

Mastering 4th Dimension

The EXECUTE Command

by Walt Nelson

The EXECUTE command?

Many highly experienced 4D developers mention the **EXECUTE** command in the same tone of voice that they might mention a hurricane or a flood: something to avoid, if at all possible. This is unfortunate, because the **EXECUTE** command is fundamental to 4D. In fact, I will begin this article by making two bold statements:

1. The **EXECUTE** command is the most powerful command in 4th Dimension.
2. The **EXECUTE** command is the most often used command in 4th Dimension.

Right now, if you are a long-time 4D developer, you are probably thinking: "Well, Walt has finally done it; he has gone insane." But wait: stay with me a moment while I back up those two statements with facts.

The definition of EXECUTE

The 4D Language Reference describes the **EXECUTE** command as follows:

"**EXECUTE** executes [a] statement as a line of code. The statement string must be one line."

In other words, when you use the **EXECUTE** command, you are telling 4D: "Treat this string of text as if it were a line of code written directly into a 4D method." 4D evaluates the string and executes it as 4D code.

Behind the scenes

With that description of the **EXECUTE** command in mind, let's examine my second statement. I said that the **EXECUTE** command is the most often used command in 4th Dimension. Look at the following list. All of these 4D commands, functions, menu items, and plug-ins either use the **EXECUTE** command directly, or they pass the name of a method to be called by the **EXECUTE** command:

APPLY FORMULA	APPLY TO SELECTION
CT ON EVENT	CT ON ERROR
CT ON MENU	CT SET TIPS ATTRIBUTES
DB SET ERROR HANDLER	DB SET MESSAGE HANDLER
DR ON ERROR	DR ON EVENT
DR ON MENU	Evaluate 4D Command (4D Calc)
EXECUTE	
EXECUTE ON CLIENT	Execute on Server
New Process	OC SET ERROR HANDLER

OD ON ERROR CALL	ON ERR CALL
ON EVENT CALL	ON SERIAL PORT CALL
OP EXECUTE ON SERVER	SP ON MENU
SP ON ERROR	SP ON EVENT
WR ON ERROR	WR ON EVENT
WR ON MENU	

Did you notice that the command **New Process** is a member of this distinguished group? How many times have you used this command? Did it ever occur to you that the **New Process** command is just another form of the **EXECUTE** command?

And oh yes, I forgot to mention that virtually every third party 4D plug-in uses **EXECUTE** when it needs to ask 4D to call a 4D command or method. Example: *AL_SetCallbacks from AreaList*.

And there is more to the story. Here is another category of commands that use a variation of the **EXECUTE** command:

CT DO COMMAND	DR DO COMMAND
SP DO COMMAND	WR DO COMMAND
WR EXECUTE COMMAND	

These commands are from the 4D family of plug-ins. The difference between these commands and the classic **EXECUTE** commands is that the commands in the **...DO COMMAND** category execute 4D menu options by their numbers instead of by their names. Functionally, however, they are members of the **EXECUTE** family of commands.

Throughout the 4D universe, there are so many uses of **EXECUTE** that it is possible that I may not have mentioned them all. If I left out your favorite use of the **EXECUTE** command, I apologize.

Using the EXECUTE Command

The **EXECUTE** command is extremely useful and versatile; however, many 4D developers do not use it. Perhaps they feel that they don't understand it. Actually, it is very simple. All uses of the **EXECUTE** command will perform one of the following two actions:

1. Call a 4D method; or
2. Evaluate a string and execute it as a line of 4D code.

In my upcoming book, "*Mastering 4th Dimension Volume 1 - Learning to Think in 4D*," I include several examples of how to use the **EXECUTE** command. In this *Dimensions* article, I

will give you one example in each category of usage: calling a 4D method, and evaluating a string as a line of 4D code.

Using the EXECUTE command to call a 4D method

Let's suppose that you are looking for a solution to the following problem: The On Load form event only occurs when a form is loaded; the event does not occur when a record is loaded. When the user clicks a record navigation button in an input form—the First Record, Previous Record, Next Record, or Last Record button—the record is loaded into the input form, but the On Load form event does not occur! If you are using the On Load form event to do some setup prior to displaying a record in an input form, that setup code will only run when the user first double-clicks an output list to go to an input form; the code will not run when the user navigates to a different record by using the record navigation buttons.

Let's assume that, to solve this problem, you have written several project methods, each method containing the input form's On Load form event code for an individual table (the actions for each table are different, so it would have been awkward to put all that code into one method). You want to be able to call the appropriate project method in the On Load form event, and also call that same project method whenever the user clicks a record navigation button.

Since you like to code in a generic style, you have decided on a naming convention: each of these On Load form event project methods will be named OnLoad_TableName (OnLoad plus an underscore plus the name of the table). So you have created the following project methods: OnLoad_Clients, OnLoad_Invoices, OnLoad_Payments, OnLoad_Inventory, and so on.

Now you are ready to write a generic routine, OnLoad_Execute, that will be called in the On Load event of every input form, and in the On Clicked event of every record navigation button. We call them "record navigation buttons" but they will actually be NO ACTION buttons; in your On Clicked event code, you will use a 4D command to move the record pointer and navigate the selection. You will call the method OnLoad_Execute as follows:

```
OnLoad_Execute(Current form table;"Message")
```

The OnLoad_Execute method will respond to five messages: "Open", "First", "Previous", "Next", "Last." We will give you two versions of this method. The first version uses the **EXECUTE** command to call a 4D method; the second version uses the **EXECUTE** command to call a 4D method, and it also uses the **EXECUTE** command to evaluate a string and execute the string as 4D code.

```
-----
`OnLoad_Execute example #1: call a 4D Method
C_POINTER ($pTablePtr)
C_STRING (31;$sTableName)
C_STRING (7; $1; $sMessage)
$sMessage:=$1
```

```
$pTablePtr:=Current Form Table
`Get a pointer to the current table
$sTableName :=Table Name ($pTablePtr)
`Get the name of the current table
EXECUTE("SAVE RECEORD(["+$sTableName+"])")
Case of
:($sMessage="Open")
`The User double-clicked an output form.
`Do nothing--don't move the record pointer.
:($sMessage="First")
FIRST RECORD ($pTablePtr->)
:($sMessage="Previous")
PREVIOUS RECORD ($pTablePtr->)
:($sMessage="Next")
NEXT RECORD ($pTablePtr->)
:($sMessage="Last")
LAST RECORD ($pTablePtr->)
End Case

EXECUTE ("OnLoad_"+$sTableName)
`Execute the OnLoad method for this table
-----
```

As you can see, this is a simple, yet powerful, use of the **EXECUTE** command. Although we used a specific case, this use of the **EXECUTE** command is actually a generic solution with many applications. Any time you have a series of associated methods, and you need to decide which method to call based on the circumstances, you can use naming conventions plus the **EXECUTE** command to solve the problem in a way that is easy to maintain: in this case, as you add new tables and new input forms, you don't need to make any changes in the method OnLoad_Execute. The method will automatically work, as long as you follow the naming conventions.

Using the EXECUTE command to execute a line of 4D code

The second syntax of the **EXECUTE** statement—and the one that many 4D developers advise against using—is the syntax in which you build a string that executes as a line of 4D code. We can make the method OnLoad_Execute even more generic by using the second syntax to replace our **Case** statement.

```
-----
`On_Load_Execute Example #2:
`call 4D method, evaluate a string as 4D Code
C_POINTER ($pTablePtr)
C_STRING (31;$sTableName)
C_STRING (7; $1; $sMessage)
$sMessage:=$1
$pTablePtr:=Current Form Table
`Get a pointer to the current table
$sTableName :=Table Name ($pTablePtr)
`Get the name of the current table
EXECUTE("SAVE RECORD(["+$sTableName+"])")
`Move the record pointer.
`For Example: FIRST RECORD ((Clients))
EXECUTE($sMessage+" RECORD(["+$sTableName+"])")
`Execute the OnLoad method for this table
EXECUTE ("OnLoad_"+$sTableName)
-----
```

The second syntax of the **EXECUTE** command is even more powerful than the first syntax; however, with more power comes more responsibility. The “command line” syntax of the **EXECUTE** command requires more coding time because it is more difficult to debug, and because it requires that you know the 4D rules of syntax very well. However, you should not let that stop you from using this technique. When you learn to use it, you will be taking your 4D programming to another level: you will be able to do things that would have been difficult, cumbersome, or impossible without the **EXECUTE** command.

Is the EXECUTE command slow?

Back in the days of 60mhz Macintosh machines, speed of execution of the **EXECUTE** command might have been something to be concerned about; however, with today’s 200mhz and faster CPUs, it is not necessary to worry about how long it takes 4D to interpret a string and execute that string as a command. The elapsed time—if we could measure it at all—would be in thousandths of a second.

Summary

The uses of the **EXECUTE** command are limited only by your imagination. 4th Dimension itself, and the ACI plug-

ins, frequently use the **EXECUTE** family of commands. Every third party plug-in that needs to ask 4D to call a command or a method, uses the **EXECUTE** command. Now that you have a deeper understanding of the **EXECUTE** command, you too should use the **EXECUTE** command.

To end this article, I will repeat my two bold statements and I think that this time, you might agree with them:

1. The **EXECUTE** command is the most powerful command in 4th Dimension.
2. The **EXECUTE** command is the most often used command in 4th Dimension.

About the author

Walt Nelson is a writer, trainer, motivational speaker, and 4D developer/consultant. Walt has been doing 4D development since 1987; he was a participant in the original “Silver Surfer” 4D Beta Test project. From his home base in the San Francisco Bay Area, Walt manages his consulting, publishing, and 4D Mentoring business. Walt is currently writing a series of books about 4D: three volumes that will be called the *Mastering 4th Dimension* series. Volume 1 is scheduled for release before the end of 1999.