GETTING WINDOWS APPLICATIONS TO WORK TOGETHER

CONVENTIONAL WISDOM ABOUT WINDOWS HOLDS THAT ITS KEY CONTRIBUTION to computing is graphics. As is usual with conventional wisdom, that's only partly correct. Windows graphics are important but only because they make possible a much more important benefit: application integration. In DOS, each application is an island, isolated, totally dominating its surroundings, unassailable by other programs and hardly able to share with them at all. In Windows, each program and document is treated as an independent object (represented by a graphic icon). Objects are malleable, can flow into one another, can borrow elements from each other. So the graphic character of Windows naturally leads toward greater integration of applications.

It is this ability to integrate that makes Windows so special. Before Windows, every PC ran the same operating system, the same applications, in the same way, all the time, everywhere. What kind of competitive advantage is possible with those kinds of cornerstones? But, with Windows' talent for integration, one person can use a PC more effectively than another by mixing applications, documents, and knowledge. You can make a better recipe from the basic ingredients than the other guy. Now, with Windows, your PC can really cook!

Basic Windows Integration

Windows is an integrating environment right down to its roots. From the moment it loads, Windows lets programs share the screen, run side by side, and pass resources back and forth. In that sense, virtually every tip in this book is about some aspect of integration. Here, though, we'll look at some tricks completely focused on getting applications to run side by side most productively.

Load Applications as Icons

If you are working in one Windows application and would like to load another one so that you can easily switch to it, you can automatically load the second application minimized. For example, if you are working in Write in one window and want easy access to the Calculator, but aren't ready to use it yet, switch to Program Manager and hold down the Shift key while double-clicking on the icon. Calculator will be running, but it will appear minimized at the bottom of your screen, as in Figure 9.1, ready for you switch to it with Windows' shortcut keys-Alt+Tab or Alt+Esc.

Make the Windows Clipboard Work for You

The most fundamental integration tool built into windows is the Clipboard. Here, information culled from any Windows program (and most DOS programs) can be stored and pasted into any other program. The notion of cut-and-paste as a way to move information is basic to Windows, and the Clipboard makes it happen. With a little imagination, you can use Clipboard cutting and pasting for all kinds of interesting results, as shown in the tips that follow.

Screen Captures with Clipboard If you need to paste a screen image into a Windows application, just press the Print Screen key on your keyboard. Your cursor will vanish for a moment as the screen image is copied to the Clipboard. If you open the Clipboard Viewer (double-click on the Clipboard icon in the Main program group) you'll see a copy of it there, as shown in Figure 9.2. You can then insert the image into any application by using the Paste command from the Edit menu or pressing Shift+Ins. To copy the active window only, press Alt+PrScr.

Copy Text from DOS Apps You can use the Clipboard's copy-and-paste capabilities with a DOS application running under Windows, provided that the DOS application is running in a window. Use the mouse to highlight the desired text, open the Control menu in the upper-left

corner of the window, and choose Copy from the Edit menu. To copy the entire DOS screen of information, press the Print Screen key and the information will be copied to the Clipboard.

Use the Windows Clipboard to Convert Formats The Windows Clipboard is a versatile utility because it can copy and paste data that's in many different formats. This feature is useful for transferring data that's in one format into an application that isn't compatible with that specific format. For example, you can paste a formatted text document from a word processor into a communications program that handles only ASCII files. Instead of having the formatting characters appear as gibberish in the communications program, the Clipboard will remove all the formatting, and paste in the document as plain ASCII text.

Keyboard Shortcuts: Clipboard Mice are nice, but the keyboard is often quicker for Clipboard actions. Most Windows programs that support the Clipboard offer a common set of keyboard shortcuts, including Shift+Del for Cut, Ctrl+Ins for Copy, and Shift+Ins for Paste. Newer applications have switched to Ctrl+X for Cut, Ctrl+C for Copy, and Ctrl+V for Paste, but they still support the older shortcuts as well.

Use Macros to Integrate Windows Applications

Virtually all Windows applications offer sophisticated macro recording capabilities that allow you to create scripts for application tasks. Often, these macros can extend to other applications from the same vendor, or even to other Windows programs. The macro recording capabilities of several dozen popular Windows applications are presented in detail in Part 2 of this book.

Windows' Own Macro Recorder Integrates Applications Too

Windows provides its own macro recorder, which can record and play back keystrokes or mouse actions from Windows itself and from just about any application that can run under Windows. The advantage of Windows' macro recorder is that it can integrate both application and Windows tasks into a single operation. Windows' macro recorder is covered fully in Chapter 6.

Make Sure You Remove Windows Applications Safely

One of the dangers of running applications together is that one can bring down many. One unexpected place this can happen is when an application is deleted. Few, if any, Windows applications include a routine for removing the application from your system. Before you start hastily deleting files, make sure that removing the app doesn't interfere with any of your other Windows applications, or Windows itself. Here's a systematic approach to removing a Windows application:

1.Use SysEdit or a text editor to open your WIN.INI file and locate any sections or lines that refer to the application, then delete them. If you are in doubt about a line's or section's function, do not delete it.

2.Save the edited WIN.INI file.

3.Open your CONFIG.SYS and AUTOEXEC.BAT files and also look for lines or references to the specific application. Remove or change these as well.

4.If you have any data files saved within the application's directory, make sure to move them or copy them to a floppy if you will need them later.

5.Then use the File Manager to delete the application's directory and all of its subdirectories and files.

6.Finally, in Program Manager, remove the application's icon by selecting it and choosing Delete from the File menu or pressing the Delete key. Windows will ask you to confirm that you want to delete this item, as shown in Figure 9.3.

Coping with Conflicts

Applications running together in Windows 3.0 provoked endless arrays of unrecoverable application errors (UAEs)-the problem from which no program returns. Officially, UAEs are gone in Windows 3.1. In truth, while they are far less devastating, come with much better information on their source, and have a dandy new name (general protection faults), these application collisions remain a headache. We've included information here on both techniques for working with Windows 3.0 UAEs and Windows 3.1 problems. Even if you have already upgraded to 3.1, some of the 3.0 tips covered here may still be of value.

The Lingering Spectre of Windows 3.0 UAEs

If you receive a UAE that results in the termination of your current application, the error was probably caused by the application writing to a region of memory it doesn't have access to. When this happens, whatever is already stored in that area of memory can become corrupted. As a result, Windows becomes unstable. Once an application causes this UAE, other applications that you run may generate the same error, even though they are not the cause of the problem.

To avoid this situation, exit Windows and reboot your computer as soon as the application is terminated. Then check the following to find out the cause of the UAE:

Are you running Windows applications that were designed for Windows 2.x? These older applications should only be run in Real mode, as you are told by Windows when you attempt to start them in Standard or Enhanced mode.

Was an incorrect machine or network selected during Windows setup? Check the installed settings by running the Windows Setup program from the DOS prompt and entering the correct settings.

Are incompatible TSRs running on your system? Remove all TSRs and see if the application now runs without errors. If it does, try adding the TSRs back one at a time to isolate the one causing the problem.

Do you have a page-mapping conflict when running in Enhanced mode? To test for this conflict, run Windows in Standard mode (WIN/S). If there is no problem in Standard mode, you'll have to determine which memory address is causing the problem. Open your SYSTEM.INI file in SysEdit or a text editor, and add the following line to the [386Enh] section to exclude Windows from using the adapter segment area of memory:

```
EMMExclude=A000-EFFF
```

Now exit Windows, restart it again in Enhanced mode, and run the problematic application. If the problem is solved, try pinpointing the exact location of the hardware that is causing memory conflicts so that you don't have to exclude the entire area.

Are you using a correct version of DOS? Systems often come with versions of DOS customized for their brand of PC, and these should be used rather than a generic version of DOS. Likewise, these DOS versions should only be used on the brand of machine with which they're supplied.

Are free system resources low? To find out, use the About command in the Program Manager's Help menu to see if they are below 20 percent. If they are, you'll need to keep fewer windows or applications open in future Windows sessions to avoid UAEs. Get in the habit of regularly checking free system resources so that when they start getting low you can close windows and applications to prevent a UAE.

Do you have enough file handles available for applications? UAEs are often caused when the system doesn't have any file handles available for an application. You should increase the

FILES= command in your CONFIG.SYS file to meet the high demands of Windows, which generally has multiple applications open at once. The default number of file handles that DOS has available is only 8, but Windows users will need to up the ante to around 50.

Is your environment large enough? Applications that are denied environment space are another cause of UAEs. Increasing the environment to 1 or 2K should alleviate the problem. To make this change, edit your CONFIG.SYS file so that the line

SHELL=C:\DOS\COMMAND.COM

includes the E switch for specifying the environment variable. To set the environment to 2K, change the line to read

SHELL=C:\DOS\COMMAND.COM/E:2048

Keep the Dr. Watson Utility on Call

In Windows 3.1, UAEs are a thing of the past; they've been renamed general protection faults. While a name change doesn't make them any less daunting, at least now you've got some help in tracking down the cause of these system errors. Windows 3.1 comes with a handy diagnostic utility called Dr. Watson, which provides you with feedback when an application error occurs. Unfortunately, Dr. Watson won't do you any good unless you manually install it on your system. To add Dr. Watson so that it runs in the background every time you start Windows, make the StartUp group the active window in Program Manager, choose New from the File menu, click on OK to select Program Item in the New Program Object dialog box, fill in Dr. Watson as the description, and specify DRWATSON.EXE in the Command Line text box.

Once you've installed Dr. Watson, you'll be ready if an application error occurs, because the program creates a log with detailed system information of what happened. Armed with DRWATSON.LOG, you can troubleshoot problems yourself or provide valuable detail when you to talk to technical support people.

Dr. Watson will also work with Windows 3.0; you just need to download it, along with a file called TOOLBOX.DLL, and place it in your Windows directory. You can download Dr. Watson from the Microsoft Product Support Download Service. The number is (206) 637-9009, and the filename is WW0440.EXE.

To automatically start Dr. Watson in version 3.0, you have to edit your WIN.INI file using a text editor such as Notepad or SysEdit. In the [windows] section of the file, which should be the first section, add DRWATSON.EXE to the Load= line. If there's already a program after Load=, just leave a space between it and DRWATSON.EXE. Once you've done so, the first few lines of WIN.INI might look like this:

```
[windows]
Load=DRWATSON.EXE
Run=
Beep=Yes
Spooler=Yes
```

Advanced Windows Integration

The real meat of Windows integration doesn't lie on the desktop or in the Clipboard. It can be found in the rich tools Windows makes available to programmers, and through them to users, for tying together applications-their information, commands, and capabilities-to create customized programs. Two approaches dominate this area: dynamic data exchange (DDE) and its newer partner, object linking and embedding (OLE). We'll take a quick look at both here.

Get the Most from DDE

Do you wish you could get someone else to exchange information between Windows apps instead of relying on yourself and the trusty Windows Clipboard? Call up dynamic data exchange

instead. When it works, it's slicker than a rain-soaked L.A. freeway. But when it doesn't, it's enough to make you scream at your computer. DDE links are fragile, so you'll find it useful to understand how they are built, how they operate, and how to fix them when they go awry.

A DDE link (called a *conversation*) is formed by two Windows programs, called the *server* and *client* applications. (The server application is the source of the data.) Both parties must support DDE, and these days, most general-purpose Windows programs do. Excel, Word for Windows, Ami Pro, WordPerfect for Windows, and 1-2-3 for Windows, for example, are all on the roster. The server application supplies data to the client and updates it either automatically or on demand. One common DDE scenario is a spreadsheet server application providing figures for a table in a client word processor.

Setting up the link is straightforward. Select the data in your server program and either press Ctrl+Ins or use the Copy command on the program's Edit menu. Then open the client document, put your cursor where you want the data to appear, and use the client program's Paste Link command, like the one shown in Figure 9.4 (sometimes it's a subset of the Edit menu's Paste Special option). That's all there is to it.

If the Paste Link command is unavailable (gray), you won't be able to establish a link. There are two possible causes for this. Either you've copied data from a program that isn't DDE-capable or the copied data came from a file that wasn't saved to disk. Many DDE server programs refuse to link data until it has been saved at least once.

Once you've set up the link, you need to know about three potential problems: performance drag, inability to find the server's data, and inability to launch the server app. To help you cope with these hazards, your client program should provide you with a command called Edit Links, Link Options, Change Links, just plain Links, or something similar.

Most DDE clients create automatic links that are almost instantly updated whenever the data from the server changes. An automatic link is also commonly called a *hotlink*. If you don't need continuous updating, you can improve the performance of your client application by changing the automatic link to an inactive or manual one. When you want to update your client document thereafter, just select the link and click the Update button.

In the example of linking a spreadsheet range to a table in a word processing document, it's important to know how the link is defined. You could create a link to the cell range that you desire-for example A1:F5. But if you change the layout of your worksheet, the desired information will reside in some other range and your document will be connected to a meaningless patch of cells. The solution is to name the spreadsheet range and then tell the document to look for the named range rather than the cell addresses.

If you ever move or rename your server document, you have to edit the link that's been created. To fix the link, point to the new location of the server document.

A more difficult problem sometimes arises when you try to refresh a DDE link to a server document that is not open. Your client program in that case should offer to launch the server application and document for you. Under some circumstances, you might accept the offer, look at an hourglass pointer awhile, and then receive a maddeningly uninformative failure message from the client application. If this happens, check the following:

Do you have enough memory and system resources to launch the server program? Switch to Program Manager and use the About Program Manager option on the Help menu to check. If you're low on either memory or system resources, close a few programs and then try again.

Is the server program stored in a directory in your DOS path? If it isn't, quit Windows, modify your AUTOEXEC.BAT file to change the path, then reboot or simply type **autoexec** at the DOS prompt, and then try again.

If memory is plentiful and the server is in the path, something else may be spoiling your fun. Excel 3.0, for example, may refuse to launch via DDE if any add-in macros (.XLA files) are stored in your XLSTART directory. But you won't find that information in the Excel manuals.

The bottom line is that you should test any DDE link thoroughly before you depend on it. And bear in mind that although DDE will hang around forever on the macro level because of the

volume of code that's already been written to use it, you can expect object linking and embedding to supersede many DDE functions.

OLE Secrets

Object linking and embedding (OLE) is new to Windows 3.1. It represents a huge potential benefit for users and is a sign of the extraordinary customizability that lies ahead for PC applications. While DDE creates a user-configurable channel between applications, OLE goes much further. OLE allows you to link programs in complex ways and to actually place one program within another program's document. Through OLE, for instance, not only a copy of a spreadsheet, or the data drawn across a link to a spreadsheet, but the full capabilities of the spreadsheet program could be plunked in the middle of a word processor's report document. It works as follows.

Windows' OLE capability allows you to create an object-such as a drawing or sound file-in one Windows application and then insert it into another file. This object can either be linked, in which case it actually exists in a separate file, or embedded, in which case it exists within the primary file. For example, suppose you are working on a Write document and you use OLE to tie a Paintbrush drawing to it. If the drawing is linked, when you double-click on it, you get the parent application (Paintbrush) with the original document window as it was saved to disk. If the drawing is embedded, when you double-click on it, you get a window from the parent application that points to the embedded data. (Furthermore, the File menu changes to read Update rather than Save, and Exit and Return rather than just Exit.)

As we discuss OLE, you'll need to be familiar with the following terms:

Object: Data encapsulated in a document so that it can be displayed and manipulated by the user. This creates a so-called compound document. Any kind of data can be made into an object if it was created in a Windows application that supports OLE.

Package: The icon that represents the object that is embedded in a document.

Client application: A Windows application that can accept, display, and store objects. The client application stores information about embedded objects: the page position, how the object is activated, and which server application is associated with the object.

Server application: A Windows application in which you can edit an object that is embedded in the client application.

Source document: The file where the data or object was originally created.

Destination document: The document where the data is embedded or linked.

Although Windows' applets, such as the Clipboard, and programming services, such as dynamic link libraries (DLLs), facilitate OLE, it's Windows applications that make OLE work. The server application provides data that is either linked to or embedded in a client application's document. Linked data is updated whenever you modify the original document. On the other hand, embedded data isn't modified until you click on it, which launches the server app from within the client document.

Setting Up an OLE Link When setting up an OLE link (from a spreadsheet program, for instance), you first copy data from the server application. This sends the selection to the Clipboard in several formats-a bitmapped image of the data (.BMP or metafile), the data in its native file format (such as .XLS for Excel), and others, such as Rich Text Format (.RTF). It also sends an OLE marker file called Link.

In the client application's compound document, you choose the OLE paste command (Paste Special, Paste Link, or another variant, depending on the application). This command sends a call to OLE'S DLLs, which in turn search the Clipboard for the Link marker file. If the call

finds Link, the client application then calls the OLE DLLs to ask the server application to make a link.

Making the link takes two steps: First, the data is pasted from the Clipboard to the client application's file. Then the client app's paste operation attaches reminders that the link exists to the original data file and to the document in the client application.

The server file's reminder tells it to contact the OLE libraries anytime someone modifies the data that has been pasted into the client document. The client file's reminder tells it to alert the OLE libraries whenever the file is opened. If the server file's data has changed, the client file also changes.

Embedding an Object Setting up object embedding is similar to the linking process in that you copy data in several formats to the Clipboard, but is different in that it also stores code that acts as a pointer to the server application, rather than to a data file.

From the client application, you choose the OLE Paste command, and the client app sends the data, screen representation, and pointer to the server application. The embedding process is then complete.

When you double-click on the embedded object, the client application uses the pointer to search the OLE DLLs for the OLE Registry, which records every OLE-capable application you've installed. This finds the server app's code.

Once the client app locates the server program, the Registry calls a library function, which in turn launches the server program (or brings it to the foreground if it's already running in the background). The client application then sends the embedded graphics object to the server application that just came up. Once you've made and saved modifications, the link updates the embedded object and closes the server app (or returns it to the background).

When to Link and When to Embed The only challenge to using OLE is deciding whether to link or embed. Linking stores only a pointer to the server data, while embedding stores a copy of the data in the client program. If you're tying together data from various corners of your hard disk or network, you can avoid unnecessary file fattening by linking instead of embedding. Linking is also preferable when the same data must be available to different client documents.

On the other hand, use embedding when the client document is likely to go off-line from the server. Getting that linked-in sound annotation to speak may be a little tough if you're on the road and the server's back home. You should also choose embedding if there's a possibility you may move or rename the server document. Linked objects have memory, but they can't find documents that have been moved.

Register Applications That Support OLE In order to take advantage of OLE, a Windows application must be registered in the Registration Database. Many newer Windows applications come with a registration file (.REG) that contains information about how the application uses OLE. To install an existing .REG file into the database, run the Registration Editor from Program Manager by selecting Run from the File menu and typing **regedit** in the Command Line text box. Then, from the Registration Editor's File menu, choose Merge Registration File. Locate the .REG file for the application that you want to add to the database, and then select OK. The file will be added to the database, and the application will be able to take advantage of OLE functionality.

Another way to merge a registration file into the database is through File Manager. Locate the .REG file in File Manager and double-click on it to install it automatically.

Quickly Restoring the Registration Database If you've tinkered with settings in the registration files for your Windows applications, you may end up with a Registration Database that is corrupted. But never fear, you can restore the database to its original condition. To do so, exit Windows and delete the file REG.DAT, which is located in the WINDOWS directory. Restart Windows and run the File Manager. In the WINDOWS\SYSTEM subdirectory, locate the file SETUP.REG and double-click on it.

A message appears to confirm that the information has been registered. The database now contains the original registration information that was installed with Windows. If any of your other applications have .REG files, you can add them to the restored database by choosing Merge Registration File from the File menu, as described in the previous section.

Drag and Drop OLE You can use Windows' drag and drop capabilities to easily embed objects in documents, provided that the server application has been written for Windows 3.1. If you are using an older Windows application, you have to use the Object Packager (described in the next section) to embed the object. Of course, if you use drag and drop from File Manager you will be embedding entire files, such as a spreadsheet, instead of just part of one, for instance a range of cells.

For example, to embed an Excel spreadsheet in a Write document, follow these steps:

1.Have Write running in one window and File Manager running in another so that they are both visible on the screen at the same time.

2.In File Manager, select the name of the file that you want to embed, and drag it to the spot in the Write document where you want the package to appear.

3.When you release the file an Excel package will appear, as shown in Figure 9.5.

The only drawback to embedding large objects, such as a sound file, is that the object is a duplicate of the original file. This means that you are using precious hard disk space to store the same file twice.

Take Advantage of Object Packager Object linking and embedding is a great way to funnel data between two applications, but for this integration to work, both programs must support OLE. Fortunately, Windows 3.1 ships with an accessory called Object Packager (its icon is shown in Figure 9.6), which offers a way around the OLE-support requirement. Object Packager lets you embed an iconic representation of a data object into another application (which must still support OLE as a client application). But the embedded data need not originate in an OLE server. For example, you could create a Cardfile database of all of your .INI files, like that shown in Figure 9.7. Each card describes a program's initialization file and includes a Notepad icon that represents the file in question. Double-click on the icon to call up Notepad, with the appropriate .INI file loaded and ready to edit.

Without Object Packager, this link would be impossible to make, because although Cardfile in Windows 3.1 is an OLE client application, Notepad doesn't know a thing about OLE. The linkage is possible only because the program that did the embedding-namely Object Packager-is an OLE server. And Object Packager can encapsulate any data file, regardless of its origin, into an object package.

For the click-and-edit procedure to work, the .INI extension must be associated with the application NOTEPAD.EXE. If this association doesn't already exist on your system, you can create it by using the Associate command in File Manager's File menu.

There are several ways to create a packaged object, but we'll follow one of the more straightforward methods here. In a File Manager window, select the file you want to package and choose the Copy command from the File menu (or cut to the chase by pressing F8, the keyboard shortcut). In the dialog box that appears (shown in Figure 9.8), choose the Copy to Clipboard option and click on OK. Now start Object Packager, which appears in the Accessories program group by default.

Object Packager has two windows, named Appearance and Content. Select the Content window and pull down the Edit menu. Now you can use either the Paste or Paste Link command to create your package. The choice between these two commands is important. When you transfer your finished package to your client application, you will be embedding that package, not linking it. It's a given that the packages themselves cannot be linked, but what's *inside* the

package can be either embedded or linked data. If you create the package with Object Packager's Paste command, you'll be embedding. If you use Paste Link, you'll be linking.

The pros and cons here are exactly the same as they are for any other linking-versus-embedding decision. The short form of the decision is this: Choose embedding if you want to encapsulate the entire source file and ensure that it will always be available, in its current form, to the client document. Choose linking if you want to encapsulate only a pointer to the source file. You should also choose linking if you plan to include the package in many different client documents and you want to ensure that each client is hooked up to exactly the same data.

In the case of the .INI file database, where all you need are pointers to the .INI files, linking makes sense and embedding does not. Embedding would not only create an overweight Cardfile document but also store frozen copies of the .INI files. As soon as an .INI's parent application made changes to the .INI file, the Cardfile document would be out of date.

Once you choose Paste or Paste Link, you can customize the package in several ways-not the least of which is changing the icon and the descriptive text, as shown in Figure 9.9.

When the package is ready, use the Edit menu's Copy Package command to put it on the loading dock-the Windows Clipboard. The rest of the procedure is standard OLE. Just activate your client program and use the appropriate Paste command.

This example shows how Object Packager can connect an OLE client document to a non-OLE document. But Object Packager's talents don't end there. Using it, you can also embed an iconic representation of data from an OLE server application. This technique annotates compound documents with tidy icons instead of cluttering client documents with visible data. Interested readers can check annotations by clicking the icons; browsers can choose to skip right over them. You can even create an elaborate hypertext document by embedding additional packages within packaged files.

Object Packager has another important virtue: You can use this program to embed a Windows command line attached to an icon. For example, you can package startup commands for your DOS applications, your Excel macros, and your favorite games. If the program you use most often is an OLE server, you can use it to create a launch document full of program-starter packages. With a little imagination, you'll find that once you get started with the Object Packager, the possibilities are virtually endless.

Embed a Windows Accessory Because the Object Packager allows you to embed most any kind of data, you can use it to embed a Windows accessory, not a specific data file, into an application. For example, if you were editing a large document such as an annual report in your word processing program, you might want the ability to easily access the Notepad, where you could jot down notes about the report, things to do, whatever comes to mind. Embed a package for Notepad and place it at the beginning of the report. If your word processor can split the screen so that the top part is always in view, you will have instant access to the Notepad, as shown in Figure 9.10.

To create the Notepad package, start the Object Packager and activate the Content window by clicking on it or using the Tab key. From the Edit menu, select Command Line, and enter **notepad.exe** in the Command text box. Activate the Appearance window, and select the Insert Icon button. From the Insert Icon dialog box, choose an icon or select Browse to have access to more icon choices, as shown in Figure 9.11. Next choose Label from the Edit menu and type in a label for the icon. Finally, select Copy Package from the Edit menu. In your word processor, position the cursor where you want the icon to appear, select Paste Special, and choose the object.

Reinstall Windows 3.1 without Losing OLE Functionality If you are planning to reinstall Windows 3.1 to a new location, you don't want to have go through the trouble of reregistering all your Windows applications that support OLE. Reregistering applications usually means spending time reinstalling the applications, because for many apps that's the only way to properly register them in the database. You can avoid going through this time-consuming process by copying your existing REG.DAT file into the new location where you'll be installing Windows.

If you've already installed Windows to the new location and just realized that this means recreating your Registration Database, you can still avoid reinstalling Windows applications if you haven't yet deleted your previous copy of Windows. Exit to DOS and copy the REG.DAT file from the old Windows directory into the new one. Start Windows and then start the Registration Editor. Select the file SETUP.REG and double-click on it to execute it. Setup will read the information in the REG.DAT file and make sure that your new Windows installation uses these OLE settings.

Quicker Objects Some Windows applications that support OLE have a command that will save you time when creating objects to embed within that application. When you want to embed an object, select the Insert Object command and select the type of object that you want to embed. For example, in Microsoft Word for Windows you can choose the Insert Object command from the Insert menu and then select the appropriate object, as shown in Figure 9.12. This shortcut saves you from having to switch to the server application and copy the desired information to the Clipboard; the object is automatically placed into the document once it is created or retrieved.

Troubleshooting OLE If an object that is embedded in an application does not launch the server application when it is double-clicked, check your WINDOWS\SYSTEM subdirectory for the files OLECLI.DLL and OLESRV.DLL. OLE will not work if either of these files is missing. If they are there, one or both of the files may be corrupted. To restore these important files you'll have to expand them from your original Windows disks and place them in the SYSTEM subdirectory.

Application Development Tools for Windows Integration

All of the integration methods described in this chapter can be manipulated in various ways by the many application development tools available for the Windows environment. These range from sophisticated scripting tools, such as Asymetrix Toolbook, through programs designed to integrate data from other applications, like Borland's ObjectVision, to complete programming languages, such as Microsoft's Visual Basic. The three products mentioned here are covered in Part 2 of this book. An excellent source for understanding how to use all Windows development tools to create custom applications is Paul Bonner's *PC/Computing Customizing Windows 3.1* (Ziff-Davis Press, 1992).