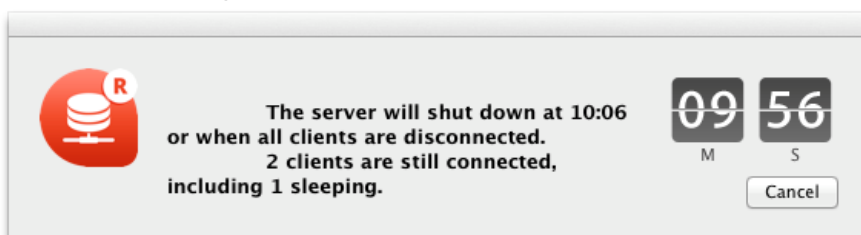
2. Enter the number of minutes in which you want the server to shut down, or choose the “Wait for all Users to disconnect” option.

As soon as you do this, no new client can connect to the server.

The following options are available:

- **Disconnect from Server in XX min.**

After the specified period of time, the server quits and all users are disconnected, including any clients in sleep mode. The following window appears on the server:



An identical window appears on each remote 4D machine. This window is repeated or updated on each client machine every 20 seconds or so, in order to prompt them to quit. When the time limit is reached, the server quits even if there are client machines still connected.

- **Wait for all clients to disconnect**

The server will only quit after all clients, including those in sleep mode, have disconnected. This option could be inappropriate for maintenance operations run during lunch time, for instance, since there are likely to be clients in sleep mode.

- **Wait for active clients to disconnect (Ignore sleeping clients)**

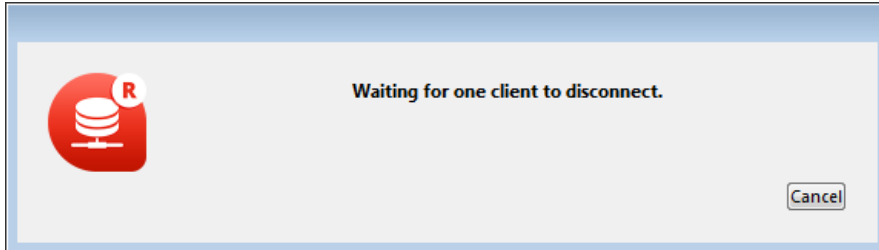
The server will only quit after all *active* clients have disconnected (in other words, all client machines that are not in sleep mode). With this option, any clients in sleep mode are not considered as connected. Use this option if you want to perform maintenance operations during lunch time, for example.

When this option is used, any clients in sleep mode will have a connection error when they wake up.

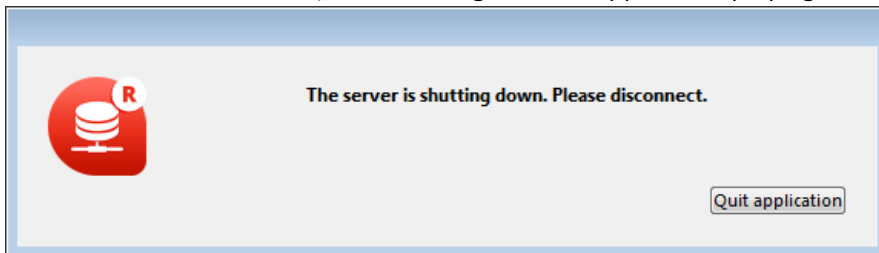
Note: This option requires the use of the *ServerNet* network layer. For more information, refer to [New ServerNet Network Layer \(compatibility\)](#).

Note: A *sleeping client* refers to a remote 4D application on a machine that has switched to sleep mode while the connection to the server machine was still active. For more information about this point, refer to [Managing sleeping users](#).

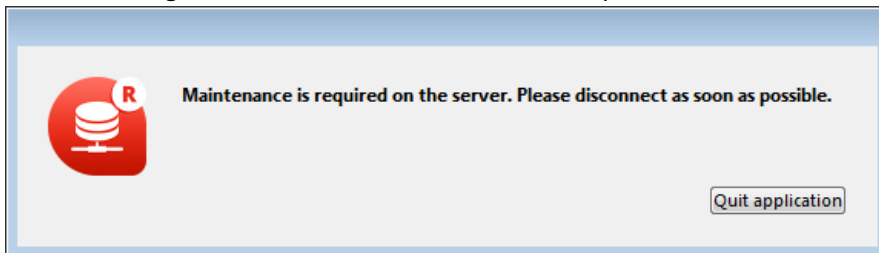
When you choose one of these options, the following window appears, which indicates the number of clients that are still connected:



On each 4D client machine, the following window appears displaying a default message:



If you entered a custom message in the 4D Server shutdown dialog box, it is displayed instead of the default message on each client machine. For example:



- **Disconnect all clients and quit**

The server ends all processes and all connections and quits after a few seconds.

Notes:

- In all cases, if no client is connected to the server when the shutting down window is validated, 4D Server quits immediately.
- If you click **Cancel** in the 4D Server shutdown window, the process of shutting down the server is canceled.
- You can close the database (and disconnect the clients) without quitting the 4D Server using the **Close database...** command. For more information, please refer to the [4D Server Menus](#) section.

Using 4D Server 64-bit version (Windows)

Beginning with version 12, 4D Server supports Windows 64-bit operating systems. The main advantage of 64-bit technology is the fact that more RAM memory can be addressed.

This section covers the particularities concerning the implementation and use of the 64-bit version of 4D Server.

Windows versions

A 64-bit 4D Server requires a Windows 64-bit operating system. Please refer to the certification matrices available on the 4D Web site to know which Windows operating systems are compatible with respect to your 4D Server release.

Architecture

The 4D Server.exe application intended for 64-bit architectures is a specific version, dedicated to this environment. It will not run on a 32-bit system.

Conversely, if you launch a 4D Server 32-bit version application on a Windows 64-bit system, it will run, but in emulation mode.

Any 4D client application, 32-bit or 64-bit, OS X or Windows, can connect to a 64-bit version of 4D Server (see the diagram below). 4D client applications include 4D in remote mode and applications merged with 4D Volume Desktop.

Compatibility note: In 4D v16, the 4D client 64-bit version applications for Windows are provided in pre-release version.

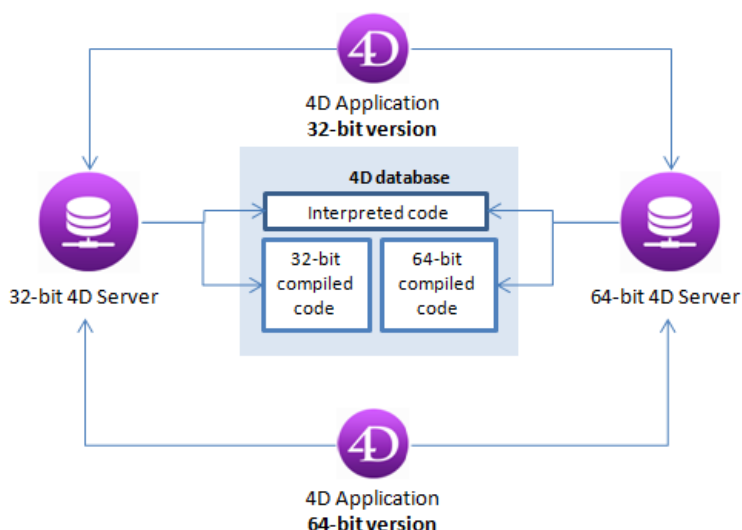
Compatibility

In interpreted mode, the same 4D databases can be executed with a 64-bit 4D Server or a 32-bit 4D Server. Development is identical regardless of which application is used (except with regard to the limitations described below).

In compiled mode, the databases must have been compiled for 64-bit processors in order to be executed with a 64-bit 4D Server (see the "64-bit compilation" paragraph).

A database that has been compiled in 32-bit only and that does not contain interpreted code cannot be executed with a 64-bit 4D Server.

General view of 4D Server 32-bit and 64-bit architecture

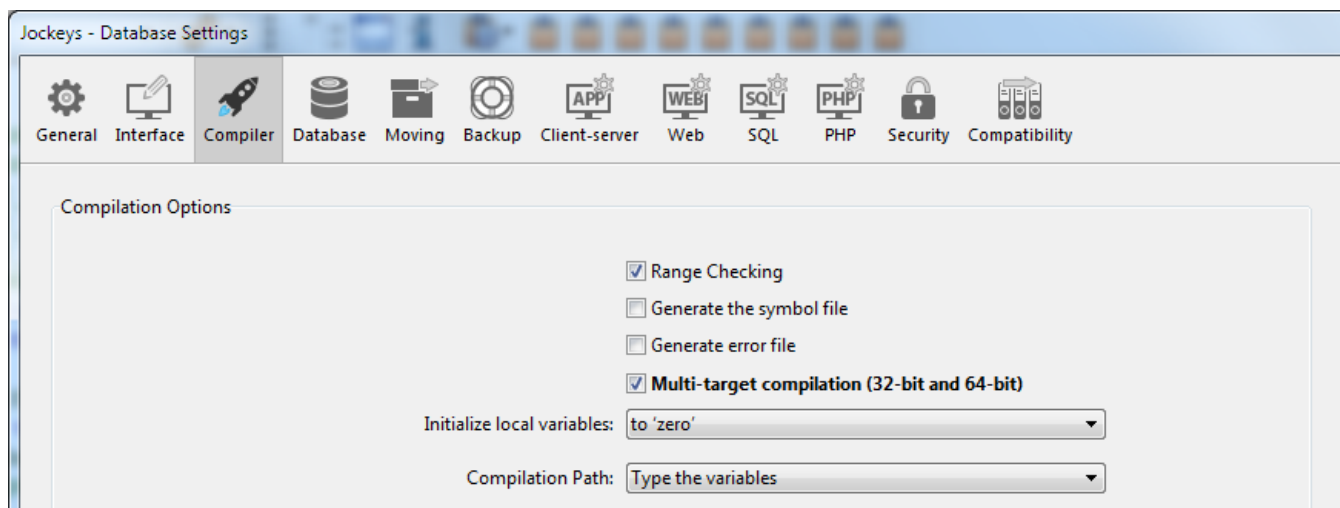


During execution, the following differences should be noted:

- Only the plug-ins compiled specifically in 64-bit mode will be loaded by the 64-bit 4D Server. A 64-bit plug-in must be built as a folder and placed in the Plugins folder of the server (the former architecture based on the .4DX and .RSR files placed in the Win4DX folder is no longer supported). 32-bit plug-ins are not loaded by a 64-bit 4D Server. They can however be stored in the Plugins folder of the server and distributed on the remote machines, without initialization. Mechanisms that call the server (for example, writing or reading templates on the server via the **WR SET AREA PROPERTY** command of 4D Write) will not function. The .4DX file for 32 bits must be placed in a subfolder named /Contents/Windows and the file for 64 bits must be placed in a subfolder named /Contents/Windows64.
- The compiled 4D components used with a 64-bit 4D Server must have been compiled in 64 bits.
- The amount of memory that can be used for the BLOBs loaded by the application remains limited to 2 GB.
- QuickTime is not supported by a 64-bit 4D Server on Windows. Generally, we do not recommend working with pictures in PICT format: if the picture is 100% Quickdraw, it can be handled by a 64-bit version of 4D Server, but if it contains Quicktime, it cannot be loaded.
- The use of integrated Web Kit in Web areas is not supported by a 64-bit 4D Server on Windows.

64-bit compilation

4D applications can be compiled for 32-bit and 64-bit processors. To do this, the **Multi-target compilation (32-bit and 64-bit)** option is available on the "Compiler" page of the Database Settings:



When this option is checked, the compiler includes the 64-bit code and the 32-bit code in the .4DC and .4DB files. These files can then be run with either a 32-bit or a 64-bit 4D Server. By default, this option is not checked.

Note: In order to compile a 64-bit version of the database, it must be operating in Unicode mode. Otherwise, an error is generated during compilation.

Size of cache memory

Since 64-bit architectures allow up to 1 TB (1000 GB) of RAM memory to be addressed, the cache memory that can be allocated to the 64-bit 4D Server is virtually unlimited.

Note: By comparison, 32-bit architectures are limited to 4 GB of RAM.

If the quantity of cache specified on the "Database/Memory" page of the database Settings cannot be obtained, 4D Server allocates the largest size possible and informs the user on startup of the application. It is then possible to quit or continue with the size offered.

Size of the process stack

The process stack running on a 64-bit version of 4D Server requires more memory than on a 32-bit version (about twice as much). When you create a process on the server using the **Execute on server** or **New process** commands with a 64-bit version of 4D Server, we recommend that you pass a minimum value of 128,000 bytes in the *stack* parameter and increase it when handling a sizeable call chain or if you receive an "out of stack" error. Make sure that you check this parameter when your code is intended for execution on a 64-bit 4D Server.

Using 4D Server 64-bit version (OS X)

Starting with version 15.1, a 64-bit version of 4D Server is provided for OS X. Thanks to this new product, your 4D Server applications can take full advantage of the power of 64-bit Apple machines. The main advantage of 64-bit technology is that more RAM memory can be addressed.

This section covers particularities concerning the implementation and use of the 64-bit version of 4D Server on OS X.

Minimum OS X version

A 64-bit 4D Server requires OS X version **10.9** (Mavericks) or higher. Please refer to the certification matrices available on 4D's web site to find out which operating systems are compatible with your version of 4D Server.

Architecture

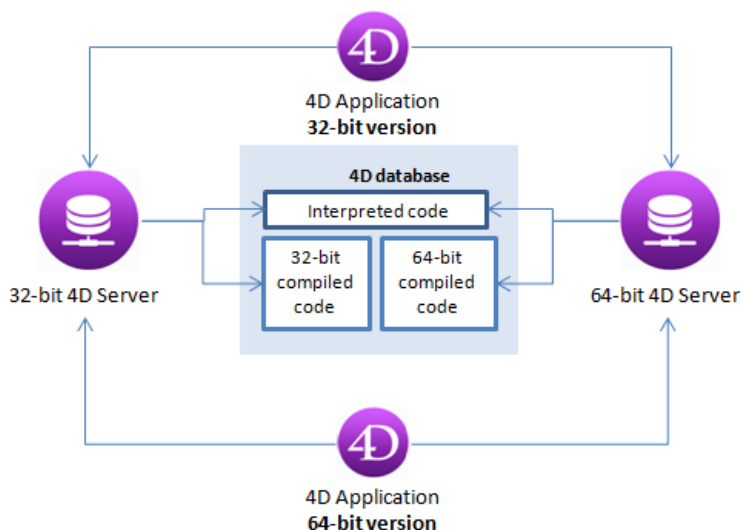
The 4D Server application intended for 64-bit architectures is a specific version, dedicated to this environment (it would not run on a 32-bit OS).

Any 4D client application, 32-bit or 64-bit, OS X or Windows, can connect to a 64-bit version of 4D Server (see the diagram below). 4D client applications include 4D in remote mode and applications merged with 4D Volume Desktop.

In interpreted mode, the same 4D databases can be executed with a 64-bit 4D Server or a 32-bit 4D Server. Development is identical regardless of which application is used (except for the limitations listed below).

In compiled mode, the databases must have been compiled for 64-bit processors in order to be executed with a 64-bit 4D Server. A database that has been compiled in 32-bit only and that does not contain interpreted code cannot be executed with a 64-bit 4D Server.

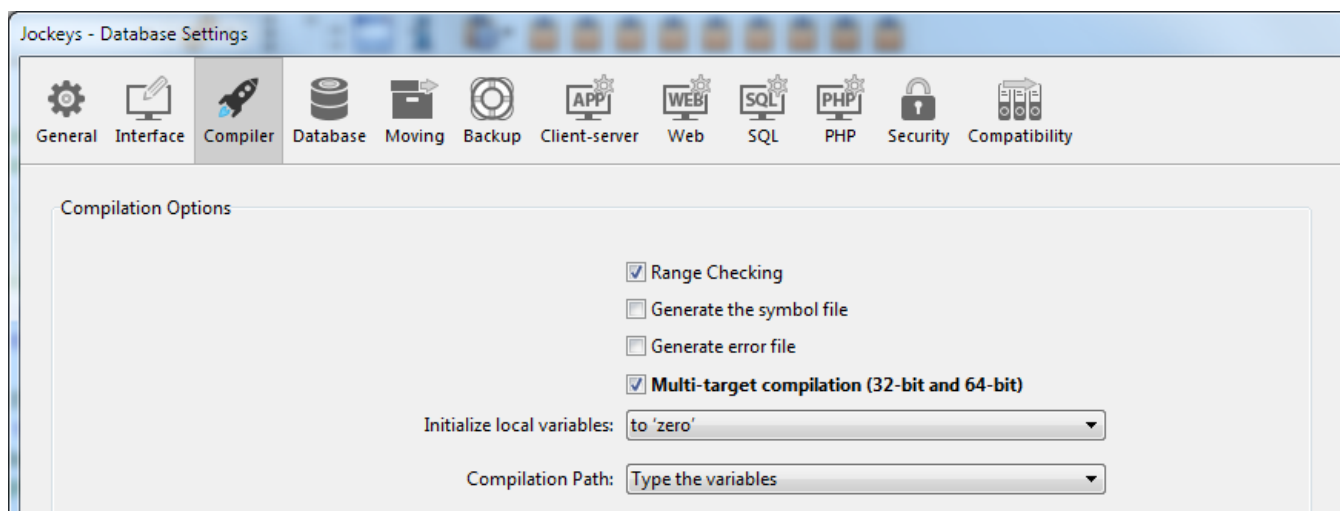
General view of 4D Server 32-bit and 64-bit architecture



If the database uses plug-ins, they must also be installed in 64-bit version for OS X on the server machine.

64-bit compilation

4D applications can be compiled for 32-bit and 64-bit processors. To do this, you need to use the **Multi-target compilation (32-bit and 64-bit)** option available on the "Compiler" page of the Database settings:



When this option is checked, the compiler includes the 64-bit code and the 32-bit code in the .4DC and .4DB files. These files can then be run with either a 32-bit or a 64-bit 4D Server. By default, this option is not checked.

Compiled Code Compatibility

In order to support OS X 64-bit architecture, the 4D built-in compiler was modified. Consequently, only databases compiled with 4D v15 or higher (**Note:** *the modification of the compiler has been effective since 4D v14 R3*) can run on OS X 64-bit. This means that:

- if you want to use existing 4D databases on OS X 64-bit in compiled mode, you need to recompile them with 4D v15 (or higher),
- if your databases use compiled components, you need to recompile the components with 4D v15 (or higher).

Size of the process stack

The process stack running on a 64-bit version of 4D Server requires more memory than on a 32-bit version (about twice as much). When you create a process on the server using the **Execute on server** or **New process** commands with a 64-bit version of 4D Server, we recommend that you pass the default value (0) or a minimum value of 512 KB in the *stack* parameter and increase it when handling a sizeable call chain or if you receive an "out of stack" error. Make sure that you check this parameter when your code is intended for execution on a 64-bit 4D Server.

Unsupported Features

The following features or technologies are not supported in the current 64-bit version of 4D Server for OS X:

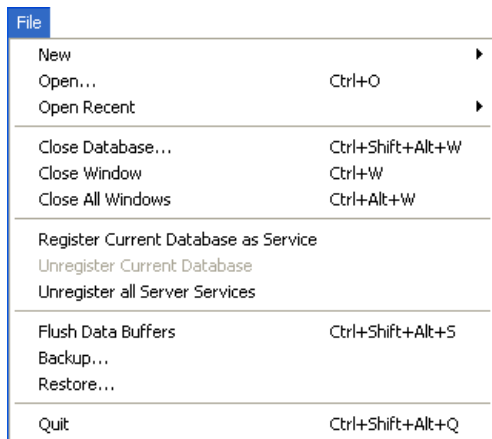
Feature/Technology	Comment
XSLT with Xalan	_o_XSLT APPLY TRANSFORMATION , _o_XSLT SET PARAMETER , and _o_XSLT GET ERROR will not work. Use the PHP <i>libxslt</i> module instead.
PICT format	'Unsupported image format' picture + file extension will be displayed instead (see Unavailable picture format). PICT format is globally deprecated in 4D, see also _o_AP Is Picture Deprecated
cicn icons	GET ICON RESOURCE command is not supported on the server (*)
Database .RSR files	Database .RSR files will not be opened automatically. You need to use Open resource file .
Writable resource files	_o_Create resource file is not supported on the server (*); you can only open resource files in read-only. Reminder: <i>Mac OS resource files have been deprecated since 4D v11.</i>
_o_Font number	This command is not supported on the server (*).
ASCII compatibility mode	Only Unicode mode is supported
Legacy network layer	Only <i>ServerNet</i> is supported (see New ServerNet Network Layer (compatibility))
Import/Export dialog boxes	Not available
Label editor	Not available
Using integrated Web Kit in Web areas	Not available

(*) An error is returned if it is executed on the server.

4D Server Menus

The interface of the 4D Server application is made up of the following menus: **File, Edit, Window, Help**. Under Mac OS, certain commands are found in the **4D Server** menu (application menu).

File



New

This hierarchical command has submenus that can be used to create a database or a new data file on the server machine.

The database creation commands are detailed in the [Creating a New 4D Server Database](#) section.

Open.../Open Recent

These commands can be used to open a database with 4D Server. The **Open Recent>** command displays a submenu listing databases that have been opened recently by 4D Server. To reset this menu, choose the **Clear Menu** command.

The database opening commands are detailed in the [Creating a New 4D Server Database](#) section.

Close Database...

This command closes the current database without exiting the 4D Server application. When you choose this command, the server shutdown dialog box appears so that you can set the disconnection mode for any connected clients(see the [Exiting 4D Server](#) section).

Close Window

This command closes the window in the foreground of the 4D Server application.

Close All Windows

This command closes all the windows of the 4D Server application. Note that in this case only the fact that the **Close Database...** command is activated in the **File** menu will indicate whether the database is still published.

Register Current Database as Service/Unregister Current Database/Unregister All

Server Services

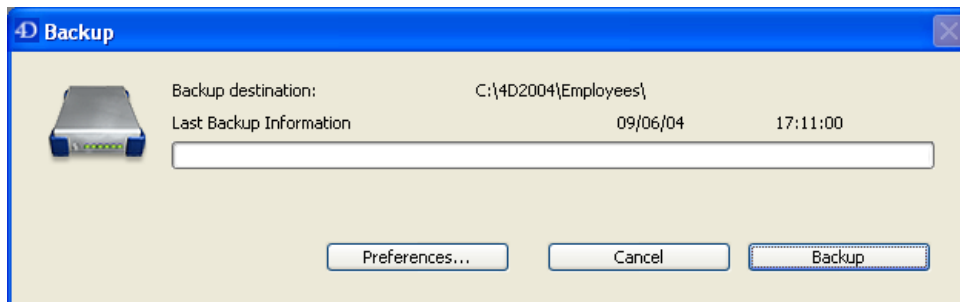
(Commands available under Windows) These commands are used to manage the registering of the database as a Service. This function is detailed in the [Registering a Database as a Service](#) section.

Flush Data Buffers

This command can be used to "force" the saving of data in the cache to the disk. By default, 4D Server automatically flushes the cache after a time limit set in the database preferences (Database/Data Management page).

Backup

This command lets you launch a back-up of the database at any time. When you select this command, the following dialog box appears:



- The **Backup** button immediately launches a back-up that takes the parameters set in the Preferences dialog box of the application into account (files to be backed up, location of archives, number of sets kept, etc.).
- The **Preferences** button opens the "Backup" theme of the Preferences, which lets you view and, if necessary, modify the current back-up settings.
- The **Cancel** button interrupts the back-up process.
For more information about back-up configuration, refer to the Design Reference manual of the 4D documentation.

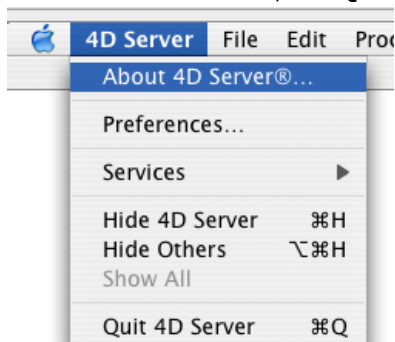
Restore...

This command displays an open file dialog box so that you can select the archive to restore.

Quit

This command lets you close the 4D Server application. For more information, refer to the [Exiting 4D Server](#) section.

Note: Under Mac OS X, the **Quit** command is located in the **4D Server** menu (application menu).



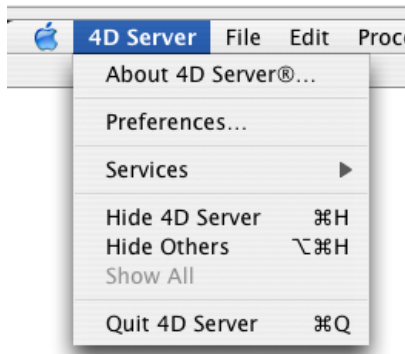
Edit



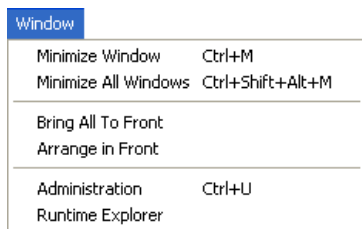
The **Edit** menu of 4D Server includes standard copy/paste commands, the **Show Clipboard** command, etc.

This menu also includes (under Windows) the **Preferences...** command, which displays the Preferences dialog box of the application. This dialog box is used to define numerous functions of the database. For more information about this dialog box, refer to the Design Reference manual of the 4D documentation. The preferences specific to 4D Server are described in the **Network and Client-Server options** and **IP Configuration** sections.

Note: Under Mac OS, the **Preferences...** command is found in the **4D Server** menu (application menu).



Window



The first part of the **Window**

menu includes standard commands for organizing workspace windows (these commands differ depending on the platform).

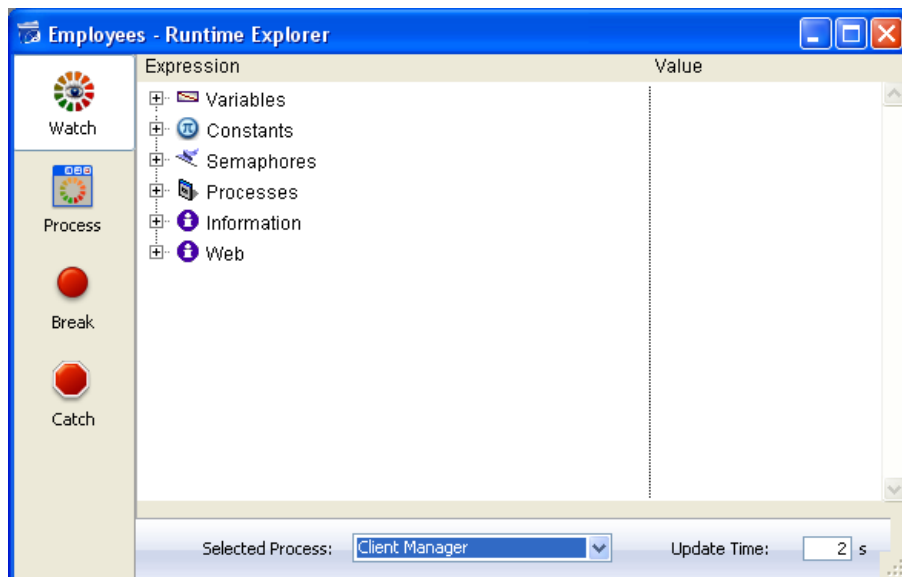
It also contains display commands for windows specific to 4D Server:

Administration

This command displays the 4D Server Administration window if it has been closed or minimized. This window is detailed in the 4D Server Administration Window chapter (see the **Monitor Page** section).

Runtime Explorer

This command displays the 4D Server Runtime Explorer window.



The Runtime Explorer enables you to view the status of the database various structural elements and to check that the available resources are correctly managed. The Runtime Explorer is particularly useful while developing or analyzing a database.

The Runtime Explorer window contains four pages that can be accessed by clicking on the following buttons: Watch, Process, Break and Catch. The Runtime Explorer works the same way in 4D Server and 4D. For more information, please refer to the 4D Design Reference manual (see chapter [Runtime Explorer](#)).

Help

Help

Maintenance Security Center
Update License...
About 4D Server@...

Maintenance Security Center

This command displays the Maintenance and Security Center (MSC) which groups together all the tools required for verification, analysis, maintenance, back-up and compacting of data and structure files.

This command is available even when no database is opened by 4D Server: in this case, it can be used to open a database in "maintenance mode" (it displays the standard open file dialog box so that you can designate the database to be opened). Maintenance mode is used more particularly for operations such as compacting or opening damaged databases.

For more information about the MSC, please refer to the Design Reference manual.

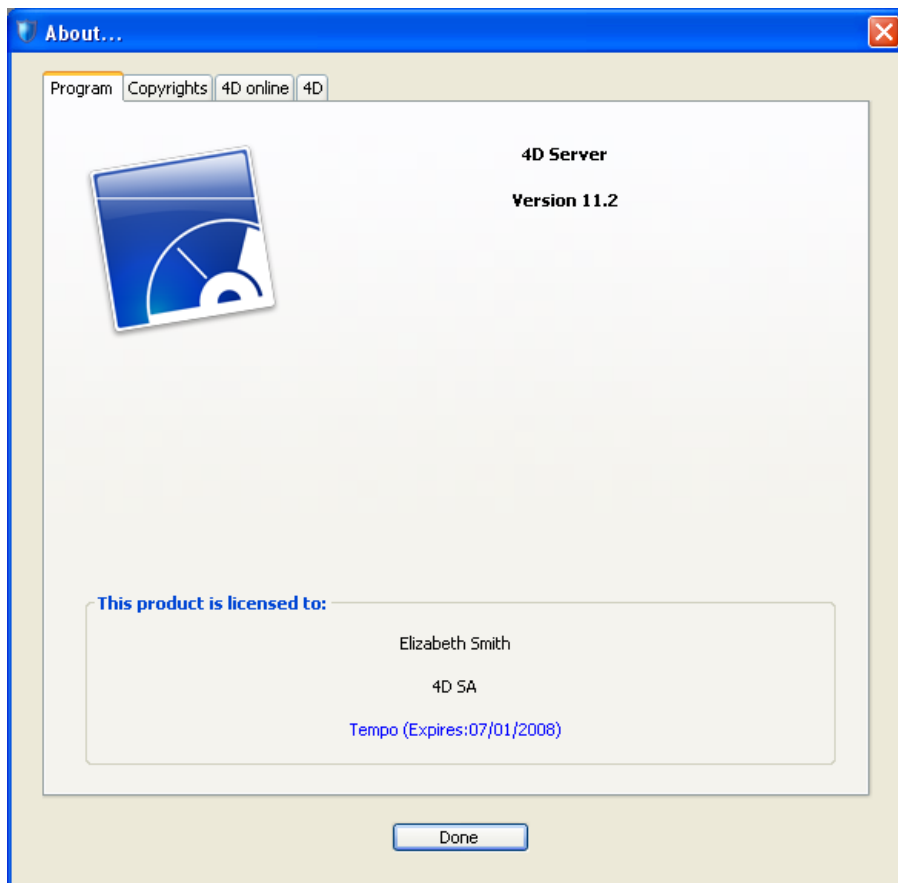
Update License...

This command displays the window used to activate additional licenses in your 4D environment.

For more information about this dialog box, refer to the 4D Installation Guide manual.

About 4D Server...

This command displays the 4D Server About... window, which provides various information organized in different pages that can be accessed via tabs:

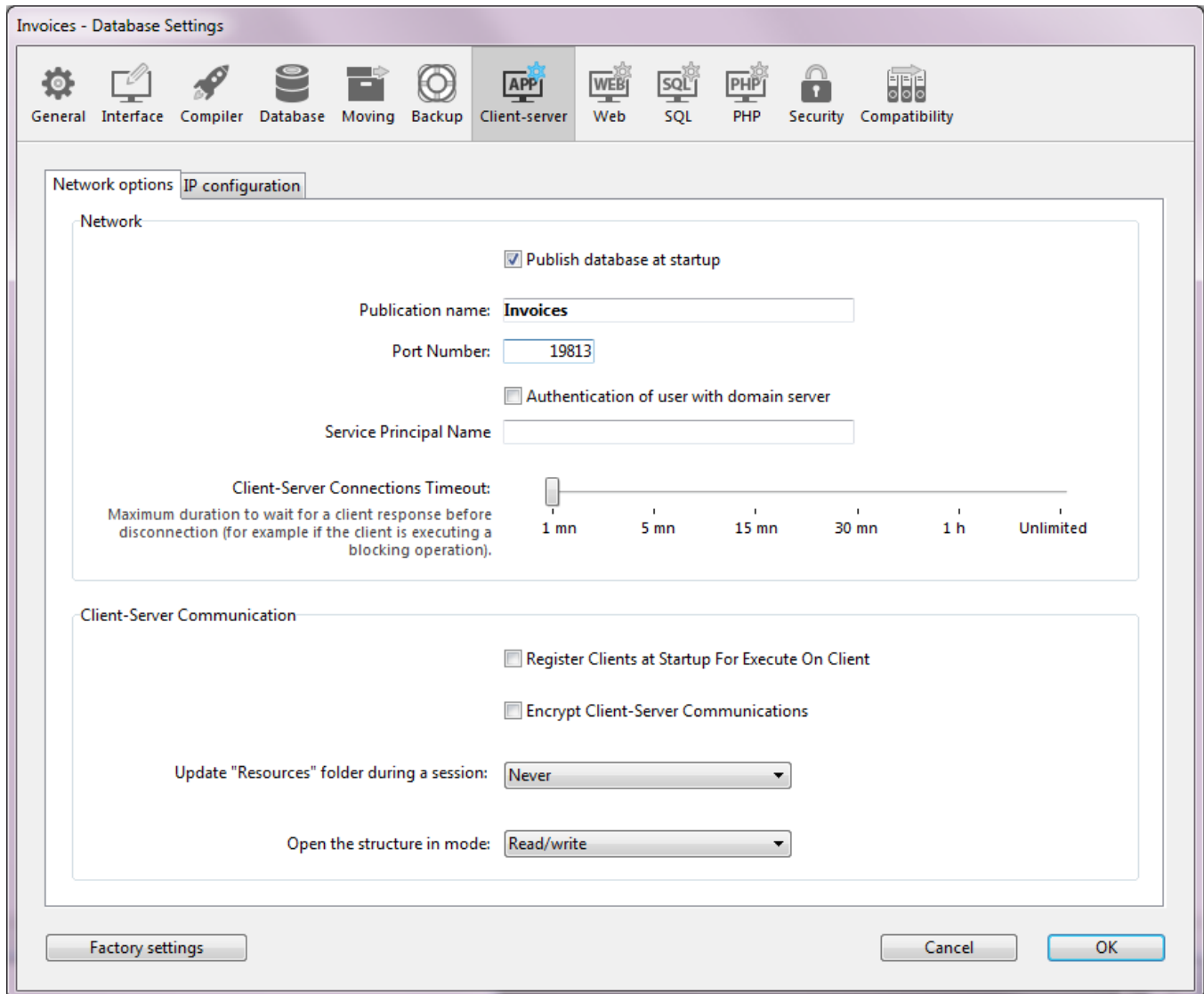


- Program: 4D Server version and license
- Copyrights: legal mentions
- 4D online: additional resources related to 4D that can be accessed on line
- 4D: a view of the company 4D SAS around the world.

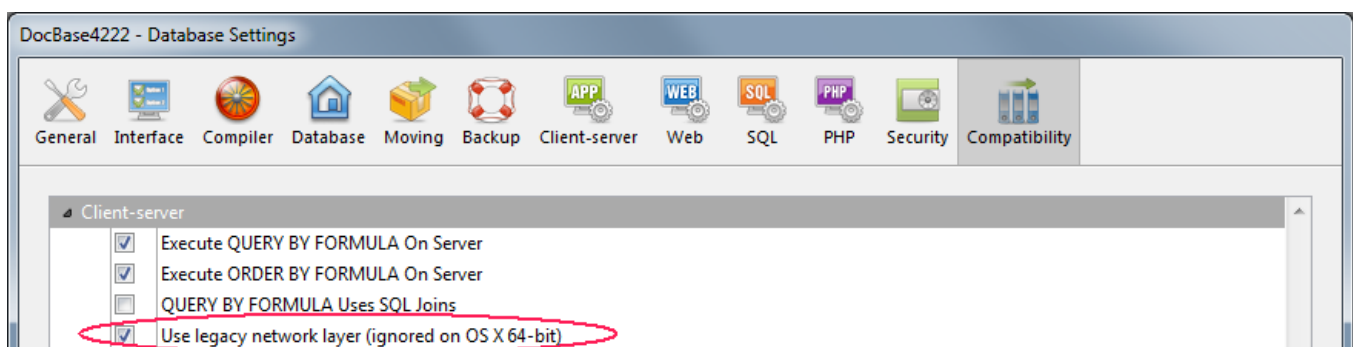
Note: Under Mac OS, the **About 4D Server** command is found in the **4D Server** menu (application menu).

Network and Client-Server options

You can set various parameters concerning the network and the client-server communication on the "Network options" tab of the **Client-Server** page in the Database settings (accessible on both 4D in remote mode and 4D Server):



Moreover, beginning with 4D Server v14 R5, a compatibility option allows you to enable or disable the former network layer at any time:



These parameters are detailed in this section.

Network

Publish database at startup

This option lets you indicate whether or not the 4D Server database will appear in the list of published databases.

- When this option is checked (default option), the database is made public and appears in the list of published databases (**TCP/IP** page).
- When the option is not checked, the database is not made public and it does not appear in the list of published databases. To connect, users must manually enter the address of the database on the **Custom** page of the connection dialog box.

Note: If you modify this parameter, you must restart the server database in order for it to be taken into account.

Publication name

This option lets you change the publication name for a database published by 4D Server, i.e., the name displayed on the dynamic TCP/IP publication page of the connection dialog box (see the **Connecting to a 4D Server Database** section).

By default, 4D Server uses the name of the database structure file. You can enter any custom name you want.

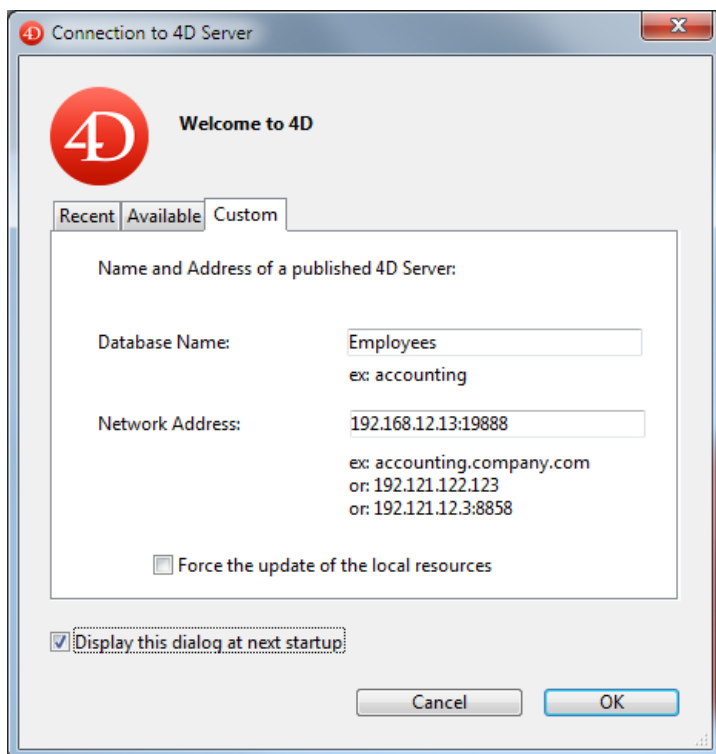
Note: This parameter is not taken into account in custom client-server applications. In theory, the client application connects directly to the server application, without passing by the connection dialog box. However, in the event of an error, this dialog box will appear; in this case, the publication name of the server application is the name of the compiled database.

Port Number

This option lets you change the TCP port number on which 4D Server publishes the database. This information is stored in the structure of the database and on each client machine. By default, the TCP port number used by 4D Server and 4D in remote mode is 19813.

Customizing this value is necessary when you want to use several 4D applications on the same machine with the TCP protocol; in this case, you must specify a different port number for each application.

When you modify this value from 4D Server or 4D, it is automatically passed on to all the 4D machines connected to the database. To update any other client machines that are not connected, you just need to enter the new port number (preceded by a colon) after the IP address of the server machine on the Custom page of the connection dialog box at the time of the next connection. For example, if the new port number is 19888:



Note: Only databases published on the same port as the one set in 4D client are visible on the TCP/IP dynamic publication page.

4D Server and port numbers

4D Server uses three TCP ports for communications between internal servers and clients:

- **SQL Server:** 19812 by default (can be modified via the "SQL/Configuration" page of the Preferences).
- **Application Server:** 19813 by default (can be modified via the "Client-Server/Configuration" page of the Preferences, see above).
- **DB4D Server** (database server): 19814 by default . This port number cannot be modified directly but it always consists of the application server port number + 1.
When a 4D client connects to 4D Server, it uses the TCP port of the application server (19813 or the port indicated after the colon ':' in the IP address shown in the connection dialog box). Connection to other servers via their respective ports is then automatic; it is no longer necessary to specify them.
Note that in the case of access via a router or a firewall, the three TCP ports must be opened explicitly.

Authentication of user with domain server

This option allows you to implement SSO (*Single Sign On*) capabilities in your 4D Server database on Windows. When you check this option, 4D transparently connects to the Active directory of the Windows domain server and gets the available authentication tokens.

This option is described in the [Single Sign On \(SSO\) on Windows](#) section.

Service Principal Name

When Single Sign On (SSO) is enabled (see above), you must fill in this field if you want to use Kerberos as your authentication protocol.

This option is described in the [Single Sign On \(SSO\) on Windows](#) section.

Client-Server Connections Timeout

This thermometer is used to set the timeout (period of inactivity beyond which the connection is closed) between 4D Server and the client machines connecting to it.

The Unlimited option removes the timeout. When this option is selected, client activity control is eliminated.

When a timeout is selected, the server will close the connection of a client if it does not receive any requests from the latter during the specified time limit.

Client-Server Communication

Register Clients at Startup For Execute On Client

When this option is checked, all the 4D remote machines connecting to the database can execute methods remotely. This mechanism is detailed in the section [Stored procedures on client machines](#).

Encrypt Client-Server Connections

This option lets you activate the secured mode for communications between the server machine and the 4D remote machines. This option is detailed in the [Encrypting Client/Server Connections](#) section.

Update Resources folder during a session

This setting can be used to globally set the updating mode for the local instance of the **Resources** folder on the connected 4D machines when the **Resources** folder of the database is modified during the session (the **Resources** folder is automatically synchronized on the remote machine each time a session is opened). Three settings are available:

- **Never:** The local **Resources** folder is not updated during the session. The notification sent by the server is ignored. The local **Resources** folder may be updated manually using the **Update Local Resources** command.
- **Always:** The synchronization of the local **Resources** folder is automatically carried out during the session whenever notification is sent by the server.
- **Ask:** When the notification is sent by the server, a dialog box is displayed on the client machines, indicating the modification. The user can then accept or refuse the synchronization of the local **Resources** folder. The **Resources** folder centralizes the custom files required for the database interface (translation files, pictures, etc.). Automatic or manual mechanisms can be used to notify each client when the contents of this folder have been modified. For more information, please refer to the [Managing the Resources folder](#) section.

Open the structure in mode

This option sets the opening mode for the database structure on client machines. By default, the **Read/Write** mode is set but you can also set it to **Read only** in order to prevent the structure from being modified.

New ServerNet Network Layer (compatibility)

Starting with 4D v14 R5, 4D applications contain a new network layer, named *ServerNet*, to handle communications between 4D Server and remote 4D machines (clients). *ServerNet* is based on a modern and robust API. It is easy to maintain and will facilitate the implementation of the latest network technologies while providing a high level of performance.

Using *ServerNet* is transparent from the user's point of view. Keep in mind, however, that when *ServerNet* is used, the names of databases published in secured mode are no longer preceded by a "^" character as was the case with the former network layer (see [Encrypting Client/Server Connections](#)).

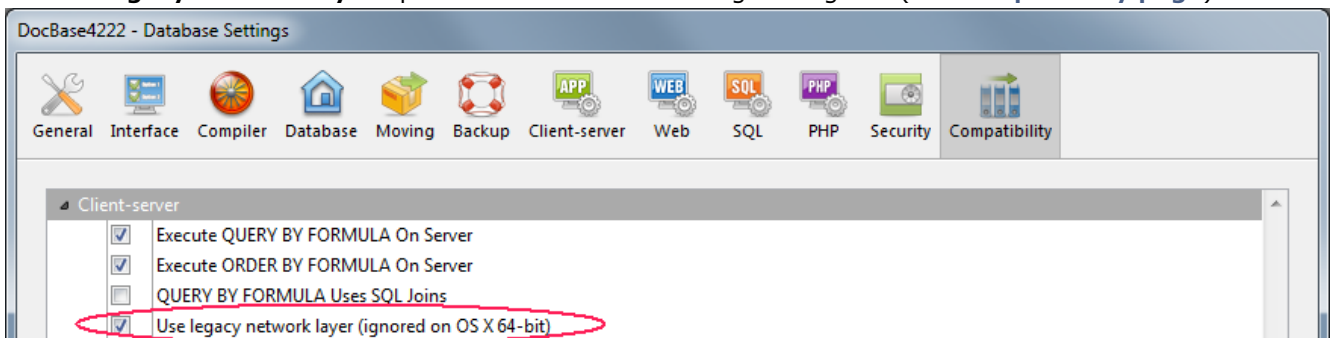
The former "legacy" network layer is still supported in order to ensure compatibility for existing databases. *ServerNet* is automatically used in newly created databases.

There are options that allow you to enable/disable the *ServerNet*. To ensure that your applications will benefit from future network evolutions, we recommend that you enable *ServerNet* gradually in all your databases.

Activating or deactivating the legacy network layer

You can enable or disable the legacy network layer in your 4D Server at any time. You can use either:

- the [Use legacy network layer](#) constant with the **SET DATABASE PARAMETER** command, or
- the **Use legacy network layer** option in the Database Settings dialog box (see [Compatibility page](#)):



Note: As indicated in its title, this option is ignored in the 64-bit version of 4D Server for OS X; only *ServerNet* can be used on this platform.

By default, the *serverNet* network layer is:

- automatically enabled in new databases, created with 4D v14 R5 or subsequent versions,
- automatically disabled in converted databases.

Migrating merged 4D clients

When you activate the *ServerNet* layer in your existing server application, only compliant 4D client applications will be able to connect:

- Client applications in version 15 (starting with 4D v14 R4) can connect with no modification.
- Client applications in previous versions (v14.x and all previous v14 'R' releases) must first be upgraded in order to be able to connect to the server.

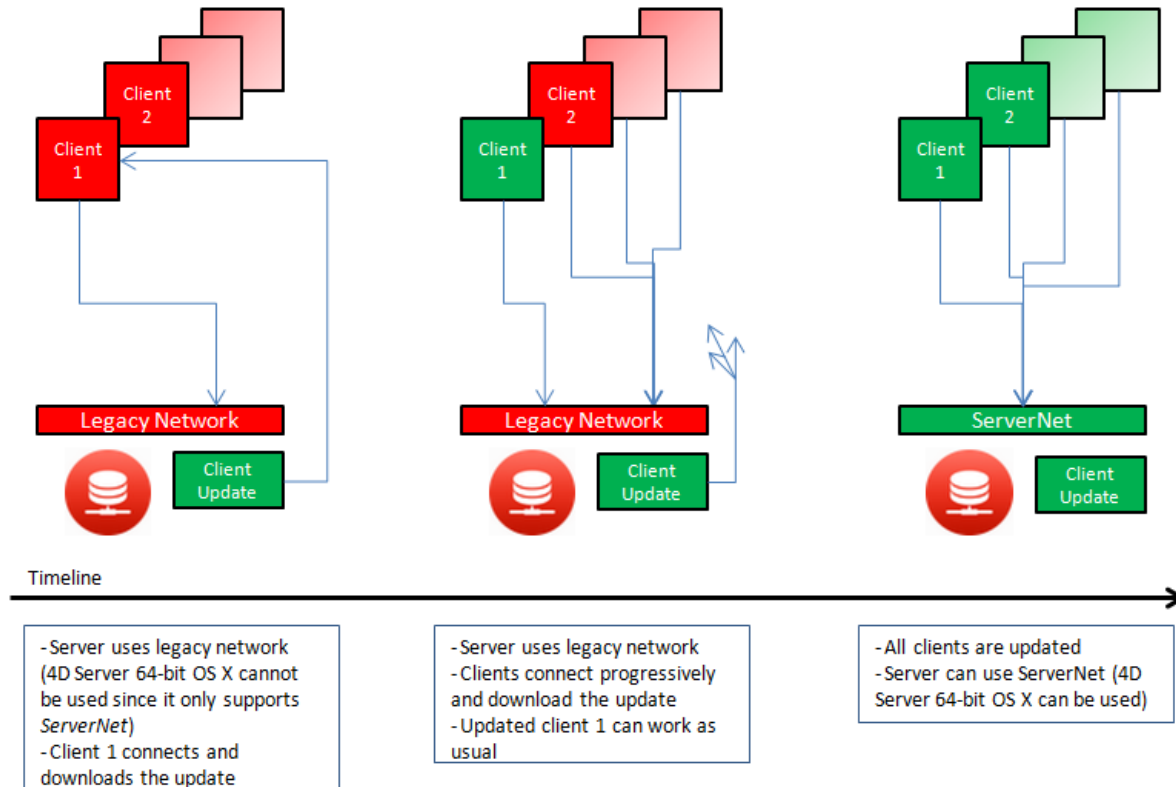
If your application works with merged 4D Volume Desktop clients in versions earlier than v14 R4, and you want to use the 4D Server automatic mechanism to distribute your updated client applications over the network, you need to establish a migration strategy. This strategy must be based upon the following principles:

- Non-compatible clients can only connect to a 4D Server that uses the legacy network layer.
- Updated clients can adapt their protocol dynamically so they can connect to 4D Server v15 and higher, regardless of the network layer that the server is using.

Your migration strategy should follow these steps:

1. Build updated client applications with 4D v15 or higher.
 2. Run 4D Server v15 or higher with the "Use legacy network layer" parameter enabled.
This configuration allows all clients to connect.
- Note:** Keep in mind that the 64-bit version of 4D Server for OS X does not support this option.
3. Wait for a given period of time until every client has connected and downloaded the new version.
This may last for a day, a week or even longer. During this transition period, both previous and updated clients are able to connect to the legacy network server.
 4. Once all the clients have been updated, you can deactivate the legacy network layer and finally switch to the *ServerNet* on 4D Server.

This strategy is depicted in the following diagram:



Logging client requests

During the migration process, we recommend that you enable the "Diagnostic log recording" file. When this file is activated, 4D Server logs each client update request in this file, allowing you to monitor the process. This log is not activated by default: you need to call the **SET DATABASE PARAMETER** command with the [Diagnostic log recording](#) constant set to 1.

For each update request, the following information is logged:

- client IP
- client version
- "Update client" event

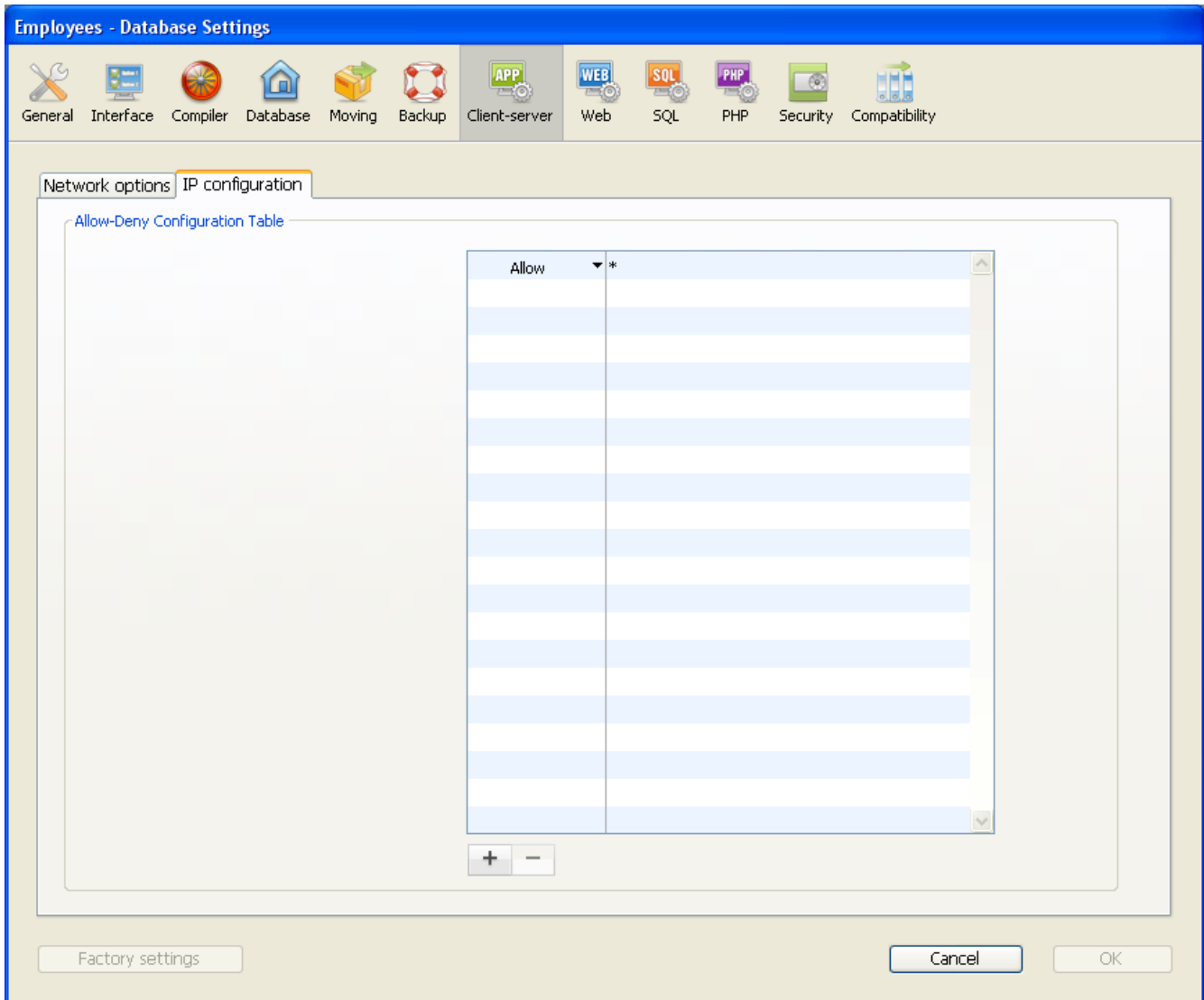
Monitoring the log file is also useful *after* you have switched the server to the *ServerNet* network layer, to make sure all the clients have been updated properly. If a non-compatible client attempts to connect, the server logs the following entry:

- client IP
- client version
- "Fail to connect" event

In this case, you can decide, for example, to update client manually.

IP Configuration

You can set parameters concerning the network configuration of 4D Server databases on the “IP configuration” tab of the **Client-Server** page in the Database settings (accessible on both remote 4D machines and 4D Server):



These parameters are detailed in this section.

Allow-Deny Configuration Table

This table allows you to set access control rules for the database depending on 4D remote machine IP addresses. This option allows reinforcing security, for example, for strategic applications.

Note: This configuration table does not control Web connections.

The behavior of the configuration table is as follows:

- The “Allow-Deny” column allows selecting the type of rule to apply (Allow or Deny) using a pop-up menu. To add a rule, click on the Add button. A new row appears in the table. The Delete button lets you remove the current row.
- The “IP Address” column allows setting the IP address(es) concerned by the rule. To specify an address, click in the column and enter the address in the following form: 123.45.67.89. You can use an * (asterisk) character to specify “starts with” type addresses. For example, 192.168.* indicates all addresses starting with 192.168.

- The application of rules is based on the display order of the table. If two rules are contradictory, priority is given to the rule located highest in the table.
You can re-order rows by modifying the current sort (click the header of the column to alternate the direction of the sort). You can also move rows using drag and drop.
 - For security reasons, only addresses that actually match a rule will be allowed to connect.
In other words, if the table only contains one or more Deny rules, all addresses will be refused because none will match at least one rule. If you want to deny only certain addresses (and allow others), add an Allow * rule at the end of the table. For example:
 - Deny 192.168.* (deny all addresses beginning with 192.168)
 - Allow * (but allow all other addresses)
- By default, no connection restrictions are applied by 4D Server: the first row of the table contains the Allow label and the * (all addresses) character.

Encrypting Client/Server Connections

You can configure the client/server connections so that 4D Server and 4D remote workstations communicate in secured mode.

The secured client/server communication is based on TLS (Transport Layer Security) protocol, which was formerly the SSL (Secured Socket Layer).

TLS Protocol and Client/Server Connections

The TLS protocol has been designed to secure data exchanges between two applications —primarily between a Web server and a browser. It is designed to authenticate the sender and receiver, and to guarantee the confidentiality and integrity of the exchanged information. For a detailed description of the secured protocol, refer to section [Using TLS Protocol](#) in the 4D Language Reference.

Between 4D Server and a remote 4D, authentication and integrity mechanisms are handled transparently by 4D Server and do not require any additional user setting.

Note: Encrypting client/server connections slows connections.

Settings

At the network level, the TLS protocol is inserted between the TCP/IP layer (low level) and the high level protocol. To use TLS in a “classic” client/server architecture, the following certificate files must be present:

- *key.pem*: document containing private encryption key
- *cert.pem*: document containing certificate.

These files must be located in the *Resources* subfolder of the 4D Server and 4D applications. They must be present on the server machine and on each remote machine. Default *key.pem* and *cert.pem* files are provided with 4D. For a higher level of security, we strongly recommend that you replace these files with your own certificates. For more information about creating custom certificates, refer to the [How to get a certificate?](#) section of 4D's Web server documentation (the procedure is identical).

You must also enable connections in secured mode.

To do this, open the “Client-Server/Network options” page of the Database settings dialog box and select the option **Encrypt Client/Server Connections** in the “Encryption” area (see the [Network and Client-Server options](#) section).

By default, the option is not checked.

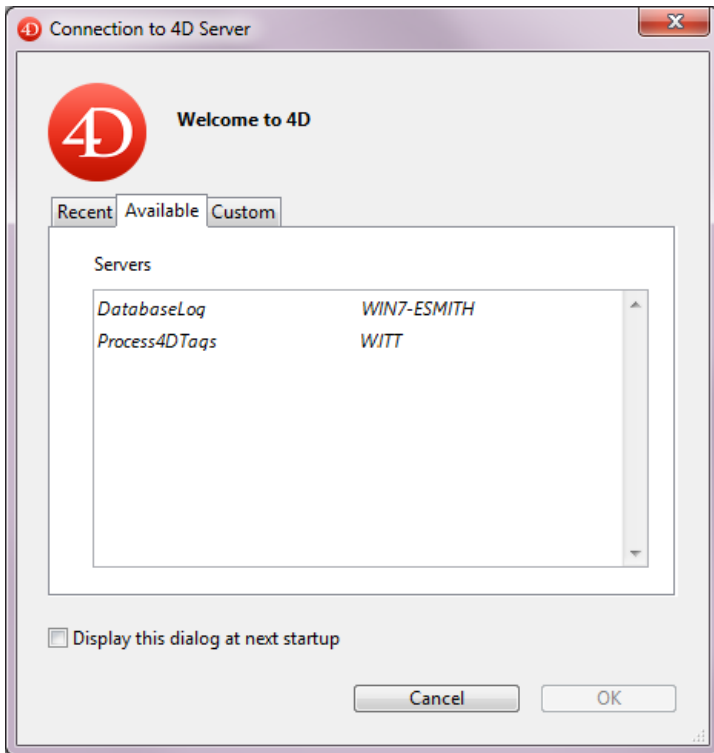
You must then quit and relaunch 4D Server so that this setting is taken into account.

All 4D remote stations will then connect in secured mode.

Connection in Secured Mode

- **Use of ServerNet**

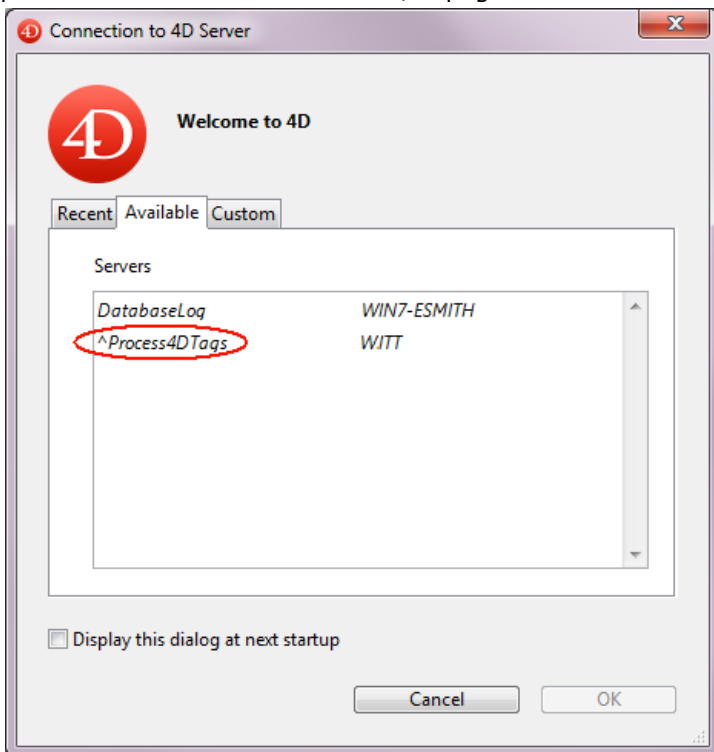
When the new *ServerNet* network layer is used, the activation of the secured protocol on the server is transparent (the publication mode does not appear in the connection dialog box). The connection and switch to secure mode takes place following an internal negotiation between the server application and the remote application.



For more information about the *ServerNet*, refer to the **Network and Client-Server options** section.

- **Use of legacy network layer**

When the legacy network layer is used, the “circumflex accent” (^) is placed before the name of the databases published in TLS mode in the TCP/IP page of the connection dialog box in 4D Server:



Note:When a database is not published dynamically on the TCP/IP page of the Connection dialog box, the user can enter its name in the **Custom** page (see the **Connecting to a 4D Server Database and IP Configuration** sections). In this case, a ^ (circumflex accent) must be placed before the database name if the database is published in secured mode; otherwise the connection will be rejected .

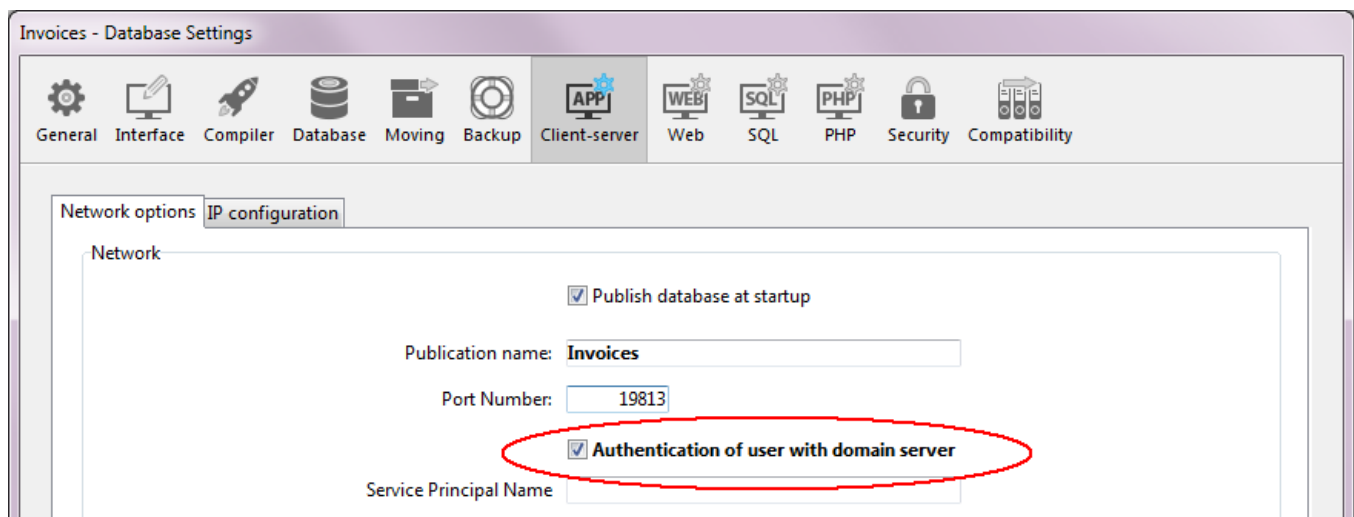
Single Sign On (SSO) on Windows

4D Server allows you to implement SSO (*Single Sign On*) capabilities in your client-server solutions on Windows. Implementing SSO in your 4D solutions will allow users to access the 4D application on Windows without needing to reenter their password when they are already logged into the Windows domain of their company (using Active Directory). Behind the scenes, the 4D Server application delegates the authentication to Active Directory and gets the Windows session login, which you can use to log the 4D user into the database by means of your standard login method.

Note: SSO is only available on 4D Server (4D single-user applications do not support SSO).

Enabling the SSO feature

By default, the SSO feature is not enabled in 4D Server. To benefit from this feature, you need to set the **Authentication of user with domain server** option on the Client-Server/Network options page of the Database Settings dialog box of 4D Server:

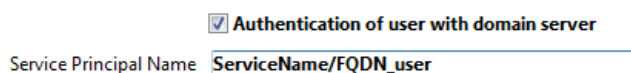


When you check this option, 4D transparently connects to the Active directory of the Windows domain server and gets the available authentication tokens.

This option offers standard authentication through the NTLM protocol. 4D supports both NTLM and Kerberos protocols. The protocol used is automatically selected by 4D depending on the current configuration (see [Requirements for SSO](#)). If you want to use the Kerberos protocol, you need to fill in the additional SPN field as well (see below).

Enabling Kerberos

If you want to use Kerberos as your authentication protocol, you also need to fill in the **Service Principal Name** option on the Client-server/Network options page of the Database Settings dialog box:



This option declares the SPN as set in the Active Directory configuration. A service principal name is a unique identifier of a service instance. SPNs are used by Kerberos authentication to associate a service instance with a service logon account. This allows a client application to request that the service authenticates an account even if the client does not have the account name. For more information, please refer to the [SPN page on the MSDN web site](#).

The SPN identifier must respect this pattern:

- "ServiceName/FQDN_user" if the SPN is a computer attribute

- "ServiceName/FQDN_computer" if the SPN is a user attribute

Where:

- *ServiceName* is the name of the service which the client wants to authenticate.
- The *Fully Qualified Domain Name (FQDN)* is a domain name that specifies its exact location in the tree hierarchy of the Active Directory for both computers and users.

In 4D databases, the SPN can be set:

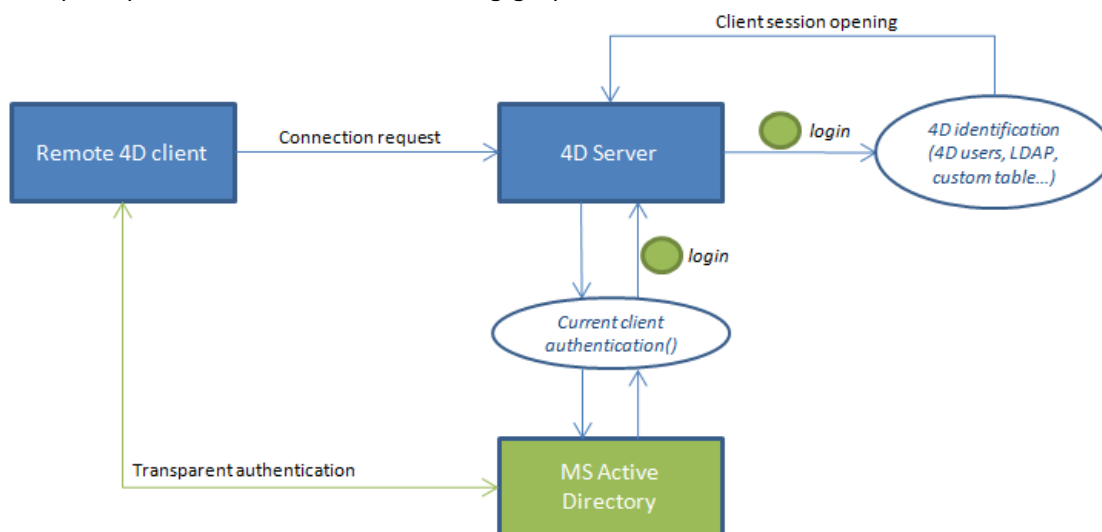
- in the database structure settings, for use with 4D Server.
- or in the user settings (*settings.4DSettings* file stored in the database's Preferences folder) for deployment needs.

Implementing SSO

When SSO features are enabled (see above), you can rely on user authentication based on Windows session credentials to open a user session on 4D Server.

Keep in mind that the SSO feature only provides you with an authenticated login; it is up to you to pass this login to your standard 4D login method. When a 4D remote application tries to connect to the server, you have to call the 4D **Current client authentication** command, which will return the user login, as defined in the Active Directory. You can then pass this login to your own identification system (using the built-in user and groups, the LDAP commands, or any custom mechanism) to open the appropriate session for the remote user in your 4D application.

This principle is illustrated in the following graphic:



The **Current client authentication** command must be called in the **On Server Open Connection Database Method**, which is called each time a remote 4D opens a new connection to the 4D Server database. If authentication fails, you can return a non-null value in \$0 to reject the connection.

Using the Current client authentication command

To call the **Current client authentication** command, use the following syntax:

```
login:=Current client authentication(domain;protocol)
```

Where:

- *login* is the ID used by the client to log into the Active Directory (text value). You need to use this value to identify the user within your database.
If the user is not correctly authenticated, an empty string is returned and no error is returned.
- *domain* and *protocol* are optional text parameters. They are filled by the command and allow you to accept or reject connections depending on these values:
 - *domain* is the Active Directory domain name
 - *protocol* is the name of the protocol used by Windows to authenticate the user.

For more information on this command, please refer to the [Current client authentication](#) command description.

Requirements for SSO

4D Server handles various SSO configurations, depending on the current architecture and settings. The protocol used for authentication (NTLM or Kerberos) as well as information returned by the [Current client authentication](#) command depend on the actual configuration, if all requirements are respected (see below). The protocol actually used for authentication is returned in the *protocol* parameter of the [Current client authentication](#) command.

The following table provides the **requirements for using NTLM or Kerberos** authentication:

	NTLM	Kerberos
4D Server and 4D remote are on different machines	yes	yes
4D Server user is on domain	yes	yes
4D remote is on the same AD as 4D Server user	yes or no(*)	yes
SPN is filled in on 4D Server	no	yes(**)
Information returned by Current client authentication if requirements are respected	<i>user</i> =expected login, <i>domain</i> =expected domain, <i>protocol</i> ="NTLM"	<i>user</i> =expected login, <i>domain</i> =expected domain, <i>protocol</i> ="Kerberos"

(*) The following specific configuration is supported: the 4D remote user is a local account on a machine that belongs to the same AD as 4D Server. In this case, the *domain* parameter is filled with the 4D Server machine name. Note that the support depends on actual user settings: if not available, empty strings are returned.

(**) If all Kerberos requirements are respected but the [Current client authentication](#) command returns "NTLM" in *protocol*, this means that you are facing one of the following situations:

- The SPN syntax is not valid; in other words, it does not respect the [constraints imposed by Microsoft](#).
- Or, the SPN has duplicates in the AD. This issue needs to be fixed by the AD administrator.

Note: A valid syntax does not mean that the SPN declaration itself is correct; more specifically, if the SPN does not exist in the AD, [Current client authentication](#) returns empty strings.

Managing the Resources folder

The **Resources** folder of the database can be used to share custom data (pictures, files, subfolders, etc.) between the server machine and all the client machines. On the server machine, the **Resources** Resources folder must simply be located next to the database structure file.

All referencing mechanisms associated with the **Resources** folder are supported in client/server mode (.lproj folder, XLIFF, pictures and so on). For more information about this point, please refer to the 4D Design Reference manual.

Each client has a local copy of this folder. The contents of the local folder are automatically synchronized with that of the server each time the client connects.

Moreover, client machines can be dynamically "notified" during a session when the contents of the **Resources** folder of the server database are modified by a developer. This notification can be triggered:

- either automatically by the server, two minutes after the last modification made by a client (this delay helps to avoid inopportune notification in the case where numerous files are being copied).
- or manually via the **Notify clients** command in the action menu of the resources explorer on the client machine at the origin of the modification.
- or by programming, via a **NOTIFY RESOURCES FOLDER MODIFICATION** command. This command is useful when the contents of the **Resources** folder are modified on the server machine via a stored procedure.

On the client side, the way the notification of any modifications will be handled depends on how the "Update "Resources" folder during a session" preference is set. This can also be set individually via the **SET DATABASE PARAMETER** command. Three choices are available: **no synchronization**, **auto synchronization** or **ask**. For more information, please refer to the **Network and Client-Server options** section and the description of the **SET DATABASE PARAMETER** command.

Lastly, each client machine can synchronize itself with the server at any time via the **Update Local Resources** command in the action menu of the resources explorer. For more information about the resources explorer, please refer to the 4D Design Reference manual in the **Resources explorer** chapter.

Compatibility Note: In previous versions of 4D Server, the transfer of custom data was carried out via a folder named "Extras" that was placed next to the structure file. This folder is now obsolete and its use is no longer recommended. It nevertheless remains supported by 4D Server in order to preserve the compatibility of existing applications.

Registering a Database as a Service

Under Windows, 4D Server can be launched as a Service.

Compatibility note: This function is no longer available under Mac OS starting with version 12 of 4D Server.

A 4D Server application registered as a service is automatically launched on start-up of the machine with the current database, even before a user session is opened. It is not closed when the user exits their session.

This operation lets you guarantee the availability of a 4D Server database even in the event of an incident that requires restarting the machine. Maintenance can be carried out remotely.

Notes:

- On a 64-bit Windows platform, a 4D Server application registered as a service is executed with no interface (the server administration window does not appear).
- For more information about the mechanisms for managing Services, refer to the documentation of your operating system.

To register a 4D Server database as a Service, select **Register Current Database as Service** in the **File** menu of 4D Server. The next time the machine is started, 4D Server will be launched automatically and the current database opened. You can register any number of databases. Each database can be registered only once.

Note: Under Windows, this command may be grayed out when access to the service management functions is restricted. In this case, to be able to use this command, you must launch 4D Server with an administrator level (to do this, right click on the application icon and choose the **Run as administrator** command in the context menu).

Warning: Be sure to use a valid account when you open the session, otherwise an error message will be displayed. By default, 4D Server is executed with the "Local System Account", which may not necessarily have the settings required to use your application. In particular, if you want to be able to print, you must open the session with a user account which has default print settings. The issue is similar if you want to access network volumes. We recommend to change the account: go to **Control Panel > System and Security > Administrative Tools > Services**. In the Services list, right-click on 4D Server, choose the **Properties** option, then go to the **Log On** tab and specify the account under which the server must run (setting used at next startup).

To unregister your database, select **Unregister Current Database** from the 4D Server **File** menu. This command is dimmed if the database is not registered as a service.

To unregister all 4D Server databases at once, select **Unregister all Server Services** from the 4D Server **File** menu. This command is dimmed if no 4D Server service is enabled.

You cannot change the service registration status of 4D Server from within 4D Server, if the application has been launched as a service on start-up. In this case, the three menu items are disabled. To stop the service, use the Services control panel.

Setting up a logical mirror

4D Server offers an integrated solution that allows the setting up of a backup system via a logical mirror. This solution is based on two commands: **New log file** and **INTEGRATE MIRROR LOG FILE**.

What is a logical mirror?

A logical mirror is a sophisticated backup mode, primarily intended for critical or high-load databases.

Using a logical mirror consists in operating a database on one machine and keeping a copy of it that is periodically updated on a second machine. Both machines communicate via the network with the machine in operation regularly transmitting any changes made in the data to the mirror machine via the intermediary of the log file.

In this way, when there is an incident affecting the operational database, the mirror database can be used to get things back in working order quickly without any data loss. Moreover, the operational database is never “blocked” by backup operations.

Why choose to back up using a logical mirror?

The use of a logical mirror corresponds to specific needs. The standard strategy based on periodic backups and the use of a log file in most cases offers a simple, reliable and inexpensive solution. The database is backed up regularly (every 24 hours in general). During backup, all processes are frozen. This period of partial unavailability is very short, and even in the case of large databases (greater than 2 GB), it lasts no longer than 5 minutes. This operation can be programmed to take place outside of normal periods of database usage.

Nevertheless, for certain kinds of organizations, such as hospitals for instance, critical databases must be entirely operational 24 hours a day. The database cannot be "being backed up" (and thus unavailable), even for a very short period of time. In this case, setting up a logical mirror is an appropriate solution.

Note: The mirror database only reflects changes made to the **data**. This backup mode is not suitable for databases in the process of development, where frequent structural modifications will make the mirror rapidly obsolete or will require repeated updating of the mirror database structure.

How it works

Setting up a backup system using a logical mirror is based on two new commands: **New log file** and **INTEGRATE MIRROR LOG FILE**. These commands are described in the 4D Language Reference manual.

The following principles are implemented:

- The database is installed on the main 4D Server machine (operational machine) and an identical copy of the database is installed on the 4D Server mirror machine.
- A test on startup of the application (for instance, for the presence of a specific file in a subfolder of the 4D Server application) is used to distinguish between the two versions (operational and mirror) and thus execute the appropriate operations.
- On the 4D Server machine in operation, the log file is “segmented” at regular intervals using the **New log file** command. Since no backup is carried out on the main server, the database remains permanently available in read-write mode.
- Each “segment” of the log file is sent to the mirror machine, where it is integrated into the mirror database using the **INTEGRATE MIRROR LOG FILE** command.

Setting up this system requires programming specific code, in particular:

- A timer on the main server for managing the execution cycles of the **New log file** command,
- A transfer system for the “segments” of the log file between the operational machine and the mirror machine

- (using 4D Internet Commands for a transfer via FTP or messaging systems, Web Services, etc.),
- A process on the mirror machine intended to supervise the arrival of new "segments" of the log file and to integrate them using the **INTEGRATE MIRROR LOG FILE** command,
 - A communication and error-handling system between the main server and the mirror server.

WARNING: A backup system using a logical mirror is not compatible with "standard" backups on a database in use since the simultaneous use of these two backup modes would lead to the desynchronization of the operational and mirror databases. Consequently, you must be sure that no backups, whether automatic or manual, are carried out on the operational database. On the other hand, it is possible to back up the mirror database or to set up a "mirror-mirror" (see following paragraph).

Backup of a mirror database and mirror-mirror

4D Server can be used to carry out backups of the database on the mirror machine.

Any conventional means can be used to carry out backups on the mirror machine: manual backups using the command in the **File** menu, scheduled backups set in the Database Settings or programmed backups using language commands.

To avoid risks of desynchronization with the operational machine, 4D automatically locks the mirror machine when it is carrying out one of two basic operations: the integration of the log file from the operational machine and the backup of the mirror database.

- When a log file is being integrated, it is not possible to carry out a backup. If the **BACKUP** command is used, the error 1417 is generated (see the **Backup Manager Errors (1401 -> 1421)** section in the 4D Language manual).
- When a backup is underway, all the processes are frozen and it is not possible to launch the integration of a log file.

Beginning with 4D v14, you can enable the current log file on the mirror machine, which means that you can set up a "mirror-mirror" (or even a series of mirrors), or a "hub-and-spoke" mirror architecture (several mirrors for the same operational database). In the first case, the current log file of the mirror is sent in turn to another mirror (the "mirror-mirror") for integration, and so forth if you use a series of mirrors. In the second case, the current log is sent directly to several identical mirror servers. This type of redundancy ensures the continuous availability of the server, even in the case of simultaneous failure of the server and the main mirror.

Operating scenario for a logical mirror

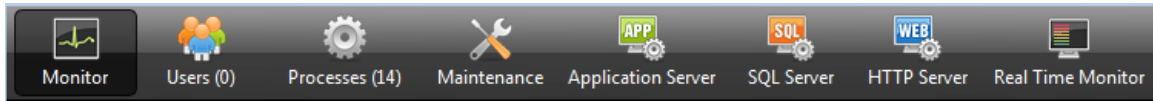
The following scenario illustrates, from the viewpoint of each 4D Server machine, the setting up and operation of a backup system using a mirror:

Step	Operational machine	Mirror machine
1	<p>Start up of the application, back up of the data file. The log file is activated by default; for better security, store this file on a separate hard drive.</p> <p>4D creates the MyDatabase.journal file.</p> <p>The application is exited.</p>	
2	<p>Copy of all the database files (log file included) onto the mirror machine.</p> <p>Restarting of the application and beginning of operation (verify that there is not a full backup programmed).</p>	<p>Start up of the mirror application. 4D Server requests the current log file: Select the MyDatabase.journal file that was transferred from the operational database. This file will be used when setting-up a mirror-mirror.</p>
3	<p>Decision made to update the mirror (for example, after a certain period of operation).</p>	
	<p>Execution of the method containing the New log file. The file saved is named MyDatabase[0001-0001].journal.</p>	
	<p>Sending of the MyDatabase[0001-0001].journal file via programming to the mirror machine (using 4DIC, Web Services, etc.).</p>	
	<p>The database is operating.</p>	<p>Detection of a file that is waiting to be integrated. Execution of the method containing the INTEGRATE MIRROR LOG FILE command in order to integrate the MyDatabase[0001-0001].journal file. If you are using a mirror-mirror, execution on the mirror machine of a procedure similar to step 3 (to be repeated each time a log is integrated).</p>
4	<p>Incident occurs on the machine; the database is unusable. Decision made to switch to the mirror machine.</p>	
	<p>Copy of the current log file MyDatabase.journal onto the mirror machine, via the usual destination folder</p>	
6	<p>Analysis of incident and repair.</p>	<p>Detection of a file that is waiting to be integrated. Execution of the method containing the INTEGRATE MIRROR LOG FILE command in order to integrate the MyDatabase.journal file.</p> <p>The database is operating.</p>
7	<p>The machine is repaired. Replacement of the database files by those of the mirror database. Start up of the application. 4D Server requests the log file: Select the file that was transferred from the mirror database.</p>	<p>The database is exited. Return to step 2.</p>

4D Server Administration Window









4D Server has a comprehensive and user-friendly administration window.

This window provides different analysis and control tools for published databases. The window contains several pages that can be accessed using the buttons at the top:



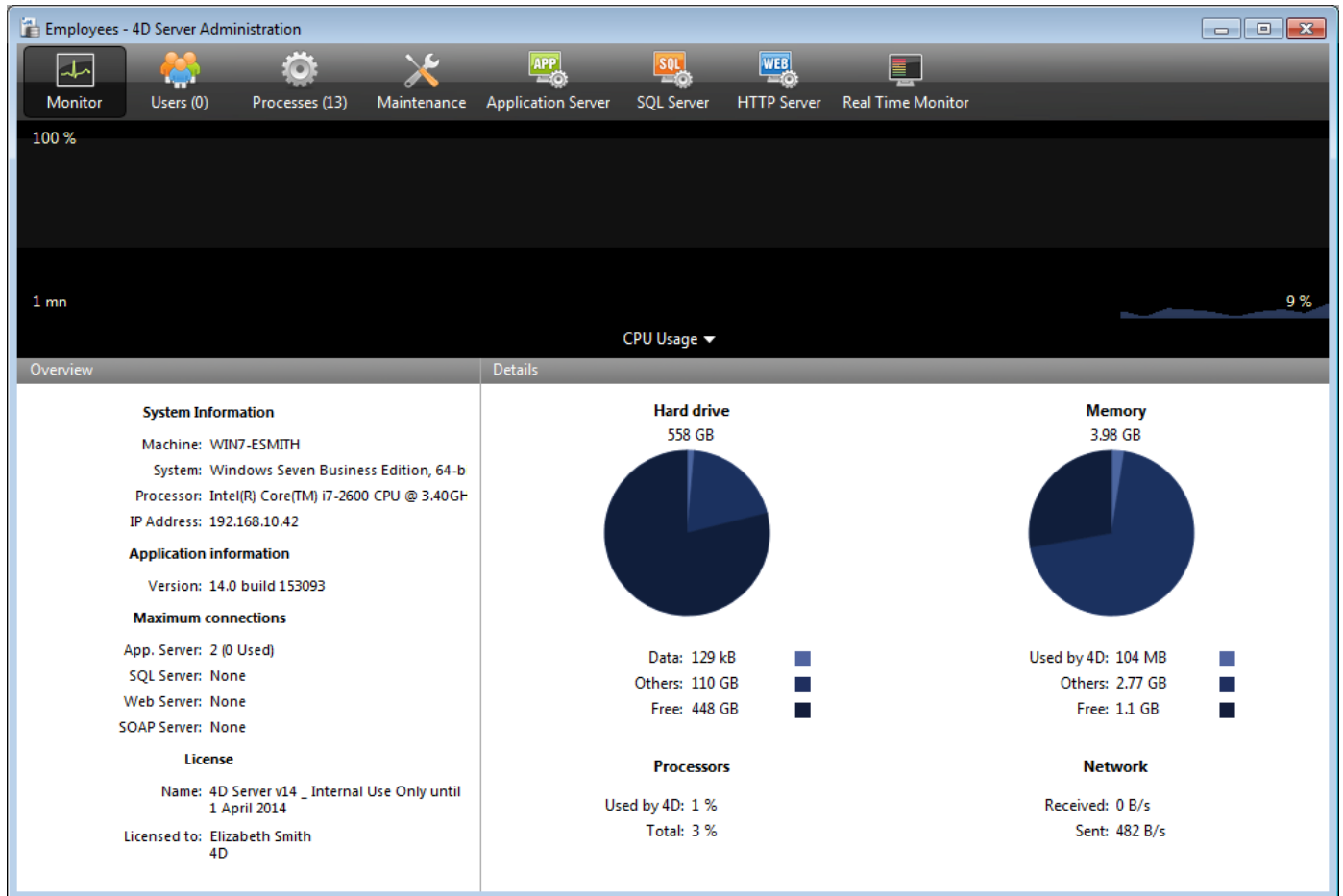
Each page is detailed in a section of this chapter.

Note: The administration window can be accessed from a remote 4D machine. For more information about this point, please refer to [Administration from Remote Machines](#) section.

-  Monitor Page
-  Users Page
-  Processes Page
-  Maintenance Page
-  Application Server Page
-  SQL Server Page
-  HTTP Server Page
-  Real Time Monitor Page

Monitor Page

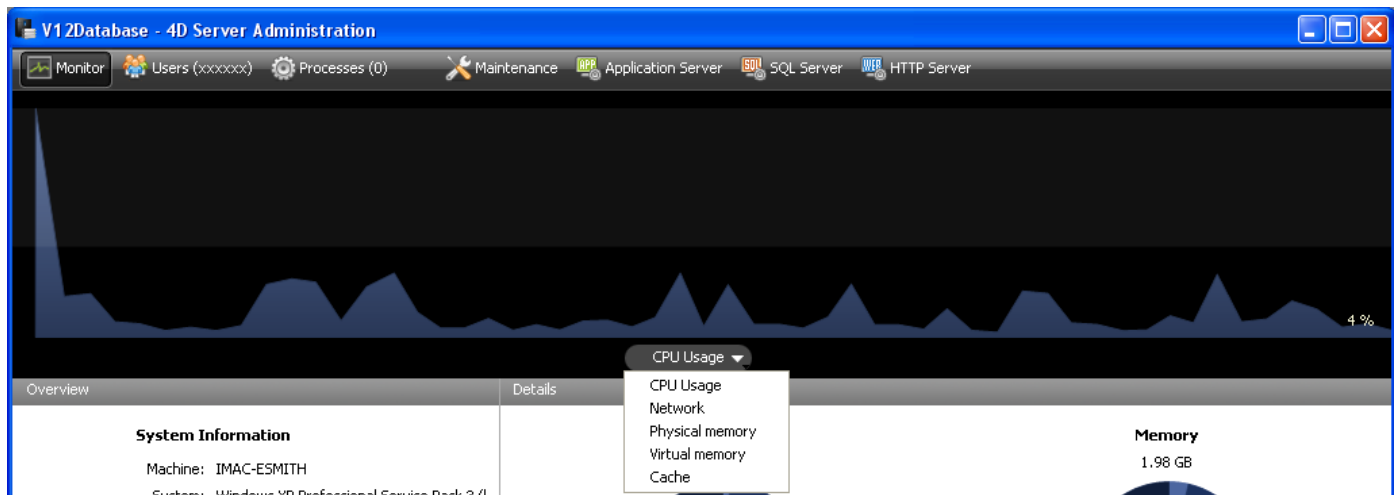
The Monitor Page displays dynamic information concerning database use as well as information about the system and the 4D Server application:



Note: Under Windows, the display of this information is related to the rights of the user that opened the session. For more information, please refer to the "Display of Monitor Information (Windows)" paragraph.

Graphic Area

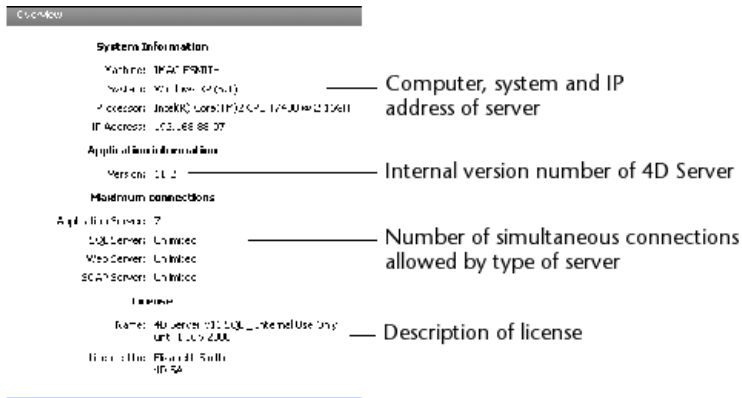
The graphic area lets you see the evolution in real time of several parameters: the CPU usage, network traffic and memory. You select the parameter to be displayed via a menu found in the center of the window:



- **CPU Usage:** Overall CPU usage of the machine, for all applications taken together. The specific part of 4D Server in this usage rate is provided in the "Processors" information area.
- **Network:** Number of bytes received per second by 4D Server. The number of bytes sent by 4D Server is provided in the "Network" information area.
- **Physical memory:** Quantity of RAM memory of machine used by 4D Server. A more detailed view of memory use is provided in the "Memory" information area.
- **Virtual memory:** Displays in the graph area the quantity of virtual memory used by the 4D Server application. This memory is allocated by the system according to the application needs. The value found at the bottom right of the area indicates the quantity of memory currently being used. The value found at the top left indicates the maximum quantity of usable virtual memory. The maximum value is calculated dynamically according to the general memory settings of the application.
- **Cache:** Displays in the graph area the quantity of cache memory used by the 4D Server application. The value found at the bottom right of the area indicates the quantity of memory currently being used. The value found at the top left indicates the total size of the cache memory, as set via the Database Settings. Note that when this option is selected, the graph area scrolling is slowed down since an efficient analysis of the cache is generally carried out over a fairly long observation period.

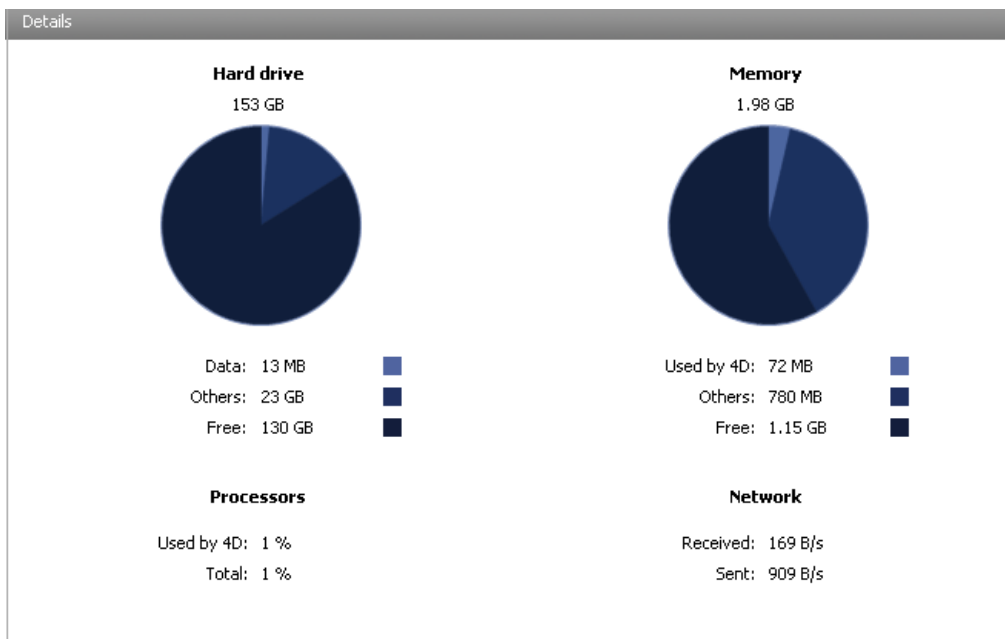
Overview Area

The "Overview" area provides various information concerning the system, application and licenses installed on the 4D Server machine.



Details Area

The "Details" area repeats part of the information displayed in the graphic area and provides additional information as well.



- **Hard drive:** Overall capacity of the hard disk and distribution of the space used by the database data (data file

+ data index), the space used by other files and the free space available.

- **Memory:** RAM memory installed on the machine and amount of memory used by 4D Server, by other applications or that is free.
The memory used by 4D Server can also be displayed dynamically in the graphic area.
- **Processors:** Instant occupancy rate for processor(s) of the machine by 4D Server and by other applications. This rate is constantly recalculated.
The occupancy rate by 4D Server can also be displayed dynamically in the graphic area.
- **Network:** Instantaneous number of bytes received via the network by 4D Server and number of bytes sent by the application. This value is updated constantly.
The number of bytes received by 4D Server can also be displayed dynamically in the graphic area.

Display of Monitor Information (Windows)

Under Windows, some of the system information displayed on the Monitor page are retrieved via the Windows "Performance Analyzer" tools. These tools can only be accessed when the user that opened the session where 4D Server was launched has the necessary authorization. This user must either:

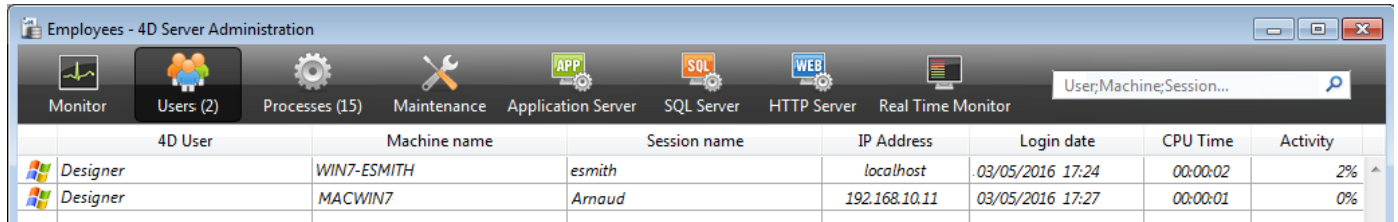
- belong to the "Administrators" group,
- under Windows Vista: belong to the "Power Users" (for a user that is not an Administrator).

To place a user that is not an Administrator in the "Power Users" group under Windows Vista (you will need to use an Administrator account to carry out these operations):

1. Go to the Control Panel and open the "User Accounts" panel.
2. Click on the "Advanced" tab and then on the "Advanced" button in the "Advanced user management" section.
The "Local Users and Groups" application is run.
3. Double-click on the "Groups" folder.
4. Double-click on the "Power Users" group.
A window named "Power Users Properties" appears.
5. Click on the **Add...** button in order to add a user.
6. In the text area titled "Enter the object names to select," enter the user names that you want to authorize.
7. Click **OK** (twice).
Close the "Local Users and Groups" application then the "User Accounts."

Users Page

The Users Page lists the 4D users connected to the database:



The screenshot shows the 'Users (2)' tab in the 4D Server Administration interface. The table below lists the active users:

4D User	Machine name	Session name	IP Address	Login date	CPU Time	Activity
Designer	WIN7-ESMITH	esmith	localhost	03/05/2016 17:24	00:00:02	2%
Designer	MACWIN7	Arnaud	192.168.10.11	03/05/2016 17:27	00:00:01	0%

The "Users" button indicates, in parentheses, the total number of users connected to the database (this number does not take into account any display filters applied to the window).



This page also contains a dynamic search area and control buttons.

You can modify the order of the columns by dragging and dropping their header areas. You can also sort the list of column values by clicking on its header:

Click several times to specify in turn an ascending/descending order.

List of Users

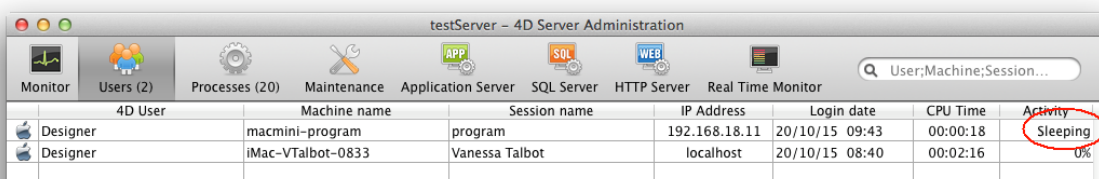
For each user connected to the database, the list provides the following information:

- System of the client machine (Mac OS or Windows) as an icon.
- 4D User: Name of the 4D user. If passwords are not activated, all users are named "Designer."
- Machine name: Name of the client machine.
- Session name: Name of the session opened on the client machine.
- IP Address: IP address of the client machine.
- Login date: Date and time of the client machine connection.
- CPU Time: CPU time consumed by this user since connecting.
- Activity: Ratio of time that 4D Server devotes to this user (dynamic display). "Sleeping" if the client machine has switched to sleep mode (see below).

Managing sleeping users

4D Server specifically handles cases where a machine running a 4D remote application switches to sleep mode while its connection to the server machine is still active. In this case, the connected 4D remote application automatically notifies 4D Server of its imminent disconnection.

On the server, the connected user changes to a **Sleeping** Activity status:



The screenshot shows the 'Users (2)' tab in the 4D Server Administration interface. The table below lists the active users, with one user in a sleeping state:

4D User	Machine name	Session name	IP Address	Login date	CPU Time	Activity
Designer	macmini-program	program	192.168.18.11	20/10/15 09:43	00:00:18	Sleeping
Designer	iMac-VTalbot-0833	Vanessa Talbot	localhost	20/10/15 08:40	00:02:16	0%

This status frees up resources on the server. In addition, the 4D remote application reconnects to 4D Server automatically after waking up from sleep mode.

The following scenario is supported: a remote user stops working for awhile, for example during a lunch break, but keeps the connection to the server open. The machine switches to sleep mode. When the user returns, they wake the machine up and the 4D remote application automatically recovers its connection to the server as well as the session context.

Search/filtering Area

User;Machine;Session...

This function can be used to reduce the number of rows displayed in the list to those that correspond to the text entered in the search area. The area indicates the columns where the search/filtering will be carried out. On the Users page, it will be the 4D User, Machine name and Session name columns.

The list is updated in real time as you enter text in the area.

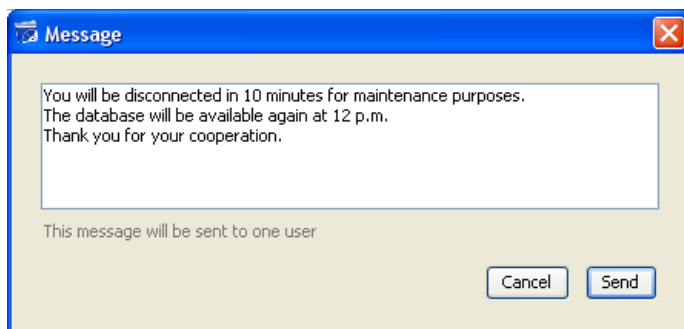
It is possible to enter more than one value to be searched for: separate the values with a semi-colon. The OR type operator is used in this case.

For example, if you enter "John;Mary;Peter," only rows with John OR Mary OR Peter in the target columns will be kept.

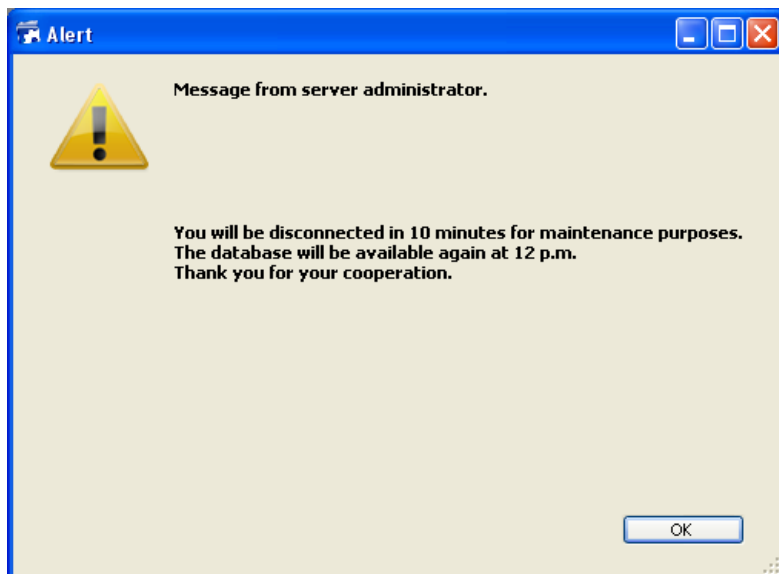
Administration Buttons

This page includes three control buttons. These are active if at least one row is selected. You can select several rows by holding down the Shift key for an adjacent selection or **Ctrl** (Windows) / **Command** (Mac OS) for a non-adjacent selection.

- **Send message:** This button can be used to send a message to the 4D users selected in the window. If no user is selected, the button is not active.
When you click on this button, a dialog box appears that lets you enter the message. The dialog box indicates the number of users that will receive this message:



The message will be displayed as an alert on the client machines:



- **Watch Processes:** This button can be used to directly show the processes of the user(s) selected on the Process page of the administration window. When you click on this button, 4D Server switches to the Processes page and enters the selected user names in the search/filtering area of this page. For more information, please refer to the description of this page.
- **Drop user:** This button can be used to force the selected user(s) to disconnect.
When you click on this button, a warning dialog box appears so that you can confirm or cancel this operation.

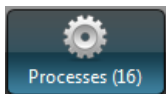
Note: You can also disconnect the selected user(s) directly without displaying the confirmation dialog box: to do so, hold down **Alt** (Windows) or **Option** (Mac OS) while clicking on the **Drop user** button.

Processes Page

The Processes Page lists all the processes underway:

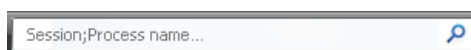
Process name	Session	Type	Num	State	CPU Time	Activity
Client Manager	-	Application server	3	Waiting for event	00:00:04	0 %
DB4D CRON	-	DB4D Server	0	Running	00:00:00	0%
DB4D Flush	-	DB4D Server	0	Running	00:00:01	0%
DB4D Index builder	-	DB4D Server	0	Running	00:00:00	0%
DB4D Server	-	DB4D Server	0	Running	00:00:00	0%
DB4D Sockets	-	DB4D Server	0	Running	00:00:00	0%
Garbage Handler	-	DB4D Server	0	Running	00:00:00	0%
Internal Timer Process	-	Application server	2	Executing	00:00:13	0 %
Task managers	-	SQL Server	0	Running	00:00:00	0%
TCP connection listener	-	TCP Connection listener	0	Running	00:00:00	0%
TCP connection listener	-	SQL Server	0	Running	00:00:00	0%
User Interface	-	Application server	1	Waiting for event	00:02:09	10 %
Application process	esmith	4D Client Process	4	Postponed	00:00:08	0%
Application process	esmith	4D Client Process	5	Running	00:00:13	0%

The "Processes" button indicates, in parentheses, the total number of processes running in the database (this number does not take into account any display filters applied to the window nor the state of the **Display processes by groups** option).

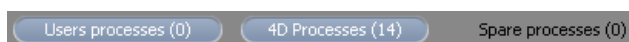


You can change the order of the columns by simply dragging and dropping the column header areas. You can also sort the list of column values by clicking on its header.

Like the **Users Page**, this page contains a dynamic search/filtering area that can be used to reduce the number of rows displayed in the list to those that correspond to the text entered in the search area. The search/filtering is carried out in the Session and Process name columns.



There are also three shortcut buttons that can be used to filter by the type of process displayed in the window:

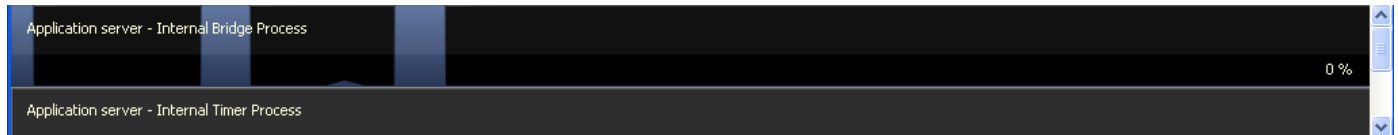


- Users processes: Processes generated by and for the user sessions. These processes are preceded by an icon in the form of a figure.
- 4D Processes: Processes generated by the 4D Server engine. These processes are preceded by an icon in the form of a notched wheel.
- Spare processes: Processes that are inactive but kept temporarily and that can be reused at any time. This mechanism optimizes the reactivity of 4D Server. These processes are preceded by an icon in the form of a dimmed figure.

The **Display processes by groups** option lets you group together the internal processes of 4D Server as well as the client processes, for better readability. When you check this option:

- the "twinned" 4D client processes (main 4D client process and 4D client base process, see the "Process Type" paragraph) are grouped as one,
- a "Task managers" group is created; it includes the internal processes dedicated to dividing up tasks (Shared balancer, Net session manager, Exclusive pool worker),
- a "Client managers" group is created; it includes various client internal processes.

The lower area of the window is used to display the graphic representation of the activity of the selected process(es):


























Note: You can select several rows by holding down the **Shift** key for an adjacent selection or **Ctrl** (Windows) / **Command** (Mac OS) for a non-adjacent selection.

The activity of the process is the percentage of time that 4D Server has devoted to this process (ratio). As in previous versions of 4D Server, the window provides the following information for each process:

- Type of process (see below),
- Session (blank in the case of a 4D process and with the 4D user name in the case of a user process),
- Name of the process,
- Number of the process (as returned by the **New process** function for example). The process number is the number assigned on the server. In the case of a global process, this number may be different from that assigned on the client machine.
- Current state of the process,
- Running time (in seconds) of the process since its creation,
- Percentage of time that 4D Server has devoted to this process (ratio).

Process Type

Each process is identified by an icon as well as a type. The color and form of the icon indicates the type of process:

-  Application server
-  SQL Server
-  DB4D Server (database engine)
-  Web Server
-  SOAP Server
-  Protected 4D client process (development process of a connected 4D)
-  Main 4D client process (main process of a connected 4D). Collaborative process, equivalent on the server of the process created on the client machine)
-  4D client base process (process parallel to a 4D client process. Preemptive process responsible for controlling the corresponding main 4D client process)
-  Spare process (former or future "4D client database process")
-  SQL server worker process
-  HTTP server worker process
-  4D client process (process running on the connected 4D)
-  Stored procedure (process launched by a connected 4D and running on the server)
-  Web method (launched by a 4DACTION for example)
-  Web method (preemptive)
-  SOAP method (launched by a Web Service)
-  Logger
-  TCP connection listener
-  TCP session manager
-  Other process
-  Worker process (cooperative)
-  Stored procedure (preemptive process)
-  Worker process (preemptive)

Note: Each main 4D client process and its "twinned" 4D client base process are grouped together when the **Display processes by groups** option is checked.

Administration Buttons

The page also has five control buttons that act on the selected process(es). Note that only user processes can be acted upon.

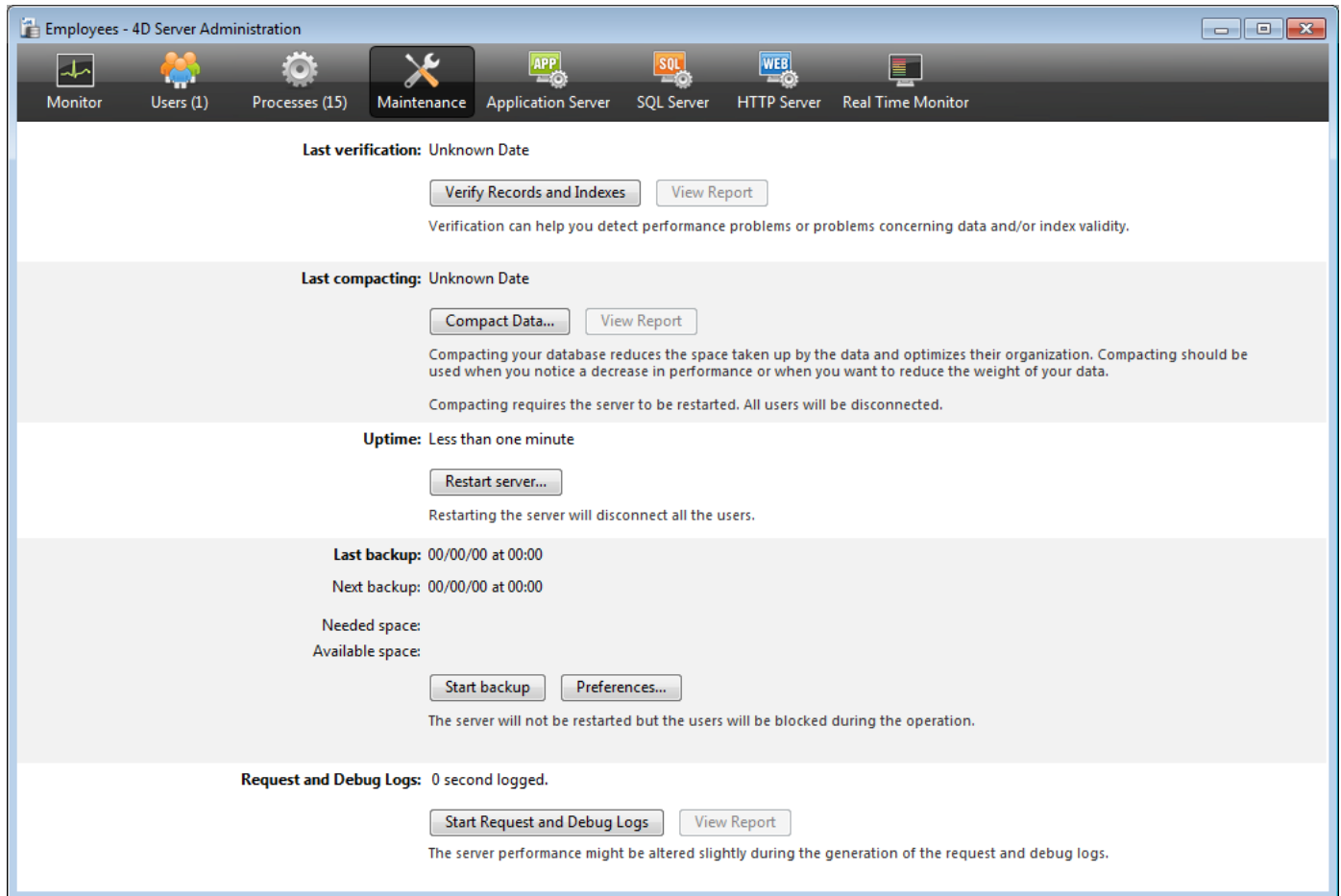


- **Abort Process:** can be used to abort the selected process(es). When you click on this button, a warning dialog box appears so that you can confirm or cancel the operation.
Note: You can also abort the selected process(es) directly without displaying the confirmation dialog box by holding down the **Alt** (Windows) or **Option** (Mac OS) button while clicking on this button.
- **Pause Process:** can be used to pause the selected process(es).
- **Activate Process:** can be used to reactivate the selected process(es). The processes must have been paused previously (using the button above or by programming); otherwise, this button has no effect.
- **Debug Process:** can be used to open on the server machine one or more debugger windows for the selected process(es). When you click on this button, a warning dialog box appears so that you can confirm or cancel the operation.
 Note that the debugger window is only displayed when the 4D code is actually executed on the server machine (for example within the framework of a trigger or the execution of a method having the "Execute on Server" attribute).
Note: You can also debug a process directly without displaying the confirmation dialog box by holding down the **Alt** (Windows) or **Option** (Mac OS) key while clicking on this button.
- **Watch users:** is used to display, on the Users page, all the processes of the selected user(s). This button is

active when at least one user process is selected.

Maintenance Page

The Maintenance Page provides information concerning the current operation of the database. It also provides access to basic maintenance functions:



- **Last verification:** This area indicates the date, time and status of the last data verification carried out on the database. For more information about the data verification procedure, please refer to the Design Reference manual.
The **Verify Records and Indexes** button can be used to launch the verification operation directly, without interrupting the server. Note that the server may be noticeably slowed down during the operation. All the records and all the indexes of the database are verified. If you want to be able to target the verification or have additional options available, you will need to use the Maintenance and Security Center (MSC). After verification, a report file is generated in XML and HTML format on the server in the Logs folder that is placed next to the database structure file. The **View Report** button (named **Download report** if the operation was carried out from a client machine) lets you display the file in your browser.
- **Last compacting:** This area indicates the date, time and status of the last compacting operation carried out on the database data. For more information about the data compacting procedure, please refer to the Design Reference manual.
The **Compact Data...** button can be used to launch a data compacting operation directly. This operation requires stopping the server: when you click on this button, the 4D Server database shutdown dialog box appears so that you can choose how to interrupt the operation:

For more information about this dialog box, please refer to the [Exiting 4D Server](#) section.

After the actual interruption of the database, 4D Server carries out a standard compacting operation on the database data. If you want to have additional options available, you will need to use the Maintenance and Security

Center (MSC).

Once the compacting is finished, 4D Server automatically restarts the database. The 4D users can then be reconnected.

Note: If the request for compacting was carried out from a remote 4D client machine, this machine is automatically reconnected by 4D Server.

A report file is generated in XML and HTML formats on the server in the Logs folder that is placed next to the database structure file. The **View Report** button (named **Download report** if the operation was carried out from a client machine) lets you display the file in your browser.

- **Uptime:** This area indicates the duration of the server operation since the last time it was started (days, hours and minutes).

The **Restart server...** button can be used to immediately restart the server. When you click on this button, the 4D Server database shutdown dialog box appears so that you can choose how to interrupt the operation (see the **Exiting 4D Server** section). After restarting, 4D Server automatically relaunches the database. The 4D users can then be reconnected.

Note: If the request for restarting was carried out from a remote 4D client machine, this machine is automatically reconnected by 4D Server.

- **Last backup:** This area indicates the date and time of the last backup of the database and provides information about the next scheduled automatic backup (if any). Automatic backups are configured using the "Scheduler" page of the database Preferences.
 - Next backup: date and time of next automatic backup.
 - Needed space: estimated space needed for the backup. The actual size of the backup file may vary according to the settings (compression, etc.) and according to variations of the data file.
 - Available space: space available on the backup volume.

The **Start backup** button can be used to backup the database immediately using the current backup parameters (files backed up, location of archives, options, etc.). You can view these parameters by clicking on the **Preferences...** button. During a backup on the server, the client machines are "blocked" (but not disconnected) and it is not possible for any new clients to connect.

- **Request and Debug logs:** This area indicates the duration of recording log requests and debugging events, when they are activated.
 - The request log file stores information concerning the requests received by the server (excluding Web requests): time, process number, user, request size, processing time, etc. that can be used to analyze the server operation. This file is named 4DRequestsLog_X (X being the sequential number of the file) and is stored in the Logs folder of the database. Once the file reaches the size of 10 MB, it is closed and a new file is generated, with an incremented sequential number.
 - The debugging events file stores each execution of a method, 4D command or plug-in command in a file named "4DDebugLog_X.txt", which is automatically placed in the **Logs** subfolder of the database, next to the structure file. Each event is systematically recorded in the file before its execution, which ensures its presence in the file even if the application quits unexpectedly. Note that this file is erased and rewritten each time you launch the application. You can configure this file using the **SET DATABASE PARAMETER** command.

The **Start Request and Debug Logs** button starts the requests log and records the debugging events. Since this may noticeably deteriorate server performance, it is to be reserved for the development phase of the application.

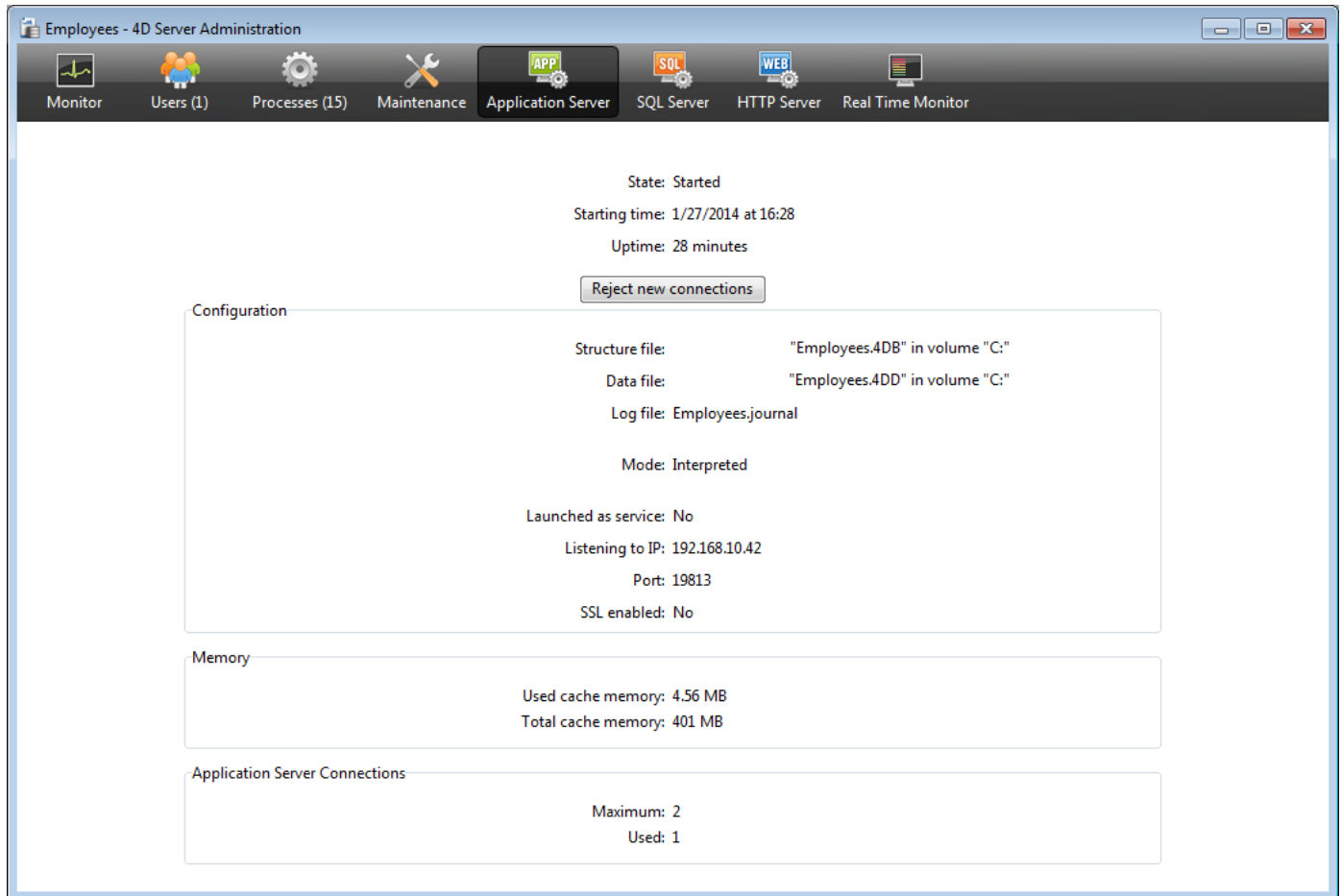
Once the request log has been activated, the button title changes to **Stop Request and Debug Logs**, so that you can stop recording requests at any time. Keep in mind that restarting the log after stopping it "erases" the previous file.

Note: It is possible to start and stop these logs by programming via the **SET DATABASE PARAMETER** command.

The **View Report** button (named **Download report** if the operation was carried out from a client machine) lets you open a system window displaying the request log file.

Application Server Page

The Application Server Page groups together information about the database published by 4D Server and can be used to manage this publication:



Status Information

The upper part of the page provides information about the current status of the 4D Server application server.

- State: Started or Stopped.
- Starting time: Date and time the server database was launched. This date corresponds to the opening of the database by 4D Server.
- Uptime: Time elapsed since last opening of database.

Accept/Reject New Connections Button

This button toggles and can be used to manage the access of new client machines to the application server.

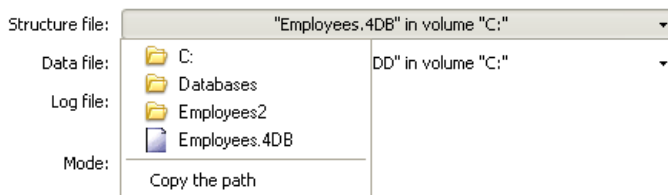
- By default, when the database is published:
 - The button is titled "Reject new connections."
 - New clients can connect freely (within the limit of the connections permitted by the license).
 - The database name is published in the connection dialog box (if the "At Startup Publish Database Name in the Connection Dialog" option is checked in the Preferences).
- If you click on the **Reject new connections** button:
 - The button title changes to "Accept new connections."

- No new client can then connect.
- The database name no longer appears in the connection dialog box.
- Clients that are already connected are not disconnected and can continue to work normally.
- If you click on the **Accept new connections** button, the database returns to its default state.

This function permits, for example, an administrator to carry out various maintenance operations (verification, compacting, etc.) just after having started the server. If the administrator uses a client connection, they can be certain to be the only one modifying the data. It is also possible to use this function in preparation of a maintenance operation which requires that there be no client machine connected.

Configuration

This area provides information about the 4D database published by the server: name and location of data and structure files and name of database log file. You can click on the structure or data file name in order to view its complete pathname:



Launched as service: No

Listening to IP: 192.168.88.108

Port: 19813

SSL enabled: No

The "Mode" field indicates the current execution mode of the database: compiled or interpreted.

The lower part of the area indicates the server configuration parameters (launched as service, port and IP address) and the enabling of SSL for client-server connections (does not concern SQL nor Web connections).

Memory

This area indicates the **Total cache memory** (parameter set in the database Preferences) and the **Used cache memory** (dynamic allocation by 4D Server according to its needs).

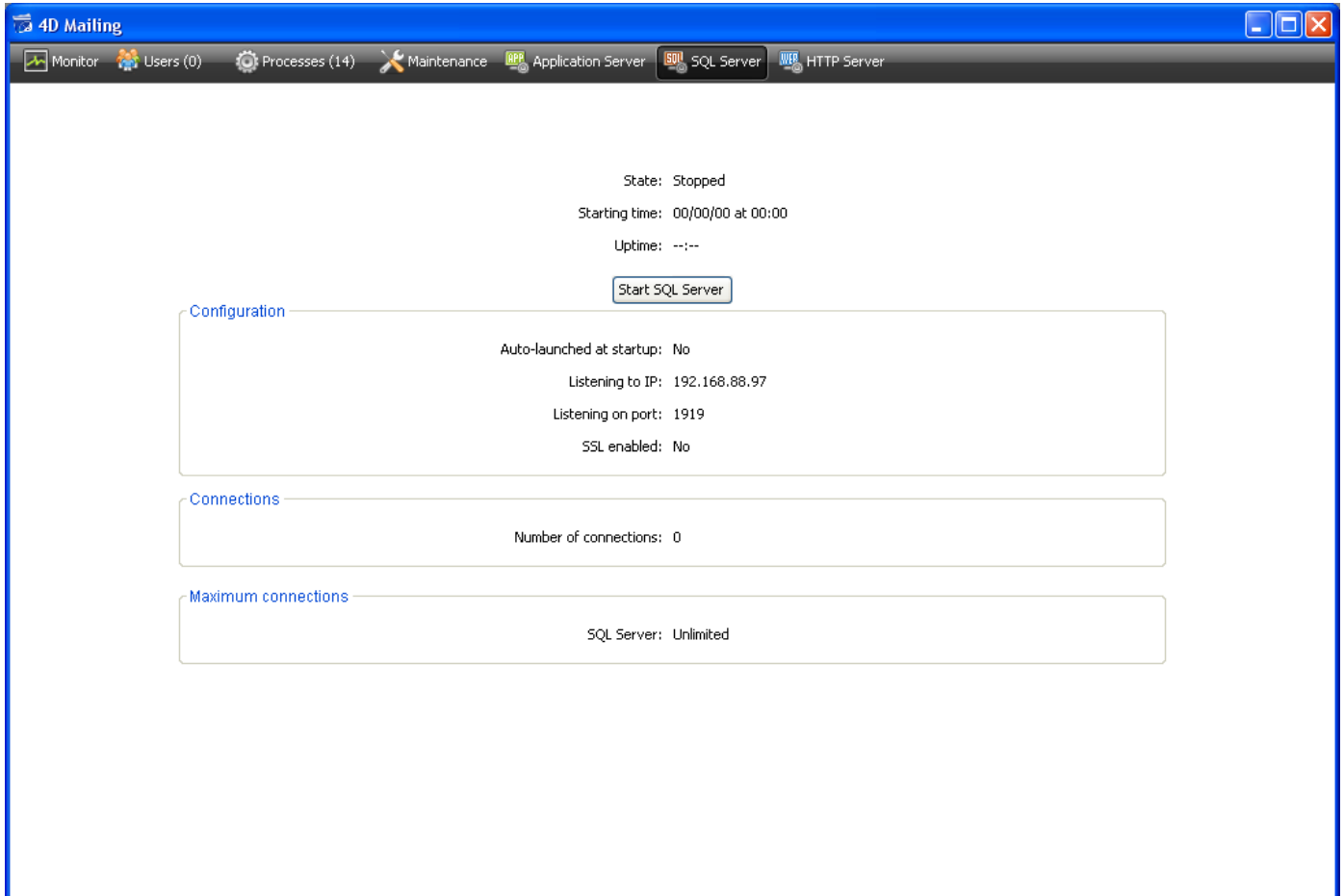
Application Server Connections

"Maximum:" indicates maximum number of simultaneous client connections allowed for the application server. This value depends on the license installed on the server machine.

"Used:" indicates actual number of connections currently being used.

SQL Server Page

The SQL Server Page groups together information about the integrated SQL server of 4D Server. It also includes a button that can be used to control the activation of the server:



Status Information

The upper part of the page provides information about the current status of the SQL server of 4D Server.

- State: Started or Stopped
- Starting time: Date and time the SQL server was last launched. This value may differ from that of the application server when the launching of the SQL server is not carried out "on startup."
- Uptime: Time elapsed since last startup of the SQL server.

Start / Stop SQL Server button

This button toggles and can be used to control the activation of the 4D Server SQL server.

- When the SQL server state is "Started," the button is titled **Stop SQL Server**. If you click on this button, the 4D Server SQL server is immediately stopped; it no longer replies to any external SQL queries received on the designated TCP port.
- When the SQL server state is "Stopped," the button is titled **Start SQL Server**. If you click on this button, the 4D Server SQL server is immediately started; it replies to any external SQL queries received on the designated TCP port. Note that you will need a suitable license to be able to use the 4D SQL server.

Note: The SQL server can also be launched automatically on application startup (option in the Preferences) or by

programming.

Configuration

This area provides information about the SQL server configuration parameters: automatic launching on startup, listening IP address, TCP port (19812 by default) and enabling of SSL for SQL connections (does not concern 4D nor Web connections).

These parameters can be modified via the 4D Preferences.

Connections

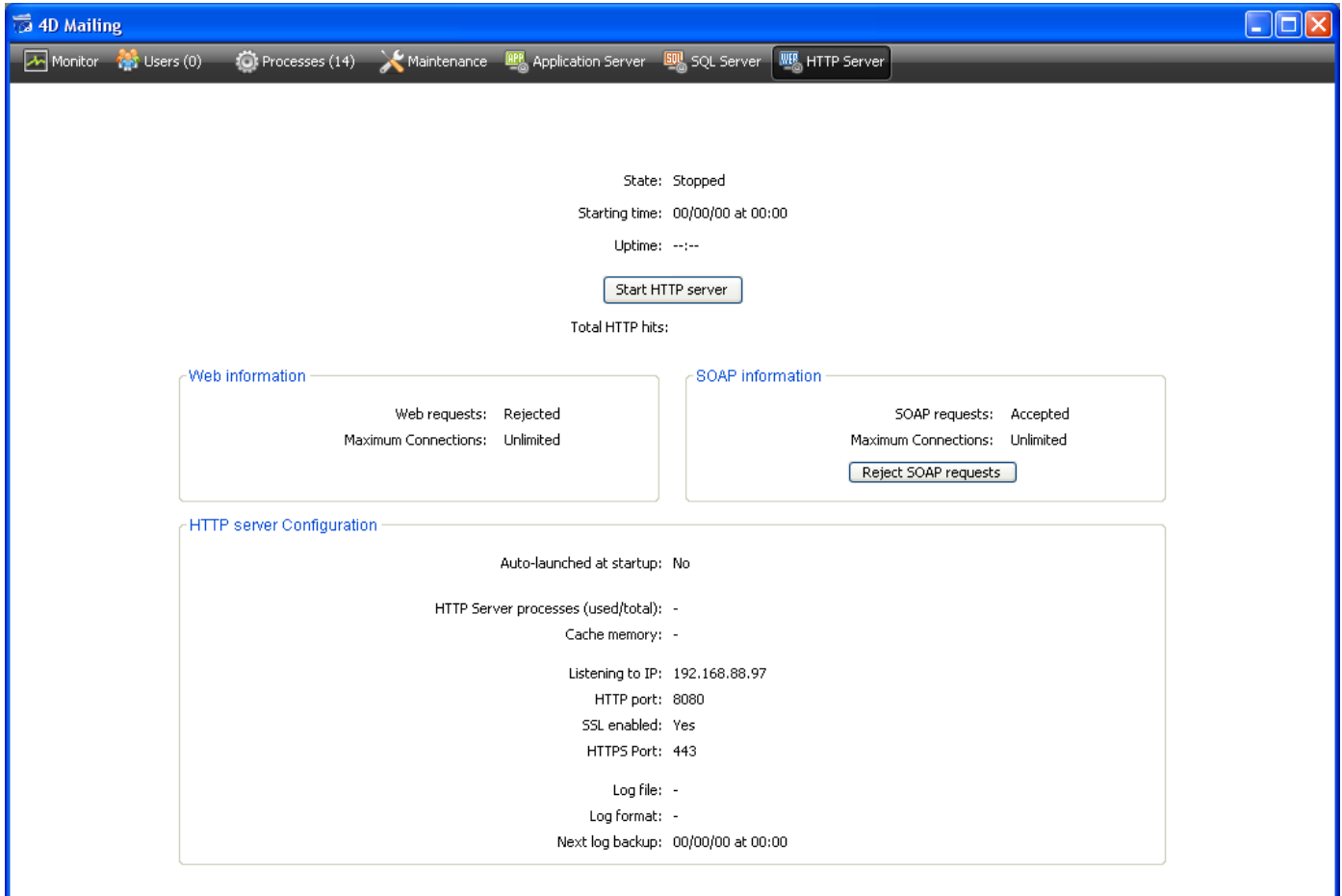
Number of SQL connections currently open on 4D Server.

Maximum Connections

Maximum number of simultaneous SQL connections allowed. This value depends on the license installed on the server machine.

HTTP Server Page

The HTTP Server Page groups together information about the operation of the Web server and SOAP server of 4D Server. The Web server lets you publish Web content such as HTML pages or pictures bound for Web browsers. The SOAP server manages the publication of Web Services. These two servers rely on the internal HTTP server of 4D Server. This page also includes buttons that can be used to control the activation of these servers:



Status Information

The upper part of the page provides information about the current status of the HTTP server of 4D Server.

- State: Started or Stopped éarré ou Arrêté
- Starting time: Date and time the HTTP server was last launched. This value may differ from that of the application server when the launching of the HTTP server is not carried out "on startup."
- Uptime: Time elapsed since last startup of the HTTP server.
- Total HTTP hits: Number of (low level) HTTP hits received by the HTTP server since it was started.

Start / Stop HTTP Server button

This button toggles and can be used to control the activation of the 4D Server HTTP server.

- When the HTTP server state is "Started," the button is titled **Stop HTTP Server**. If you click on this button, the 4D Server HTTP server is immediately stopped; the Web server and SOAP server no longer accept any requests.
- When the HTTP server state is "Stopped," the button is titled **Start HTTP Server**. If you click on this button, the 4D Server HTTP server is immediately started; Web and SOAP requests are accepted (note that it is

possible to stop the SOAP server separately, see the "SOAP Information" paragraph below).

Notes:

- You must have a suitable license in order to be able to start the HTTP server.
- The HTTP server can also be launched automatically on application startup (option in the Preferences) or by programming.

Web Information

This area provides specific information about the Web server of 4D Server.

- **Web requests: Accepted or Rejected.** This information indicates whether the Web server is activated. Since the Web server is directly linked to the HTTP server, Web requests are accepted when the HTTP server is started and rejected when it is stopped.
- **Maximum connections:** Maximum number of Web connections allowed. This value depends on the license installed on the server machine.

SOAP Information

This area provides specific information about the SOAP server of 4D Server and includes a control button.

- **SOAP requests: Accepted or Rejected.** This information indicates whether the SOAP server is activated. In order for SOAP requests to be accepted, the HTTP server must be started and the SOAP server must explicitly accept the requests (see the Accept/Reject button).
- **Maximum connections:** Maximum number of SOAP connections allowed. This value depends on the license installed on the server machine.
- **Accept/Reject SOAP requests** button: This button toggles and can be used to control the activation of the 4D Server SOAP server. This button modifies the value of the Allow SOAP Requests option on the "Web Services/SOAP" page of the Preferences (and vice versa).
If you click on the **Accept SOAP requests** button and the HTTP server is stopped, 4D automatically starts it.

HTTP Server Configuration

This area provides information about the configuration parameters and operation of the HTTP server:

- **Auto-launched at startup:** parameter set via the 4D Server Preferences.
- **HTTP Server processes (used/total):** number of HTTP processes created on the server (current number of processes / total of all processes created).
- **Cache memory (used/total):** size of HTTP server cache memory, when it is activated (size actually used by cache / maximum size theoretically allocated to the cache in the Preferences).
- **Listening to IP, TCP Port (80 by default), SSL enabled for HTTP connections (does not concern 4D nor SQL connections) and HTTPS Port used:** current configuration parameters of HTTP server, specified on the Web/Configuration page of the Preferences (see the **Web Server Settings** section in the 4D Language Reference manual).
- **Log file information:** location, format and date of the next automatic log backup of the HTTP server (logweb.txt file).

Real Time Monitor Page

The Real Time Monitor page monitors the progress of "long" operations performed by the application in real time. These operations are, for example, sequential queries, execution of formulas, etc.:

Start Time	Duration (ms)	Information
2014-06-03 10:09:26:562	75 829	Sequential searching on Table_1: 4398275 of 24728607 records
2014-06-03 10:10:27:910	14 481	Deleting records: 41998 of 24728607

Details

Created on client

Operation Details

Operation Type: Delete Records
Table: Table_1

Process Details

Client Process Num: 6
Process Name: P_2
4D User: Super_Utilisateur
Session Name: Arnaud Schmitt
Machine Name: MACWIN7-SCHMITT

PAUSED

Snapshot Resume

This page is available in the administration window of the server machine and also from a remote 4D machine. In the case of a remote machine, this page displays data from operations performed on the server machine.

A line is added for each long operation performed on the data. This line automatically disappears when the operation is complete (you can check the **Display operations at least 5 seconds** option to keep quick operations on screen for 5 seconds, see below).

The following information is provided for each line:

- **Start Time:** starting time of operation in the format: "dd/mm/yyyy - hh:mm:ss"
- **Duration (ms):** duration in milliseconds of operation in progress
- **Information:** title of operation.
- **Details:** this area displays detailed information which will vary according to the type of operation selected.





More specifically:

- **Created on:** indicates whether the operation results from a client action (Created on client) or if it was started explicitly on the server by means of a stored procedure or the "Execute on server" option (Created on server).
- **Operation Details:** Operation type and (for query operations) query plan.
- **Sub-operations** (if any): Dependent operations of the selected operation (e.g. deleting related records before a parent record).
- **Process Details:** Additional information concerning the table, field, process or client, depending on the type of operation

Note: Real-time monitoring page uses the **GET ACTIVITY SNAPSHOT** command internally. For more information, please refer to this command description.

The page is active and updated permanently as soon as it is displayed. It should be noted that its operation can significantly slow the execution of the application. It is possible to suspend the updating of this page in one of the following ways:

4D Server Database Methods

-  On Server Startup Database Method
-  On Server Shutdown Database Method
-  On Server Open Connection Database Method
-  On Server Close Connection Database Method

On Server Startup Database Method

On Server Startup Database Method

Does not require any parameters

The **On Server Startup Database Method** is called once on the server machine when you open a database with 4D Server. The **On Server Startup Database Method** is NOT invoked by any 4D environment other than 4D Server.

The **On Server Startup Database Method** is the perfect place to:

- Initialize interprocess variables that you will use during the whole 4D Server session.
- Start **Stored Procedures** automatically when a database is opened.
- Load Preferences or Settings saved during the previous 4D Server session.
- Prevent the opening of the database if a condition is not met (i.e., missing system resources) by explicitly calling **QUIT 4D**.
- Perform any other actions that you want performed automatically each time a database is opened.

To automatically execute code on a client machine when a remote 4D connects to the server, use the **On Startup database method**.

Note: The **On Server Startup Database Method** is executed automatically, which means that no remote 4D can connect until the method has finished executing.

On Server Shutdown Database Method

On Server Shutdown Database Method

Does not require any parameters

The **On Server Shutdown Database Method** is called once on the server machine when the current database is closed on 4D Server. The **On Server Shutdown Database Method** is NOT invoked by any 4D environment other than 4D Server.

To close the current database on the server, you can select the **Close Database...** menu command on the server. You can also choose the **Quit** command or call the **QUIT 4D** command within a stored procedure executed on the server.

When the exit from the database is initiated, 4D performs the following actions:

- If there is no **On Server Shutdown Database Method**, 4D Server aborts each running process one by one, without distinction.
- If there is an **On Server Shutdown Database Method**, 4D Server starts executing this method within a newly created local process. You can therefore use this database method to inform other processes, via interprocess communication, that they must stop executing. Note that 4D Server will eventually quit — the **On Server Shutdown Database Method** can perform all the cleanup or closing operations you want, but it cannot refuse the quit, and will at some point end.

The **On Server Shutdown Database Method** is the perfect place to:

- Stop store procedures automatically started when the database was opened.
- Save (locally, on disk) Preferences or Settings to be reused at the beginning of the next session in the **On Server Startup Database Method**.
- Perform any other actions that you want to be done automatically each time a database is exited.

Warning: If you use the **On Server Shutdown Database Method** to close stored procedures, keep in mind that the server quits once the **On Server Shutdown Database Method** (and not the stored procedures) is executed. If some stored procedures are still running at this point, they will be killed.

Consequently, if you want to make sure that the stored procedures are fully executed before being killed by the server, the **On Server Shutdown Database Method** should indicate to the stored procedures that they must end their execution (for example, using an interprocess variable) and should allow them to close (through a x seconds loop or another interprocess variable).

If you want code to be executed automatically on a client machine when a remote 4D stops connecting to the server, use the **On Exit database method**.

🔧 On Server Open Connection Database Method

\$1, \$2, \$3 -> On Server Open Connection Database Method -> \$0

Parameter	Type	Description
\$1	Longint	← User ID number used internally by 4D Server to identify users
\$2	Longint	← Connection ID number used internally by 4D Server to identify a connection
\$3	Longint	← Obsolete: Always returns 0 (but must be declared)
\$0	Longint	→ 0 or omitted = connection accepted; other value = connection refused

When is the On Server Open Connection Database Method Called?

The **On Server Open Connection Database Method** is called once on the Server machine each time a connection process is started by a 4D remote workstation. The **On Server Open Connection Database Method** is NOT invoked by any 4D environment other than 4D Server.

The **On Server Open Connection Database Method** is called each time:

- a remote 4D connects (because the Application process starts)
- a remote 4D opens the Design environment (because the Design process starts)
- a remote 4D starts a global process (whose name does not begin with "\$") which requires the creation of a cooperative process on the server (*). This process can be created using the **New process** command, a menu command or using the Execute Method dialog box.

In each case with a remote 4D, several processes are started—One on the client machine and one or two others (as needed) on the server machine. On the client machine, the process executes code and send requests to 4D Server. On the server machine, the **4D Client Process** (preemptive process) maintains the database environment for the client process (i.e., current selections and locking of records for user processes) and replies to requests sent by the process running on the client machine. The **4D Client Database process** (cooperative process) is in charge of monitoring the corresponding 4D Client process.

(*) Beginning with 4D v13, for optimization purposes, the server processes (a preemptive process for access to the database engine and a cooperative process for access to the language) are only created when necessary when executing client-side code. For example, here are the details of a 4D code sequence running in a new client process:

```
// global process begins without a new process on the server, like a local process.  
CREATE RECORD ([Table_1])  
[Table_1]field1_1:="Hello world"  
SAVE RECORD ([Table_1]) // creation here of preemptive process on server  
$serverTime:=Current time(*) // creation here of cooperative process on server  
// call to On Server Open Connection
```

Important: Web connections and SQL connections do not invoke the **On Server Open Connection Database Method**. When a Web browser connects to 4D Server, the **On Web Authentication Database Method** (if any) and/or the **On Web Connection database method** are invoked. When 4D Server receives an SQL query, the **On SQL Authentication database method** (if one exists) is called. For more information, see the description of this database method in the 4D Language Reference manual.

Important: When a Stored Procedure is started, the **On Server Open Connection Database Method** is NOT invoked. **Stored Procedures** are server processes, not 4D Client processes. They execute code on the Server machine, but do not reply to requests exchanged by a 4D client (or other clients) and 4D Server.

How is the On Server Open Connection Database Method Called?

The **On Server Open Connection Database Method** is executed on the 4D Server machine within the 4D Client process that provoked the call to the method.

For example, if a remote 4D connects to a 4D Server interpreted database, the user process, the Design process

and the client registration process (by default) for that client are started. The **On Server Open Connection Database Method** is therefore executed three times in a row—the first time within the Application process, the second time within the client registration process, and the third time within the Design process. If the three processes are respectively the sixth, seventh and eighth process to be started on the Server machine, and if you call **Current process** from within the **On Server Open Connection Database Method**, the first time **Current process** returns 6, the second time 7 and the third time 8.

Note that **On Server Open Connection Database Method** executes on the Server machine. It executes within the 4D Client process running on the Server machine, independent of the process running on the client side. In addition, at the moment when the method is invoked, the 4D Client process has not yet been named (**PROCESS PROPERTIES** will not at this point return the name of the 4D Client process).

The **On Server Open Connection Database Method** has no access to the process variable table of the process running on the Client side. This table resides on the Client machine, not on the Server machine.

When the **On Server Open Connection Database Method** accesses a process variable, it works with a private and dynamically created process variable table for the 4D Client process.

4D Server passes three Long Integer parameters to the **On Server Open Connection Database Method** and expects a Long Integer result. The method must therefore be explicitly declared with three Long Integer parameters as well as a Long Integer function result:

```
C_LONGINT ($0; $1; $2; $3)
```

If you do not return a value in $\$0$, thereby leaving the variable undefined or initialized to zero, 4D Server assumes that the database method accepts the connection. If you do not accept the connection, you return a non-null value in $\$0$.

This table details the information provided by the three parameters passed to the database method:

Parameter	Description
\$1	User ID number used internally by 4D Server to identify users
\$2	Connection ID number used internally by 4D Server to identify a connection
\$3	Obsolete: Always returns 0 but must be declared

These ID numbers are not directly usable as sources of information to be passed as, for example, parameters to a 4D command. However, they provide a way to uniquely identify a 4D Client process between the **On Server Open Connection Database Method** and the **On Server Close Connection database method**. At any moment of a 4D Server session, the combination of these values is unique. By storing this information in an interprocess array or a table, the two database methods can exchange information. In the example at the end of this section, the two database methods use this information to store the date and time of the beginning and end of a connection in the same record of a table.

Example 1

The following example shows how to maintain a log of the connections to the database using the **On Server Open Connection Database Method** and the **On Server Close Connection Database Method**. The *[Server Log]* table (shown below) is used to keep track of the connection processes:

Log ID	2 ³²
Log Date	17
Log Time	🕒
Exit Date	17
Exit Time	🕒
User ID	2 ³²
Connection ID	2 ³²
Process ID	2 ³²
Process Name	A

The information stored in this table is managed by the **On Server Open Connection Database Method** and the **On Server Close Connection Database Method** listed here:

```

` On Server Open Connection Database Method
C_LONGINT ($0; $1; $2; $3)
` Create a [Server Log] record
CREATE RECORD ([Server Log])

```

```

[Server Log]Log ID:=Sequence number([Server Log])
  \ Save the Log Date and Time
[Server Log]Log Date:=Current date
[Server Log]Log Time:=Current time
  \ Save the connection information
[Server Log]User ID:=$1
[Server Log]Connection ID:=$2
SAVE RECORD([Server Log])
  \ Returns no error so that the connection can continue
$0:=0

  \ On Server Close Connection Database Method
C_LONGINT($1;$2;$3)
  \ Retrieve the [Server Log] record
QUERY([Server Log];[Server Log]User ID=$1;*)
QUERY([Server Log]; & ;[Server Log]Connection ID=$2;*)
QUERY([Server Log]; & ;[Server Log]Process ID=0)
  \ Save the Exit date and time
[Server Log]Exit Date:=Current date
[Server Log]Exit Time:=Current time
  \ Save the process information
[Server Log]Process ID:=Current process
PROCESS PROPERTIES([Server Log]Process ID;$vsProcName;$vlProcState;$vlProcTime)
[Server Log]Process Name:=$vsProcName
SAVE RECORD([Server Log])

```

Here are some entries in the [Server Log] showing several remote connections:

Log ID	Log Date	Log Time	Exit Date	Exit Time	User ID	Connection ID	Process ID	Process Name
36	06/16/2008	18:30:29	06/16/2008	18:46:21	12274272	252872560	7	Design process
37	07/01/2008	16:26:05	07/01/2008	16:27:43	74850768	122753776	6	Application process
38	07/01/2008	16:26:10	07/01/2008	16:27:42	74850768	120343104	7	Design process
39	07/01/2008	16:29:50	07/01/2008	16:33:37	72755568	119129408	6	Application process
40	07/01/2008	16:30:10	07/01/2008	16:33:36	72755568	118734956	7	Design process
41	07/01/2008	16:32:26	07/01/2008	16:33:18	11944049	119324128	8	Application process
42	07/01/2008	16:32:43	07/01/2008	16:33:17	11944049	119718480	9	Design process
43	07/01/2008	16:32:49	07/01/2008	16:32:49	72755568	119888104	10	P_1
44	07/01/2008	16:32:51	07/01/2008	16:32:51	72755568	119888104	10	P_2
45	07/01/2008	16:33:16	07/01/2008	16:33:16	72755568	119709704	10	P_3
46	07/01/2008	16:33:17	07/01/2008	16:33:17	72755568	72783988	5	P_4
47	07/01/2008	16:33:18	07/01/2008	16:33:18	72755568	119846936	5	P_5
48	07/01/2008	16:33:30	07/01/2008	16:33:30	20728779	119324128	5	Application process
49	07/01/2008	16:33:53	07/01/2008	16:36:57	72755568	118541512	5	Application process
50	07/01/2008	16:33:58	07/01/2008	16:36:56	72755568	118581968	6	Design process
51	07/01/2008	16:35:42	07/01/2008	16:36:57	72755936	119755744	7	Application process
52	07/01/2008	16:35:45	07/01/2008	16:36:50	72755936	120055408	8	Design process
53	07/01/2008	16:36:37	07/01/2008	16:36:56	72755936	120144432	9	P_1
54	07/01/2008	16:37:07	00/00/00	00/00/00	72755568	118541512	0	
55	07/01/2008	16:37:11	00/00/00	00/00/00	72755568	119285016	0	

Example 2

The following example prevents any new connection from 2 to 4 A.M.

```

  \ On Server Open Connection Database Method
C_LONGINT($0;$1;$2;$3)

If((?02:00:00?<=Current time)&(Current time<?04:00:00?))
  $0:=22000
Else
  $0:=0
End if

```

⚙️ On Server Close Connection Database Method

\$1, \$2, \$3 -> On Server Close Connection Database Method

Parameter	Type	Description
\$1	Longint	← User ID number used internally by 4D Server to identify users
\$2	Longint	← Connection ID number used internally by 4D Server to identify a connection
\$3	Longint	← Obsolete: Always returns 0 but must be declared

Description

The **On Server Close Connection Database Method** is called once on the Server machine each time a 4D Client process ends.

As for the **On Server Open Connection database method**, 4D Server passes three Long Integer parameters to the **On Server Close Connection Database Method**. On the other hand, no result is expected by 4D Server.

The method must therefore be explicitly declared with three Long Integer parameters:

```
C_LONGINT ($1; $2; $3)
```

This table details the information provided by the three parameters passed to the database method:




Parameter	Description
\$1	User ID number used internally by 4D Server to identify users
\$2	Connection ID number used internally by 4D Server to identify a connection
\$3	Obsolete: Always returns 0 but must be declared

The **On Server Close Connection Database Method** is the exact counterpoint to the **On Server Open Connection database method**. For more information and a description of the **4D Client processes**, see the description of this database method.

Example

See the first example for **On Server Open Connection database method**.

Using a remote 4D

-  Connecting to a 4D Server Database
-  Administration from Remote Machines
-  Compilation from Remote Machines

Connecting to a 4D Server Database

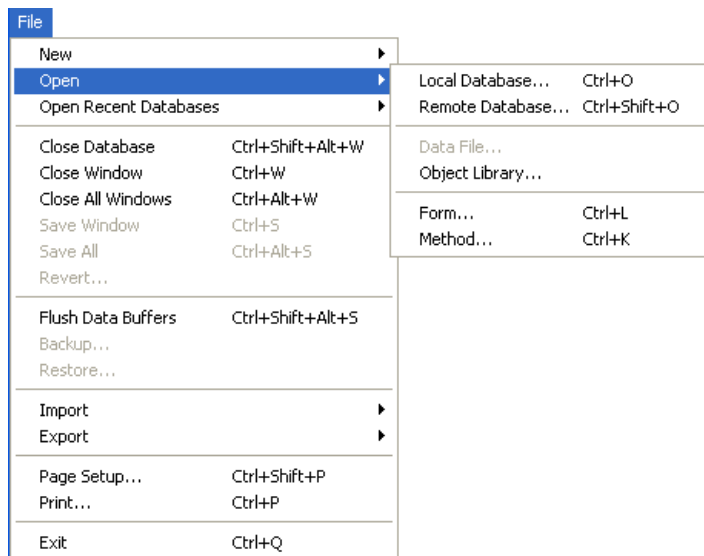
There are three ways to connect to a 4D Server database via a remote 4D:

- Using the connection dialog box
- Using the **Open Recent Databases** menu
- Using a 4DLink shortcut file containing the access parameters to the database.

Using the connection dialog box

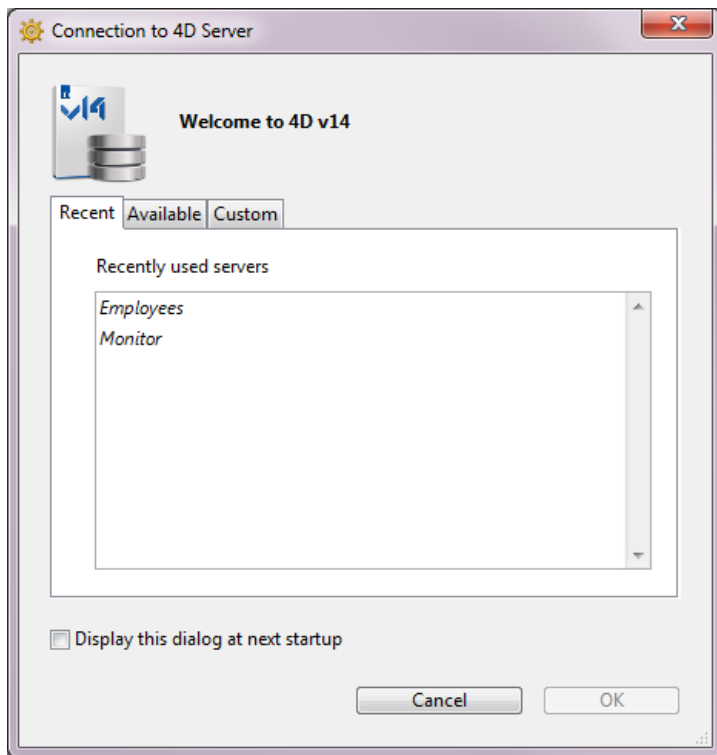
To display the 4D Server connection dialog box, first launch the 4D application.

The **Open** command of the **File** menu (or the corresponding button in the 4D tool bar) can be used to select the opening mode for the 4D database:



Choose the **Open>Remote Database...** command

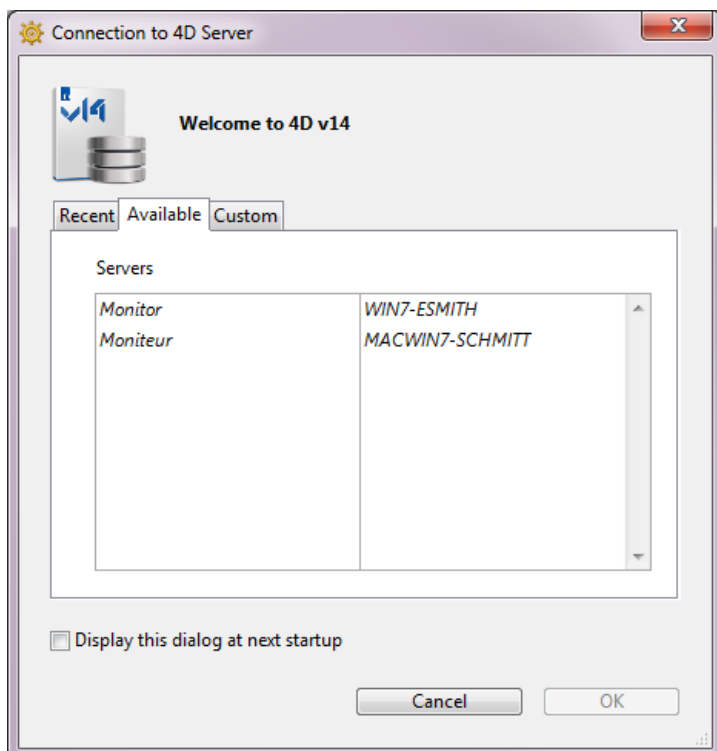
The 4D Server connection dialog box appears. This dialog box has three pages which can be accessed via the following tabs: Recent, Available and Custom:



If you check the **Display this dialog at next startup** option, this dialog box will automatically be displayed on startup of the 4D application.

Note: You can also display this dialog box by checking on the **Connect to 4D Server** link in the 4D Welcome dialog box.

“Available” tab



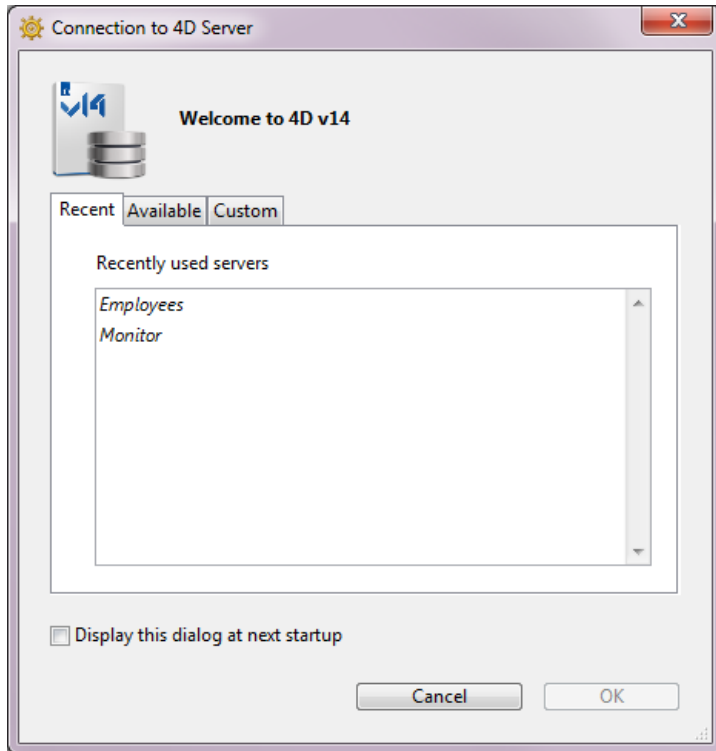
4D Server includes a built-in TCP/IP broadcasting system that publishes by default the name of the 4D Server databases available over the network. These names are listed on the **TCP/IP** Page of the connection dialog box. The list is sorted by order of appearance and updated dynamically. To connect to a server from this list, double-click on its name or select it and click the **OK** button.

Notes:

- A circumflex accent (^) is placed before the name of databases published with the encryption option. For more information, refer to section **Encrypting Client/Server Connections**.
- It is possible to prohibit dynamic publication of the database name on the network (see the **IP Configuration**

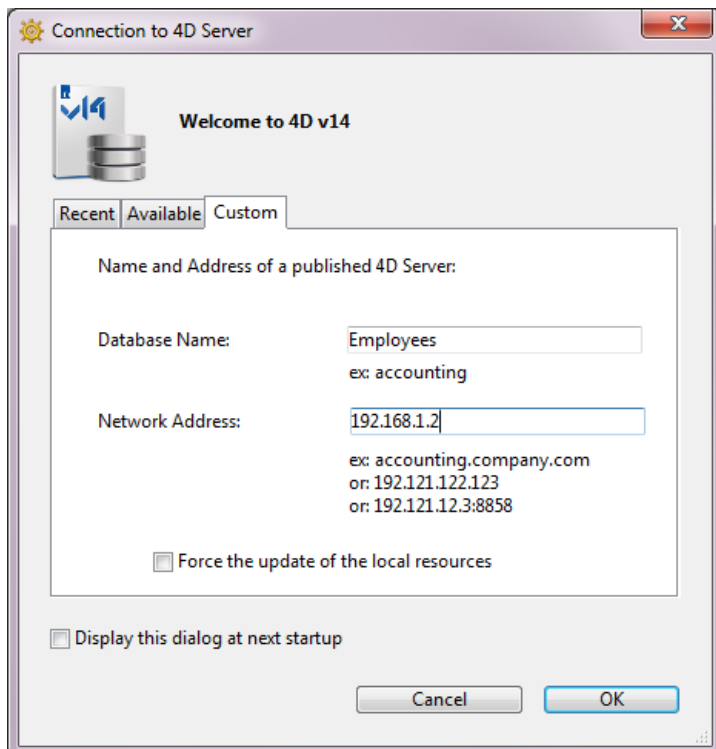
section). In this case, the connection can only be carried out manually on the "Custom" page.

"Recent" tab



The **Recent** page memorizes the list of all 4D servers recently used. The list is sorted by alphabetical order. To connect to a server from this list, double-click on its name or select it and click the **OK** button.

"Custom" tab



The **Custom** page allows assigning a published server on the network using its IP address and attributing it a customized name.

You can customize the 4D Server TCP/IP broadcasting system so that the names of server databases are not automatically published over the network (see the **IP Configuration** section). In this case, this name does not appear on the "Available" page. However, if you know the IP address of a server database whose name is not broadcast, you can manually enter its IP address.

- **Database name:** allows defining the name of the 4D Server database. This name will be used in the **Recent**

page when referring to the database.

- **Network address:** allows entering the IP address of the machine where the 4D Server was launched. If two servers are executed simultaneously on the same machine, the IP address must be followed a colon and port number, for example: 192.168.92.104:19814.
By default, the publishing port of a 4D Server is 19813. This number can be modified in the Database Settings (see **Network and Client-Server options** section).

Note: If a database was selected in the **Recent** or **Available** pages at the moment that you clicked on the **Custom** tab, the two fields display the corresponding information.

Once this page assigns a server, click the **OK** button will allow you to connect to the server. The server will then be listed in the **Recent** page.

Note: If the database is published using the encryption option, you must add a circumflex accent (^) before the name; otherwise the connection will be refused. For more information, refer to section **Encrypting Client/Server Connections**.

Force the update of the local resources

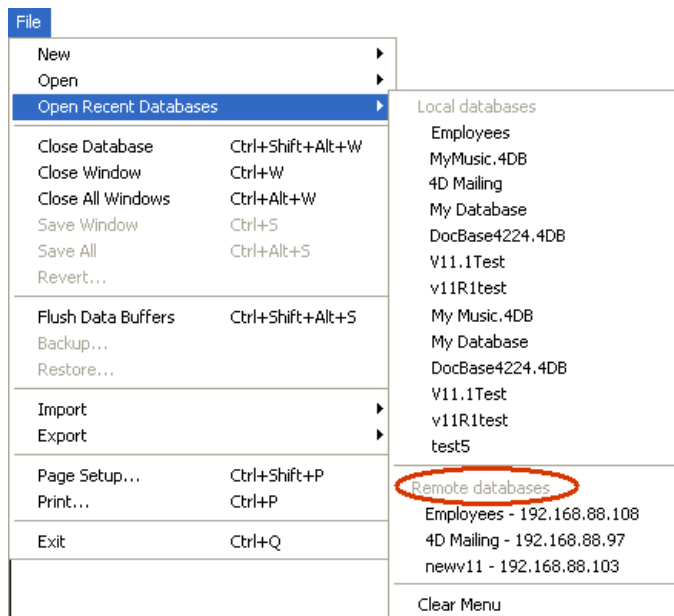
This option causes systematic updating of the local resources on the client machine when it connects. The local resources are the structural information related to the database that are stored on each client machine.

As a rule, updating of the local resources is automatic on the remote machine each time it connects, when the structure of the database has been modified between two connections. Most of the time, this option is unnecessary. Nevertheless, in certain specific cases, it may be necessary to force the update.

Using the Open Recent Databases menu

The **Open Recent Databases** menu command can be used to connect directly to a 4D Server database to which you have already connected previously.

This command is found in the 4D **File** menu. If you use the 4D application to open local databases and to connect to remote databasess, this menu will list both types of databases. The remote databases are placed at the bottom of the menu:



The IP address of the server is indicated next to the database name.

The **Clear Menu** command can be used to reset the menu.

Using a 4DLink file

You can generate database access files containing parameters intended to automate and simplify opening or connecting to 4D databases. Usually, an access file can save the address of a 4D Server remote database as well as its connection identifiers, thus eliminating several operations for the user.

Access files can also be used for opening local databases.

Creation of Files

The access files of 4D databases are XML files that have the ".4DLink" extension. 4D generates and uses this type of file to build the "recent databases" submenu: a .4DLink file is automatically generated by 4D when a local database is opened for the first time or when connecting to a server for the first time.

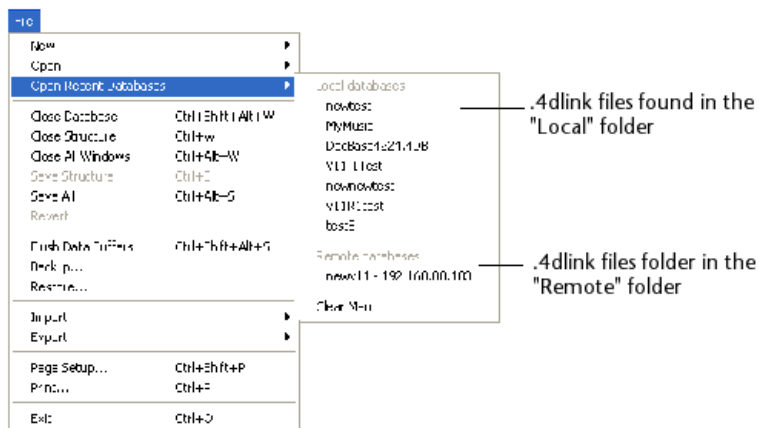
The .4DLink files that are created automatically by 4D are placed in the local preferences folder of the user. In this folder, two directories are created: Local and Remote. The Local folder contains the ".4DLink" files that can be used to connect to local databases and the Remote folder contains the ".4DLink" files that can be used to connect to remote databases.

Local preferences folders are found at the following locations:

- Windows 7 and higher: C:\Users\UserName\AppData\Roaming\4D\Favorites vXX\
- OS X: Users/UserName/Library/Application Support/4D/Favorites vXX/

... where XX represents the version number of the application (for example, "Favorites v14" for 4D v14).

The files found in these directories are displayed by 4D in the **Open Recent Databases** submenu of the **File** menu:



The ".4DLink" files can also be created with an XML editor and contain customized information such as the connection identifiers (user name and password) or the database opening mode.

4D provides a DTD describing the XML keys that can be used to build a ".4DLink" file. This DTD is named *database_link.dtd* and is found in the `Resources\DTD` subfolder of the 4D application.

Using Files

A .4DLink access file can be used to launch the 4D application and open the target 4D database. There are two different ways to use it:

- Via a double-click or drag and drop onto the 4D application,
- Via the **Open Recent Databases** submenu (file located in the local preferences folder).
A .4DLink file of the "remote database" type can be copied and used on several different machine.

Note: It is also possible to select a 4DLink file in the 4D and 4D Server opening dialog box (opening local databases only).

Administration from Remote Machines

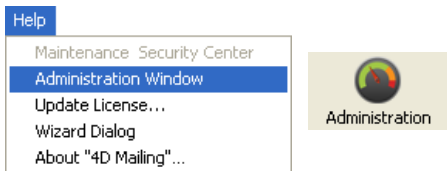
You can administer the 4D Server machine from a remote 4D (client machine) by opening the 4D Server administration window (see the [Monitor Page](#) section) on the client machine.

Opening the administration window on a remote 4D machine

To open a server administration window from a client machine, you must be connected to the remote database as a Designer or Administrator. Otherwise, when you attempt to open the administration window, a privilege error (-9991) is generated.

This window can be accessed in one of two manners:

- Choose the **Administration Window** command from the **Help** menu or click on the corresponding button in the 4D tool bar:



- Execute the **OPEN ADMINISTRATION WINDOW** command.

A server administration window then appears on the client machine:



Specificities of administration via a client machine

A client machine displaying the server administration window has access to all the available information and can act upon the processes and the starting/stopping of servers. When the server administration window is displayed on a client machine, there are nevertheless certain restrictions and specific features concerning its operation:

- On the **Process page**, it is not possible to debug a user process (since the debug window appears on the server machine).
- On the **Maintenance Page**, it is possible to execute actions that cause all the clients to be disconnected and the server to be restarted (compacting and restarting operations). In this case, the client machine requesting the operation is automatically reconnected on restarting.
- On the **Maintenance Page**, the **View Report** buttons are renamed **Download Report** after the execution of a maintenance operation. These files are downloaded into the database folder on the client machine before being displayed.

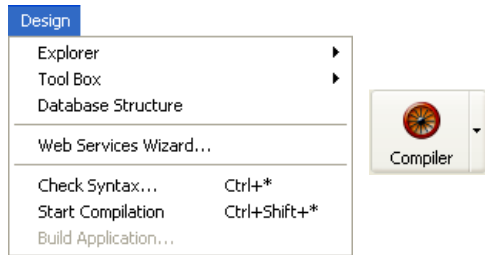
Compilation from Remote Machines

It is possible to compile a 4D application from a remote connection. In other words, it is possible to compile from a 4D client machine. In versions of 4D Server prior to v11 SQL, compilation could only be carried out from the single-user application.

Note: It is not possible to build a custom 4D application (single-user or client/server) from a remote connection. The Application builder is not accessible in this environment (the menu command is dimmed).

On the client side, the compilation interface and principles are the same as those of single-user versions.

Compilation can be triggered either from the **Design** menu or the tool bar, or from the compiler window itself:



Note: The "4D Team Server" license is required on the 4D Server side in order for the client machines to be able to access the compilation function.

Only one client machine can compile a database at a given time. Compilation by a client locks the function for other remote machines. If another client machine attempts to compile the database at the same time, a warning message appears.

While one client machine is carrying out a compilation, the other client machines can continue to work and to modify methods or any other structural element. The compiled code and the interpreted code will then be different, which means that it will be necessary to recompile the database subsequently.








The compiled code is sent in the .4DB file on the server gradually as compilation is carried out.

On the client side, after completion of the operation, it is possible to restart the server in interpreted mode or in compiled mode using the corresponding commands in the **Run** menu. When a 4D remote machine requests the restarting of the server in compiled/interpreted, the standard server shutdown dialog box appears so that a waiting time can be set or a warning message sent to the other clients (see the **Exiting 4D Server** section). When the server has been restarted, the client at the origin of the restarting operation is automatically connected again.

On the server side, restarting in compiled/interpreted requires the use of the standard open file dialog box (pop-up menu associated with the **Open** button)

Note: Compilation on the WAN network is not recommended for performance reasons (more particularly in the case of databases including numerous methods) because the operation generates a large quantity of network exchanges.

4D Server and the 4D Language

-  4D Server and the 4D Language
-  4D Server, Sets and Named Selections
-  Stored Procedures
-  Stored procedures on client machines
-  SP-Based Import (Example)
-  SP-Based Services (Example)
-  Execute on Server attribute

With 4D Server, you now have three situations in which you can execute 4D code on the server machine:

- Triggers
- Stored procedures
- Project methods with the "Execute on Server" attribute
- Database methods

Triggers

A Trigger is a method attached to a table. Triggers can prevent "illegal" operations on your database records. They are a very powerful tool to restrict operations on a table, as well as to prevent accidental data loss or tampering. For example, in an invoicing system, you can prevent anyone from adding an invoice without specifying the customer to whom the invoice is billed.

Triggers are executed on the machine where the database engine is actually located.

With 4D Server, triggers are executed within the context of processes running on the server machine, and not on the client machine. More specifically, they are executed in the context of the "twinned" processes of the user processes that call the database operation. These twinned processes share the database context with the user process on the client machine (in particular, the state of transactions and the locking of records) but do not share the language context (variables, processes, sets, current selections). Note however that the current record of the table of the trigger is the same in all contexts.

For more information about triggers, see the section [ARRAY REAL](#) of the 4D Language Reference manual.

Stored Procedures

A 4D stored procedure is project method executing a process method in a process running on the server machine (or on any client machine), instead of on the client machine which has launched the method. See the [Stored Procedures](#) section.

Methods with the "Execute on Server" attribute

Project methods that have the "Execute on Server" attribute are also executed on the server. However, unlike stored procedures, they are executed in the "twinned" process of the client process and thus benefit from its database context, like with triggers. For more information, please refer to the [Execute on Server attribute](#) section.

Database Methods

Four database methods are executed only on the server machine:

- [On Server Startup Database Method](#)
- [DISTINCT ATTRIBUTE VALUES On Server Shutdown Database Method](#)
- [On Server Open Connection Database Method](#)
- [On Server Close Connection Database Method](#)

Five other database methods can be executed either on the server machine or a client machine depending on the context:

- [On Web Authentication Database Method](#)
- [On Web Connection Database Method](#)

- **On SQL Authentication Database Method**
- **On Backup Startup Database Method**
- **On Backup Shutdown Database Method**

Three database methods can only be executed on a client machine:

- **On Startup database method**
- **On Exit database method**
- **On Drop Database Method**

For more information on the database methods see the corresponding sections in this manual and in the 4D Language Reference manual

4D Server and Variables

- 4D Server maintains one table of interprocess variables. The scope of these variables is the server machine. When running a compiled database, the interprocess variable table definition is common between the server and all the clients machines, each machine having its own instance.
- Like all processes, each stored procedure, database method and trigger has its own table of process variables. These process variables can be created and used dynamically during each phase of execution.

4D Server, Sets and Named Selections

With 4D Server, the visibility of sets and named selections depends on where they were created (server process or client process) and the type of these objects (local, process or interprocess objects). For more information, see the **4D Server, Sets and Named Selections** section.

4D Server, Sets and Named Selections

As explained in the **Sets** and **Named Selections** chapters of the 4D Language Reference manual, you can work with interprocess, process, and local sets and named selections:

- **Process sets/named selections:** A process object can only be accessed by the process in which it has been created and, if it has been created in a client process, by the "twinned" process created on the server. Process sets are cleared as soon as the process method ends. Process objects do not need any special prefix in the name.
- **Interprocess sets/named selections:** An interprocess object is visible for all the processes on the machine (client or server) where it was created. A set or named selection is an interprocess object if the name of the set is preceded by the symbols (<>) — a "less than" sign followed by a "greater than" sign.
Note: This syntax can be used on both Windows and Macintosh. Also, on Macintosh only, you can use the diamond symbol (Option-Shift-V on a US keyboard).
- **Local/Client sets/named selections:** A local/client object is only visible in the process where it was created. The name of a local/client object is preceded by the dollar sign (\$).
Note: Although its name does not begin with a \$, the *UserSet* system set is a local/client set.

The following table indicates the principles concerning the visibility of named selections and sets according to where they are created (the table is identical for both types of objects):

	Client Process	Other client processes	Other clients	Server process	Other server processes
Creation in a client process					
\$test	x				
test	x			x (Trigger)	
<>test	x	x			
Creation in a server process					
\$test				x	
test				x	
<>test				x	x

x = visible

You need to keep this visibility matrix in mind depending on the operations you want to perform. For example, if you want to do a **DIFFERENCE**, **INTERSECTION** or **UNION** type operation, make sure that all the sets are visible on the machine that is carrying out the operation.

For optimization purposes, it is recommended to choose the creation location and the scope of the objects according to their visibility requirements.

What is an SQL-based Stored Procedure?

The expression Stored Procedure comes from the SQL-based Server world. When a client workstation sends a request to an SQL-based server, it actually sends a plain text request in SQL language to the SQL-based server. This request is then parsed and interpreted on the SQL-based server before being executed. Obviously, if the source code of the request is huge and if the request is sent multiple times during a session, there is a great deal of time spent in sending the source code over the network, parsing and interpreting the request as many times as the request is sent.

So, the idea was to find a way to send that request over the network, parse and interpret it once, and then execute it only each time it was received from a client workstation. The solution was to keep the request source code (in other words, a procedure) on the server side and have the client workstation send a request consisting only of the name of the procedure to be executed. The procedure is consequently said to be “stored” on the server, thus the term “stored procedure.”

Note that an SQL-based stored procedure is a procedure that can receive parameters from a client workstation, execute the tasks it implements (synchronously or asynchronously) and eventually return a result to the client workstation. When a client workstation invokes the execution of a stored procedure, to a certain extent, it delegates code execution on the server machine.

What is a 4D Server Stored Procedure?

Although we use the industry name, the capabilities of 4D Server stored procedures significantly exceed the regular concept of stored procedures.

With 4D in local mode, when you use a command, such as **New process**, you can start a user process in which you can run a method. This method is called a **process method** (see the **Project Methods** section in the 4D Language Reference manual).

You can do the same with 4D Server, on a client machine. In addition, using the command **Execute on server** on the server machine, you can start a user process in which you can run a method. Moreover, when using the **EXECUTE ON CLIENT**, you can run a method in another process on a different client.

In both cases, the method is called a **stored procedure**, and (with an abuse of language) the process started on the server machine or another client is also called a **stored procedure**.

Important: The essential difference between an SQL-based stored procedure and a 4D Server stored procedure is that in the first case you execute an SQL procedure, in the second case, you run a stand-alone 4D process.

Architecture of 4D Stored Procedures

Like a regular process, a stored procedure has its own environment:

- **Current selection per table:** Each stored procedure has a separate current selection. One table can have a different current selection in different stored procedures.
- **Current record per table:** Each table can have a different current record in each stored procedure.
- **Variables:** Every stored procedure has its own process variables. Process variables are recognized only within the domain of their native stored procedure.
- **Default table:** Each stored procedure has its own default table.
- **Process sets:** Each stored procedure has its own process sets.
- **On Error Call:** Each stored procedure has its own error-handling method.
- **Debugger window:** Each stored procedure can have its own Debugger window.

In terms of user interface, a stored procedure can open windows and display data (i.e., **DISPLAY RECORD**).

A stored procedure executed on a 4D client machine allow data entry.

On the other hand a stored procedure executed on the server cannot invoke data entry (i.e., **ADD RECORD**); there

is no data entry kernel on the server machine.

You can start as many as stored procedures as the system authorizes (hardware and memory). In fact, the 4D Server machine should be viewed as a machine that not only replies to 4D Client and Web browsers, but also one that executes processes that interact with other processes running on the server machine and on remote 4D machines.

In the same way that 4D provides a multi-tasking environment to user processes running on the machine, 4D Server provides a multi-tasking environment to stored procedures. For example, 4D Server maintains a table of interprocess variables that can be used by the stored procedures for interprocess communications.

Note: The "Execute on Server" method property can also be used to execute a method in a process on the server, but the method uses a "twinned" process of the client process in this case, which means more particularly that it can take advantage of the environment of this client process. In this case, it is not a 4D stored procedure. For more information, please refer to the [Execute on Server attribute](#) section.

What a Stored Procedure Does?

Aside from data entry for stored procedures executed on the server, almost everything said in the 4D Language Reference manual about the capabilities of processes and commands applies to stored procedures.

A stored procedure can add, query, order by, update or delete records. A stored procedure can use sets and named selections, access documents on disk, work with BLOBs, print records and so on. Just think that instead of doing something on a local 4D machine, you are doing it on the server machine or on one or several 4D client machines.

One obvious advantage of stored procedures executed on the server is that indeed a stored procedure executes locally on the server machine, the machine where the database engine is located. For example, an [APPLY TO SELECTION](#) is not efficient over the network, but it is from within a stored procedure. The example proposed in the section [SP-Based Import \(Example\)](#) shows the magnitude of performance optimization you can achieve with "smart" stored procedure implementation.

Stored procedures executed on one or several client machines allows to optimize the task repartition and the communication between several client machines. Refer to the command [REGISTER CLIENT](#) in the Language Reference manual for an example of a stored procedures executed on several clients.

However, the most important advantage of the stored procedure architecture is the additional dimension it gives to 4D Server. Using stored procedures, you can implement your own custom 4D Server services. The only limit is your imagination. The example in the section [SP-Based Services \(Example\)](#) shows a stored procedure that provides clients with information about 4D Server or the server machine. You can, for example, list the volumes of the server machine. This example could be expanded easily for returning directory or document information to a client.

What a stored procedure does not do (executed on the server)?

Generally speaking, stored procedures executed on the server should not deal with interface items (such as menus, windows, forms...). Indeed the interface is not managed on the server's side.

Commands displaying dialog boxes on the server machine as well as dialog boxes dealing with data entry should be avoided.

Here is the list of the commands that should NOT be used within stored procedures executed on the server. These commands are organized within three groups:

- **Forbidden commands on the server**

If one of the following commands is used within a stored procedure, an alert will be displayed indicating that this command cannot be executed on 4D Server. The error #67 is returned; it can be intercepted through a method installed in the [ON ERR CALL](#) command.

[ACCUMULATE](#)

[ADD RECORD](#)

[_o_ADD SUBRECORD](#)

[APPEND MENU ITEM](#)

[BREAK LEVEL](#)

CALL PROCESS
CHANGE LICENSES
Count menu items
Count menus
CREATE USER FORM
DELETE MENU ITEM
DELETE USER FORM
DISABLE MENU ITEM
DISPLAY SELECTION
EDIT ACCESS
EDIT FORM
ENABLE MENU ITEM
FILTER EVENT
Get menu item
Get menu item key
Get menu item mark
Get menu item style
Get menu title
SET PICTURE TO LIBRARY
_o_GRAPH TABLE
INSERT MENU ITEM
Level
LIST USER FORMS
Menu selected
MODIFY RECORD
MODIFY SELECTION
_o_MODIFY SUBRECORD
ON EVENT CALL
_o_Open external window
PAGE BREAK
PAGE SETUP
PRINT SETTINGS
QUERY BY EXAMPLE
QR REPORT
Printing page
REMOVE PICTURE FROM LIBRARY
SET MENU ITEM
SET MENU ITEM SHORTCUT
SET MENU ITEM MARK
SET MENU ITEM STYLE
SET PICTURE TO LIBRARY
SHOW MENU BAR
Subtotal

• **Inappropriate commands on server**

We strongly advise you not to use the following commands in stored procedures because they are not suitable for the server executing method. They can block the server and create errors, and in any case they do not execute properly. No specific error code is returned.

ACCEPT
Activated
_o_ADD DATA SEGMENT
After
APPEND DATA TO PASTEBOARD
APPEND TO LIST
Before
BLOB TO DOCUMENT
BLOB to list

BRING TO FRONT
_o_C_GRAPH
CANCEL
CHANGE CURRENT USER
CHANGE PASSWORD
CLEAR LIST
CLEAR PASTEBOARD
Copy list
Count list items
Count screens
Create document(1)
_o_Create resource file(1)
Current form table
Current user
Deactivated
DELETE FROM LIST
DELETE USER
DIALOG
_o_DISABLE BUTTON
DRAG AND DROP PROPERTIES
DRAG WINDOW
Drop position
_o_During
_o_ENABLE BUTTON
ERASE WINDOW
EXPORT DATA(1)
FILTER KEYSTROKE
Find window
Focus object
FONT LIST
_o_Font name
_o_Font number
Form event
FORM FIRST PAGE
FORM Get current page
FORM GET PROPERTIES
FORM GOTO PAGE
FORM LAST PAGE
FORM NEXT PAGE
FORM PREVIOUS PAGE
FORM SET INPUT
FORM SET OUTPUT
Frontmost process
Frontmost window
Get edited text
GET GROUP LIST
GET GROUP PROPERTIES
GET HIGHLIGHT
GET LIST ITEM
GET LIST ITEM PROPERTIES
GET LIST PROPERTIES
GET MOUSE
GET PASTEBOARD DATA
GET PICTURE FROM PASTEBOARD
Get text from pasteboard
GET USER LIST

GET USER PROPERTIES
GET WINDOW RECT
Get window title
GOTO OBJECT
GRAPH SETTINGS
HIDE PROCESS
HIDE TOOL BAR
HIDE WINDOW
HIGHLIGHT RECORDS
HIGHLIGHT TEXT
IMPORT DATA(1)
In break
In footer
In header
INSERT IN LIST
_o_INVERT BACKGROUND
Is a list
Is user deleted
Keystroke
List item parent
List item position
LIST TO BLOB
Load list
MAXIMIZE WINDOW
Menu bar height
Menu bar screen
MINIMIZE WINDOW
Modified
New list
Next window
OBJECT GET COORDINATES
OBJECT MOVE
OBJECT SET LIST BY NAME
OBJECT SET COLOR
OBJECT SET ENTERABLE
OBJECT SET FILTER
OBJECT SET FORMAT
OBJECT SET RGB COLORS
OBJECT SET TITLE
OBJECT SET VISIBLE
Old
Open document(1)
Open resource file(1)
ORDER BY(2)
Outside call
Pasteboard data size
Pop up menu
POST CLICK
POST EVENT
POST KEY
QUERY BY FORMULA(2)
QUERY(2)
REDRAW
_o_REDRAW LIST
REDRAW WINDOW
REGISTER CLIENT

REJECT
SAVE LIST
SCREEN COORDINATES
SCREEN DEPTH
Screen height
Screen width
Select folder
SELECT LIST ITEMS BY POSITION
SELECT LIST ITEMS BY REFERENCE
SELECT LOG FILE
Selected list items
Self
SET CURSOR
SET FIELD TITLES
Set group properties
SET LIST ITEM
SET LIST ITEM PROPERTIES
SET LIST PROPERTIES
SET PICTURE TO PASTEBOARD
SET SCREEN DEPTH
SET TABLE TITLES
SET TEXT TO PASTEBOARD
SET TIMER
Set user properties
SET WINDOW RECT
Shift down
SHOW PROCESS
SHOW WINDOW
SORT LIST
User in group
Validate password
Window kind
WINDOW LIST
Window process

(1) Only when the first parameter is an empty string.

(2) Only when the syntax results in displaying a dialog box (i.e.: **ORDER BY**([Table])).

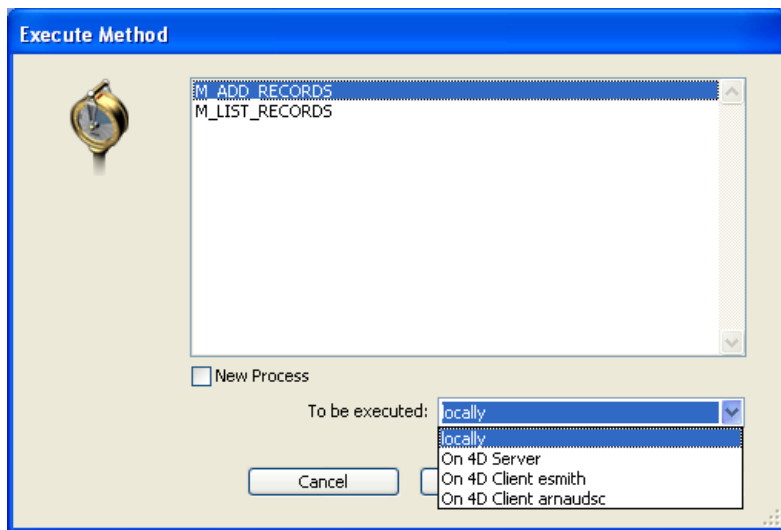
- **Commands with no effect on the server**

The following commands have no effect when they are executed within a stored procedure on the server. No specific error code is returned.

GRAPH
MESSAGES OFF
MESSAGES ON
SET MENU BAR
SHOW TOOL BAR

How to Start a Stored Procedure

- From 4D, you can manually start a stored procedure in the Execute Method dialog box:



You can execute it on 4D Server or on another 4D client machine. Please note that to display the 4D client machines in this list, they should have been first registered (see the **Stored procedures on client machines** section and the **REGISTER CLIENT** command).

- Also on 4D, you can programmatically start a stored procedure using the commands **Execute on server** or **EXECUTE ON CLIENT**.

Note: It is not possible to use the process management commands **DELAY PROCESS**, **PAUSE PROCESS** and **RESUME PROCESS** from a remote 4D with stored procedures on the server.

- A method executed on 4D Server (server database method, method with the Execute on Server attribute or stored procedure) can start a stored procedure using **Execute on server**, **New process** or **EXECUTE ON CLIENT**.

More About Interprocess Communication Between Stored Procedures and User Processes

Stored procedures can communicate between themselves using:

- interprocess variables
- local or global semaphores
- records
- interprocess sets and interprocess named selections
- the commands **GET PROCESS VARIABLE**, **SET PROCESS VARIABLE** and **VARIABLE TO VARIABLE**.

Refer to the corresponding parts of the 4D Language Reference manual. Once again, keep in mind that the 4D commands act within the scope of the server machine which is executing the stored procedure (server or clients) in the same way as they act in the scope of a client machine.

Note: The **CALL PROCESS** and **Outside call** mechanism has no meaning on the server machine, because stored procedures do not have a user interface with data entry.

There is yet another important feature: client user processes (processes running on a client machine) can read and write the process variables (*) of a stored procedure, using the commands **GET PROCESS VARIABLE**, **SET PROCESS VARIABLE** and **VARIABLE TO VARIABLE**.

(*) as well as the server machine interprocess variable.

Important: "Intermachine" process communication, provided by the commands **GET PROCESS VARIABLE**, **SET PROCESS VARIABLE** and **VARIABLE TO VARIABLE**, is possible from client to server only. It is always a client process that reads or write the variables of a stored procedure.

Stored procedures on client machines

Stored procedures can be executed on one or several 4D client machines. Stored procedures on client machines are executed the same as way as stored procedures on the server, except that on the client they can invoke data entry. Refer to the [Stored Procedures](#) section for further information.

Any client machine executing stored procedures triggered by a server or another client machine, should explicitly be registered for this session. There are two methods to register a client: it can automatically be registered when connecting or through programming.

Registering automatically each 4D client machine connecting to 4D Server

The "Register Clients at Startup For Execute On Client" check box is available in the Database Settings, on the "Network options" tab of the "Client-Server" page:

□

When this option is checked, each 4D client machine connecting to the database is automatically referenced with 4D Server as being able to execute stored procedures. A 4D Client type process named according to the client machine is created on the server. An equivalent process is also created on each client machine.

Registering 4D Client through programming

It is possible to register one or several client machines using programming. It allows you to select the client machines that needs to be registered and to define their registration name.

The "Process" theme contains the **REGISTER CLIENT** command which allows you to register a client machine under any name.

Unregistering 4D Client

No matter how the client machines have been registered, you can unregister them for the current session using the **UNREGISTER CLIENT** command ("Process" theme) for a given client. The registration process (named according to the client) disappears from the user process group on the server machine as well as on the client.

Note: You can get the list and the task distribution (number of methods still to be executed) for the clients registered for a given session using the **GET REGISTERED CLIENTS** command.

For further information on these commands, refer to the 4D Language Reference manual.

SP-Based Import (Example)

The following example shows how importing data can be dramatically accelerated in Client/Server architecture. The **Regular Import** method allows you to test how long it takes to import records using the **IMPORT TEXT** command on the Client side:

```
\ Regular Import Project Method
$vhDocRef:=Open document("")
If(OK=1)
  CLOSE DOCUMENT($vhDocRef)
  INPUT FORM([Table1];"Import")
  $vhStartTime:=Current time
  IMPORT TEXT([Table1];Document)
  $vhEndTime:=Current time
  ALERT("It took "+String(0+($vhEndTime-$vhStartTime))+ " seconds.")
End if
```

With the regular import data, 4D parses the text file, then for each record, creates a new record, fills out the fields with the imported data and sends the record to the Server machine to be added to the database. There are consequently many requests going over the network. A way to optimize the operation is to use a stored procedure to do the job locally on the Server machine. The client machine loads the document into a BLOB, then starts a stored procedure that passes the BLOB as parameter. The stored procedure stores the BLOB in a document on the server machine disk, then imports the document locally. The import of the data is therefore performed locally (at a speed comparable to that of a local version of 4D) because most the network requests have been eliminated.

Here is the **CLIENT IMPORT** project method. Executed on the Client machine, it calls the **SERVER IMPORT** stored procedure listed just below:

```
\ CLIENT IMPORT Project Method
\ CLIENT IMPORT ( Pointer ; Text)
\ CLIENT IMPORT ( -> [Table] ; Input form )

C_POINTER($1)
C_TEXT($2)
C_TIME($vhDocRef)
C_BLOB($vxData)
C_LONGINT(spErrCode)

\ Select the document do be imported
$vhDocRef:=Open document("")
If(OK=1)
  \ If a document was selected, do not keep it open
  CLOSE DOCUMENT($vhDocRef)
  $vhStartTime:=Current time
  \ Try to load it in memory
  DOCUMENT TO BLOB(Document;$vxData)
  If(OK=1)
  \ If the document could be loaded in the BLOB,
  \ Start the stored procedure that will import the data on the server machine
  $spProcessID:=Execute on server("SERVER IMPORT";32*1024;
  "Server Import Services";Table($1);$2;$vxData)
  \ At this point, we no longer need the BLOB in this process
  CLEAR VARIABLE($vxData)
  \ Wait for the completion of the operation performed by the stored procedure
  Repeat
    DELAY PROCESS(Current process;300)
    GET PROCESS VARIABLE($spProcessID;spErrCode;spErrCode)
    If(Undefined(spErrCode))
  \ Note: if the stored procedure has not initialized its own instance
```

```

` of the variable spErrCode, we may be returned an undefined variable
    spErrCode:=1
    End if
Until(spErrCode<=0)
` Tell the stored procedure that we acknowledge
    spErrCode:=1
    SET PROCESS VARIABLE($spProcessID;spErrCode;spErrCode)
    $vhEndTime:=Current time
    ALERT("It took "+String(0+($vhEndTime-$vhStartTime))+ " seconds.")
Else
    ALERT("There is not enough memory to load the document.")
End if
End if

```

Here is the **SERVER IMPORT** project method executed as a stored procedure:

```

` SERVER IMPORT Project Method
` SERVER IMPORT ( Long ; Text; BLOB )
` SERVER IMPORT ( Table Number ; Input form ; Import Data )

C_LONGINT($1)
C_TEXT($2)
C_BLOB($3)
C_LONGINT(spErrCode)

` Operation is not finished yet, set spErrCode to 1
spErrCode:=1
$vpTable:=Table($1)
INPUT FORM($vpTable->;$2)
$vsDocName:="Import File "+String(1+Random)
If(On Windows)
    $vsDocName:=$vsDocName+".txt" ` On Windows, file extension is mandatory
End if
DELETE DOCUMENT($vsDocName)
BLOB TO DOCUMENT($vsDocName;$3)
IMPORT TEXT($vpTable->;$vsDocName)
DELETE DOCUMENT($vsDocName)
` Operation is finished, set spErrCode to 0
spErrCode:=0
` Wait until the requester Client got the result back
Repeat
    DELAY PROCESS(Current process;1)
Until(spErrCode>0)

```

Note: The **On Windows** project method is listed in the **System Documents** section in the 4D Language Reference manual.

After these two project methods have been implemented in a database, you call perform a "Stored Procedure-based" import data by writing, for example:

```

CLIENT IMPORT(->[Table1];"Import")

```

With some benchmarks, you will discover that by using this method you can import records up to 60 times faster than with a regular import.

SP-Based Services (Example)

In the example discussed in the section **SP-Based Import (Example)**, a stored procedure is started and ended each time an import data operation is requested. In this example, a stored procedure is started automatically when the server database starts up, and can be ended and restarted at will by any 4D connected to the database. As soon as it runs, the stored procedure can reply asynchronously to multiple requests sent by the clients connected to the database.

While the **SP-Based Import (Example)** section shows how to implement a drastically optimized existing service provided by 4D Server, this example shows how to implement new and custom services available to all connected 4D client machines. In addition, this example can be used as a template for implementing your own services.

Automatic Start-up of the Stored Procedure

The stored procedure is automatically started by the **On Server Startup Database Method**:

```
On Server Startup Database Method
START SP SERVICES
```

Because the **On Server Startup Database Method** starts the **SP SERVICES** project method as a stored procedure, **SP SERVICES** starts running as soon as the database is opened with 4D Server, whether or not clients are actually connected to the server database. In the following figure, the 4D Server administration window shows the stored procedure running when no client is yet connected.

Process name	Session	Type	Num	State	CPU Time	Activity
DB4D Flush		DB4D Server	0	Running	00:00:00	0.00 %
DB4D Index builder		DB4D Server	0	Running	00:00:00	0.00 %
DB4D Server		SQL Server	0	Running	00:00:00	0.00 %
EXCLUSIVE pool worker 0		SQL Server	0	Running	00:00:00	0.00 %
EXCLUSIVE pool worker 1		SQL Server	0	Running	00:00:00	0.00 %
Shared Load Balancer		SQL Server	0	Running	00:00:00	0.00 %
SHARED pool worker 0		SQL Server	0	Running	00:00:00	0.00 %
SHARED pool worker 1		SQL Server	0	Running	00:00:00	0.00 %
SQL Net Session Manager Thread		SQL Server	0	Running	00:00:00	0.00 %
Client Manager	-	Application server	3	Waiting for flag	00:00:03	0.00 %
Internal Bridge Process	-	Application server	4	Waiting for flag	00:00:01	0.00 %
Internal Timer Process	-	Application server	2	Executing	00:00:01	0.92 %
User Interface	-	Application server	1	Waiting for event	00:00:11	1.85 %

Starting and Ending the Stored Procedure At Will

The **START SP SERVICES** project method is listed here:

```
START SP SERVICES Project Method
@vlSPServices:=Execute on server("SP SERVICES";32*1024;"SP SERVICES";*)
```

Since the **Execute on server** command acts like **New process** when called on the server machine, the same method (**START SP SERVICES**) can be used on the server machine or on any client machine to start, at will, the method **SP SERVICES** as a stored procedure on the server machine.

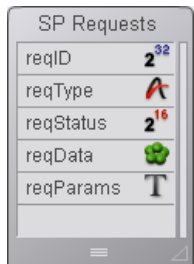
The **STOP SP SERVICES** project method "tells" the **SP SERVICES** project method to stop.

```
STOP SP SERVICES Project Method
SET PROCESS VARIABLE(@vlSPServices;vbStopSPServices;True)
```

When the **SP SERVICES** project method starts, it sets the *vbStopSPServices* process variable to False and then loops until this Boolean variable becomes True. The command **SET PROCESS VARIABLE**, enables any user process running on the server or any client machines to change the value of the *vbStopSPServices* variable, and consequently stop the stored procedure at will.

Communicating with the Stored Procedure

The stored procedure should be able to receive and reply asynchronously to client requests at any time and in any order. A straightforward way to insure this communication is to use a table.



SP Requests	
reqID	2 ³²
reqType	A
reqStatus	2 ¹⁶
reqData	🌿
reqParams	T

The *[SP Requests]* table contains the following fields:

- *[SP Requests]reqID* is set using the **Sequence number** command. This field uniquely identifies each request.
- *[SP Requests]reqType* describes the type of the request.
- *[SP Requests]reqStatus* may take one of the following values:

Value	Description
1	the request has been posted but not processed yet.
0	the request has been successfully processed.
< 0	the request has been processed but an error occurred.

Note: These values are arbitrarily chosen for this example, they are not imposed by 4D.

- *[SP Requests]reqData* is a BLOB containing the data of the request. It can contain data sent by the requester or data returned by the stored procedure to the requester.
- *[SP Requests]reqParams* optionally contains parameter values sent by the requester to the stored procedure.

Why Use a Table?

Communication between a client process and a stored procedure can be implemented using the command **GET PROCESS VARIABLE**, **SET PROCESS VARIABLE** and **VARIABLE TO VARIABLE**. For example, this is the solution used in the section **SP-Based Import (Example)**, as well as in the **STOP SP SERVICES** project method listed previously.

Here, the system must allow the stored procedure to receive and send back variable amounts of data. Arrays, including Text and Picture arrays, could be used, but there are two reasons for using a table:

- The algorithm for handling requests via records is simpler to implement. Posting a request from a client machine just consists of adding a request to the table. Replying to the request from within the stored

procedure just consists of modifying this request.

- Since the requests are stored in a table, they are stored on disk. As a result, the size of a large request is is not an issue, because it can be purged from memory (unlike data stored in arrays).

Posting a Request From the Client Machine

The *Client post request* project method is a generic method for posting a request:

```
\ Client post request Project Method
\ Client post request ( String { ; Text } ) -> Long
\ Client post request ( Request Type { ; Parameters } ) -> Request ID
CREATE RECORD([SP Requests])
[SP Requests]reqID:=Sequence number([SP Requests])
[SP Requests]reqType:=$1
[SP Requests]reqStatus:=1
If(Count parameters>=2)
    [SP Requests]reqParams:=$2
End if
SAVE RECORD([SP Requests])
$0:=[SP Requests]reqID
```

The method returns the request ID number whose unicity is guaranteed by the use of the **Sequence number** command. After the record has been added to the [SP Requests] database, the client can poll the field [SP Requets]redStatus in order to wait until the stored procedure has completely handled the requests.

Polling the Request Status and Getting the Result on the Client Machine

The *Client get result* project method is a generic method for polling the status of the request. As explained previously, as soon as the field [SP Requets]redStatus becomes different from 1, the client knows that the stored procedure has managed (successfully or not) the request.

```
\ Client get result Project Method
\ Client get result ( Long ; ->BLOB {; Long } ) -> Long
\ Client get result ( Request ID ; ->Data {; Delay } ) -> Error Code
C_LONGINT($0;$1;$v1Delay)
$0:=1
$v1Delay:=0
If(Count parameters>=3)
    $v1Delay:=$3
End if
READ ONLY([SP Requests])
Repeat
    QUERY([SP Requests];[SP Requests]reqID=$1)
    If(Records in selection([SP Requests])>0)
        If([SP Requests]reqStatus&NBSP;#&NBSP;1)
            $2->:=[SP Requests]reqData
            READ WRITE([SP Requests])
            While(Locked([SP Requests]))
                WAITING LOOP($v1Delay)
            LOAD RECORD([SP Requests])
            End while
            DELETE RECORD([SP Requests])
            $0:=[SP Requests]reqStatus
        End if
    Else
        \ Request record has been lost!
        \ It should not happen. But anyway set error to -2 (arbitrary value)
        $0:=-2
    End if
    \ The request has not been processed yet
    If($0=1)
        WAITING LOOP($v1Delay)
    End if
Until ($0&NBSP;#&NBSP;1)
```

```
READ ONLY([SP Requests])
```

If the request has been successfully managed by the stored procedure, the method copies the result (if any) from the record to the BLOB whose pointer is passed as parameter. The caller method then parses and uses the BLOB data according to the type of the request. Note that the client is in charge of deleting the [SP Requests] record once the request is completed.

The small **WAITING LOOP** project method loops until a number of ticks has elapsed:

```
` WAITING LOOP Project Method
` WAITING LOOP ( Long )
` WAITING LOOP ( Delay in ticks )
C_LONGINT($1)
$vlStartTicks:=Tickcount
Repeat
  IDLE
Until ((Tickcount-$vlStartTicks)>=$1)
```

Reminder: **DELAY PROCESS** has no effect on the Application process. Using the **WAITING LOOP** project method, the process will wait the required amount of time, even though the request originated from the User environment process of a client machine.

The Stored Procedure and Its Subroutines

The **SP SERVICES** project method is the method running as stored procedure on the server machine. The overall architecture of this method, here shown in pseudocode, is straightforward:

```
Initialize a "stop" variable
Repeat
  Look for the requests with the [SP Requests]reqStatus field equal to 1
  For each request
    Depending on the type of the request, call a subroutine
      that stores the result in the [SP Requests]reqData field
    Change the status of the request so that the client knows what happened
  End for
  "Sleep" a little bit before to start again
Until the "stop" variable becomes true
```

Here is the actual source code:

```
` SP SERVICES Project Method
` The stored procedure is starting
vbStopSPServices:=False
` The stored procedure does not need read-write access to the tables...
READ ONLY(*)
` ...except the [SP Requests] table
READ WRITE([SP Requests])
Repeat
` Look for the requests that have not been processed yet
QUERY([SP Requests];[SP Requests]reqStatus=1)
` Process these requests one after one
For($vlRecord;1;Records in selection([SP Requests]))
` If the request record is locked, wait until it becomes unlocked
  While(Locked([SP Requests]))
` Wait one second before trying again
  DELAY PROCESS(Current process;60)
` Try to get read-write access
  LOAD RECORD([SP Requests])
End while
` Assume the request will be processed successfully
  [SP Requests]reqStatus:=0
  Case of
    :([SP Requests]reqType="Server Information")
      SP DO SERVER INFORMATION
    :([SP Requests]reqType="Volume List")
```

```

SP DO VOLUME LIST
:([SP Requests]reqType="Browse Directory")
SP DO BROWSE DIRECTORY([SP Requests]reqParams)
`
...
` OTHER REQUEST TYPES COULD BE ADDED HERE!
`
...
Else
` The request type is unknown, returns error -1 (arbitrary value)
  [SP Requests]reqStatus:=-1
End case
` Force request status to be different from 1
` (in case a subroutine sets it to 1)
If([SP Requests]reqStatus=1)
  [SP Requests]reqStatus:=-3
End if
` Update the request record
SAVE RECORD([SP Requests])
` Go to the next unprocessed request
NEXT RECORD([SP Requests])
End for
` Free the last processed request record
UNLOAD RECORD([SP Requests])
` Wait one second before starting answering request again
DELAY PROCESS(Current process;60)
` Loop until the SP is told to stop execution
Until(vbStopSPServices)

```

The **SP SERVICES** project method can be used as a template for implementing new services to a database. In this section, we detail the **SP DO SERVER INFORMATION** and **SP DO VOLUME LIST** subroutines. The **SP DO BROWSE DIRECTORY** (which takes as a parameter the parameter sent by the client in the `[SP Requests]reqParams` field) is not detailed in this document.

Depending on the type of the request, the **SP SERVICES** project method calls a subroutine whose task is to store the result data in the `[SP Requests]reqData` field. Saving the record and changing the status of the request is performed by the **SP SERVICES** project method.

Here is the **SP DO SERVER INFORMATION** subroutine. It stores server-related information in the BLOB. Another project method will extract the BLOB data accordingly on the client machine.

```

` SP DO SERVER INFORMATION Project Method
TEXT TO BLOB(Application version(*);[SP Requests]reqData;UTF8 C string)
TEXT TO BLOB(Structure file;[SP Requests]reqData;UTF8 C string;*)
TEXT TO BLOB(Data file;[SP Requests]reqData;UTF8 C string;*)
PLATFORM PROPERTIES($vlPlatform;$vlSystem;$vlMachine)
VARIABLE TO BLOB($vlPlatform;[SP Requests]reqData;*)
VARIABLE TO BLOB($vlSystem;[SP Requests]reqData;*)
VARIABLE TO BLOB($vlMachine;[SP Requests]reqData;*)

```

Here is the **SP DO VOLUME LIST** subroutine. It stores volume-related information in the BLOB. Another project method will extract the BLOB data accordingly on the client machine.

```

` SP DO VOLUME LIST Project Method
VOLUME LIST($asVName)
$vlSize:=Size of array($asVName)
ARRAY REAL($arVSize;$vlSize)
ARRAY REAL($arVUsedSpace;$vlSize)
ARRAY REAL($arVFreeSpace;$vlSize)
For($vlElem;1;$vlSize)
  VOLUME ATTRIBUTES($asVName{$vlElem};$arVSize{$vlElem};$arVUsedSpace{$vlElem}
  ;$arVFreeSpace{$vlElem})
End for
VARIABLE TO BLOB($asVName;[SP Requests]reqData)
VARIABLE TO BLOB($arVSize;[SP Requests]reqData;*)
VARIABLE TO BLOB($arVUsedSpace;[SP Requests]reqData;*)
VARIABLE TO BLOB($arVFreeSpace;[SP Requests]reqData;*)

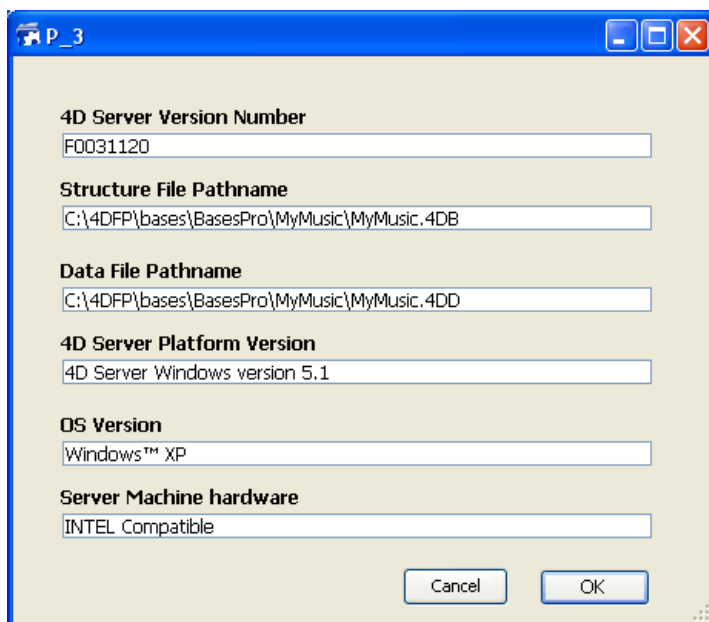
```


Showing the Server Information on a Client Machine

Using the generic *Client post request* and *Client get result* project methods, the **M_SERVER_INFORMATION** project method displays the server information returned by the stored procedure on the client machine. This method could be attached to a menu command or invoked, for instance, from a button's object method:

```
\ M_SERVER_INFORMATION
C_BLOB (vxData)
C_LONGINT ($v1ReqID;$v1ErrCode;$v1Offset)
\ Post the request
$v1ReqID:=Client post request("Server Information")
\ Poll the request status and get the result
$v1ErrCode:=Client get result($v1ReqID;->vxData;60)
\ If the request is successfully completed, display the result
If ($v1ErrCode=0)
\ Extract the result information from the BLOB
$v1Offset:=0
vsServerVersion:=BLOB to text(vxData;UTF8 C string;$v1Offset)
vsStructureFile:=BLOB to text(vxData;UTF8 C string;$v1Offset)
vsDataFile:=BLOB to text(vxData;UTF8 C string;$v1Offset)
BLOB TO VARIABLE (vxData;$v1Platform;$v1Offset)
BLOB TO VARIABLE (vxData;$v1System;$v1Offset)
BLOB TO VARIABLE (vxData;$v1Machine;$v1Offset)
\ Analyse the platform properties
vs4DPlatform:="Unknown 4D Server Version"
vsSystem:="Unknown System Version"
vsMachine:="Unknown Machine"
\ ...
\ Here is the code (not listed) that parses the $v1System and $v1Machine
\ ( see the example for the PLATFORM PROPERTIES command)
\ ...
\ Display the result information
DIALOG ([SP Requests];"SERVER INFORMATION")
Else
ALERT ("Request error "+String($v1ErrCode))
End if
\ No longer need the BLOB
CLEAR VARIABLE (vxData)
```

Here is the [SP Requests];"SERVER INFORMATION" form being executed:



The screenshot shows a dialog box with a blue title bar containing the text 'P_3'. The dialog box has a light gray background and contains several text input fields with labels to their left. At the bottom, there are two buttons: 'Cancel' and 'OK'. The data entered in the fields is as follows:

Label	Value
4D Server Version Number	F0031120
Structure File Pathname	C:\4DFP\bases\BasesPro\MyMusic\MyMusic.4DB
Data File Pathname	C:\4DFP\bases\BasesPro\MyMusic\MyMusic.4DD
4D Server Platform Version	4D Server Windows version 5.1
OS Version	Windows™ XP
Server Machine hardware	INTEL Compatible

Showing the Server Machine Volume List on a Client Machine

Using the generic *Client post request* and *Client get result* project methods, the **M_SERVER_VOLUMES** project

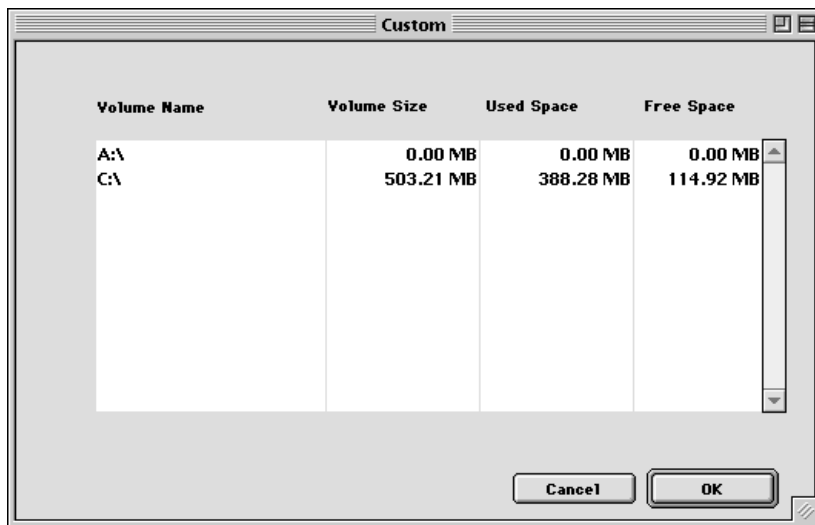
method displays, on the client machine, the server machine volume list returned by the stored procedure. This method could be attached to a menu command or invoked, for instance, from a button's object method:

```

` M_SERVER_VOLUMES
C_BLOB(vxDat)
` Post the request
$vlReqID:=Client post request("Volume List")
` Poll the request status and get the result
$vlErrCode:=Client get result($vlReqID;->vxDat;120)
` If the request is successfully completed, display the result
If($vlErrCode=0)
` Extract the result information from the BLOB
$vlOffset:=0
BLOB TO VARIABLE(vxDat;asVName;$vlOffset)
BLOB TO VARIABLE(vxDat;arVSize;$vlOffset)
BLOB TO VARIABLE(vxDat;arVUsedSpace;$vlOffset)
BLOB TO VARIABLE(vxDat;arVFreeSpace;$vlOffset)
For($vlElem;1;Size of array(arVSize))
` Convert from bytes to MB
arVSize{$vlElem}:=arVSize{$vlElem}/1048576
arVUsedSpace{$vlElem}:=arVUsedSpace{$vlElem}/1048576
arVFreeSpace{$vlElem}:=arVFreeSpace{$vlElem}/1048576
End for
` Display the result information
DIALOG([SP Requests];"VOLUME LIST")
Else
ALERT("Request error "+String($vlErrCode))
End if
` No longer need the BLOB
CLEAR VARIABLE(vxDat)

```

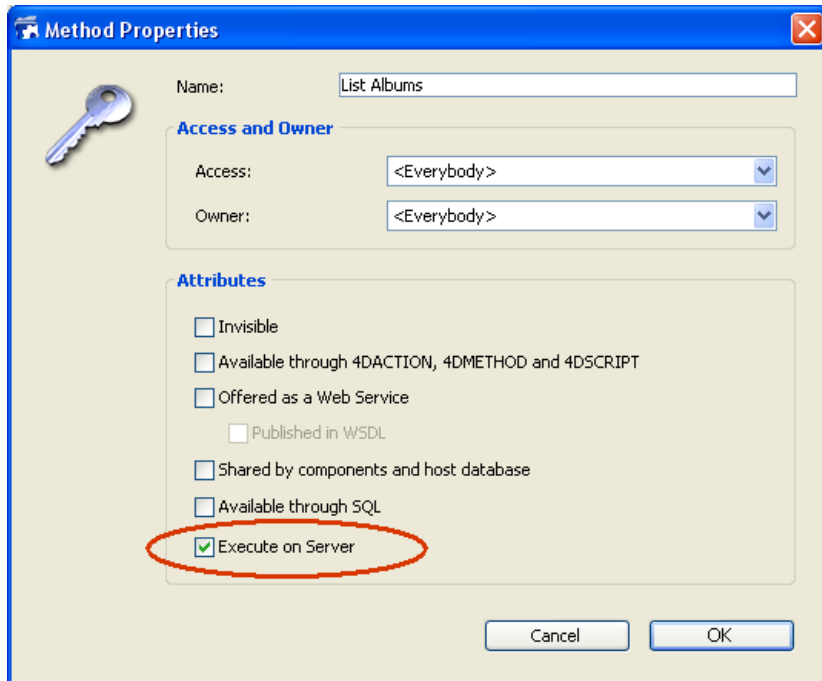
Here is the [SP Requests];"VOLUME LIST" form being executed:



Volume Name	Volume Size	Used Space	Free Space
A:\	0.00 MB	0.00 MB	0.00 MB
C:\	503.21 MB	388.28 MB	114.92 MB

Execute on Server attribute

The "Execute on Server" project method attribute can be set using the batch setting of attributes dialog box as well as the Method Properties dialog box:



When this option is checked, the project method is always executed on the server, regardless of how it is called.

Note: This attribute is only taken into account for a 4D application in client/server mode.

Execution Context

When this attribute is checked, the execution context of the project method is comparable to that of triggers (see [4D Server and the 4D Language](#)): the method on the server shares the same database context as the corresponding context on the client side for locking records and for transactions, but not the same language context (process variables, sets, current selections). However, unlike a trigger, a method executed on the server does not share the current record with the client context.

All the parameters of the method (\$1, \$2, etc.) are sent to the server and the value of parameter \$0, if used, is returned to the client.

Unlike the **Execute on server** command, this option does not create a process on the server. 4D Server uses the "twin" process of the client process that requested the execution.

Moreover, this option simplifies the principle of delegating the execution of a method on the server since the transfer of parameters is automatically carried out in both directions, as with a "normal" method call.

The **Execute on server** command functions asynchronously, therefore it requires more programming and makes use of semaphores for reading the results.

Usable Commands

Methods that have the "Execute on Server" attribute are subject to the same rules as the stored procedures as far as the use of 4D language commands is concerned. Executing certain commands is not allowed on the server, executing others is not recommended. For more information, please refer to the "What a stored procedure does not do (executed on the server)?" paragraph in the [Stored Procedures](#) section.

Pointers

If you pass a pointer to a variable (simple variable, array or array element), the pointed value is also sent to the server. If the pointed value is modified on the server by the method, the modified value is returned to the client in order to update the corresponding variable on the client side.

Pointers to a table or field are sent as references (table number, field number). The current record value is not automatically exchanged.

Note: This option works the same way in interpreted mode as in compiled mode.

Example

Here is the code for the Myappli project method which has the "Execute on Server" attribute:

```
C_POINTER($1) `Pointer to table
C_POINTER($2) `Pointer to field
C_POINTER($3) `Pointer to array
C_TEXT($4) `Value to be searched for
C_LONGINT($0) `Result

`Search and send back values for each record
QUERY ($1->;$2->=$4)
While (Not(End selection($1->)))
    APPEND TO ARRAY ($3->;myFormula($1))
    NEXT RECORD ($1->)
End while
UNLOAD RECORD ($1->)
$0:=Records in selection($1->)
```

On the client side, the method is called as follows:

```
ARRAY TEXT(myArray;0)
$vlnum :=MyAppli(->[Table_1] ;->[Table_1]Field_1 ;->myArray;"to find")
```