4D Server®

Reference
Windows® and Mac OS® Versions

4D Server Reference
Version 2004 for Windows® and Mac OS®

Copyright © 2005 4D SA / 4D, Inc.
All rights reserved

IMPORTANT LICENSE INFORMATION

Use of this Software is subject to the 4D Product Line License Agreement, which is provided in electronic form with the Software. Please read the 4D Product Line License Agreement carefully before completely installing or using the Software.

Contents

5. 4D Server Menus 89

6. 4D Server and the 4D Language 103

7. Stored Procedures 119

1

Introduction

version 2004.3 (Modified)

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

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) with 4D Connectivity Plug-ins. 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, 4th Dimension. The 4D Server application itself is divided into two parts—4D Server and 4D Client—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).

A copy of 4D Client resides on each client machine. Using 4D Client, 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 4th Dimension can be done using 4D Server and 4D Client.

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

Any 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.

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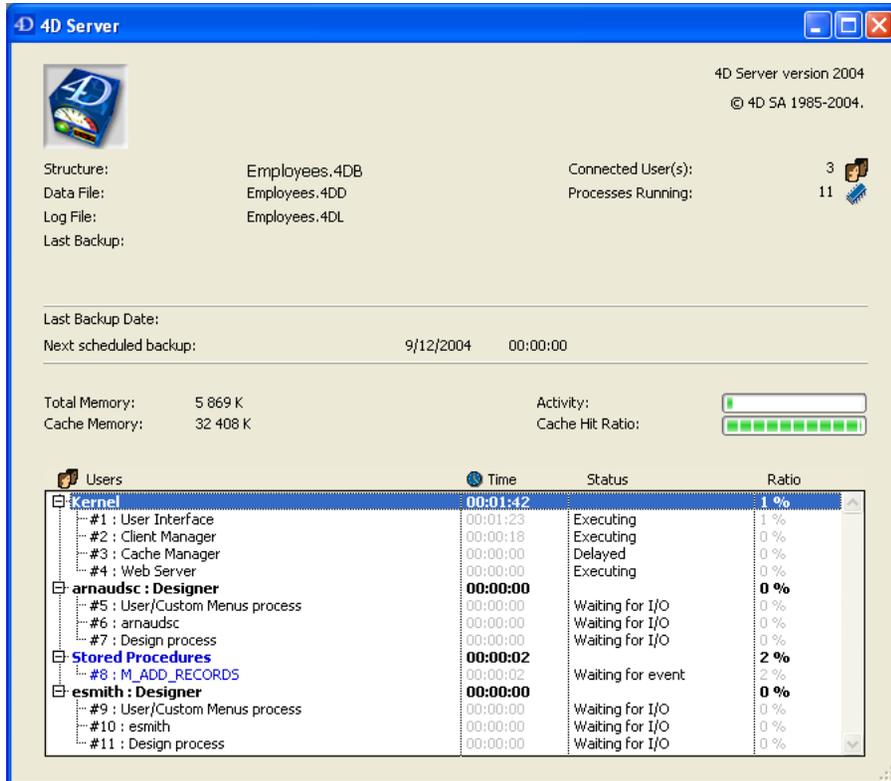
4D Server Main Window

• **Simple on-line graphical centralized information**: The 4D Server Main 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, server activity, cache ratio and monitoring of backups.

• **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 controling of client workstations**: All 4D clients are automatically and dynamically updated if the database is modified, or if a plug-in is added, removed, or modified.

• **Automatic asynchronous connections using the standard TCP/IP protocol**: 4D Server and 4D Client 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 now integrated in all operating systems, using this protocol with 4D Server does not require any additional installation.

• **Automatic session and state maintenance for 4D Clients and Web 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. In contextual mode, the Web server of 4D Server provides Web clients with all the client/server functions.

•**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 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 4th Dimension 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.

• **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 Client, no matter what the platform of the client.

• **Built-in Web Server**: Like 4th Dimension, 4D Server and each 4D Client has a Web Server 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 4D Client machine into a Web server. For more information about the built-in Web Server of 4D Server and 4D Client, see the section Web Server, Overview in the *4th Dimension Language Reference* manual.

• **Connection Security via SSL** (Secured Socket Layer) : You can encrypt 4D Server connections. In fact, the "classic" client/server architecture can make use of SSL's 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, modify, and load). You can write very simple triggers, and then make them more sophisticated. Triggers can prevent "illegal" operations within database records. They are a very powerful tool to restrict table operations, 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. With 4D Server, triggers are executed on the server machine. Any client, whether it is a 4D Client or a 4D Open-based application, is subjected to the database rules enforced by the triggers. For more information about 4D triggers, see the section Triggers in the *4th Dimension 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 4D Open. 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 in the Password Access editor. This feature allows a user to connect to a database on the server by clicking the 4D Client Path document icon. 4D Client uses the path to automatically access the correct database.

• **4D Open**: Using 4D Open, the API (Application Programming Interface) for 4D Server, users can connect to a 4D Server database from programs other than 4D Client. These programs can be either Windows or Macintosh applications. 4D Open can be included in 4th Dimension and 4D Client. Users can simultaneously connect to multiple servers, enabling departmental and interdepartmental distributed systems. Last but not least, using 4D Open from within triggers and stored procedures (thus, executing methods on the server machine), you can write systems where 4D Server connects to other 4D Servers in order to achieve automatic data replication or distribution. In short, with 4D Open, 4D Server becomes a client of its own architecture.

• **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 Client 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.

• **Remote Connections:** On Mac OS, 4D Server supports remote connections via ARA (Apple Remote Access). For further information, refer to the ARA documentation (Mac OS).
On Windows, 4D Server supports remote connections via Remote Access Windows. For more information, refer to the Windows manual.

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. In recent years, 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."

User | Network | **File server**

Creating the query

SEARCH ([People];[People]Name="Smith")

Loading index table pages from the shared file

Calculating the position of the record in the index tables for each index in the file

Loading address table pages from the shared file

Locking the record in the shared file

Loading the record the shared file

Displaying the record

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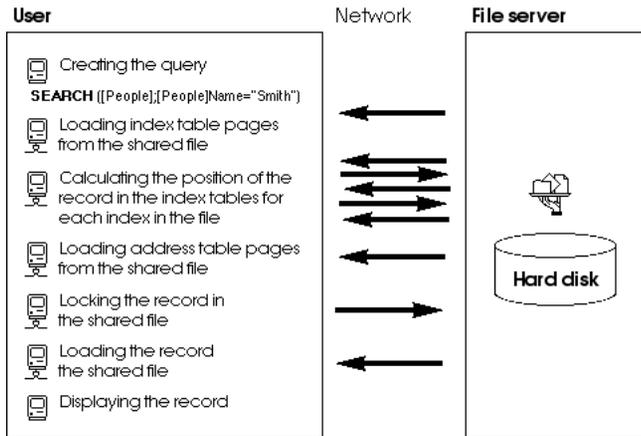
Another disadvantage of the file sharing model is the inability to use a memory cache to keep records in memory. If records were kept in memory, different users could have different versions of the same record stored in cache, leading to data inconsistency. As a result, each time a user accesses a record,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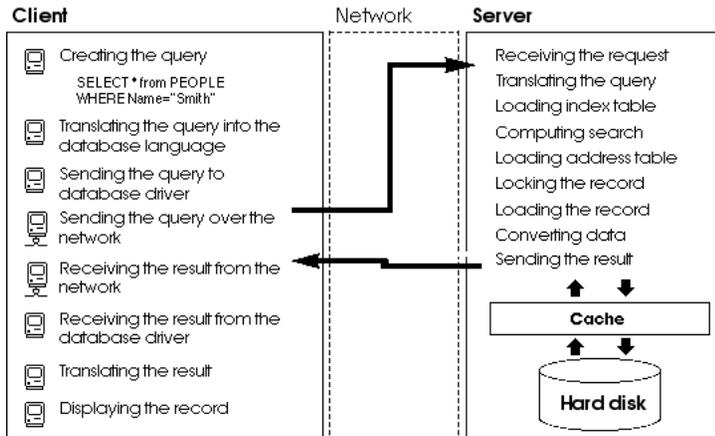
Heterogeneous Client/Server Architecture

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.

Client | Network | **Server**

Creating the query
SELECT * from PEOPLE
WHERE Name="Smith"

Translating the query into the database language

Sending the query to database driver

Sending the query over the network

Receiving the result from the network

Receiving the result from the database driver

Translating the result

Displaying the record

Receiving the request
Translating the query
Loading index table
Computing search
Loading address table
Locking the record
Loading the record
Converting data
Sending the result

Cache

Hard disk

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 Client are two applications that share the same structure and communicate directly.

Since 4D Server and 4D Client 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.
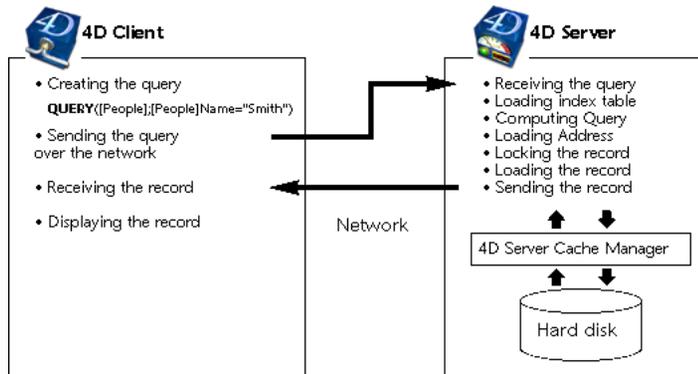
The division of labor is organized so that one request yields one response. As you can see in the diagram, the client is responsible for:

• **Requests**: 4D Client sends requests to 4D Server. These requests can be performed using the built-in editors, such as the Query and Order By editors, or using the procedural language. 4D Client provides editors in which methods can be created and modified. It also manages method components such as variables and arrays.

• **Receiving responses**: 4D Client receives responses 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 Client 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 Client. The integration of 4D Server's architecture is present at every level:

• **At the request level**: When 4D Client sends a request to 4D Server, such as a query or a sort, 4D Client 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 Client and 4D Server exchange a data or structure object, both applications use the same internal format. For instance, when 4D Client 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 Client wants to update a record, 4D Client sends the data to 4D Server, which stores it in cache exactly as it was received.

• **At the user interface level**: When 4D Client 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.

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.

2

4D Server in 10 minutes

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
• Connect with a Web browser

You will need:
• One machine with 4D Server installed
• Two machines with 4D Client installed
• One machine with a Web browser installed

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

Installed elements

If you have installed 4D Server and 4D Client in the default folders proposed by their installation programs, you should have the following folders and files on disk.

4D Server

A 4D Server folder located in the installation folder has been created.
• Under Windows, this folder contains the 4D Server application, its related files, and a 4D Extensions folder. To start 4D Server, double-click on the 4DSERVER.EXE file.
• Under Mac OS, this folder contains the 4D Server software package. To launch 4D Server, just double-click on this package.

4D Client

A 4D Client folder located in the installation folder has been created.
• Under Windows, this folder contains the 4D Client application, its related files, and a 4D Extensions folder. To start 4D Client, double-click on the 4DCLIENT.EXE file.
• Under Mac OS, this folder contains the 4D Client software package. To launch 4D Client, just double-click on this package

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

This section explains how to create a server database that can be accessed on the network using 4D Client and then using a Web Browser. Before working with 4D Server and 4D Client 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 registered your 4D Server with 4D, as described in the 4th Dimension *Installation Guide*.

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

1. Launch 4D Server by double-clicking on the 4D Server icon.
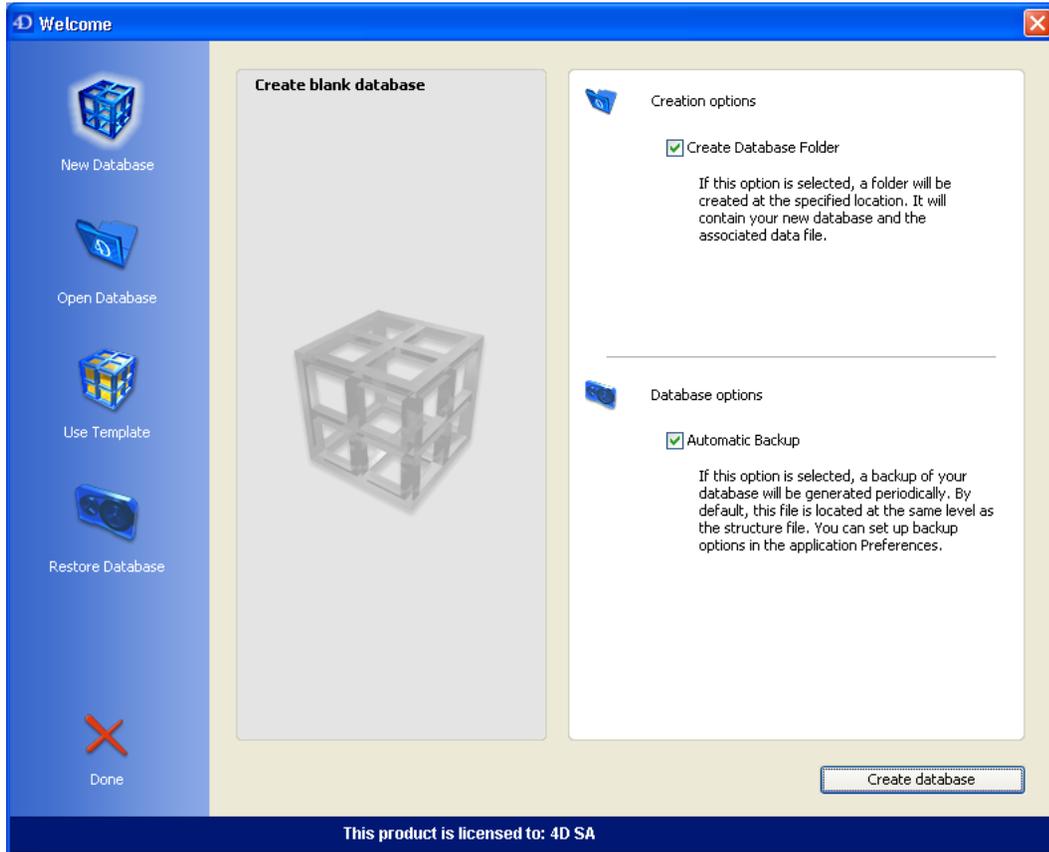
The Welcome dialog box appears, giving you the choice of opening an existing database, creating a new one (either blank or based on a template), or restoring an archived database. In this exercise, you will create a new database.
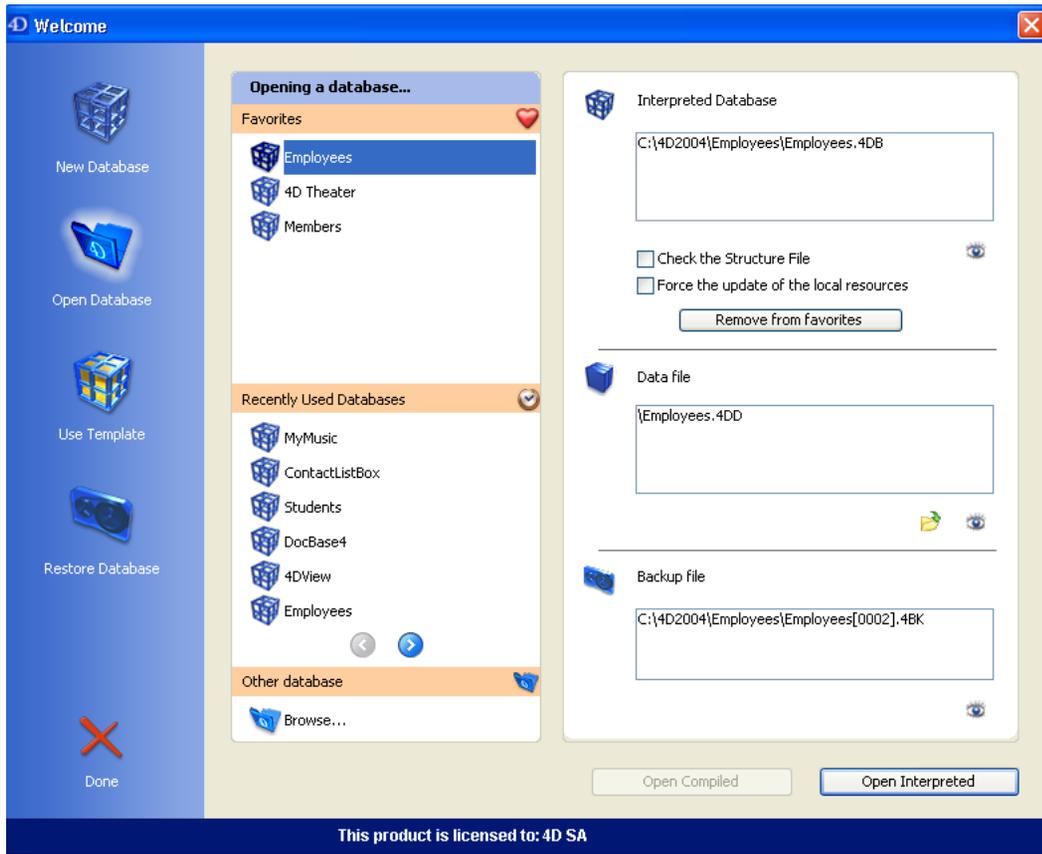
2. Click on the New Database icon on the left-hand side of the window.

The following window appears:

3. Keep the default options and click on the Create database button.
A standard save file dialog box appears, which lets you specify the name and location of the new database to be created.

4. Specify a location, then enter the name of the structure file 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 Process window appears:
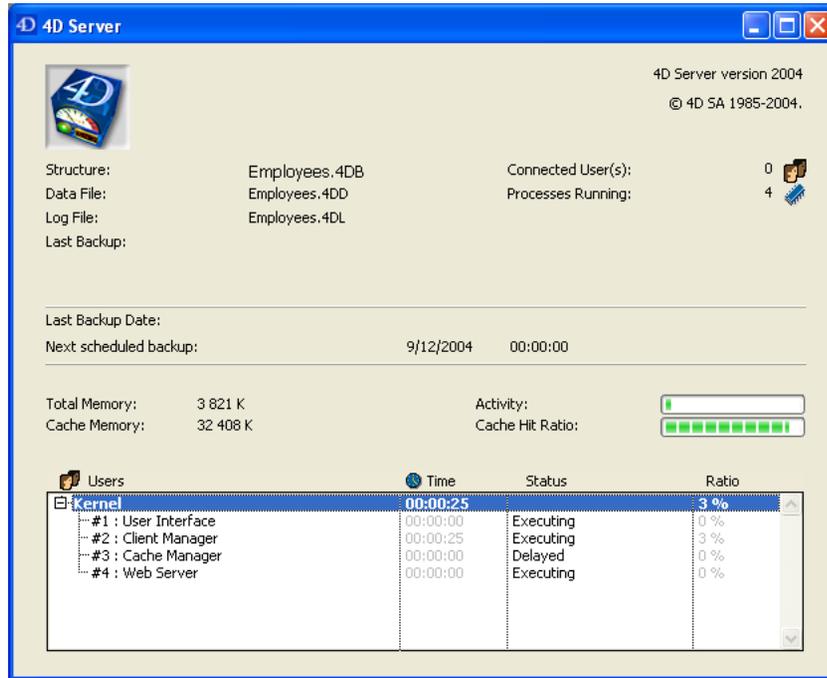
The 4D Server Process window is composed of several parts.

The upper part displays general information about the server and the files used; the central part indicates the state of database backups and information related to the memory; and the lower part displays information about the current server activity.

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. The number of processes currently running is four. These four processes are the processes created by the database engine (kernel) and the built-in Web Server.

The lower portion of the 4D Server window lists the connected users and the processes that are currently executing. When no client workstation is connected, the only information displayed consists of the server engine activity.

Where To Go From Here?

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

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

Before working with 4D Server and 4D Client for the first time, it is a good idea to check your installation. To do so, read the section Checking Your Installation.

This section discusses:
• How to connect clients 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.

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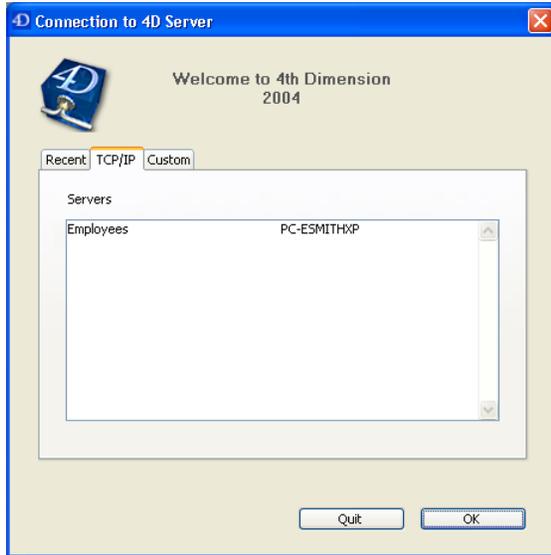
1. Double-click the 4D Client application icon.

The Connection dialog box is displayed:
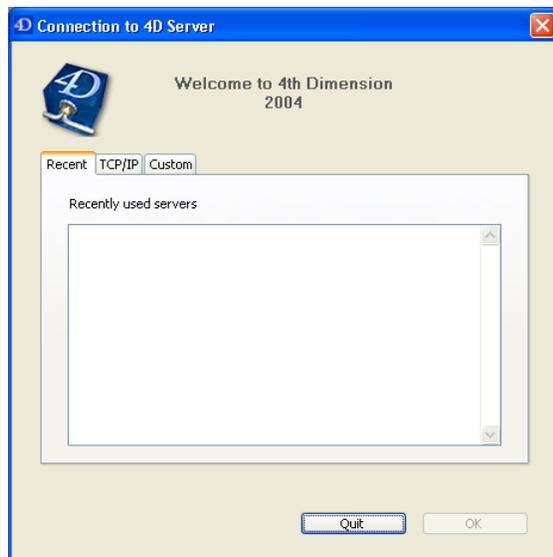
2. Click on the TCP/IP tab in order to display the list of 4D databases published on the network.

The Employees database appears in the list:

3. Select Employees.4DB and click OK.

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

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 runnning on the other machine?
• Are 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 Connecting to a 4D Server Database.

Server Activity

When you look at the 4D Server Process window, note that your network user name has been added to the list of users and the number of connected users is one.

There are now a total of six processes running:
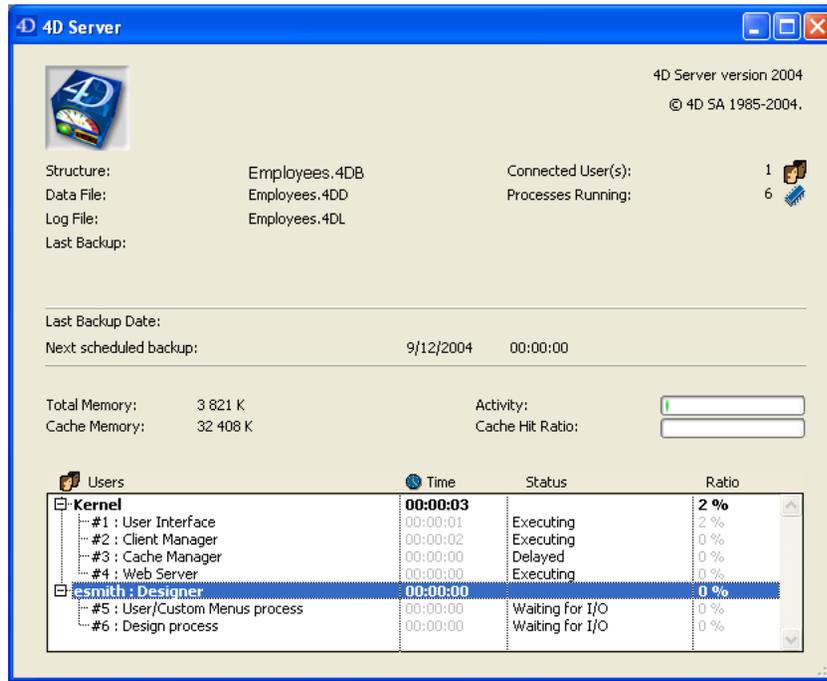
The first four processes are for 4D Server itself and were created when 4D Server was launched. The two new processes are for the first user connecting to the server:

• The Design process manages the Design environment. If you close the Design process on the client machine by choosing **Close Structure** from the **File** menu, the Design process on the server ends.

• The User/Custom Menus process manages the User and Custom Menus environments. Each additional user will add at least two processes to the list of processes. You can hide the processes of a user by clicking the cross (Windows) or the arrow (Macintosh) to the left of the user's name. To show the processes of the user, click the cross (Windows) or the arrow (Macintosh) again.

Where To Go From Here?

Now that you are connected, you can work with the database with the same feature set as that of 4th Dimension in single-user environment. First, you need to define the structure. Go to the section Defining the database structure.

After you have connected to the server database (see the section Connecting to the Server Database with 4D Client), bring the Structure window to the front.

The new database has a default first table name [Table1].

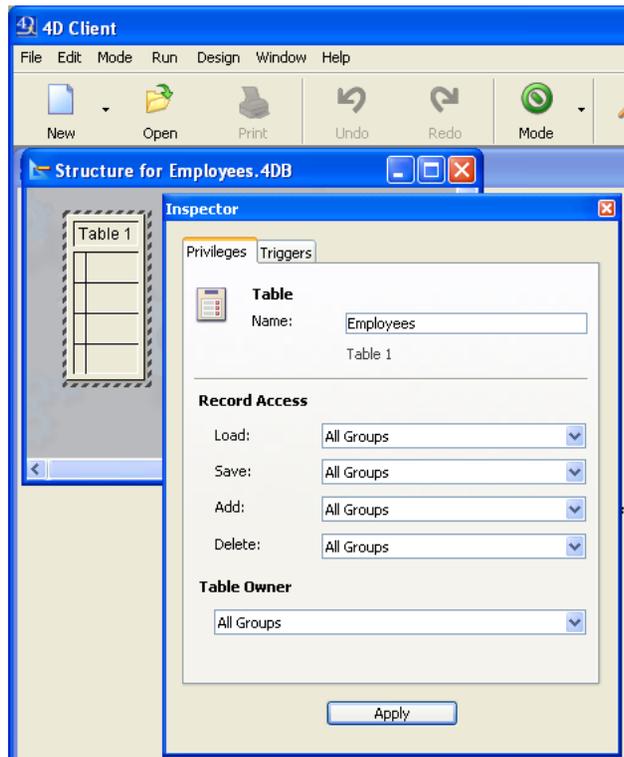**Setting the Definition of the [Employees] Table — an Example**

1. Double-click the title bar of [Table1].
OR
Click using the right mouse button or press Ctrl+click (on Mac OS) on [Table1], then select **Table Properties...** within the contextual menu.

The Inspector floating window appears.

2. Type Employees in the Name area.

3. Click the **Apply** button.

4. Click using the right mouse button or press Ctrl+click (on Mac OS) on [Table1], then select **New Field** within the contextual menu.

The field properties are displayed in the Inspector window.

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

Field Name	Field Type
Last Name	Alphanumeric (20 characters)
First Name	Alphanumeric (20 characters)
Salary	Real
Department	Alphanumeric (20 characters)

For each field, enter the name of the field in the Name area, select the field type and click **Add**.
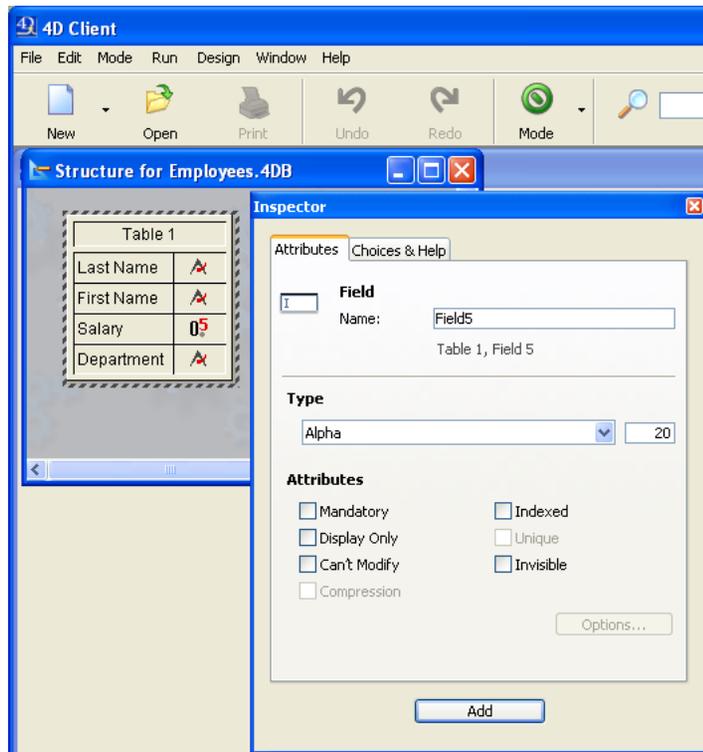
After you have added the fields to the table, close the Inspector window.

Note: If other 4D Clients 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 Foms 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 Server comes with a convenient shortcut for quickly creating default input and output forms.

1. Choose **User** from the **Mode** menu.

You switch to the User environment. 4D Server detects that the table still has no forms and asks if you want to let the program create them for you.
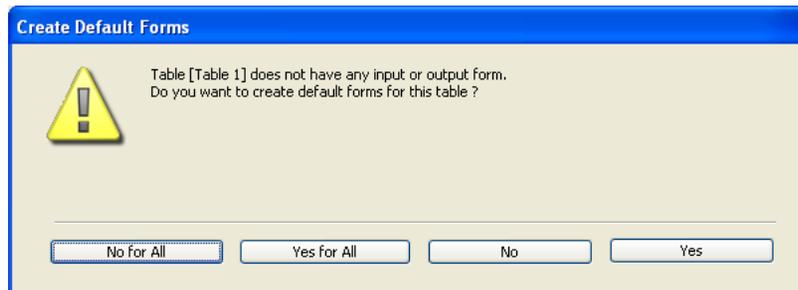
2. Click **Yes**.

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 Server create the default forms for that table. You are now ready to enter records.

Entering Records

You enter, query, print and modify records in the 4D Client User environment. If you are not yet in the User environment, choose **User** from the **Mode** menu. 4D Server displays the default output form that is created automatically, blank for the moment.

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.

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

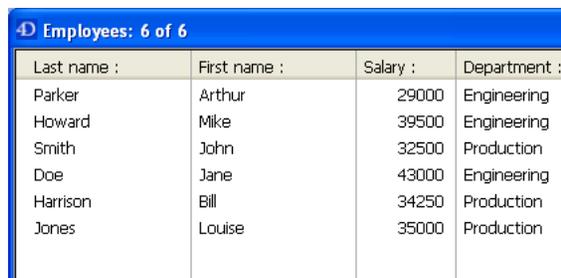A blank input form appears so you can continue adding new records.

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

5. 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.

6. If all the six records are not shown, choose **Show All** from the **Records** menu.

You should have:

The records are now stored in the database on the server machine. If a second 4D Client 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 **Find** button in the toolbar.

The Query Editor appears:
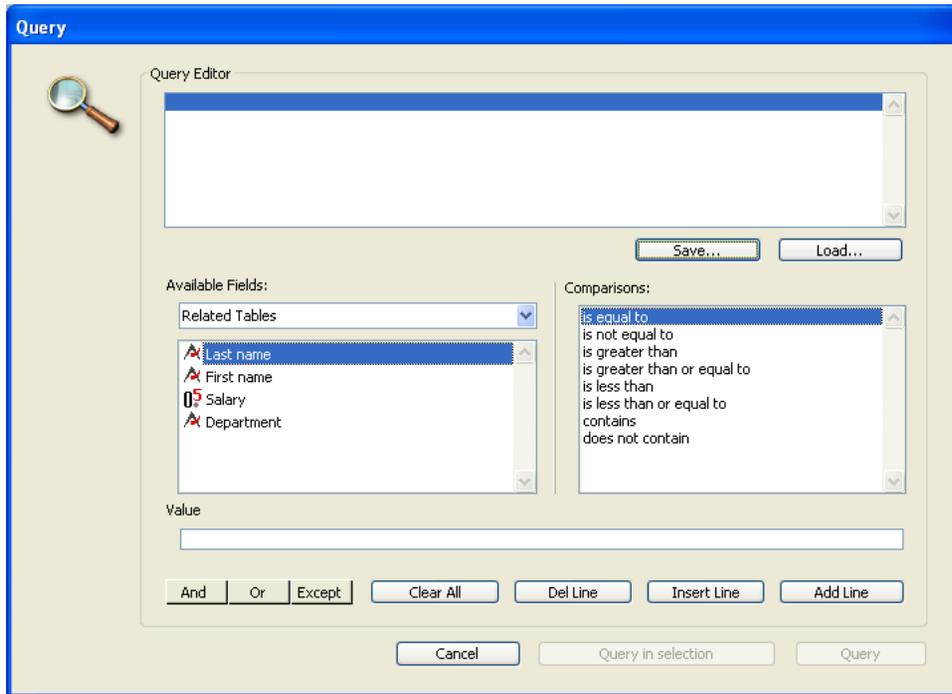
2. Click on Department in the Fields list, on "is equal to" in the Comparisons list, then enter Engineering in the Value area.

3. Click **Query**.

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

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

Making a Chart

1. Choose **Charts...** from the **Tools** menu.

The Chart Wizard appears:
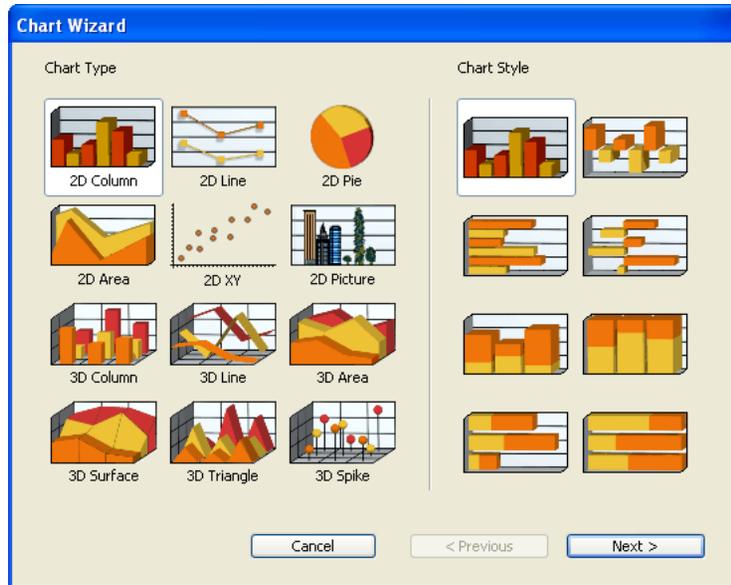
You are going to create a 2D Column chart (which is the chart type selected by default).

2. Click on the Next > button.

You must select the data that you want to use to create a new chart.

3. Drag and drop the field Last Name to the area Categories (X Axis) and the field Salary to the list Values (Z Axis):
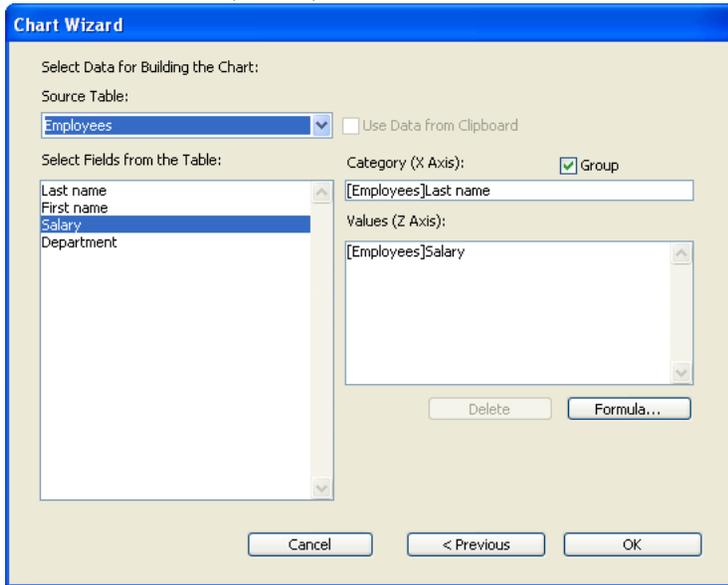
5. Click **OK**.

A Chart window appears, displaying the chart created according to your settings.

Where To Go From Here?

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

It is now time to add a custom menu bar to your database. Go to the 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. Return to the Design environment by choosing **Design** from the **Mode** menu.

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

The New Method dialog box appears.

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

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

4. Enter the code of the M_ADD_RECORDS method as shown:

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

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

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

The Menu Bar Editor appears containing a default menu bar.

2. Go to the end of the list of existing menus (click on the "Mode" menu) and choose the **Add Menu** command in the lower part of the window.

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

4. Choose **Add Item** to add a menu item to the "Tutorial" menu.

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

6. Choose **Add Item** 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:
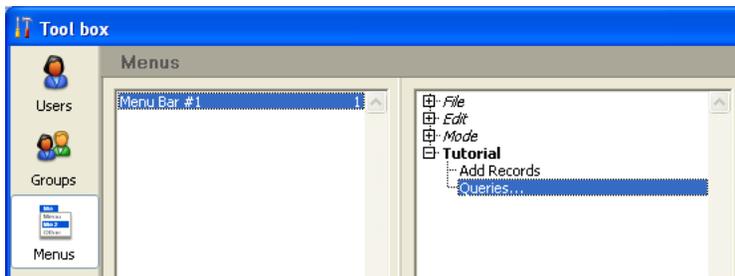
8. Click the "Add Records..." menu command and enter "M_ADD_RECORDS" in the Method Name area.

9. Click the "Queries..." menu command and enter "M_LIST_RECORDS" in the Method Name Area.

The menu bar #1 should now look like this:
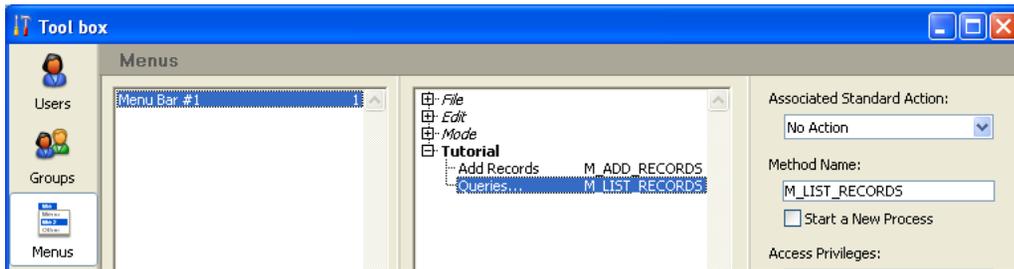
10. Close the Tool box window.

You're done!

11. Select **Custom Menus** from the **Mode** menu.

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

For example, if you select **Queries...** from the **Tutorial** menu, the Query editor (the built-in Query editor from the User environment) 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.

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 Windows 4D Client and a 4D Client Mac OS. As soon as you are connected, you can see the second user entry in the 4D Server Process window:
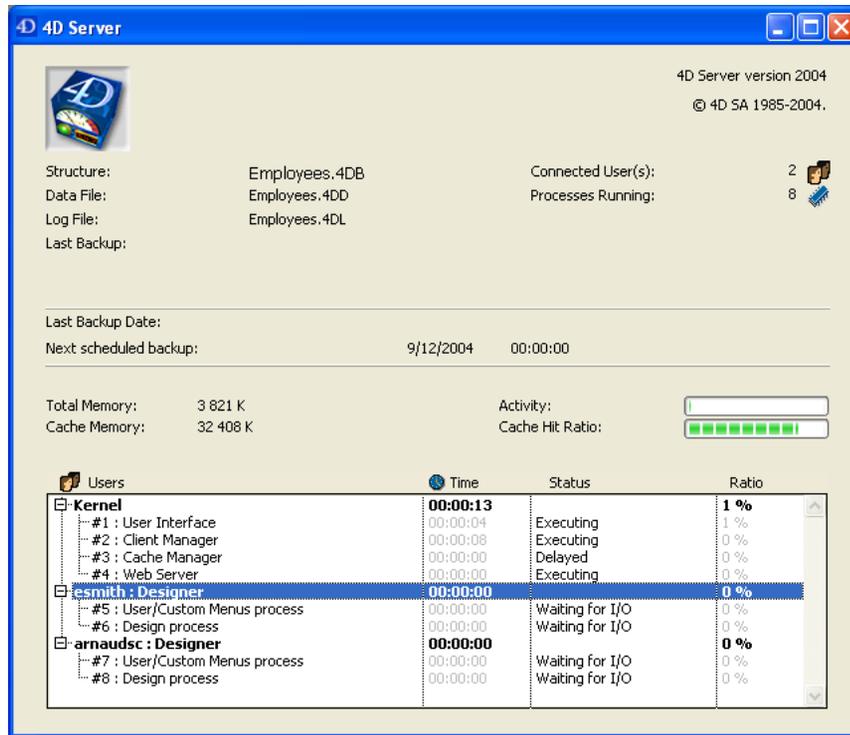
On each client machine, everything done on the other platform is instaneously and transparently reusable. Here is the Design environment on a Mac OS client:
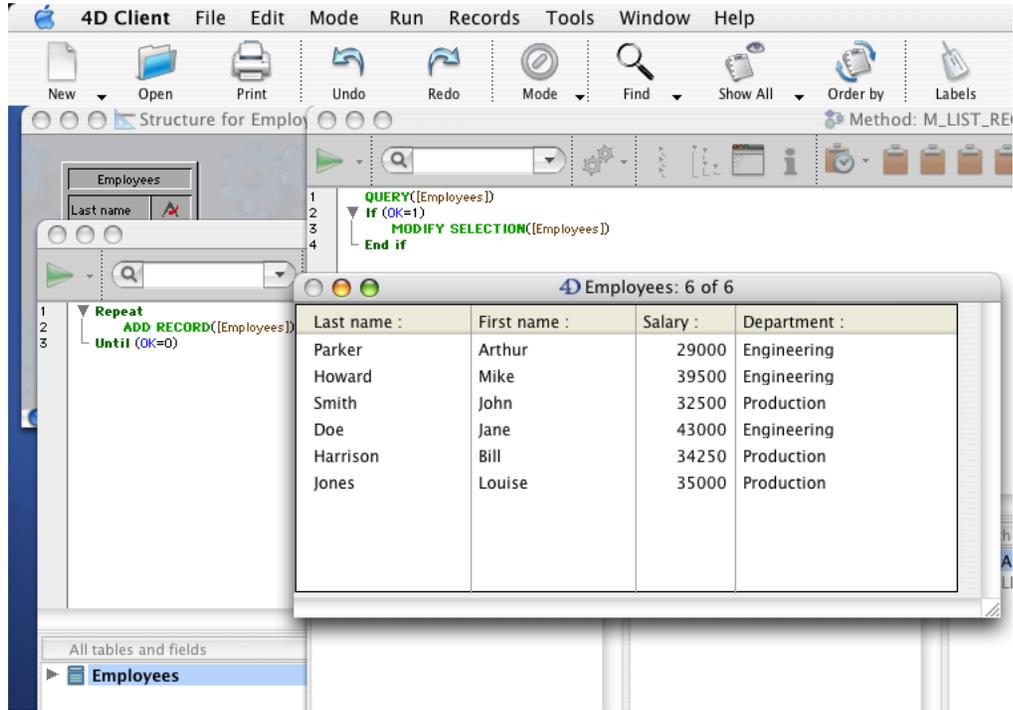
Your six records and your two methods are here!

Working Concurrently with Records

1. On the first client machine, change to Custom Menus 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 client 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:
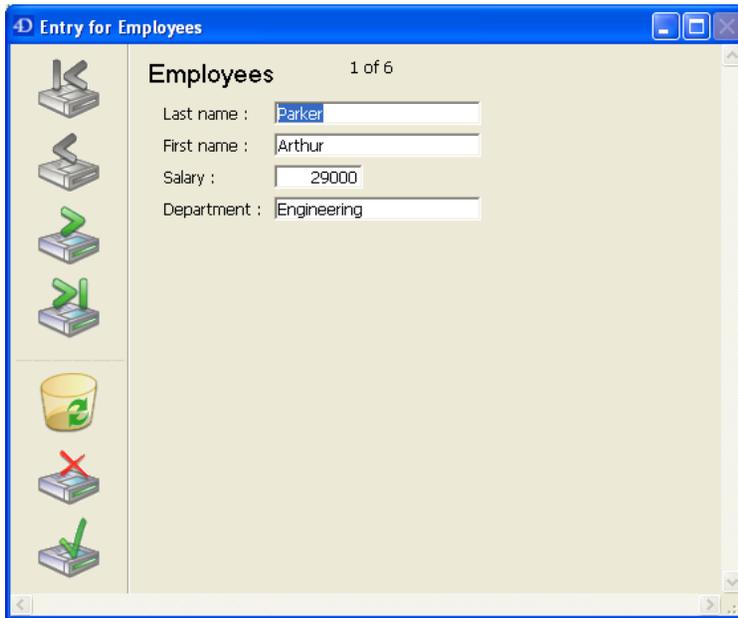
4. Do the same thing on the second machine.

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.

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!

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 **Design** 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. Expand the Employees table:
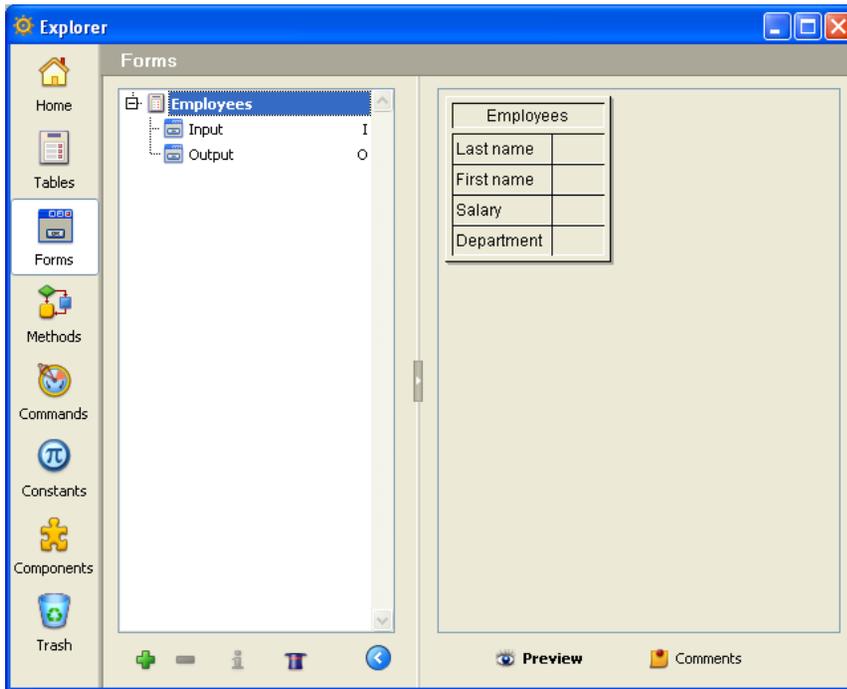
5. Double-click the Input form.

A Form Editor window is opened for the Input form:
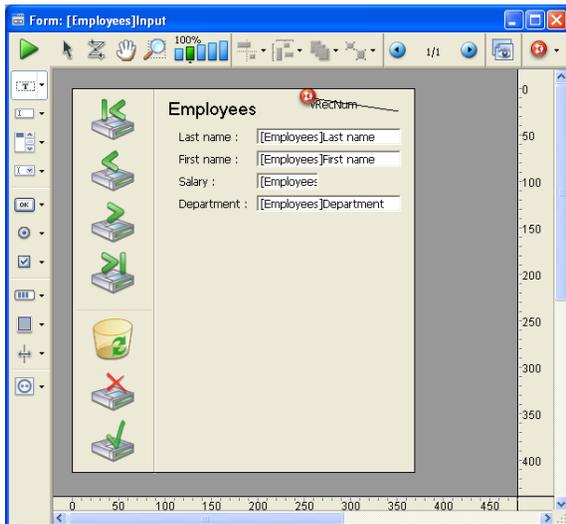
6. Do the same thing on the second machine.

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 left corner of the form. This icon reminds you that you cannot change the form.
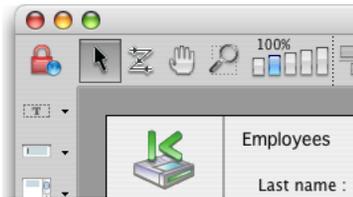
7. On the first machine, select the legend "Last name" on the left of the [Employees]Last Name field.

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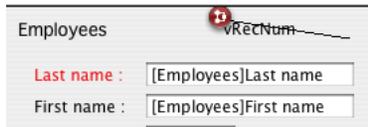
4D Server allows you to develop concurrently a database with other users!

Shutting Down the Server

In addition to informing 4D Client 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:
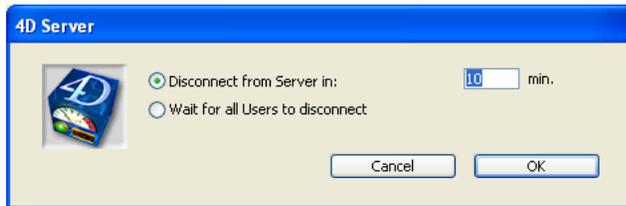
2. Click OK.

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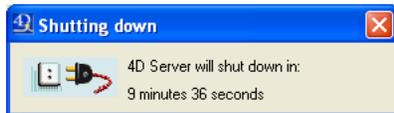
This warning dialog box is repeated regularly on each client machine.

Note: Alternatively, you can shut down the server using the option "Wait for all users to disconnect."

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

Where To Go From Here?

First, after these nine intense minutes and while the server is shutting down, you might want to have drink.

Now, the interesting point is that you did not just develop two applications—in fact, you developed three applications!

To see why, go to the section 4D Server is a Web Server. This will take about one minute.

Restart the server database by launching 4D Server again and open the Employees database you created.

Connect to the database using 4D Client.

You are going to quickly create a form adapted to Web publication.
In Design mode, choose the **New >Form**... command in the **File** menu.
The New Form Wizard appears.
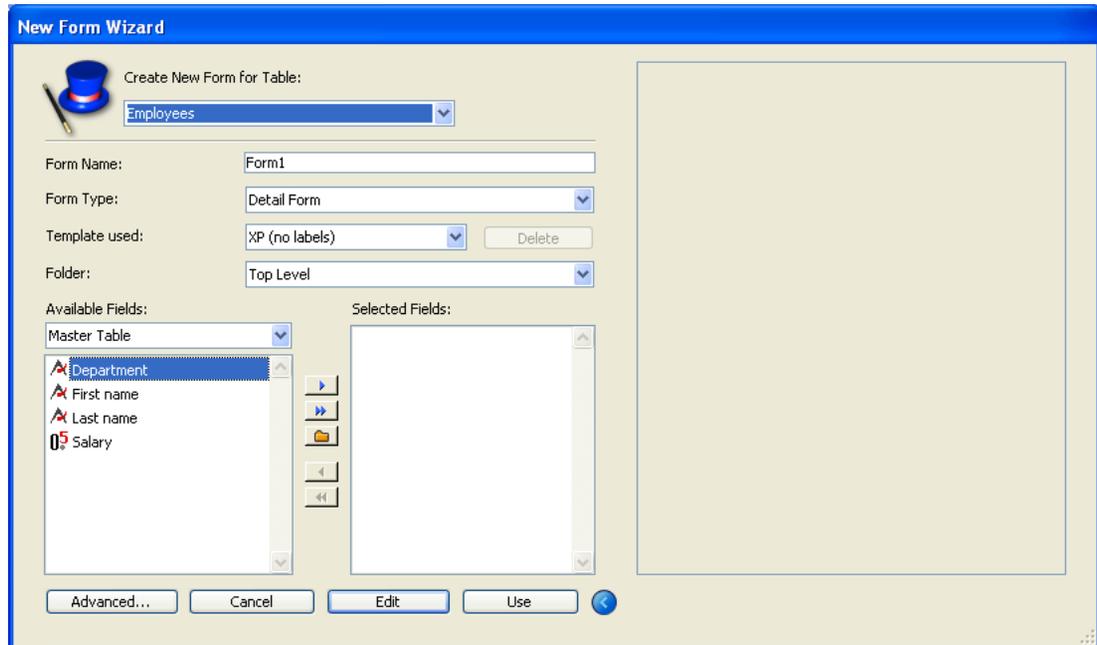
Make the following settings:
• Form Name: "Web Input"
• Form Type: Detail Form
• Template used: Web Aware
• Available Fields: Click the "Add all fields" button.

The wizard should now appear as follows:
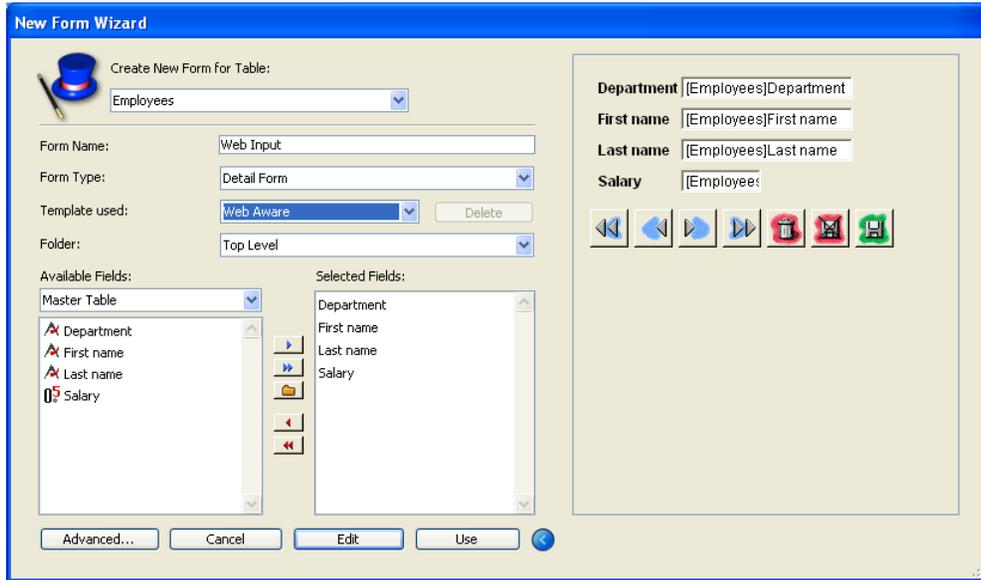
Click on the **Use** button.

On the server machine, you have one connected user:
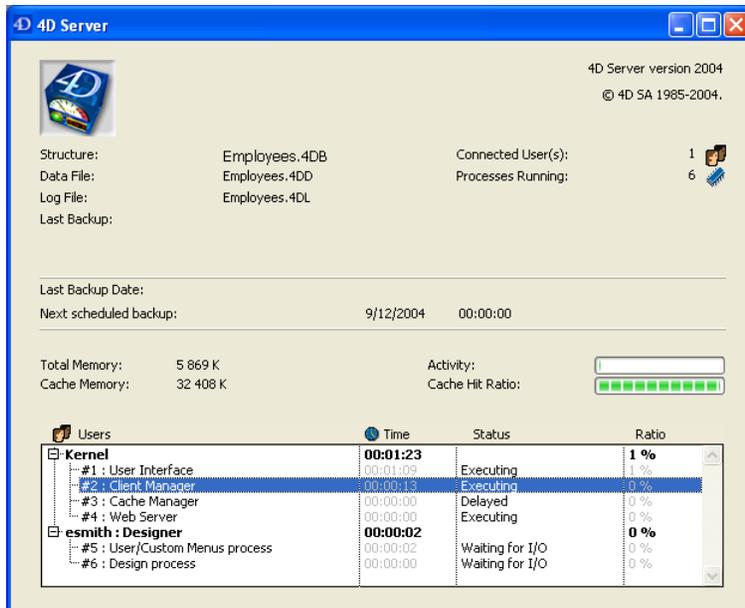
To check that the Web server is operating, start a browser from another machine and enter the IP address of the server machine in the "Address" area of the browser (usually of the 123.45.67.89 type). You should obtain the default home page of the Web server as follows:
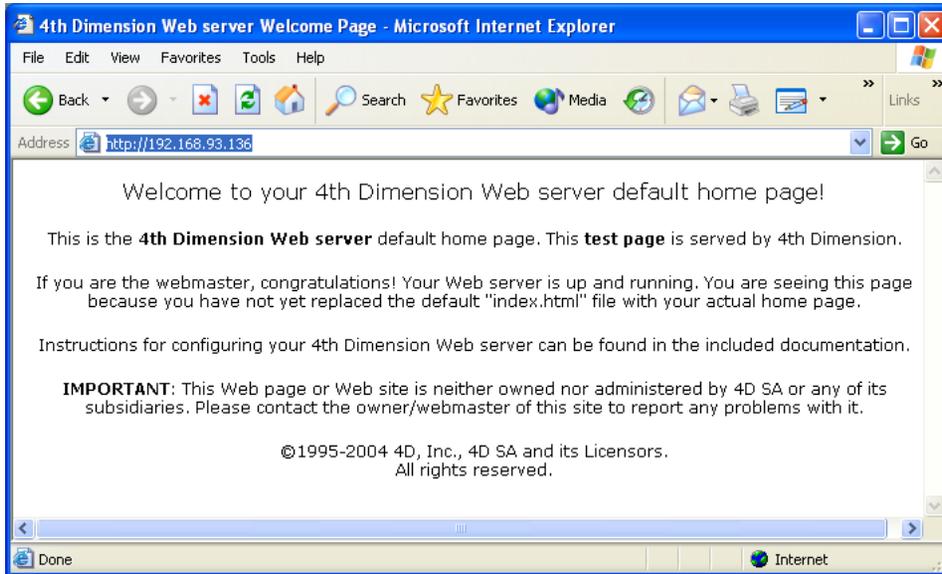
You are now going to configure the Web server so that it starts in contextual mode. In this mode, Web pages come from dynamic conversion of the 4D forms and menu bars. Contextual mode is particularly suited for fast publication of an existing database (for more information about contextual mode, refer to the section Using the Contextual Mode in the *4th Dimension Language Reference* manual).

On the server machine, choose **Preferences...** in the **Edit** menu, display the "Web (Server)/Configuration" page and make the following modifications:

• Check that the database is published at startup,
• Remove the default home page (in order to use the menu bar as the home page),
• Check the "Contextual Mode" option.

The page should appear as follows:
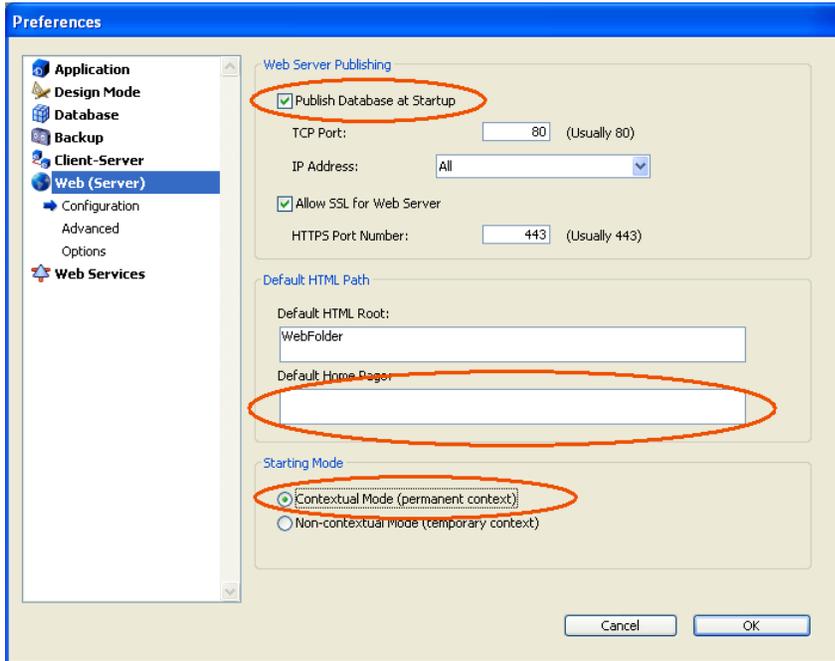
Return to the machine where the Web browser is running and refresh the server connection. You will now obtain the Web version of your custom menu bar (it may be necessary to empty the cache of the browser):
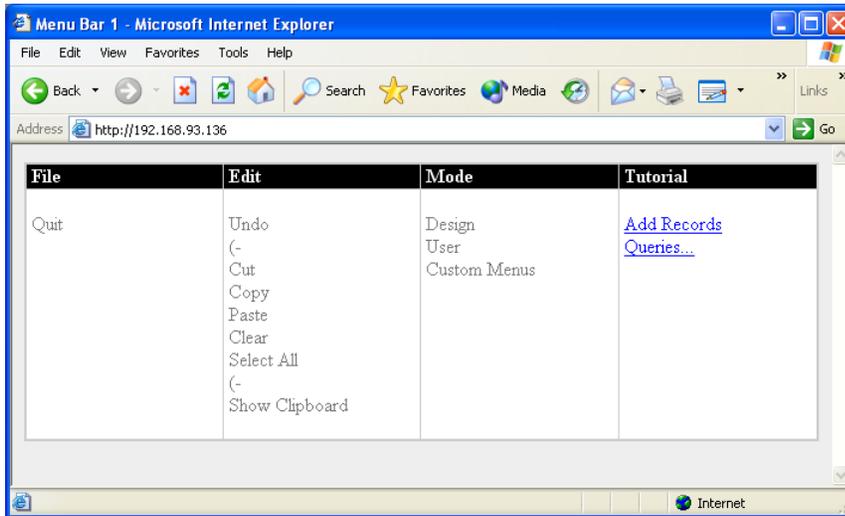
Note that you did not need to write any HTML code or implement any CGI module to get this result—4D Server is a Web server too!

On the server machine, the contextual Web connection is listed in the Process window:
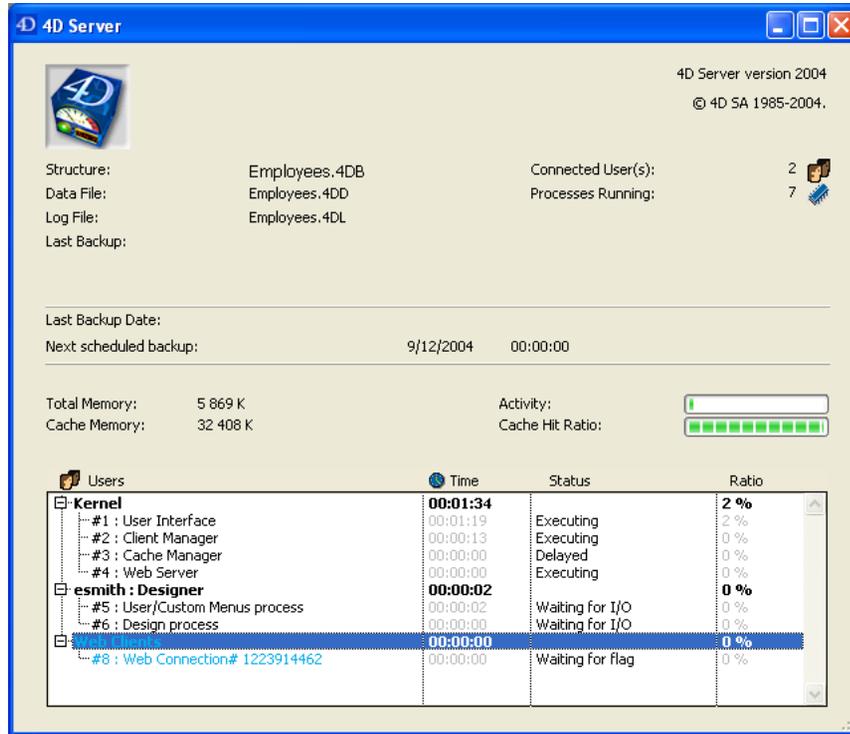
On the Web Browser side, click **Queries**. 4D Server translated its standard Query editor window into an HTML page—on the fly.

Define the query "Department is equal to Engineering".
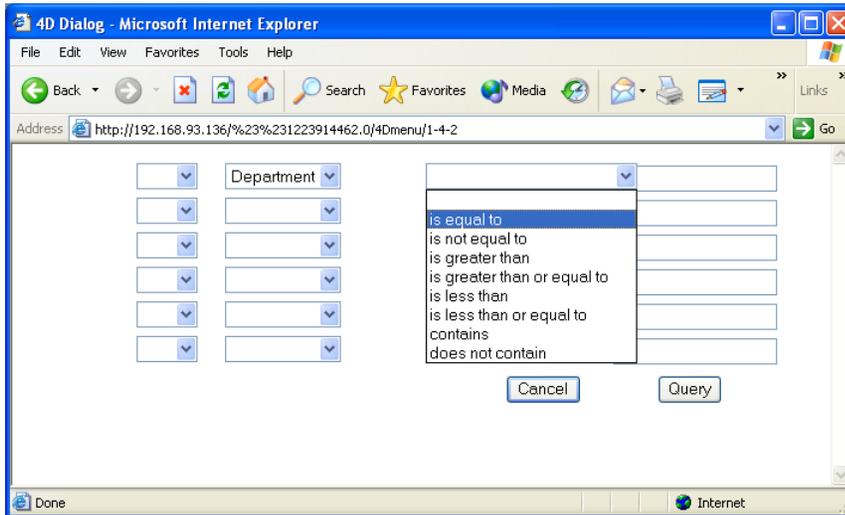
You get the results of the query:
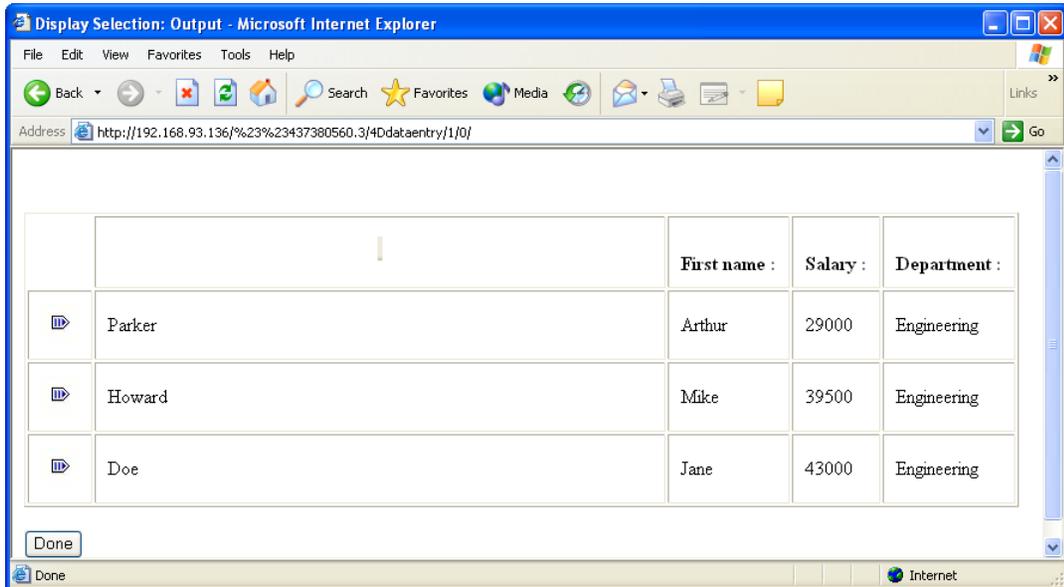
4D Server dynamically translates the 4D data into Web data and displays the list of records.

Using the icon in the left column, open the record "Parker, Arthur".

The input form you were using with 4D client appears as a Web page (transparently translated by 4D Server):
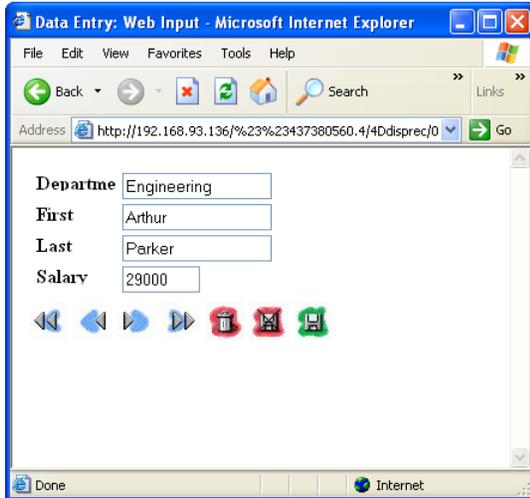
Change the first name to Ryan and validate the data entry. Go back to the list of records and note that the change shows up in the list of records:

Last but not least, go to the User environment on the 4D Client machine.

You see that the change is also available on the client:

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.
• You used the server database concurrently with 4D Client and via the Web!

To conclude, you created three custom applications (Windows, Macintosh and Web) while actually <u>developing</u> only one. In addition, if you need to use the database in single-user mode, you could open it directly, as is, with 4th Dimension or 4D Runtime.

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 *4th Dimension Design Reference* manual to learn about designing 4D databases.
• The *4th Dimension User Reference* manual to learn about the built-in editors used in this tutorial, such as the Query Editor and the Chart Wizard.
• The *4th Dimension 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.

3

Using 4D Server

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

The Welcome dialog box appears, giving you the choice of opening an existing database, creating a new one (either blank or based on a template), or restoring an archived database.
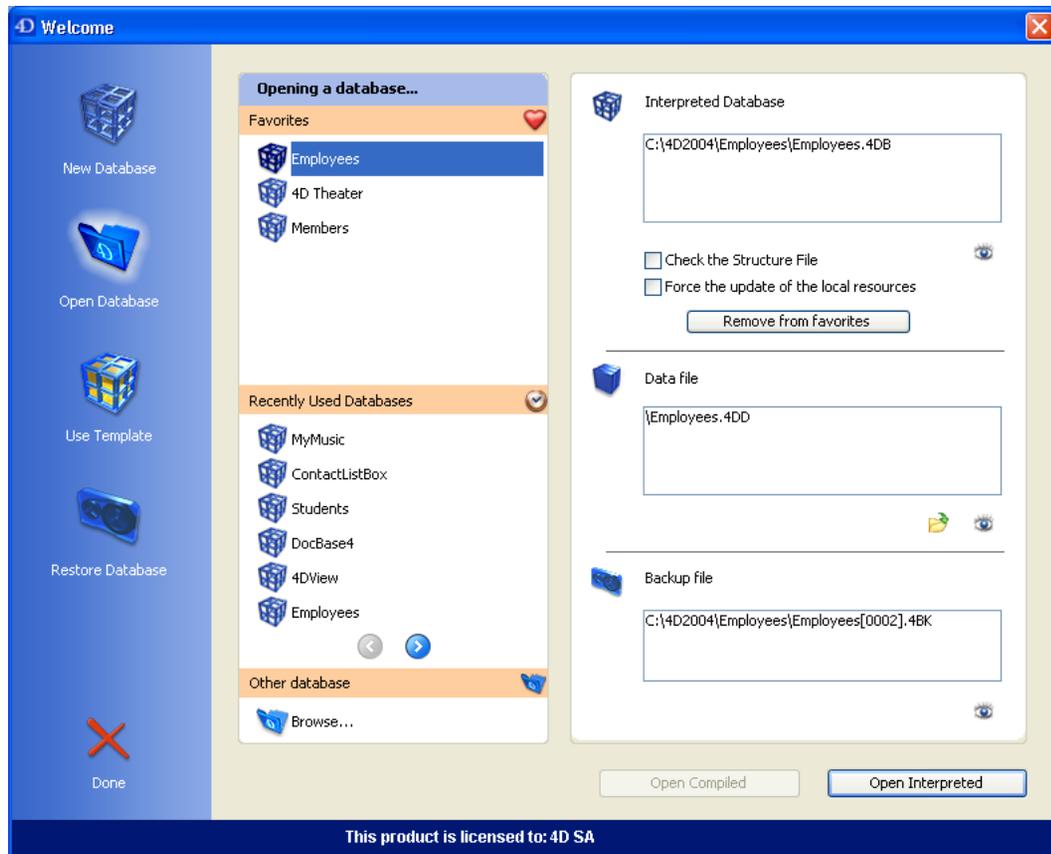
The operation of this dialog box is identical to that of the Welcome dialog box of 4th Dimension stand-alone version. For more information, refer to the *Design Reference* manual of the 4th Dimension documentation.

However, 4D Server provides an additional option in the Interpreted Database area: **Force the update of the local resources**. This option causes the systematic transmission of local resources to each client machine when it connects. Local resources are structural information related to the database that are stored on each client machine.
As a general rule, the updating of local resources on client machines is automatic whenever the database structure has been modified since the last connection. In most cases, this option is not needed. However, in certain specific cases, it may be necessary to force the update.
This option can also be set on each client machine, in which case it will only apply to the machine concerned. For more information, please refer to the Connecting to a 4D Server Database section.

To shut down the server:

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

The following dialog box is displayed on the server machine:
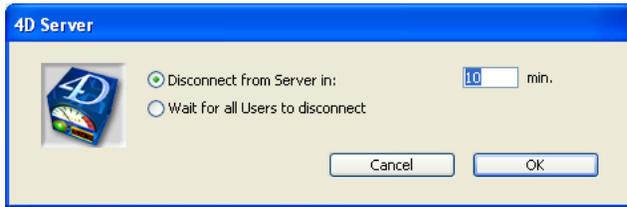
2. Enter the number of minutes in which you want the server to shut down, or choose the "Wait for all Users to disconnect" option.

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

• If you choose the "Disconnect from Server in *XX* min." option, the following window appears:
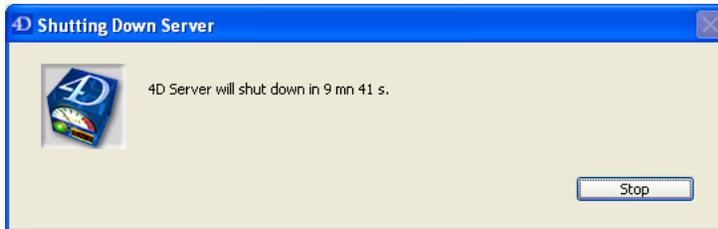
Simultaneously, the server alerts the client machines of the time remaining before disconnection:

Shutting Down window on the 4D Client machines

This dialog box 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.

• If you choose the "Wait for all Users to disconnect" option, the following window appears:
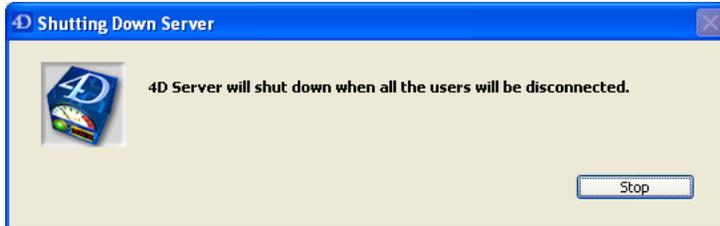
No dialog box appears on the client machines. The server will quit as soon as all the clients are disconnected.

Notes:
• In both cases, if no client is connected to the server when the shutting down window is validated, 4D Server quits immediately.
• To force 4D Server to quit with no delay, pass 0 min as the time limit.
• If you click **Cancel** in the "Shutting Down Server" window, the process of shutting down the server is cancelled.

The 4D Server window contains information that allows you to manage various aspects of your database system for administrative purposes:
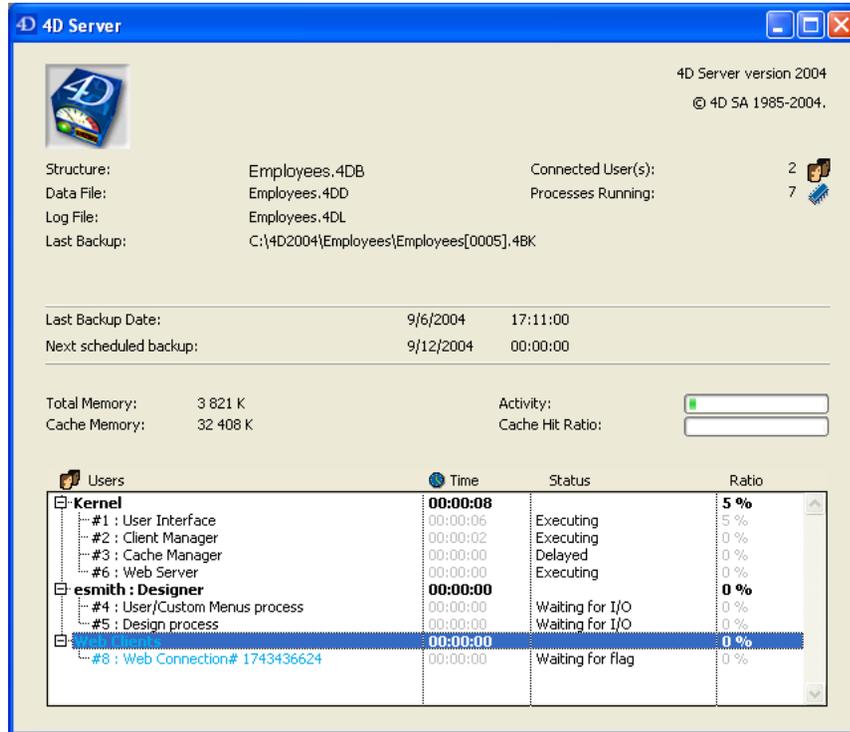
The server window is divided into two panes:

• The upper pane displays information about the files used by the server, back-ups and memory,

• The lower pane displays information about the clients connected to the server and the processes currently running.

Server Information Pane

The Server Information Pane describes general information about the server:

It contains the following data:

• **4D Server version number**: The version number of the 4D Server application currently in use.
• **Structure File**: The name of the structure file opened by 4D Server.
• **Data File**: The name of the data file associated with the structure file.
• **Log File**: The name of the log file created to keep track of database operations. A log file is created by default when the database is created.
• **Last Backup**: Access path and name of the last backup file of the database.
• **Connected Users**: The total number of users currently connected.
• **Processes Running**: The number of processes currently running. This number includes all processes (kernel, user connections, Web connections and stored procedures).
• **Last Backup Date**: Date and time when the last backup of the database was carried out.
• **Next scheduled backup**: Date and time of the next scheduled automatic backup. This information comes from the automatic backup configuration set in the Preferences dialog box of the database.
• **Total Memory**: The memory allocated to 4D Server's kernel. This value is set dynamically by the operating system depending on the use of the database and the available resources.
• **Cache Memory**: The amount of memory allocated to the cache. The size of the cache controls the amount of data that can be kept in memory to reduce the number of times the disk is accessed. Cache memory can be configured on the Database/Data Management page of the application Preferences. For more information about this, refer to the *Design Reference* manual of the 4D documentation.
• **Activity thermometer**: A thermometer indicating the current level of server activity. The Activity thermometer indicates the overall activity on the network. The more requests sent to the server, the higher the thermometer reading.
• **Cache Hit Ratio thermometer**: A thermometer indicating the level of cache use. The higher the thermometer reading, the more cache is being used. If the thermometer is always at a low level, you may want to consider decreasing the size of the cache.

The Cache Hit Ratio Window

The Cache Hit Ratio thermometer can also be used as a button. Clicking it displays the Cache Hit Ratio window:

This window provides additional thermometers indicating how the cache is being used. For each type of data that can be kept in the cache, there is a thermometer indicating how much of the data is present in the cache. A higher reading is considered advantageous, because accessing that type of data will require fewer occurrences of disk access (hits).

Records, Record Addresses and Index Page Addresses are progressively loaded in the cache when you work with records. Index Pages and Bit Tables are progressively loaded in the cache when you modify data by adding, modifying or deleting records.

Process Information Pane

The **Process Information Pane** lists the connected users and the processes that are currently running:

All client processes, the **connection processes** (excluding processes local to the workstation), are included in the list. You can hide and show processes for the kernel or for a particular user by clicking the cross (Windows) or arrow (Macintosh) to the left of the user name. When no clients are connected, only information about the server engine activity is displayed, as well as optional stored procedures.

The 4D Server engine is managed by four kernel processes:
• **User Interface**: Controls the user interface of the server window itself.
• **Client Manager**: Manages all connections to the server and ensures that each client is given processing time.
• **Cache Manager**: Manages the flushing of data to disk.
• **Web Server**: Manages Web connection attempts. Starting the Web Server process does not mean that you open an actual Web connection, it just means that you allow Web users to initiate Web connections.

For each process, the server window provides the following information:
• Name of the process
• Amount of time (in seconds) spent in each process since it started
• Status of the process
• Percentage of time spent by 4D Server in the process (ratio).

In order to differenciate processes, the Process window displays the processes as follows:
• Kernel processes in black
• User connection processes in black
• Web connection processes in cyan
• Stored procedures processes in blue
• 4D Open-based connection processes in green

Note: Processes being aborted may appear in red, for a short time.

See Also
Process Menu, Processes.

You can set various parameters concerning the network and the client-server communication on the "Configuration" page of the **Client-Server** theme in the application Preferences (accessible on both 4D Client and 4D Server):
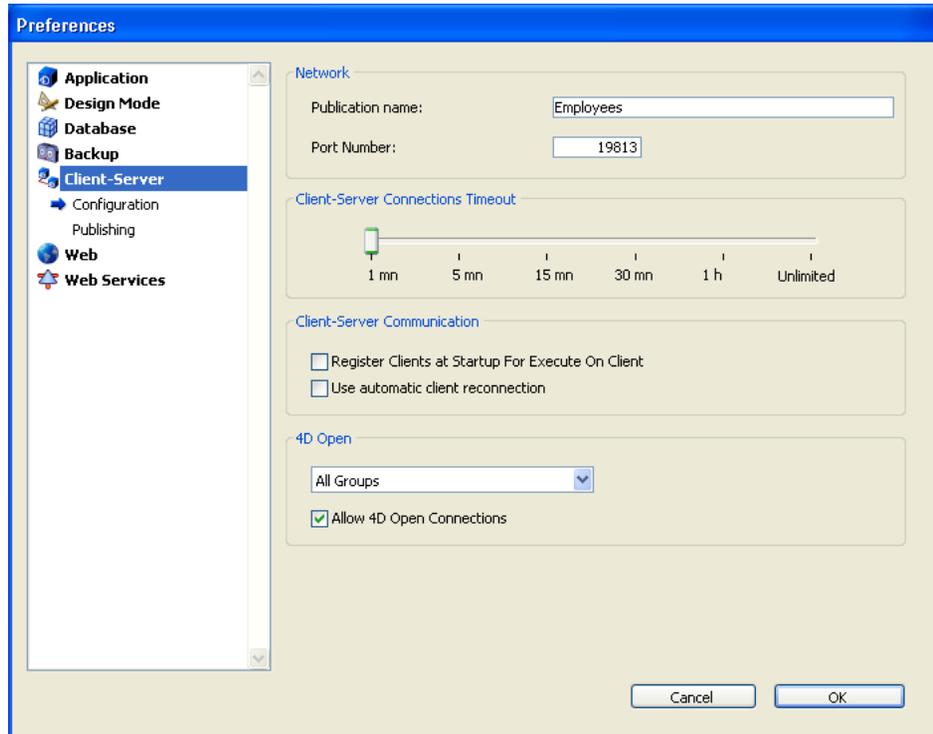
These parameters are detailed in this section.

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 Client 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 Client, it is automatically passed on to all the 4D Client 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.

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.

Register Clients at Startup For Execute On Client
When this option is checked, all the 4D Client machines connecting to the database can execute methods remotely. This mechanism is detailed in the section Registering 4D Client.

Use automatic client reconnection
This low-level function allows, in some specific configurations, the automatic reconnection of client machines in the case of unexpected disconnection.
When the option (global for all client machines) is checked, the automatic client reconnection feature is used as soon as a client connection is lost. There is only one attempt: if the reconnection fails, an error -10002 is returned. If it is successful, an alert dialog box is displayed and a notification icon appears in the Windows task bar.
For the automatic client reconnection feature to be available, the server-side timeout must be set to a value higher than 1 minute.

4D Open
4D Open is the API (Application Programming Interface) that allows non-4D Client applications to connect to 4D Server.
• **Allow 4D Open Connections**: When it is checked, this option gives the group specified by the menu the possibility of connecting to 4D Server from a 4D Open application.
• **4D Open Access menu**: This menu lets you specify the group of users allowed to connect to the 4D Server database via 4D Open, when the "Allow 4D Open Connections" option is checked.

Note: For security reasons, this option is unavailable on the server machine.

See also
Publishing preferences, Registering 4D Client.

You can set parameters concerning the security and publishing of databases by 4D Server on the "Publishing" page of the **Client-Server** theme in the application Preferences (accessible on both 4D Client and 4D Server):
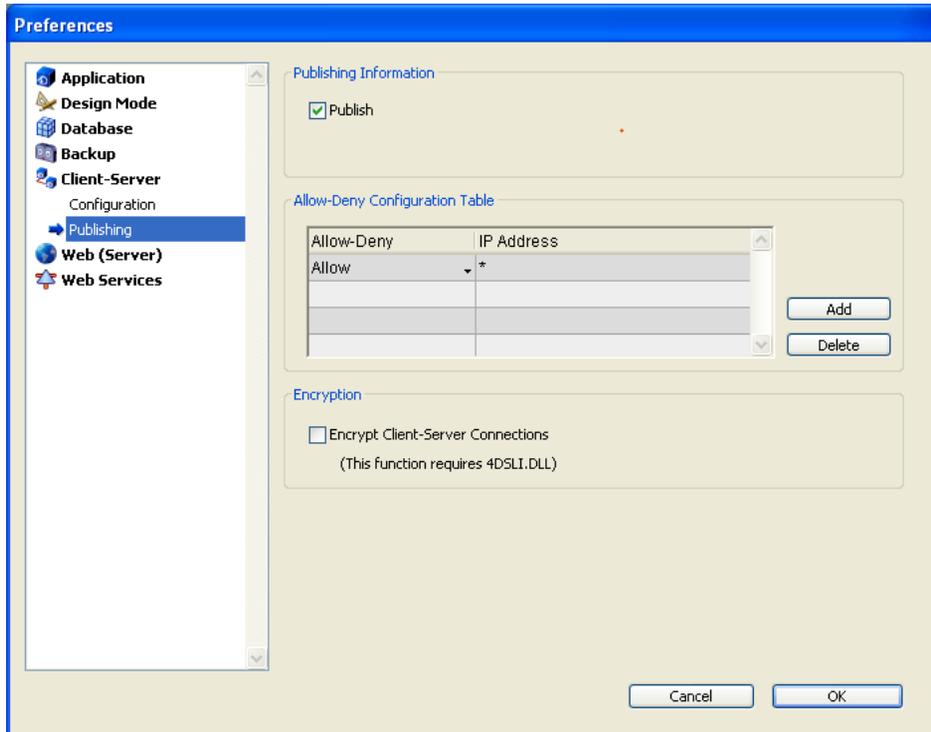
These parameters are detailed in this section.

Publish
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.

Allow-Deny Configuration Table

This table allows you to set access control rules for the database depending on 4D Client 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.

Encrypt Client-Server Connections

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

See also

Configuration Preferences, Encrypting Client/Server Connections.

You can configure the client/server connections so that 4D Server and 4D Client workstations communicate in secured mode.
The secured client/server communication is based on SSL (*Secured Socket Layer*) protocol.

SSL Protocol and Client/Server Connections

The SSL protocol has been designed to secure data exchanges between two applications —primarily between a Web server and a browser. The SSL protocol 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 SSL protocol, refer to section Web Services, Using SSL Protocol in the 4D *Language Reference*.

Regarding 4D Server and 4D Client, the SSL protocol allows communications security reinforcement. Key generation, 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 SSL protocol is inserted between the TCP/IP layer (low level) and the high level protocol.

To use SSL in a "classic" client/server architecture, make sure that on the 4D Server machine as well as on every 4D Client machine, the file 4DSLI.DLL is properly installed. This is the *Secured Layer Interface* dedicated to the SSL management. It should be placed in the [4D Extensions] folder of the 4D Server application/software package and 4D Client applications/software packages that publish the database.

This file is installed by default.

On the other hand, the encryption of 4D Server connections is not activated by default. You should activate this mode: open the "Client-Server/Publishing" page of the application Preferences dialog box and select the option **Encrypt Client/Server Connections** in the "Encryption" area:
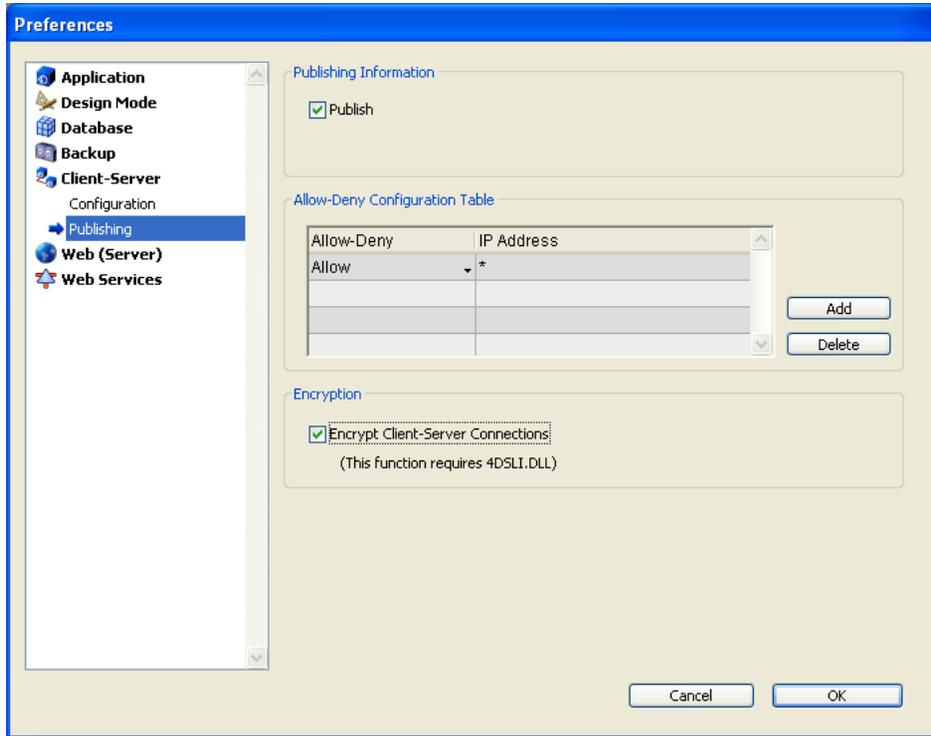
By default, the box is not checked.
You should then quit and relaunch 4D Server so that this setting is taken into account.

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

Secured Mode 4D Client Connection

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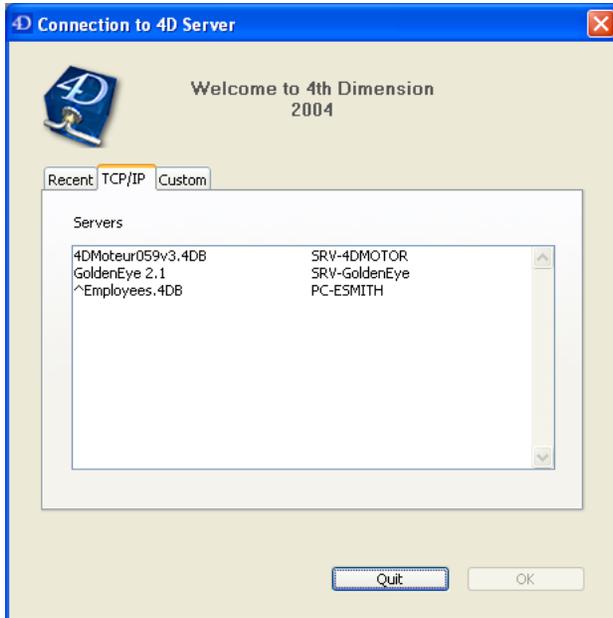
Note: When a database is not published dynamically in the Connection dialog box, the user can enter its name in the **Custom** page (see the Connecting to a 4D Server Database and Publishing preferences sections). In this case, a ^ (circumflex accent) must be placed before the database name if the database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

See Also

Using SSL Protocol.

It is possible to use a specific folder for transferring any customized elements (resource files, text documents, XML reference files, etc.) from the server to the client machines.

This folder, named **Extras**, must simply be placed next to the structure file.
4D Server automatically manages the modifications made to this folder and only transfers what is necessary. In addition, the contents of the folder are compressed in order to optimize network copying time.

Note: Avoid transferring very large volumes of data since, in this case, the duration of the connection initialization phase for client machines could be significantly increased.

On the client machine side, the Extras folder is downloaded to the same location as the other structure elements, i.e.:

System	Location of folders downloaded to the client machine
Mac OS	{Disk}:Library:Application Support:4D:DatabaseName_Address:Extras
Windows 2000 Windows XP	{Disk}:\Documents and Settings\User Name\Application Data \4D\DatabaseName_Address\Extras

The original hierarchy is reconstituted on each client machine.

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 new commands: New log file and INTEGRATE 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 database 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, the database remains accessible in read-only mode. 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 even 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 in read-only mode, 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 LOG FILE. These commands are described in the 4th Dimension *Language Referemce* 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 the 4D Extensions folder) 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 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, 4D Open for 4D, 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 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 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 either the operational or mirror database.

Operating scenario for a logical mirror

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

Operational machine	*Mirror machine*
Start up of the application, back up of the data file and activation (when necessary) of the log file. 4D creates the MyDatabase.4DL file.	
The application is exited.	
Copy of all the database files (log file included) onto the mirror machine.	
Start up of the application and beginning of operation.	Start up of the mirror application. 4D Server requests the log file: Select the MyDatabase.4DL file that was transferred from the operational database.

Decision made to update the mirror (for example, after a certain period of operation).	
Execution of the method containing the New log file command. The file saved is named MyDatabase[0001-0000].4DL.	
Sending of the MyDatabase[0001-0000].4DL file via programming to the mirror machine (using 4DIC, 4D Open for 4D, etc.).	
The database is operating.	Detection of a file that is waiting to be integrated. Execution of the method containing the INTEGRATE LOG FILE command in order to integrate the MyDatabase[0001-0000].4DL file. This file then becomes the new current log file.
Incident occurs on the machine; the database is unusable. Decision made to switch to the mirror machine.	
Copy of the current log file MyDatabase.4DL onto the mirror machine, via the usual destination folder.	
Repair of the machine...	Detection of a file that is waiting to be integrated. Execution of the method containing the INTEGRATE LOG FILE command in order to integrate the MyDatabase.4DL file. This file then becomes the new current log file.
	The database is operating.
The machine is repaired.	The database is exited.
Replacement of the database files by those of the mirror database.	
Start up of the application. 4D Server requests the log file: Select the MyDatabase.4DL file that was transferred from the mirror database.	Start up of the mirror application.
The database is operating.	

See also

INTEGRATE LOG FILE, New log file.

4

Using 4D Client

When you launch 4D Client, you are presented with a Connection dialog box containing three tabs.

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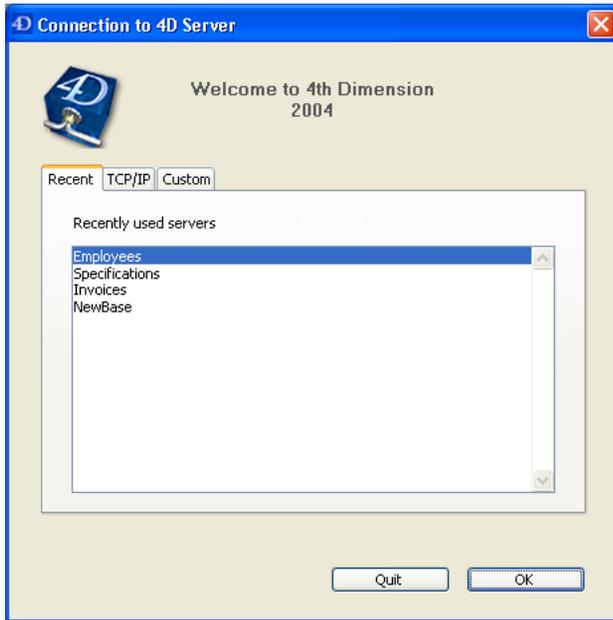
The **Recent** page memorizes the list of all 4D servers recently used. The list is sorted by alphabetical order. To connect to a server from this list, double-click on its name or select it and click the **OK** button.

To remove a server from the list, select it and hold down the **Del** or **Backspace** key.

"TCP/IP" tab
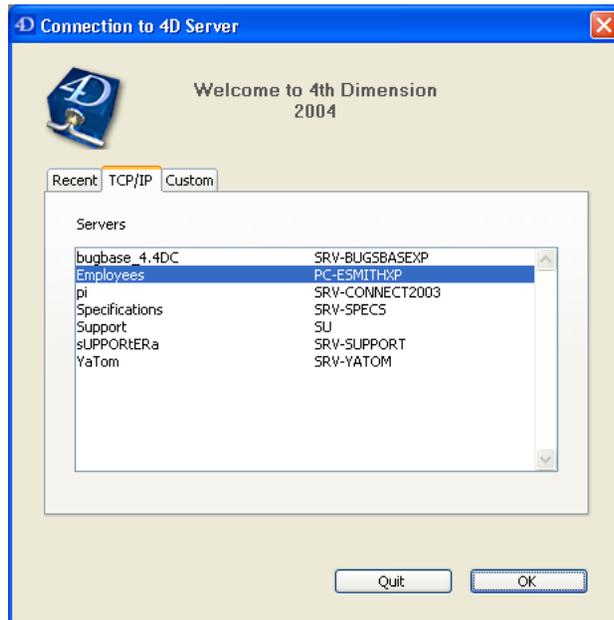
4D Server includes a built-in TCP/IP broadcasting system that publishes by default the name of the 4D Server databases published over the network. These names are listed on the **TCP/IP** Page of the connection dialog box.

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.

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 Publishing preferences section). In this case, the connection must be carried out manually on the "Custom" page.

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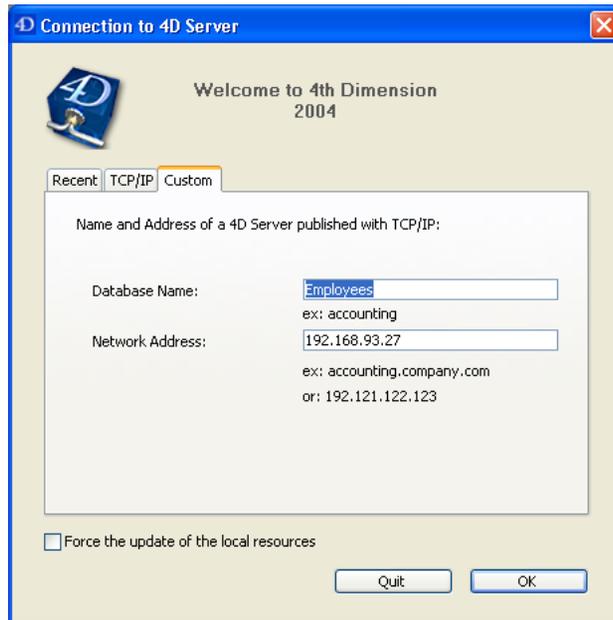
The **Custom** page allows assigning a published server on the network using its IP address and attributing it a customized name.

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 Publishing preferences section). In this case, the names do not appear in the **TCP/IP** page on the client side. 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 comma 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 application Preferences (see the Configuration preferences section).

Note: If a database was selected in the **Recent** or **TCP/IP** 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 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.
This option can also be set on the server machine; in this case, it will apply to all the client machines. For more information, refer to the Creating a New 4D Server Database section.

You can create a path document that enables you to access a 4D Server database directly from the client machine.

When you drag and drop a path document onto the 4D Client application (or double-click the file), you access the 4D Server database directly, without having to specify it in the connection dialog box.
In addition, the connection file can contain the user password (encrypted). In this case, access to the 4D Server database is immediate.

To create a path file to a database:

1. In the User Identification dialog box of 4D Client, select or enter (depending on the current configuration) the name of the user whose access you want to save, then the password.
These parameters must be valid in order to be able to access the dialog box for saving the connection file.
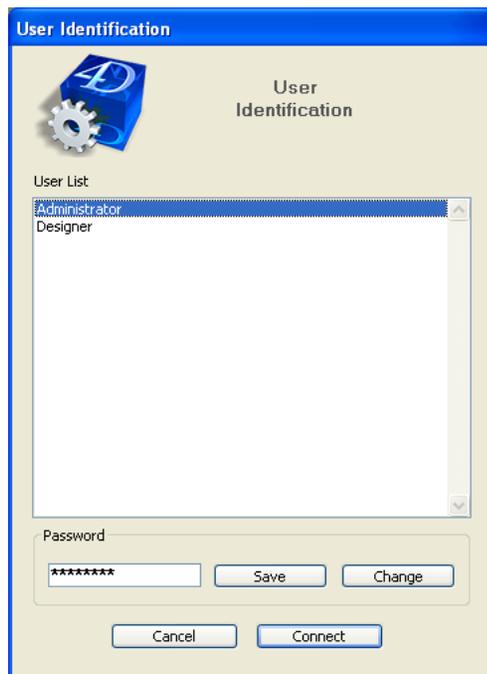
Note: This dialog box appears just after the 4D Server database is selected, when the password protection system is activated.

2. Click on the Save button.
The following dialog box appears:

You can save the path with or without the password.

• **With password**: In this case, the database access path and password (encrypted) of the user are stored in the document. On startup of 4D Client, no dialog box appears and the connection is immediate.

• **Without password**: In this case, the database access path is stored but the user must enter their password in order to open it.

3. Click on either the With password or Without password button.
A standard save file dialog box appears, which lets you specify the name and location of the file on your disk. The extension for path documents is ".pth".
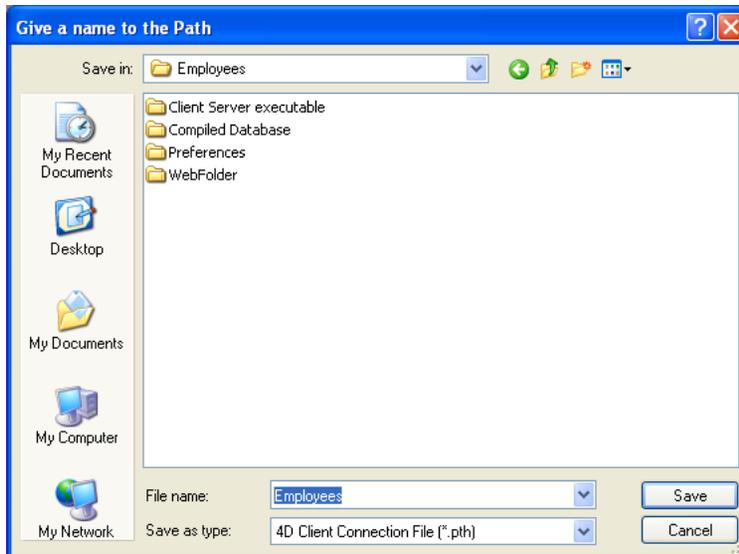
4. Choose the name and location of the file and validate the save dialog box.
The 4D Client connection file is stored on disk.

MyData.PTH

5

4D Server Menus

File
Backup…
Check Log File

Register Current Database as Service
Unregister Current Database
Unregister all Server Services

Quit   Ctrl+Q

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:

4D Backup

Backup destination: C:\4D2004\Employees\
Last Backup Information   09/06/04   17:11:00

Preferences…   Cancel   Backup
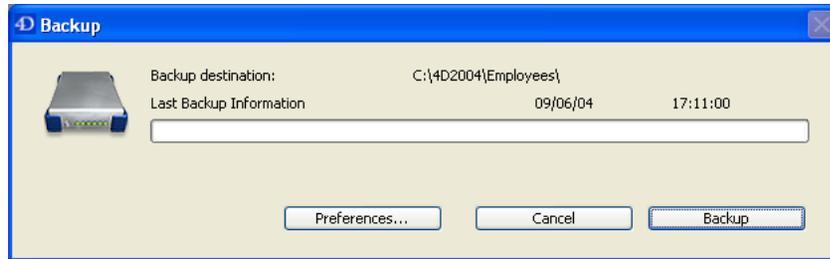
• The **Backup** button immediately launches a back-up that takes the parameters set in the
Preferences dialog box of the application into account (files to be backed up, location of
archives, number of sets kept, etc.).
• The **Preferences** button opens the "Backup" theme of the Preferences, which 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 4th Dimension documentation.

Check Log File
The log file stores all the operations carried out on the data of the database (addition,
modification, deletion) — whether they were carried out by a user (from 4D Client), a
stored procedure, a plug-in, a Web browser or yet again by 4D Open. In the event of an
incident on the database, restoring the last back-up and integrating the log file ensure
that you will recover the database in the exact state it was in prior to the incident.

The **Check Log File** command displays a window that lets you view the contents of the current log file:

	Action	Table	User	Date	Hour
57	Modification	Customers	Designer	8/24/2004	12:16:19 PM
58	Modification	Customers	Designer	8/24/2004	12:16:30 PM
59	Modification	Customers	Designer	8/24/2004	12:16:48 PM
60	Deletion	Customers	Designer	8/24/2004	12:16:51 PM
61	Deletion	Customers	Designer	8/24/2004	12:17:13 PM
62	Deletion	Customers	Designer	8/24/2004	12:17:13 PM
63	Deletion	Customers	Designer	8/24/2004	12:17:13 PM
64	Addition	Customers	Designer	8/24/2004	12:17:17 PM
65	Addition	Customers	Designer	8/24/2004	12:17:17 PM

This window is useful for analyzing the use of a database or detecting the operation(s) at the origin of errors or malfunctioning. More particularly, it lets you check the operations carried out by each client machine.

This window also lets you go back through the operations carried out in the database (**Roll back** button).

For more information about these functions, refer to the *User Reference* manual of the 4th Dimension documentation.

Register Current Database as Service
Unregister Current Database
Unregister all Server Services

4D Server can be launched as a Service under Windows and under Mac OS X.

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.

Note : 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**. 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.

Warning: Be sure to use a valid account when you open the session. In addition, this user account must have access to a printer, otherwise an error message will be displayed. Access to the services settings is generally via the Administrative Tools/Services of the Control Panel.

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.

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).
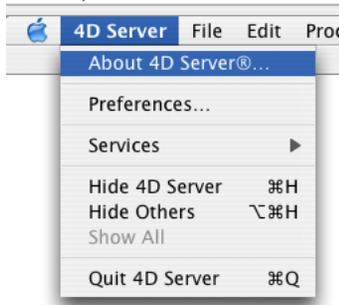
See Also
Data Menu, Edit Menu, Help Menu, Process Menu, Web Server Menu.
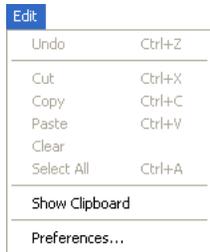
The **Edit** menu of 4D Server includes standard copy/paste commands, the **Show Clipboard** command, etc.

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.

Note: Under Mac OS, the **Preferences...** command is found in the **4D Server** menu (application menu).
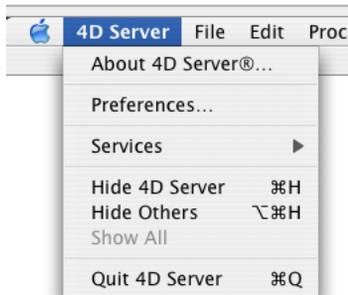
See also
Data Menu, File Menu, Help Menu, Process Menu, Web Server Menu.

version 2004 (Modified)

Abort

Warning: The **Abort** menu command is provided only for administration and debugging purposes. You should use it only when a client cannot terminate a process by itself.

The effect of the Abort command depends on the type of the process currently selected in the Process window:

• If a connection process is selected, this command aborts the process.
• If a User list is selected, this command aborts all processes belonging to this user and then removes the user from the list. This automatically disconnects the user from the server.
• If a stored procedure is selected, this command aborts the stored procedure.
• If the Stored Procedures list is selected, this command aborts all the stored procedures.
• If a Web connection process is selected, this command aborts the Web connection process.
• If the Web Clients list is selected, this command aborts all the Web connection processes.

You can abort all processes except the kernel processes. If a kernel process is selected, the Abort command is disabled.

If you attempt to use an aborted process on a 4D client machine, a warning message informs you that the process is no longer available. For example, the following alert is displayed when the Abort command has been applied to a user:

Error -10001, The actual connection to the database has been disrupted.

Note: Connection errors are listed in the section Network Errors of the 4D *Language Reference* manual.

If a workstation accidentally ends its connection to the server, it takes the server a few seconds to determine that the client has unexpectedly exited. The server then performs the following:

• Unlocks any locked records
• Cancels any transactions not yet canceled or validated
• Aborts any client processes
• Removes the user from the list of users in the 4D Server Process window

Consequently, you do not need to apply the Abort command to the user; 4D Server automatically cleans up the user list for you.

Expand All/Collapse All

These commands allow you to either expand or collapse processes for all users listed in the Process window.

Hide Process Window/Show Process Window

This command allows you to hide or show the Process window.

Show Runtime Explorer/Hide Runtime Explorer

This command allows you to hide or show the 4D Server Runtime Explorer window.
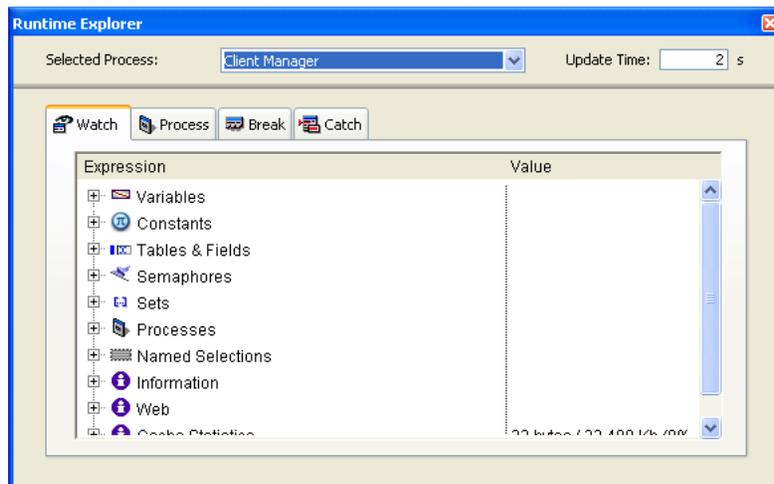
The Runtime Explorer enables you to view the status of the database various structural elements and to check that the available resources are correctly managed. The Runtime Explorer is particularly useful while developing or analyzing a database.

The Runtime Explorer window contains four pages that can be accessed by clicking on the following tabs: Watch, Process, Break and Catch. The Runtime Explorer works the same way in 4D Server and 4th Dimension. For more information, please refer to the *4th Dimension Design Mode* manual.

Trace

The **Trace** command can be applied to the following processes:
• Connection process for tracing Triggers executing within the context of that process
• Web connection process for tracing the code executed on the server machine in response to the requests sent by the Web browser
• Stored Procedure for tracing the project method executed as stored procedure.

Choosing the **Trace** command displays a Debugger window for the selected process as soon as this process starts executing code. You can also display the Debugger window for a process by calling the TRACE command from within a method executing in that process (see screen shot below).

The interesting point here is that 4D Server "memorizes" the Trace request:
• If the process is currently executing code, the Debugger immediately appears for that process.
• If the process is not currently executing code (i.e., the process is waiting for an event in data entry mode), the Debugger will appear right after the process resumes executing the code.

For additional information about the **Trace** command, refer to the *4th Dimension Language Reference* manual.

See Also
Data Menu, Edit Menu, File Menu, Help Menu, Web Server Menu.

Data
Segments...

The **Segments...** command displays the **Segments** dialog box:
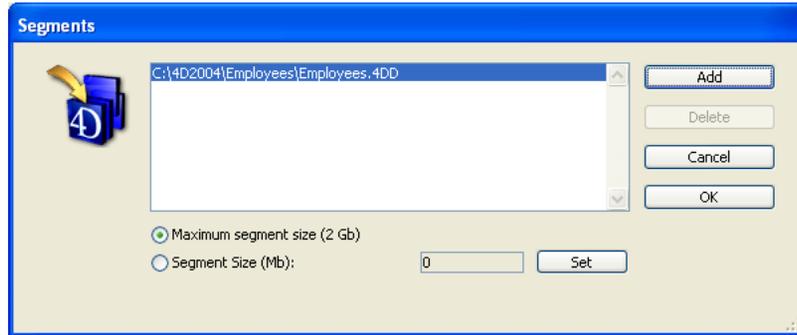
You can either segment a data file at the time you create the database or after you begin to use it. You should segment a new data file if you expect the data file to become very large. Segmenting a data file allows a virtually unlimited amount of data to be stored.

Note: You do not need to create any data segments unless you have more than two gigabytes of data or your hard disk cannot accommodate the size of your data file.

When segmenting a data file, you divide the data file into segments and then specify the volume on which each segment is to be stored. For example, four gigabytes of data could be divided into two segments of two gigabytes each. Each segment can be limited in size, so you can reserve space on your hard disk for other files and thus avoid a completely full volume.

To increase the size of an existing data file beyond two gigabytes, you can add data segments, each of which can contain up to two gigabytes of data. You use the Segments dialog box to create segments for existing data on the server machine.

For a detailed description about segmenting a data file, refer to the *4th Dimension Design Reference* manual.

See Also
ADD DATA SEGMENT, Edit Menu, File Menu, Help Menu, Process Menu, Web Server Menu.

Web Server
  Start Web Server
  Stop Web Server

This menu lets you start and stop the 4D Web server at any time.

Note: Under Mac OS, this menu may not work if the Web server is started using a temporary root session (see the Web server configuration and connection management section in the *4th Dimension Language Reference* manual).

The 4D Web Server can be started in three different ways:

• Using the **Web Server** menu from the menu bar of 4D Server or the **Run** menu of 4D Client. These menus allow you to start and stop the Web server at your convenience.

• Automatically, by publishing the database each time it is opened. The **Publish Database at Startup** option on the "Web>Configuration" page of the application Preferences should be checked. Once this is done, the database will be published on the Web automatically each time you open it with 4th Dimension or 4D Server.

• Programmatically, by calling the command START WEB SERVER.

See Also
Data Menu, Edit Menu, File Menu, Help Menu, Process Menu, Web server configuration and connection management, Web Server, Overview.

The first commands of the **Help** menu allow you to view on-line help for 4D Server.

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 *4th Dimension Installation Guide* manual.

The **About 4D Server...** command displays the About dialog box. This dialog box provides information regarding the 4D Server application (name and location of application files), the running database and system files. You can display different information using the dialog box tabs:
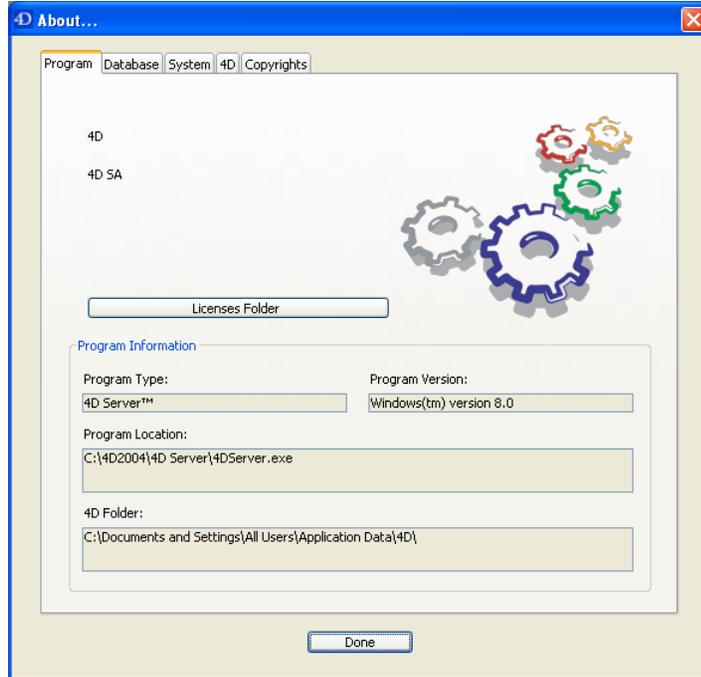
In addition, the **4D** page allows accessing to the 4D Server on-line registration feature.

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

See Also
Data Menu, Edit Menu, File Menu, Process Menu, Web Server Menu.

6

4D Server and the 4D Language

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

• Triggers
• Stored procedures
• 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 preventaccidental 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 <u>on the machine where the database engine is actually located</u>. On 4D Server, triggers are executed within the context of the acting process <u>on the server machine</u>, not on the client machine. With 4D Server, a trigger executes within the context of the user processes that invokes the database operation. The trigger, however, has no access to the process variables of the user process.

For more information about triggers, see the section Triggers of the *4th Dimension Language Reference* manual.

Stored Procedures

A stored procedure is project method executing a process method in a process running on the server machine (or, starting from 4D Server version 6.5, on any client machine), instead of on the client machine which has launched the method. See the section Stored Procedures.

Database Methods

Four database methods are executed only on the server machine:

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

Four 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 Backup Startup Database Method
• On Backup Shutdown Database Method

See the corresponding sections in this manual and in the *4th Dimension Language Reference* manual for more information on the database methods.

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.

• Each stored procedure has its own table of process variables. When running a compiled database, the process variable table definition is common between between all stored procedures and user process running on all clients machines, each process having its own instance.

• In interpreted mode, the database methods and triggers can create and use process variables dynamically during each phase of execution. This is not true in compiled mode. When running a compiled database, database methods and triggers share only one common table of process variables (whose definition is identical to that of the other processes).

4D Server and Sets

With 4D Server, interprocess and process sets are maintained on the server machine, while local sets are maintained on the client machines. For more information, see the section 4D Server and Sets.

As explained in the section Sets of the *4th Dimension Language Reference* manual, you can work with interprocess, process, and local sets:

• **Process sets**: A process set can only be accessed by the process in which it has been created. UserSet and LockedSet are process sets. Process sets are cleared as soon as the process method ends. Process sets do not need any special prefix in the name.
• **Interprocess sets**: A set is an interprocess set 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 Sets/Client Sets**: The name of a local/client set is preceded by the dollar sign ($). **Note**: Although its name does not begin with a **$**, the UserSet system set is a local/client set.

With 4D Server, interprocess and process sets are maintained on the server machine, while local sets are maintained on the client machines.

Tip: Usually, you will use interprocess and process sets because they optimize set handling in Client/Server architecture.

4D Server and the Sets Commands

This section describes the behavior of the Sets commands in Client/Server architecture when they are executed on the Client machine:

• CREATE EMPTY SET
The empty set is created on the server machine. A local set is then copied over the network from the server machine to the client machine. An interprocess or a process set stays and is maintained on the server machine.

• CREATE SET
• CREATE SET FROM ARRAY
The set is created on the server machine. A local set is then copied over the network from the server machine to the client machine. An interprocess or a process set stays and is maintained on the server machine.

• USE SET
A local set is first copied over the network from the client machine to the server machine, then it is used on the server machine to change the selection of the table. An interprocess or process set is used locally on the server machine to change the selection of the table.

• SAVE SET
A local set is saved locally on the client machine.
An interprocess or process set is first copied <u>over the network</u> from the server machine to the client machine, then is saved locally on the client machine

• LOAD SET
A local set is loaded from the disk locally on the client machine.
An interprocess or process set is first loaded from the disk locally on the client machine, then is copied <u>over the network</u> from the client machine to the server machine.

• CLEAR SET
• ADD TO SET
• REMOVE FROM SET
• Is in set
• Records in set

These five commands access a local set locally on the client machine. With an interprocess or process set, a request is sent <u>over the network</u> to the server machine to get the information or perform the action.

• DIFFERENCE
• INTERSECTION
• UNION

These three commands require the three set parameters to be on the same machine. Consequently, they must be all local sets or none of them must be local.

• COPY SET

Using COPY SET, you can copy any set into another one. For example, you can copy a local set into an interprocess or process set. In this case, the set is copied <u>over the network</u> from the client machine to the server machine.

4D Server and the UserSet

4D Client creates the UserSet as a local set (although its name does not begin with a $) for optimizing the creation of a set according to the user actions performed in a MODIFY SELECTION or DISPLAY SELECTION form.

If you plan to pass UserSet as a parameter to DIFFERENCE, INTERSECTION or UNION when the other parameters are interprocess or process sets, you must first copy UserSet (a local set) into an interprocess or process set and use that set with the command.

Example:

```
ALL RECORDS ([aTable])
    ` Let the user select some records
MODIFY SELECTION ([aTable];*)
    ` Check if the user has selected some records
If (Records in set("UserSet")>0)
        ` Query the records to be excluded
    QUERY([aTable];[aTable]aFlag#0)
        ` Create a set from the resulting selection
    CREATE SET([aTable];"To be excluded")
    If (Application type = 4D Client)
            ` UserSet is local set, copy it to a non-local set
        COPY SET ("UserSet";"UserSelection") ` ← Copied over the network
            ` Call DIFFERENCE passing 3 non-local set parameters
        DIFFERENCE ("UserSelection";"To be excluded";"UserSelection")
    Else
            ` Call DIFFERENCE
        DIFFERENCE ("UserSet";"To be excluded";"UserSelection")
    End if
    CLEAR SET ("To be excluded")
    USE SET("UserSelection")
    CLEAR SET ("UserSelection")
End if
```

The same thing can be achieved with the following code:

```
ALL RECORDS ([aTable])
    ` Let the user select some records
MODIFY SELECTION ([aTable];*)
    ` Check if the user has selected some records
If (Records in set("UserSet")>0)
        ` Query the records to be excluded
    QUERY([aTable];[aTable]aFlag#0)
    If (Application type = 4D Client)
            ` Create a local set from the resulting selection
        CREATE SET([aTable];"$To be excluded") ` ← Copied from Server to Client
            ` Call DIFFERENCE passing 3 local set parameters
        DIFFERENCE ("UserSet";"$To be excluded";"UserSet")
    Else
            ` Create a non-local set from the resulting selection
        CREATE SET([aTable];"To be excluded")
            ` Call DIFFERENCE
        DIFFERENCE ("UserSet";"To be excluded";"UserSelection")
    End if
    CLEAR SET("$To be excluded")
    USE SET("UserSet") ` ← Copied from Client to Server
End if
```

In the first example, three sets are created and one is copied over the network. In the second example, two sets are created and two are copied over the network. Depending on your needs, choose a solution similar to one of these examples.

4D Server and the LockedSet

The LockedSet is a process set created and maintained on the server machine.

See Also

4D Server and the 4D Language, COPY SET, Sets.

version 6.8 (Modified)

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 4D Client connects to the server, use the On Startup database method.

See Also

Database Methods, On Server Shutdown Database Method, SP-Based Import (example).

The On Server Shutdown Database Method is called once on the server machine when you shut down 4D Server and thereby quit the database. The On Server Shutdown Database Method is NOT invoked by any 4D environment other than 4D Server.

A server database is exited if the user selects the menu command **Quit** on the server or if a call to the QUIT 4D command is issued by a stored procedure.

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 4D Client stops connecting to the server, use the On Exit Database Method.

See Also

Database Methods, Methods, On Server Startup Database Method.

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 client workstation. The On Server Open Connection Database Method is NOT invoked by any 4D environment other than 4D Server.

4D Client
With 4D Client, the On Server Open Connection Database Method is called each time:
• 4D Client connects (because the User environment process starts)
• 4D Client opens the Design environment (because the Design process starts)
• 4D Client starts a non-local process, using the New Process command
• A non-local process is started by a menu or using the Execute Method dialog box

In each case with 4D Client, two processes are started—One on the Client machine, one on the Server machine. On the Client machine, the process executes code and send requests to 4D Server. On the Server machine, the process maintains the database environment for the client process (i.e., current selections for user processes) and replies to requests sent by the process running on the Client machine. This is why the process running on the server is said to be a **connection process**—on the Server machine, over the network, via a connection, it does what the process running on the Client machine would also do if you were running as single-user rather than Client/Server.

4D Insider
When you connect to 4D Server from 4D Insider, a connection process is started on the Server machine in order to maintain a working environment for 4D Insider. This process replies to the requests sent by 4D Insider.

4D Open-based Applications
Each time a 4D Open-based application initiates a connection to 4D Server, a connection process is started on the Server machine. This process replies to the requests sent via 4D Open and maintains the database context of the connection (i.e., current selections).

Important: Web 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. For more information, see the description of this database method in the *4th Dimension 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 connection processes. They execute code on the Server machine, but do not reply to requests exchanged by 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 connection process that provoked the call to the method.

For example, if a 4D Client connects to a 4D Server interpreted database, the User Environment, 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 User Environment connection process, the second time within the client registration process, and the third time within the Design connection process. If the three process 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 connection 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 connection process has not yet been named (PROCESS PROPERTIES will not at this point return the name of the connection 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.

With an interpreted database, 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 connection process. Because the On Server Close Connection Database Method will eventually be invoked within the same connection process, you may think that you can maintain information between the two methods using process variables. **This will not work in compiled mode**.

With a compiled database, the On Server Open Connection Database Method shares, with other database methods as well as triggers, a common process variable table maintained on the Server machine. This architecture has two purposes: to allow compiled code to run and to reduce memory consumption. First, you can access any process variable from within a database method or a trigger. The process variables need to be there. Second, creating one process table for each database method or trigger phase would consume memory and initialization time. Conclusion: Do NOT rely on process variables when executing On Server Open Connection Database Method and On Server Close Connection Database Method. Use data stored in interprocess variables or in table.

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	Network protocol ID number used internally by 4D Server

Note: Since 4D Server version 6.8, the $3 parameter always returns 2 or 29.

These three 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 connection 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 three 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.

Examples

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:

Server Log	
Log ID	L
Log Date	D
Log Time	H
Exit Date	D
Exit Time	H
User ID	L
Connection ID	L
NC ID	L
Process ID	L
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
[Server Log]NC ID:=$3
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]NC ID=$3;*)
QUERY([Server Log]; & ;[Server Log]Process ID=0)
    ` Save the Exit date and time
[Server Log]Exit Date:=Current date
[Server Log]Exit Time:=Current time
    ` Save the process information
[Server Log]Process ID:=Current process
PROCESS PROPERTIES([Server Log]Process ID;$vsProcName;$vlProcState;$vlProcTime)
[Server Log]Process Name:=$vsProcName
SAVE RECORD([Server Log])
```

Here are some entries in the [Server Log] showing several 4D Client connections as well as a 4D Insider connection:

Log ID	Log Date	Log Time	Exit Date	Exit Time	User ID	Connection ID	NC ID	Process ID	Process Name
1	05/24/97	16:24:03	05/24/97	16:25:50	17892880	17901424	4	5	User/Custom Menus process
2	05/24/97	16:24:06	05/24/97	16:24:18	17892880	17902504	4	6	Design process
3	05/24/97	16:24:33	05/24/97	16:24:44	17892880	17902504	4	6	Design process
4	05/24/97	16:25:07	05/24/97	16:25:32	17903568	17903968	4	6	4D Insider
5	05/24/97	16:26:07	05/24/97	16:26:50	17892880	17901424	4	6	User/Custom Menus process
6	05/24/97	16:26:11	05/24/97	16:26:51	17892880	17902504	4	5	Design process

Server Log: 6 of 8

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
```

See Also

Database Methods, On Server Close Connection Database Method.

The On Server Close Connection Database Method is called once on the Server machine each time a connection 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	Network protocol ID number used internally by 4D Server

Note: Since 4D Server version 6.8, the **$3** parameter always returns 2 or 29.

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 **connection processes**, see the description of this database method.

Example

See the first example for On Server Open Connection Database Method.

See Also

Database Methods, On Server Open Connection Database Method.

7

Stored Procedures

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.

Using a 4D 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 *4th Dimension Language Reference* manual).

You can do the same with 4D Server, on a 4D Client machine. In addition, using the command Execute on server on the server machine, you can start a user process in which you are 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 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 one or several client machines invoke 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 4D Client machines.

In the same way that 4th Dimension and 4D Client provide a multi-tasking environment to user processes running on a workstation, 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.

What a Stored Procedure Does?

Aside from data entry for stored procedures executed on the server, almost everything said in the *4th Dimension 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 and so on. Just think that instead of doing something on a 4D Client 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 new 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 catched through a method installed in the ON ERR CALL command.

ACCUMULATE
ADD RECORD
ADD SUBRECORD
APPEND MENU ITEM
BREAK LEVEL
CALL PROCESS
CHANGE LICENSES
Count menu items
Count menus
CREATE DATA FILE
DELETE MENU ITEM
DISABLE MENU ITEM
DISPLAY SELECTION
EDIT ACCESS
ENABLE MENU ITEM
FILTER EVENT
Get menu item
Get menu item key
Get menu item mark
Get menu item style
Get menu title
GRAPH TABLE
HIDE MENU BAR
INSERT MENU ITEM
Level
Menu selected
MODIFY RECORD
MODIFY SELECTION
MODIFY SUBRECORD
ON EVENT CALL
OPEN DATA FILE
Open external window
PAGE BREAK
PAGE SETUP
PRINT FORM
PRINT LABEL
PRINT RECORD
PRINT SELECTION
PRINT SETTINGS
Printing page
QR REPORT
QUERY BY EXAMPLE
REMOVE PICTURE FROM LIBRARY

SET ABOUT
SET MENU ITEM
SET MENU ITEM KEY
SET MENU ITEM MARK
SET MENU ITEM STYLE
SET PICTURE TO LIBRARY
SET PRINT PREVIEW
SHOW MENU BAR
Subtotal

• Unappropriate 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
ADD DATA SEGMENT
After
APPEND TO CLIPBOARD
APPEND TO LIST
Before
BLOB to list
BRING TO FRONT
BUTTON TEXT
CANCEL
CHANGE CURRENT USER
CHANGE PASSWORD
CLEAR CLIPBOARD
CLEAR LIST
Copy list
Count list items
Count screens
Create document (1)
Create resource file (1)
Current form page
Current form table
Current user
C_GRAPH
Deactivated
DELETE LIST ITEM
DELETE USER
DIALOG
DISABLE BUTTON
DRAG AND DROP PROPERTIES
DRAG WINDOW

Drop position
During
ENABLE BUTTON
ERASE WINDOW
EXPORT DATA (1)
FILTER KEYSTROKE
Find window
FIRST PAGE
Focus object
FONT
FONT LIST
Font name
Font number
FONT SIZE
FONT STYLE
Form event
Frontmost process
Frontmost window
GET CLIPBOARD
Get edited text
GET FORM PROPERTIES
GET GROUP LIST
GET GROUP PROPERTIES
GET HIGHLIGHT
GET LIST ITEM
GET LIST ITEM PROPERTIES
GET LIST PROPERTIES
GET MOUSE
GET OBJECT RECT
GET PICTURE FROM CLIPBOARD
Get text from clipboard
GET USER LIST
GET USER PROPERTIES
GET WINDOW RECT
Get window title
GOTO AREA
GOTO PAGE
GRAPH SETTINGS
HIDE PROCESS
HIDE TOOL BAR
HIDE WINDOW
HIGHLIGHT RECORDS
HIGHLIGHT TEXT
IMPORT DATA (1)
In break
In footer
In header
INPUT FORM

INSERT LIST ITEM
INVERT BACKGROUND
Is a list
Is user deleted
Keystroke
LAST PAGE
List item parent
List item position
LIST TO BLOB
Load list
MAXIMIZE WINDOW
Menu bar height
Menu bar screen
MINIMIZE WINDOW
Modified
MOVE OBJECT
New list
NEXT PAGE
Next window
Old
Open document (1)
Open resource file (1)
ORDER BY (2)
OUTPUT FORM
Outside call
Pop up menu
POST CLICK
POST EVENT
POST KEY
PREVIOUS PAGE
QUERY BY FORMULA (2)
QUERY (2)
REDRAW
REDRAW LIST
REDRAW WINDOW
REGISTER CLIENT
REJECT
SAVE LIST
SCREEN COORDINATES
SCREEN DEPTH
Screen height
Screen width
Select folder
Selected list items
SELECT LIST ITEMS BY POSITION
SELECT LIST ITEMS BY REFERENCE
SELECT LOG FILE
Self

SET CHOICE LIST
SET COLOR
SET CURSOR
SET ENTERABLE
SET FIELD TITLES
SET FILTER
SET FORMAT
Set group properties
SET LIST ITEM
SET LIST ITEM PROPERTIES
SET LIST PROPERTIES
SET PICTURE TO CLIPBOARD
SET RGB COLORS
SET SCREEN DEPTH
SET TABLE TITLES
SET TEXT TO CLIPBOARD
SET TIMER
Set user properties
SET VISIBLE
SET WINDOW RECT
Shift down
SHOW PROCESS
SHOW WINDOW
SORT LIST
Test clipboard
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.: SORT ([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
MENU BAR
MESSAGES OFF
MESSAGES ON
SHOW TOOL BAR

How to Start a Stored Procedure

• With 4D Client, you can manually start a stored procedure in the Execute Method dialog box:
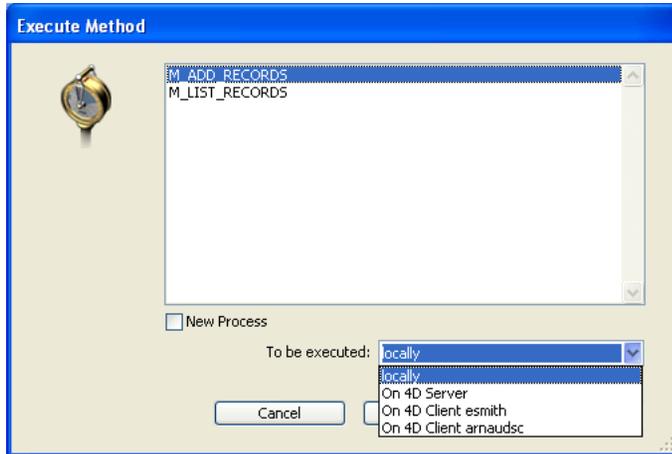
You can execute it on 4D Server or on one or several 4D Client machines. Please note that to display the 4D Clients in this list, they should have been first registered (see the Registering 4D Client section and the REGISTER CLIENT command).

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

• A method executed on 4D Server (server database method or stored procedure) can start a stored procedure using Execute on server or 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 *4th Dimension 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.

See Also

SP-Based Import (Example), SP-Based Services (Example).

Stored procedures can be executed on one or several 4D Client. 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.

Furthermore, 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 4D Client: 4D Client can automatically be registered when connecting or through programming.

Registering automatically each 4D Client connecting to 4D Server
The "Register Clients at Startup" check box is available in the application Preferences, on the "Configuration" page of the "Client-Server" theme:
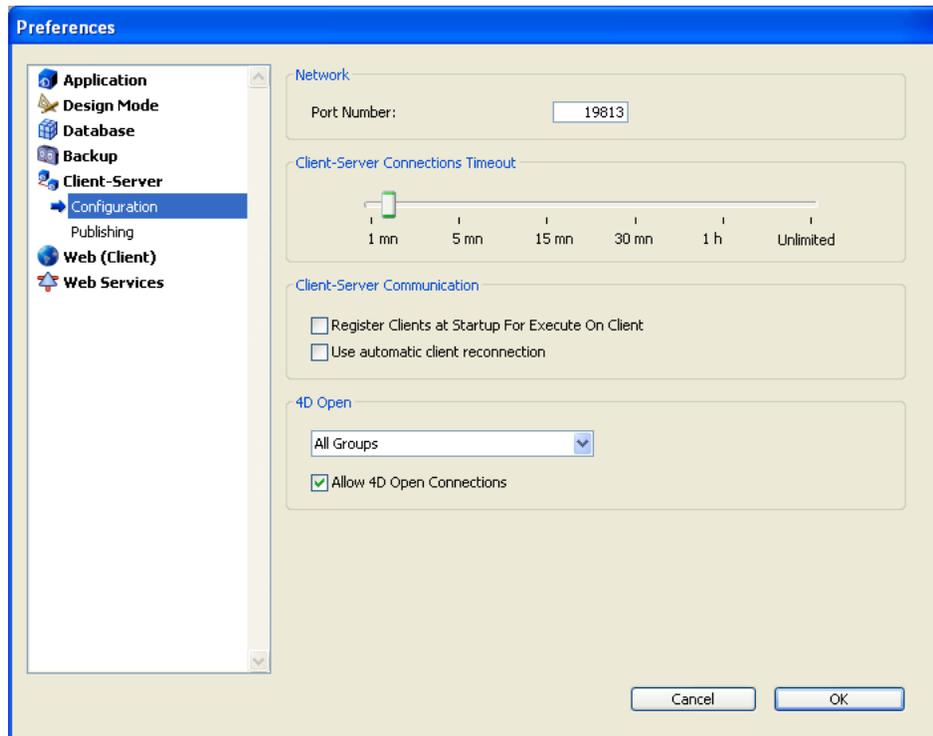
When this option is checked, each 4D Client connecting to the database is automatically referenced with 4D Server as being able to execute stored procedures. A process named according to the client machine is created on the server, in the user process group. A process is also created on each client machine.

Registering 4D Client through programming
It is possible to register one or several 4D Client 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 *4th Dimension Language Reference* manual.

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 Client 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 data is therefore performed locally (at single-user speed) 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 ; String )
        ` CLIENT IMPORT ( -> [Table] ; Input form )

C_POINTER($1)
C_STRING(31;$2)
C_TIME($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 ; String ; BLOB )
   ` SERVER IMPORT ( Table Number ; Input form ; Import Data )

C_LONGINT($1)
C_STRING(31;$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 required
    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.

See Also

Execute on server, GET PROCESS VARIABLE, SET PROCESS VARIABLE, SP-Based Services (Example), Stored Procedures.

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 <u>automatically</u> when the server database starts up, and can be ended and restarted at will by any 4D Client 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 4D Clients. 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 Process window shows the stored procedure running when no client is yet connected.
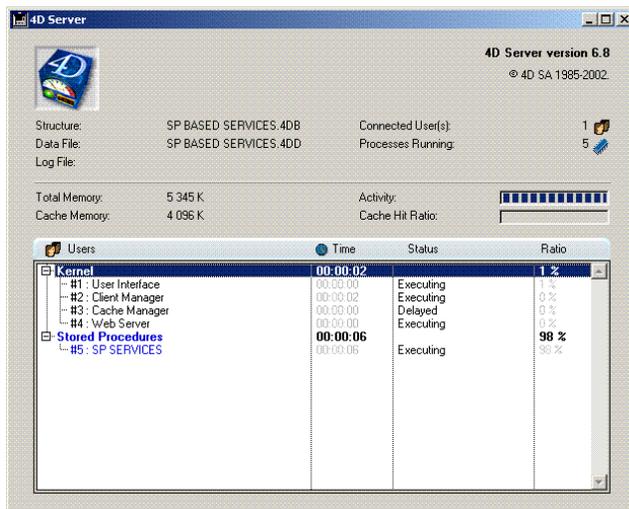
Starting and Ending the Stored Procedure At Will

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	**L**
reqType	A
reqStatus	I
reqData	X
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 choosen 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;$1;$vlDelay)
    $0:=1
    $vlDelay:=0
    If (Count parameters>=3)
        $vlDelay:=$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 # 1)
                $2->:=[SP Requests]reqData
                READ WRITE([SP Requests])
                While (Locked([SP Requests]))
                    WAITING LOOP ($vlDelay)
                    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 ($0=1)
      WAITING LOOP ($vlDelay)
   End if
Until ($0 # 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: The DELAY PROCESS has no effect on the User environment 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;Pascal string)
   TEXT TO BLOB(Structure file;[SP Requests]reqData;Pascal string;*)
   TEXT TO BLOB(Data file;[SP Requests]reqData;Pascal 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;*)
```
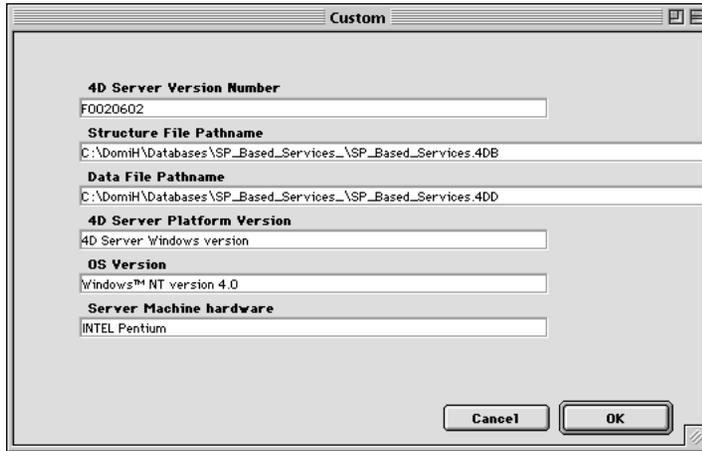
Showing the Server Information on a Client Machine

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($vlReqID;$vlErrCode;$vlOffset)
    ` Post the request
$vlReqID:=Client post request ("Server Information")
    ` Poll the request status and get the result
$vlErrCode:=Client get result ($vlReqID;->vxData;60)
    ` If the request is successfully completed, display the result
If ($vlErrCode=0)
        ` Extract the result information from the BLOB
    $vlOffset:=0
    vsServerVersion:=BLOB to text(vxData;Pascal string;$vlOffset)
    vsStructureFile:=BLOB to text(vxData;Pascal string;$vlOffset)
    vsDataFile:=BLOB to text(vxData;Pascal string;$vlOffset)
    BLOB TO VARIABLE(vxData;$vlPlatform;$vlOffset)
    BLOB TO VARIABLE(vxData;$vlSystem;$vlOffset)
    BLOB TO VARIABLE(vxData;$vlMachine;$vlOffset)
        ` 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 $vlSystem and $vlMachine
        ` ( see the example for the PLATFORM PROPERTIES command)
        ` ...
        ` Display the result information
    DIALOG([SP Requests];"SERVER INFORMATION")
Else
    ALERT("Request error "+String($vlErrCode))
End if
    ` No longer need the BLOB
CLEAR VARIABLE(vxData)
```

Here is the [SP Requests];"SERVER INFORMATION" form in the User or Custom Menus environments:

In this window, information coming from a Windows NT-based 4D Server machine is displayed on a Macintosh client machine.

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(vxData)
      ` Post the request
$vlReqID:=Client post request ("Volume List")
      ` Poll the request status and get the result
$vlErrCode:=Client get result ($vlReqID;->vxData;120)
      ` If the request is successfully completed, display the result
If ($vlErrCode=0)
      ` Extract the result information from the BLOB
   $vlOffset:=0
   BLOB TO VARIABLE(vxData;asVName;$vlOffset)
   BLOB TO VARIABLE(vxData;arVSize;$vlOffset)
   BLOB TO VARIABLE(vxData;arVUsedSpace;$vlOffset)
   BLOB TO VARIABLE(vxData;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(vxData)
```

Here is the [SP Requests];"VOLUME LIST" form in the User or Custom Menus environments:

In this window, disk information coming from a Windows server machine is displayed on a Macintosh client machine.

See Also

BLOB Commands, Execute on server, SP-Based Import (example), Stored Procedures.