Build PerfectOffice Apps

With a suite of three VBXs in hand, VB developers can build cool custom solutions in PerfectOffice—without relying on DDE.

BY STEVE MANN

The newest version of Novell's PerfectOffice suite offers robust and varied custom development tools, but little is known about them. I see numerous messages on the WPUSERS Forum on CompuServe asking about various ways to automate PerfectOffice through Visual Basic. Until now Dynamic Data Exchange (DDE) was the only way to accomplish this with PerfectOffice's predecessor, Borland Office 2.0, or with WordPerfect for Windows. DDE works, but the method usually doesn't offer the level of control necessary to create really cool solutions based on common desktop applications.

PerfectOffice 3.0, released in December 1994, breaks through that limitation, opening the full functionality of PerfectOffice to Visual Basic programmers through three VBXs. The mechanism is simple, robust, and the back-end functionality is extensive. Armed with your existing knowledge of Visual Basic, all you'll need is this introduction to PerfectOffice and you'll be coding Visual Basic solutions with PerfectOffice like an old pro.

Let's start with an overview of the PerfectOffice automation architecture. The foundation of PerfectOffice's development architecture is PerfectScript, a full scripting system. PerfectScript has application-independent language commands (also called product commands) such as If, While, and Call. Each application exposes its product-specific command set to PerfectScript through a Product Interface Descriptor, or PID file. This file details every product command and its parameter types for each product's native functions.

When you execute a script in the PerfectScript system, it interprets its own programming commands and passes product commands to the appropriate application for it to execute. Because PerfectScript is independent of the applications, it is a true cross-application controller, where a single script can simultaneously control individual PerfectScript applications. A command prefix tells PerfectScript which application a command should be sent to. For example, you could use these lines to set up a script for WordPerfect for Windows (WPWin) and GroupWise (PerfectOffice's e-mail package, which retains the WPOffice name in PerfectScript for backward compatibility):

```
Application (WP; _
  "WordPerfect"; "US")
Application (GW; _
  "WPOffice";"US")
```

After these lines, any statement preceded by GW is sent to GroupWise, while WP tells PerfectScript to send the command to WPWin. The other parameter tells PerfectScript the language, in this case U.S. English. You can also use a Default parameter to indicate that product commands without a prefix by default go to a specific application:

```
Application (WP; "WordPerfect"; _
   Default; "US")
Application (GW; "WPOffice";"US")
```

These application prefixes need to be defined only once within a script. The first time a product command is executed for a

Steve Mann is group leader of the PerfectOffice development team at WordPerfect, the Novell Applications Group. Steve is also coauthor of Que's Using Novell PerfectOffice: Special Edition. *Reach Steve on CompuServe at 74777,2716.*

new application (new to this PerfectScript session), PerfectScript will launch the application if necessary. If you want to see more on how PerfectScript works, experiment with recording some cross-application scripts in PerfectOffice. During the record session, you can move from one PerfectOffice application to another. As you record the various product commands, the PerfectScript recorder will insert the appropriate command prefixes to direct the commands to the correct application.

VB DOES PERFECTSCRIPT

The Visual Basic interface into PerfectOffice is built on this PerfectScript foundation. Instead of using PerfectScript's programming commands, you use Visual Basic as the programming tool. When you need to communicate with or control an application, you use a VBX to send the application's product command. PerfectScript acts as the traffic controller, forwarding the command to the proper application. This combination gives you the full power and familiarity of Visual Basic with the native command set of the application.

The VBX connects to PerfectScript through the CIWIN20.DLL, bypassing PerfectScript's language interpreter, but passing application commands into PerfectScript's command marshaling facility (see Figure 1).

Now you're ready to start coding. First, you need to make sure the VBXs are installed. If you do a standard install of PerfectOffice, the VBXs will not be installed. To install them, you need to do a custom install, select Shared components, and check the box next to "Visual Basic VBXs." PerfectOffice will install the VBXs (there are three), and a constants text file for each one, into your Windows system directory. Of the three VBXs, I'll focus on the most general and most useful one: the WPCIWIN.VBX. The CIWIN stands for Command Interface for Windows, meaning that this VBX (really the CIWIN20.DLL behind it) gives Visual Basic an alternate command interface to the PerfectOffice applications. Just like PerfectScript, this VBX allows you to play scripts in any of the PerfectOffice applications, or to send individual commands or command sequences directly to any specific PerfectOffice app.

When you are ready to include the WPCIWIN VBX in your project, use Add File to add it to the toolbar, select it, and create a frame on your form. Double-click on the control and open your project's Properties window to view the Property list. Installed with the VBX is a text file that defines the WPCIWIN constants you'll use in the code throughout this article (see Table 1).

When you create the VBX instance in your form, it will be named PSCRIPT1.

Property	Possible Values	Purpose
CommandTargetApp	WORDPERFECT = "WordPerfect" GROUPWISE = "WPOffice" PRESENTATIONS = "WPPrWin" QUATTRO_PRO = "QuattroPro"	The name of the application that will receive subsequent commands.
CommandLangCode		PerfectScript's international capabilities are very robust. This Language code allows the developer to specify which language should be used. Language codes are always two-letter codes, such as US, FR, etc. This property only applies if launching the application. If it is already running, the language is already determined.
CommandString	Any product command	This is a hidden property. The specific product command to be sent to the application specified in the last CommandTargetApp statement. CommandString is one of two trigger properties.
CommandReturn		Some product commands return a value. This property will hold the return value of the last command to return a value.
ScriptType	PS_TEMPLATE_SCRIPT = 0 PS_FILE_SCRIPT = 1	PerfectScript allows applications to store scripts in two locations, either within a script file or in a template file.
ScriptName		Full path and file name of the script file, or of the name of the script within a template file (depending on the value of the ScriptType property).
Action	PS_ACTION_PLAYSCRIPT = 1 PS_ACTION_GETVARIABLE = 2 PS_ACTION_SETVARIABLE = 3 PS_ACTION_DELVARIABLE = 4 PS_ACTION_EXISTSVARIABLE = 5	Action is a trigger property that executes based on the settings of other properties. For example, you would use Action when running a script from VB.
Name		The name of this instance of the control. By default this is PScript1 for the first instance. If you change the name, then all commands that reference the control must begin with the new name.

CONTINUED ON PAGE 58.

TABLE 1 ***VBX Properties.*** *While they are few in number, these simple VBX properties give VB programmers access to thousands of product-specific commands to control PerfectOffice. For example, WordPerfect alone has 1824 product commands.*

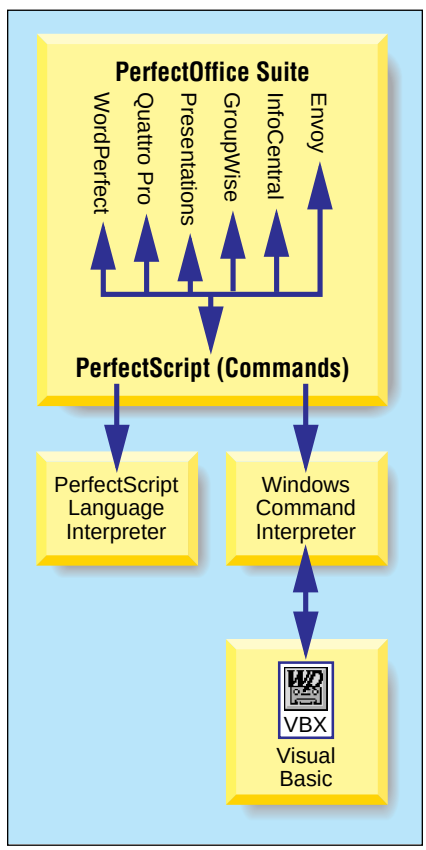

FIGURE 1 ***Architecture of PerfectScript.*** *PerfectScript is the foundation of the PerfectOffice's automation architecture, and a part of the PerfectFit technology. The native scripting mechanism in PerfectOffice is PerfectScript's language interpreter and command dispatcher. CIWIN exposes an alternate interface for dispatching commands. Through the WPCIWIN VBX, Visual Basic can take advantage of this alternate interface and talk directly to PerfectScript's command facility to automate PerfectOffice.*

You can include more than one instance in a form or project, although it is rarely necessary. The only exception might be when you want to drive more than one application, and you set your properties at design time.

All of the properties, their purposes, and their enumerated values from the WPCIWIN.TXT file are shown in Table 1. The main properties used to control PerfectOffice applications are CommandTargetApp, ScriptName, ScriptType, Action, and CommandString. CommandTargetApp contains the name of the application that will receive subsequent commands. ScriptName is a full path name to a script file containing PerfectScript commands to be executed. ScriptType is either a script file or a template file with an embedded script. The Action property is a trigger command to run a script or carry out other actions specified in the VBX properties.

You can control PerfectOffice from VB by playing a prerecorded script, or you can send product commands (also called tokens) directly to the application. If you have a simple interaction with PerfectOffice, and the steps are known in advance, then playing a script make sense. For example, if your company requires employees to send a status report to a supervisor every week, you could automate that process from VB. Record a script (say SENDRPT.WCM) in WordPerfect and GroupWise, and then run that script from VB. Set the properties either at run time or design time:

```
CommandLangCode = US
CommandTargetApp = WordPerfect
ScriptType = WP_FILE_SCRIPT
ScriptName = _
   C:\OFFICE\WPWIN\MACROS\SENDRPT.WCM
```

In this case, the CommandTargetApp could be any of the applications, because PerfectScript will recognize the Application statements in the script and fire up WordPerfect and GroupWise. However, you can't just leave the CommandTargetApp property blank. Once these properties are set, at run time you trigger the running of the script by tying a single VB statement to a button (or some other event) like this:

```
Sub SendRptBtn_Click ()
   PScript1.Action = _
         PS_ACTION_PLAYSCRIPT
End Sub
```

Sending individual commands to the applications requires setting the value of the hidden property, CommandString. Because it is hidden, it can be used only at run time. CommandString enables the VBX to send individual product commands to a specific product. Sending one command at a time gives you greater control and interaction with the applications in your custom solution. You can drive the app, query for state information, and have greater runtime flexibility based on dynamic user interaction with the VB front end.

When sending tokens between VB and PerfectScript, you must follow a few additional procedures for the process to work correctly. First, the CommandTargetApp becomes critical. When running a script, the application statement in the script tells PerfectScript which application to communicate with. When sending individual commands, it's up to you to tell PerfectScript where you want commands to go.

Second, when you send individual commands, PerfectScript puts the application into a remote-control mode—much like a script play-back mode. You decide when the application is in or out of that mode. When your app is in this remote control mode, some of the product's functionality becomes

unavailable. For example, while the VBX is controlling WordPerfect, the user cannot exit. Quit is a special command that tells PerfectScript you are through with a command or series of commands. You should always follow a runtime command, or series of commands, with a Quit command.

A third difference is that the CommandString property requires no other trigger. CommandString, like the Action property, is itself a trigger property. As soon as you assign a value (or command) to the CommandString property, it is sent immediately to the TargetApp.

What does a runtime sequence of commands look like? Here's a series of steps that will copy the text area of a GroupWise e-mail message into a new document in WordPerfect:

```
PScript1.CommandLangCode = "US"
PScript1.CommandTargetApp = GROUPWISE
PScript1.CommandString = _
   "FocusSet(MESSAGE!)"
PScript1.CommandString = "PosTextTop"
PScript1.CommandString = _
   "SelectToEndText"
PScript1.CommandString = "EditCopy()"
PScript1.CommandTargetApp = _
   WORDPERFECT
' the next line is ugly.
' all it does is create a new file
' based on WPWin's Standard Template
PScript1.CommandString =_
   "TemplateSelect(""C:\OFFICE\WPWIN\_
   TEMPLATE\STANDARD.WPT"")"
PScript1.CommandString = _
   "EditPaste ()"
PScript1.CommandString = "Quit"
```

MASTERING PERFECTOFFICE COMMANDS

You now know everything you need to about VB and the PerfectOffice VBX to start building your own PerfectOffice apps. But there is still one problem. Where can you find the right product commands to send to the applications?

If you want help building product commands, there are two places you can turn. The first is the application's own script recording capability. Plan out the steps you want to follow, then go to the applications involved and manually go through the sequence. Use filler file names or values as you go. Once you've completed the recording, you've completed 80 percent or more of the work.

From there, copy the commands into your VB code and add:

```
PScript1.CommandString =
```

in front of each command. Of course you'll need to edit in quotation marks around the commands, and change any recorded values to variables, but recording is a great head start.

Second, the PerfectOffice VBX surfaces the PerfectScript product command browser. This feature, called the command inserter in PerfectScript, is accessed in a somewhat unconventional way, but you'll find it invaluable in building product commands (see Figure 2). For each product with a PID, PerfectScript gives you access to every command within that PID and all its parameters. You access this browser from the Properties window of the VBX control. You can either double-click on the Command Inserter property itself, or select it, then click on the ellipse button on the upper-right corner of the Properties Window. (The down-arrow button changes to an ellipse when you select the Command Inserter property.)

From this dialog, you first select the product whose command set you want to browse. Every product with a PID shows up in the drop-down list. After you select the product and click on OK, you'll see a list of every command in that product's

PID. For example, the WordPerfect product set contains 1824 commands. As you move through the list, you'll see the parameter list for the selected command in a second list box. Each parameter identifies the parameter's type and any enumerations for that parameter.

PerfectScript commands contain two peculiar conventions. Any command that begins with a question mark (or *hook* in the PerfectScript parlance) is a Query command. These commands return state information from the product, such as an open file name or the current font. The return value from a query command can be retrieved by the CommandReturn property of the VBX control. Here's an example of a query command that will get the current font name from a WordPerfect document:

```
PScript1.CommandLangCode = "US"
PScript1.CommandTargetApp = _
   WORDPERFECT
PScript1.CommandString = "?Font"
PScript1.CommandString = "Quit"
CurFont$ = PScript1.CommandReturn
```

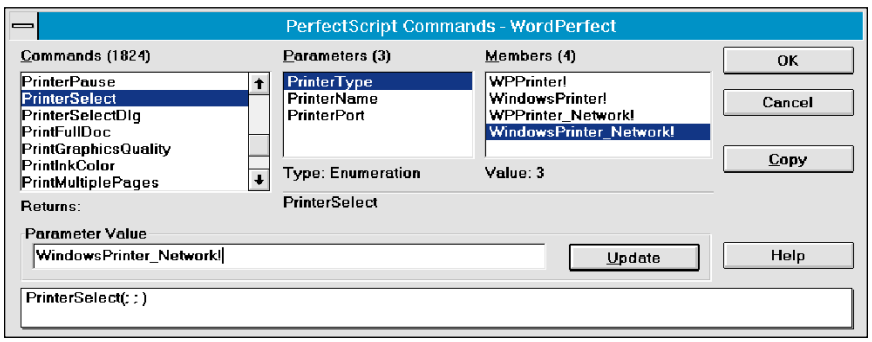

FIGURE 2 ***Can I See My Script?*** *The Command Inserter dialog is the VB programmer's tool for learning PerfectOffice's product command syntax. The dialog displays every product command, the parameter types, enumerated values, and a brief description of the command.*

Property	Possible Values	Purpose
CONTINUED FROM PAGE 53.		
PSVariableName		Sets the name of a variable in PerfectScript's persistent variable pool.
PSVariableValue		The value, in string format, of the variable.
PSVariableType	PS_STRING_TYPE = 0 PS_INTEGER_TYPE = 1 PS_FLOAT_TYPE = 2	The type of the variable.
CommandErrorCode	NO_ERROR = 0 COMMAND_NOT_FOUND = 1 INVALID_APP_PROPERTY = 2 COMMAND_TOO_LARGE = 3 NO_PRODUCTS_FOUND = 4 COMMAND_SYNTAX_ERROR = 5 PRODUCT_NOT_RESPONDING = 6 MACRO_OR_TEMPLATE_NOT_FOUND = 7 VARIABLE_ERROR = 8 PERFECTSCRIPT_SYSTEM_ERROR = 9 PID_NOT_ENABLED = 10	If a command does not execute correctly, PerfectScript returns an error code, which you'll find in the CommandError Code property.
ExistsResult	PS_VAR_NOT_EXISTS = 0 PS_VAR_EXISTS = 1	Used to test for the existence of a persistent variable in the PerfectScript persistnet variable pool. The variable name is set in the PSVariableName.
Top		The top position of the upper-left corner of the control in the form. Because the control is invisible at run time, it is not necessary to change this value.
Left		The left position of the upper-left corner of the control in the form. Because the control is invisible at run time, it is not necessary to change this value.
Index		Required by Visual Basic for VBX control.

Enumerations usually have text values associated with the numeric equivalents. These enumerated values end with an exclamation point "!" (called a *bang*). For example, the FileSave command has three parameters: the file name, export type, and overwrite. Overwrite is a numeric parameter, with three possible values, zero for no (don't overwrite), 1 for yes, and 2 for prompt (prompt the user). You'll see that these values can be represented as No! (0), Yes! (1), and Prompt! (2). If you pass No! to this command it will translate it to zero. These two lines are equivalent:

```
PScript1.CommandString = _
   "FileSave(""REPORT.WPD""; _
   WordPerfect_60!;No!)"
PScript1.CommandString = _
   "FileSave(""REPORT.WPD""; _
   WordPerfect_60!;0)"
```

While in the command inserter dialog you can build your commands and then use the copy button to place them on the clipboard. Go back to VB and paste the code into the appropriate code window. The command inserter is an invaluable tool for assembling product commands and learning the basics of PerfectOffice product command syntax.

To use the techniques outlined here you must be sure you are using the latest versions of the PerfectOffice applications: WordPerfect for Windows 6.1, GroupWise 4.1a, Presentations 3.0, and Quattro Pro 6.0. InfoCentral and Envoy are not accessible through the PerfectOffice VBXs.

There's more to cover, but you've learned the basics of using VB with PerfectOffice. Topics for future investigation include PerfectScript's persistent variable pool—a mechanism to pass data between PerfectOffice applications or between the apps and VB. The PSVariableName, PSVariableType, and PSVariableValue properties all make use of this pool.

You can also experiment with two additional VBXs. The WPDLG VBX exposes the PerfectOffice File Open, Save, Save As, and Select Directories dialogs to VB. With these dialogs, VB programs can take on the look and feel of a PerfectOffice application. The third VBX gives some extra control over Quattro Pro for Windows.

If you're looking for more information on programming VB and PerfectOffice, try checking the WPUSERS Forum on CompuServe. This independent forum is frequented by several PerfectScript and VB experts. Section 19 is the preferred forum for PerfectOffice discussions. Finally, the PerfectOffice SDK, due later this year from Novell, will document the VBXs and other PerfectOffice APIs. ■