



Summit '97

V6 Programming Fundamentals: Part 1 Stored Procedures and Beyond

by David Adams & Dan Beckett

©1997 David Adams & Dan Beckett. All rights reserved.

Content adapted from *Programming 4th Dimension: The Ultimate Guide*,
by David Adams & Dan Beckett, published by Foresight Technology, Inc.

V6 Programming Fundamentals: Part 1

Stored Procedures and Beyond

Overview

V6 *fundamentally* changes the rules for programming with 4D Server. If you understand where 4D executes code, and how to exploit that knowledge, you can optimize parts of your applications by orders of magnitude. The great thing is that it's not hard. The keys to high-performance multi-user systems are [Server-Optimized Commands](#), [Stored Procedures](#), and [Batch Processing](#). We explain these concepts and techniques in plain language, and point out the common pitfalls to avoid. You'll learn that taking advantage of these techniques is easy, and that it makes your systems faster and more useful.

Server-Optimized Commands

Concept

Commands can execute on the server machine, the client machine, or split the work between the two. This is true with 4D Server 1.x and 4D Server V6. Commands that execute on the server machine increase performance by reducing network traffic.

Where Does Code Run?

The exact location that code executes depends on the commands you are using, what context they're executing in, and the version of 4D Server you are using. Here is a summary:

Where Code Executes

Code	4D Server V1.x	4D Server V6
Server optimized command	Server machine	Server machine
Non-server optimized command	Client machine	Client machine
Trigger code	Not available	Server machine
Stored procedure	Not available	Server machine

List of Commands

This table summarizes where 4D commands execute:.

Where Commands Execute

Area	4D Client	4D Server
Queries	QUERY BY FORMULA "Contains" queries	All indexed queries: QUERY QUERY SELECTION QUERY BY EXAMPLE
Sorts	ORDER BY FORMULA	All other sorts
Record Modification	APPLY TO SELECTION	SELECTION TO ARRAY SUBSELECTION TO ARRAY ARRAY TO SELECTION
Lists		ARRAY TO LIST LIST TO ARRAY
Selections		ALL RECORDS RELATE ONE RELATE ONE SELECTION (formerly JOIN) RELATE MANY RELATE MANY SELECTION (formerly PROJECT SELECTION) REDUCE SELECTION SCAN INDEX GOTO RECORD GOTO SELECTED RECORD CLEAR NAMED SELECTION COPY NAMED SELECTION

Where Commands Execute (Continued)

Area	4D Client	4D Server
Sets	<p><i>All V3 sets and V6 local sets:</i></p> <p>CREATE EMPTY SET CREATE SET USE SET CLEAR SET ADD TO SET REMOVE FROM SET Is in set Records in set</p> <p><i>V3:</i></p> <p>DIFFERENCE INTERSECTION UNION</p>	<p><i>V6 process and interprocess sets:</i></p> <p>CREATE EMPTY SET CREATE SET USE SET CLEAR SET ADD TO SET REMOVE FROM SET Is in set Records in set</p> <p><i>V6: All sets</i></p> <p>DIFFERENCE INTERSECTION UNION</p>
Information	<p>Current date Current time</p>	<p>Current date (*) Current time (*) Table Field name Table Table name</p>
Statistics		<p>Average Max Min Std deviation Sum Sum squares Variance</p>

Who Cares?

Server optimized commands were the cornerstone of many server-based optimization strategies before V6. These strategies work in V6, and are still the easiest way to improve performance. V6 takes this idea one step further with stored procedures.

Stored Procedures

Concept

A stored procedure is custom code that runs in its own process on the server machine itself. Think of it as a “server process”. 4D Server prior to V6 did not execute any of your code on the server machine, except for server optimized commands. 4D Server V6 allows you to execute almost any command in a stored procedure on the server machine, effectively turning any command into a server optimized command.

Stored procedures are great for optimizing network-intensive tasks. On the other hand, since code executes on the server machine itself, whatever you do in a stored procedure can affect *all* users, for better or for worse. If you crash or hang the server machine, monopolize the CPU, or display a modal dialog then all users are effected. Stored procedures are one of the most powerful tools in the 4D Server environment, but they need to be used correctly and appropriately.

We group appropriate uses of stored procedures into four categories:

- ❖ [Optimizing Network-Intensive Tasks.](#)
- ❖ [Maintaining Server-Side Data Structures.](#)
- ❖ [Sharing Access to External Resources.](#)
- ❖ [Centralizing Business Logic.](#)

Optimizing Network-Intensive Tasks

Commands that spend the majority of their time in network traffic are ideal candidates for optimization with a stored procedure. Good examples include all formulaic commands, like **QUERY BY FORMULA** and **APPLY TO SELECTION**.

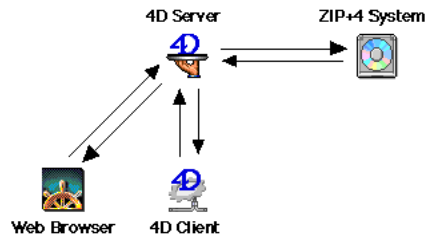
Maintaining Server-Side Data Structures

Why not cache frequently used information on the server? Then all users can use or copy these objects without recalculating or loading data. Good candidates for server-side data structures include sets, arrays, and BLOBs.

Sharing Access to External Resources

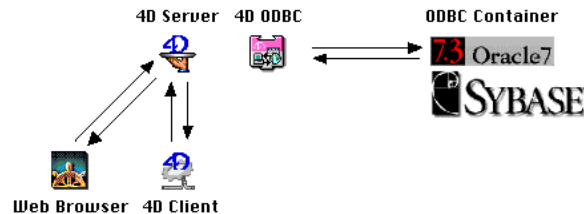
Stored procedures can distribute access to external resources. If, for example, you have a ZIP+4 CD-ROM on the server machine, you can use a stored procedure to communicate with the CD-ROM,

allowing all 4D Clients to use the same ZIP Code data without importing all of it into the database. (A ZIP+4 database can contain more than 500Mb of data that changes every two months.)



With a stored procedure all clients can share a server-side resource.

You can also use stored procedures to read data from remote hosts, read documents from the server machine's hard disk, and even perform database-to-database communication:



Centralizing Business Logic

If you use a variety of interfaces onto data stored in 4D Server—like 4D Open, 4D forms, and interfaceless processes, stored procedures can make your life a lot easier. You can centralize standard routines in a stored procedure, and then create small “interface” routines that invoke the stored procedure from whatever interface you like.

Implementing Stored Procedures

Starting a Stored Procedure

Stored procedures can start from 4D Client or from 4D Server itself and are identical to a regular 4D global processes. A stored procedure runs as a process on the server machine with a process ID that is unique for that machine. To start a stored procedure from 4D Client, use the **Execute on server** function:

```
$My_SP_ID:=Execute on server ("MY_SP";32*1024;"My Stored Proc.")
```

The **Execute on server** function has the same parameters as the **New process** function. There are only a few differences between a global process on a client and a stored procedure:

- ❖ Stored procedures run on the server machine.
- ❖ Client processes can read data from stored procedures.
- ❖ There are a few restrictions on the commands you can use in a stored procedure.

Stored Procedure ID's

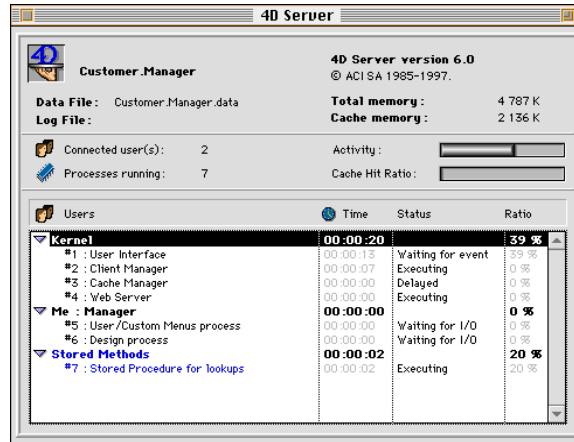
When a stored procedure starts on the server machine, it is assigned a unique ID like any other process. Knowing a stored procedure's ID number allows a client machine to communicate with that stored procedure. If a 4D Client launched the stored procedure using **Execute on server**, then that 4D Client knows the resulting process ID. Often, however, the stored procedure is started on the server machine itself, or by a different 4D Client. Any 4D Client can find a stored procedure's ID based on its name by using the **Process number** function:

```
$Stored_Procedure_ID:=Process number ("My main stored procedure")
```

Called from a client machine, the **Process number** function returns 0 if no matching process name is found, a negative number if a match is found and the process is a stored procedure, and a positive number if a match is found on the current machine. Here is a summary of this behavior:

Process/Stored Procedure ID Numbers

ID	Meaning
Negative number	ID of stored procedure when called from a client machine.
Zero	No such process.
Positive number	ID of process on current machine. The ID for a stored procedure is negative when used from a client, and positive when used from the server machine itself.



The stored procedure is process #7 on the server machine.

Client-Stored Procedure Communication

Stored procedures do *not* communicate with client machines. A stored procedure never “pushes” information to a client machine. The stored procedure can change the values of variables and arrays it uses, but it can’t read and write variables in a client-side process.

It is easy, however, for code on a client machine to communicate with a stored procedure. 4D V6 includes new commands that allow you to implement interprocess communication—even across machines—using arrays and variables. Communication between a client machine and a stored procedure is accomplished through reading and setting process variables and arrays using the **SET PROCESS VARIABLE**, **GET PROCESS VARIABLE**, and **VARIABLE TO VARIABLE** commands. You can still use global semaphores and records to exchange information between workstations, as in earlier versions of 4D Server, but these new commands can make inter-process/inter-workstation communication quicker and easier to implement. Let’s look at an example.

Example

In this example there are two routines, one that wants a query performed, and one that performs the query. The `START_STORED_PROCEDURE` method starts a stored procedure using **Execute on server**, but would

V6 Programming Fundamentals: Part 1 Stored Procedures and Beyond

work the same way if you started another process on the same workstation using **New process**. Here's a look at what the two routines do:

Steps in Stored Procedure Communication

Runs On Client	Runs On Server
START_STORED_PROCEDURE	SERVER_SIDE_LOOKUP
Start the stored procedure called SERVER_SIDE_LOOKUP.	
Wait for the stored procedure to finish.	Perform the query.
	Put the results in a variable
	Set a variable indicating the query is finished.
	Wait for the calling client to say it is OK to stop.
Read the result variable.	
Tell the stored procedure it can stop.	Stop.

Most of the work is devoted to keeping the two processes in step. This is the sort of work that 4D Server performs for you transparently when you use global processes. With a stored procedure you do a little more coding, but get great speed benefits because of it. Here is the code that performs the tasks diagrammed above:

` START_STORED_PROCEDURE

` Tells the stored procedure not to stop.

vb_Keep_Lookup_Process_Alive:=**True**

` Indicates that the stored procedure is still working.

vb_Processing_Lookup:=**True**

` Start the stored procedure and get its ID.

` Notice that you can pass parameters to the new procedure.

vl_Stored_Procedure_ID:=**Execute on server**("SERVER_SIDE_LOOKUP"; 32*1024;"Stored Procedures lookups";"William";"Blake")

` This routine waits for vb_Processing_Lookup to be

` set to False in the stored procedure, meaning the

` stored procedure has completed the query, and is waiting.

GET PROCESS VARIABLE(vl_Stored_Procedure_ID;
vb_Processing_Lookup;vb_Processing_Lookup)

` Remain in an endless loop until the

` stored procedure finishes.

While (vb_Processing_Lookup)

DELAY PROCESS(Current process;30)

` Find out if the store procedure has finished.

GET PROCESS VARIABLE(vl_Stored_Procedure_ID;vb_Processing_Lookup;
vb_Processing_Lookup)

End while

` OK, the stored procedure is done with the lookup,

` get the values back:

GET PROCESS VARIABLE(vl_Stored_Procedure_ID;vs_Customer_ID;
vs_Customer_ID)

` We've read the values, so now it's OK to let the

` stored procedure stop:

vb_Keep_Lookup_Process_Alive:=**False**

` End of method.

Here is the `SERVER_SIDE_LOOKUP` routine:

```
` SERVER_SIDE_LOOKUP

` This was passed as a parameter to the Execute on server command.
$First_name:=$1 ` Passed parameter to Execute on server.
$Last_name:=$2 ` Passed parameter to Execute on server.

vb_Keep_Lookup_Process_Alive:=True
vb_Processing_Lookup:=True

` Perform the query:
QUERY([Customers];[Customers]First Name= $First_name;*)
QUERY([Customers];&;Last Name=$Last_Name)

vs_Customer_ID:=[Customers]ID

` Done processing. The calling process checks this value.
vb_Processing_Lookup:=False

` The client reads vs_Customer_ID and then sets
` vb_Keep_Lookup_Process_Alive to False.
While (vb_Keep_Lookup_Process_Alive=True)
  DELAY PROCESS(Current process;30)
End while

` End of method.
```

There are many ways to manage interprocess communication. This example is simple because it is meant to illustrate the mechanics of **SET PROCESS VARIABLE** and **GET PROCESS VARIABLE**. In a production system, stored procedures are often used by multiple workstations, requiring a global semaphore to insure that only one client at a time reads variables from and writes variables to the stored procedure.

Use Stored Procedures Wisely

Stored procedures run on the server machine, so whatever happens in them affects the entire system. 4D imposes few limits on what you can do in a stored procedure. You cannot use **ADD RECORD** or call **QUIT 4D** in a stored procedure, but you can perform virtually any other task. It is up to you to use stored procedures appropriately. If you put up a modal dialog, crash, or spend an enormous amount of time on a task then all clients are effected. For these reasons the following rules of thumb should be followed:

- ❖ Make your stored procedures fast.
- ❖ Never use modal windows (dialogs, alerts, etc.).
- ❖ Trap all errors.
- ❖ Avoid operations likely to produce OS errors.
- ❖ Avoid synchronous routines and plug-ins.

- ❖ Keep stored procedures to a minimum.

Batch Processing

Concept

Stored procedures are a great way to improve performance, but overloading the server machine slows down the entire system. Also, several tasks—like printing and faxing—are not appropriate for stored procedures because they are highly exception prone. Distributing tasks amongst several workstations is a flexible and scalable way of increasing overall system performance without over-taxing the server. You can implement batch workstations in 4D Server 1.x as easily as in 4D Server V6.

Batch Workstations vs. Stored Procedures

The kinds of tasks performed by a batch workstation can also be performed using stored procedures under 4D Server V6. The advantage of using a batch workstation is that it does not unnecessarily burden the server machine. The important questions to ask when considering whether to implement a feature as a batch workstation task or a stored procedure are:

- 1) How quickly does the user really need the results?
- 2) How processor-intensive is the task?

What you are really trying to determine is, what impact will processing on the server machine have on the users?

In general, a batch workstation is appropriate for processor-intensive tasks that do not need immediate attention. (Immediate as in, “Anytime in the next second would do.”) This prevents the server machine from slowing down all other users to, say, print a report that the requesting user would happily wait ten minutes for.

Stored procedures are appropriate for quick tasks that a user is waiting on, like looking up a ZIP Code, or assembling a customer history. These tasks do not take long to perform, so there is little impact on the other users and the benefit to the requesting user is great.

Appropriate Uses

Let’s explore several kinds of tasks that a batch workstation can perform.

- ❖ [Updating Summary Data.](#)

- ❖ [Managing Shared Resources.](#)
- ❖ [Distributing Processing.](#)
- ❖ [Automating Report Production.](#)

Updating Summary Data

Many of the most powerful optimization techniques depend on periodic data updates. Both post-and-reconcile systems and periodically refreshed saved sets are good examples. Tasks such as these are best handled automatically and on a regular basis. Using a batch workstation is the natural way to ensure that these important, but potentially time-consuming, tasks are taken care of automatically, regularly, and in the background

Managing Shared Resources

Systems often benefit from integration with other programs or from access to special devices and services. Common examples include:

- ❖ CD-ROMs with special data.
- ❖ External programs, including mapping systems, Web-based services, and other databases.
- ❖ Printers.

It is not always practical or cost-effective to duplicate every resource on each client machine. For example, if you use an expensive Geographic Information System in conjunction with 4D it may be licensed on a per CPU basis, and may require that each user have special software or files on their computer. If you can afford it (and if the software runs on your users' operating system) this may be a good solution. An alternative is to allow users to submit requests to a single workstation which processes the requests and returns results.

Distributing Processing

People hate waiting (just ask them). People particularly hate waiting for their computer to finish a task before they can do something else. They hate this with a hatred that is sharp and spiky, and directed at you. If a task takes ten minutes and keeps users from doing anything else, they are going to curse you, at least to themselves. This is natural, and you may feel the same way during a lengthy compile or Internet download.

If, however, you allow users to send jobs to another machine for background processing, they are usually willing to wait *longer* for the job to finish. It is painful to wait ten minutes while your machine

is tied up. It is not as painful to do something else for 15 minutes while your job is processed somewhere else. The point here is not that submitting jobs to a batch workstation makes them take longer—this may or may not be the case in your environment—but rather that letting users retain control of their computers makes them happy.

Automating Report Production

Databases often contain far more *data* than *information*. Data is the material you collect, *information* is a further refinement or distillation of your data. Reports, charts, and summaries are examples of information derived from database data. Good return on investment from information systems comes from the system producing information that improves business decisions. Information production is too often neglected because of the time it takes to produce reports. An automated report production system lowers the barriers to fruitful information production. With an automated report production system you amplify the value of your programming *exponentially*. If you write a routine that generates a complex and useful summary of some of your system's data, you can automatically reproduce the report regularly with no extra effort. If you do not do this already, this is arguably the single most important way you can improve the value of your system. Information that leads to better business decisions justifies the costs of building and maintaining a business system in the first place. Here are some examples of reports you can generate automatically with a batch workstation:

- ❖ Trending and cross-tabs saved into a BLOB, 4D Calc area, text field, or export document.
- ❖ Automatic e-mail and fax reports.
- ❖ Automatic integrity and exceptions tests.

How It Works

There are a lot of ways to program a batch workstation. We'll outline a record-based, on-demand batch workstation. It's conceptually simple. When a job is needed a record is created describing the job. Batch processes scan the job file for new jobs and perform the necessary work. Here's a brief outline of how this works:

Batch Processing

Requestor	Batch Processor
Create request description record	Scan for unprocessed requests
	Load and lock new request
	Perform requested work.
	Return results in request record.
	Mark request record as "processed".
Read completed request record (optional)	

(Implementing a batch workstation is outside the scope of this discussion. The book *Programming 4th Dimension: The Ultimate Guide* contains a complete Batch Workstation module that you can use in your own systems.)