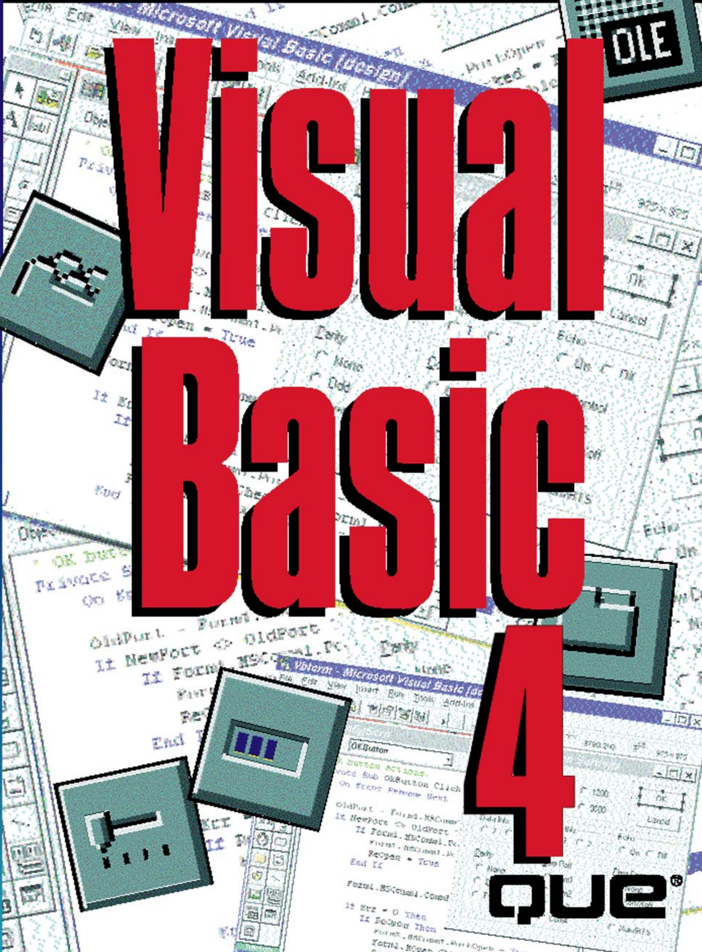
The Official Visual Basic Programmer's Journal Guide To Visual Basic 4

Covers All The HOT Topics Developers Want to Learn Including:

- Optimization
- Database Access
- Using the Win32 API
- OLE
- Client/Server Development
- Multimedia
- And More!

Developed & Co-published by Visual Basic Programmer's Journal

Includes a CD with All the Source Code, Project Files and more!



que®

By Roger Jennings

Roger Jennings is a consultant specializing in Windows database and multimedia applications. He is the author of Que's *Unveiling Windows 95*; *Using Windows Desktop Video,* Special Edition; *Discover Windows 3.1 Multimedia* and *Access for Windows Hot Tips;* and was a contributing author to Que's *Using Windows 95,* Special Edition; *Excel Professional Techniques*; *Killer Windows Utilities*; and *Using Visual Basic 3.* He has written two other books about creating database applications with Access and Visual Basic, co-authored a book on using Microsoft Visual C++ for database development, is a contributing editor of *Visual Basic Programmer's Journal,* and has written articles for the *Microsoft Developer Network News* and the MSDN CD-ROMs. Roger is a principal of OakLeaf Systems, a Northern California software development and consulting firm; you may contact him via CompuServe (ID 70233,2161), on the Internet (70233.2161@compuserve.com), or on The Microsoft Network (Roger_Jennings).

Introducing 32-Bit OLE 2.1

Object Linking and Embedding has come a long, long way from its beginnings as a built-in element of early versions of Microsoft PowerPoint. Subsequently, 16-bit Microsoft Office applications for Windows 3 adopted OLE 1.0 as their primary method of inter-process communication, supplementing Clipboard copy-and-paste applications and DDE. Windows 3.1 added OLE 1.0 services (provided by OLECLI.DLL and OLESVR.DLL) as a basic component of the Windows graphical user interface. In 1993, Microsoft introduced 16-bit OLE 2.0, which added a variety of new features to OLE 1.0 through a collection of seven DLLs stored in \WINDOWS\SYSTEM. The most important of the OLE 2.0 additions were in-place activation and editing of embedded objects, nested embedded objects, the ability to drag and drop objects into compound documents, and OLE Automation. In just a couple of years, OLE Automation has proven to be the most important feature of OLE 2x.

Visual Basic 3 provided the MSOLE2.VBX custom control and a helper DLL, MSOLEVBX.DLL, to provide basic OLE 2.0 container capabilities. Visual Basic 3 also supported the client side of OLE Automation with its **CreateObject** and **GetObject** functions. At the time of Visual Basic 3's release, there were no commercial OLE Automation server applications, which explains the lack of real-world OLE Automation code examples in the Visual Basic 3 *Programmer's Guide*. Visio Corporation (then Shapeware) introduced the first OLE Automation server; Microsoft quickly followed with Excel 5.0, Word 6.0, and Project 4. Access 2.0, an OLE Automation client, included support for a prototype version of today's OLE (Custom) Controls.

Windows 95, Windows NT 3.51, and 32-bit OLE 2.1 eliminate many of the problems associated with the initial implementations of 16-bit OLE 2.0. The most vexing problem for users of OLE 2.0 has been Out of Memory messages when attempting to link or embed objects created with Microsoft Office mega-apps, such as Word and Excel. The 32-bit OLE 2.1 also promises improved performance as applications are upgraded and tuned to take full advantage of 32-bit operating system features, such as multithreading and (under Windows NT 3.51+) symmetrical multiprocessing. Ultimately, Network OLE will eliminate the single-workstation orientation of OLE 2.1.

This chapter provides an overview of Visual Basic 4's role as an OLE 2.1 client application. All the examples in this chapter are 32-bit, because the future of OLE is inexorably tied to Microsoft's 32-bit operating systems. The majority of the examples, however, are equally applicable to 16-bit Visual Basic applications and 16-bit OLE 2.0 servers. The chapter begins with a definition of important OLE 2x terms, explains the registration of OLE servers in the Windows 95 and Windows NT Registry, goes on to describe how to use the OLE container control and insertable object controls in a variety of scenarios, and concludes with an introduction to the client side of 32-bit OLE Automation.

Defining OLE 2.1 Terms

Getting a grip on OLE 2.1 requires an updated glossary of OLE terminology. Following are definitions of the most important of the current OLE 2x buzz words:

- *Component Object Model* (COM) is Microsoft's infrastructure for code modules (called *objects*) that are independent of programming languages and computer platforms. COM is oriented to C++ programming on Intel PCs, but Windows NT 3.51+ also supports COM on MIPS, DEC Alpha, and PowerPC RISC systems. COM defines a set of *interfaces*, implemented as a virtual table (Vtbl), that points to entries in member function tables. COM requires that all objects support the IUnknown interface, which has AddRef, Release, and QueryInterface member functions. The AddRef and Release functions maintain a count of the number of instances of the object in use. The QueryInterface function returns information on the capabilities of the object. COM is implemented by Compobj.dll under Windows 95 and Compob32.dll under Windows NT. OLE is Microsoft's high-level COM implementation.

- *OLE Documents* allow an application to share data created by another OLE-enabled application. A document that contains data created within another application is called a *compound document*, which can be stored in a *compound file*. (OLE 1.0 called compound documents *destination documents*.) Data can be embedded in an OLE Document or created by a link to a file containing the data. Embedded data is contained entirely within the OLE Document, whereas liked data relies on an external file. Monikers (IMoniker interface) identify and manage references to embedded or linked OLE objects, including their data element(s). Uniform data transfer allows objects that implement the IDataObject interface to pass data via the Windows Clipboard, through OLE drag-and-drop methods, and within a compound document.

- *Container applications*, previously called *OLE client applications*, are capable of displaying and manipulating OLE Documents. The 16-bit and 32-bit versions of Microsoft Excel, Word, Access, PowerPoint, Project, and Visual Basic 4 are OLE container applications.

- *Local servers* are executable applications that are capable of embedding or linking objects within a container application's OLE Document. Local servers, as the name implies, must reside on the same computer as the container application. Local servers are classified as *full servers*, which provide both linking and embedding capabilities, and *mini-servers*, which only can embed data in a compound document and don't open or save files. Most container applications are local servers; however, neither Visual Basic 4 or Access are local servers. Microsoft Graph 5.0 is an example of a mini-server.

- *Automation servers* expose *programmable objects* for manipulation by client applications that include an (OLE) Automation-compliant programming language, such as VBA. The Automation client usually is, but need not be, an OLE container application. An Automation server can be implemented as an *in-process* .dll, which shares the same address space as the Automation client, or an *out-of-process* .exe, which uses LRPCs (Lightweight Remote Procedure Calls) to communicate with the Automation client. Access 95 and Schedule+, although not local servers, are automation servers. Visual Basic 4 is capable of creating both in-process and out-of-process local Automation servers as well as remote Automation servers.

- *OLE Custom Controls*, are a special class of in-process Automation server that also expose events. You can add an OLE Control to any container application, but events exposed by the control are accessible only to container applications such as Visual Basic 4 and Access 2.0/95, which are specifically designed to act as OLE Control containers. Future versions of Microsoft Office applications, plus authoring applications for The Microsoft Network and the Internet (codenamed Blackbird during the beta cycle), are expected to support OLE Controls. Chapter 11, "VBX versus OCX," describes OLE Controls in detail.

- *Remote Automation servers*, also called *Remote Automation Objects* (RAOs), are *out-of-process* Automation servers that communicate via Remote Procedure Calls (RPCs) with networked Automation clients. Visual Basic 4 is the first programming language to be capable of creating RAOs. RAO technology in Visual Basic 4 is the precursor to Network OLE. One of the most important applications for RAOs is developing three-tier client/server database applications, where an RAO (called a *LOBject* for Line-of-Business object) is used in the middle tier to implement business rules. RAOs are one of the subjects of Chapter 18, "Using Remote Data Objects."

One of the objectives of COM is create *component applications*, which are made up of a combination of local servers, plus in-process or out-of-process automation servers to implement high-end features. The mega-apps that comprise the Microsoft Office software suite are logical candidates for componentizing. As an example, you might want to incorporate only the basic functionality of an Excel worksheet or use Word as a minimal word processor in a Visual Basic application. Although you don't need all the features of either Excel or Word, the present versions of these Microsoft productivity applications are monolithic. Large, monolithic applications require a long time to load and consume resources big time. Componentizing best-selling applications raises the specter of licensing individual pieces of highly profitable mainstream software; thus it may be some time before you see "pieceware" versions of Microsoft productivity applications.

Note

> To qualify for use of Microsoft's trademarked "Designed for Windows 95" logo, 32-bit Windows applications must support OLE 2x containers, objects, or both containers and objects. This requirement, which does not apply to utilities or applications that run exclusively in full-screen mode, is sure to increase the number and variety of OLE servers and, to a lesser extent, OLE container applications. The level of required OLE support depends on whether an application uses files. Applications that use files must support OLE drag and drop. Support for in-place activation, compound files, and OLE Automation is "strongly recommended," but not mandatory.

Interoperating with 16-Bit and 32-Bit OLE Applications

OLE 2.1 running under Windows 95 and Windows NT provides support for interoperability of 16-bit and 32-bit by a process called *thunking*. Following are the rules for using 16-bit and 32-bit OLE objects with container applications:

- 32-bit OLE 2x container applications can embed or link objects created by 16-bit OLE local servers, including OLE 1.0 servers.
- 16-bit OLE 1.0 container (client) applications cannot embed or link objects created by 32-bit OLE 2x local servers. 16-bit OLE 2.0 containers can (or should be able to) embed or link objects created with 32-bit OLE 2x local servers by thunking.
- 32-bit OLE 2x clients can call 16-bit out-of-process Automation servers, and vice versa. All Automation servers are OLE 2x objects. Out-of-process servers that have custom interfaces, however, must have the same "bitness."
- In-process OLE .dlls and OLE Controls must have the same "bitness." Thus separate 16-bit and 32-bit versions of OLE Controls are provided with Visual Basic 4.

Note

> Windows 3.1+, Windows 95, and Windows NT 3.5+ use different thunking processes (Universal, Flat, and Generic, respectively.) For further information on the thunking process for OLE 2x operations, see the "Thunk Layer Operation" chapter of the *OLE Programmer's Reference* in the Win32 SDK.

Registering OLE Servers

All OLE servers, including Automation servers and OLE Controls, must be registered before you can use them. Windows 3.1+ registers OLE servers in its registration database, REG.DAT. Windows 95 and Windows NT uses the Registry (System.dat), a more robust version of REG.DAT, to store information about the location and capabilities of OLE servers. Well-behaved OLE servers of all types are self-registering on installation and should de-register themselves as part of the uninstall process required for compliance with the "Designed for Windows 95" logo guidelines. All OLE servers aren't likely to be well-behaved, so you should be familiar with methods of checking, editing, and deleting registry entries. The following two sections describe how to use Windows 95's Registry Editor and the Regsvr32.exe application.

Using Windows 95's Registry Editor

Familiarity with the Windows 95 and/or Windows NT registry is a must for 32-bit Visual Basic 4 developers. As an example, if you manually move an OLE server's executable file (instead of uninstalling and reinstalling the application), you must alter the contents of the Registry to reflect the new well-formed path to the server. You launch Windows 95's Registry Editor, Regedit.exe, from the \Start\Programs\Accessories\System Tools menu. Unlike Windows 3.1's REGEDIT.EXE, you do not need to specify verbose mode with a /v command-line switch to display detailed Registry entries. Windows 95's Registry Editor uses the Explorer model to display the hierarchy of Registry entries, which is much more complex than that of Windows 3.1+'s REG.DAT.

OLE 2 server registration takes place in the HKEY_CLASSES_ROOT\CLSID hive of the Registry. CLSID (Class ID) is a 32-character GUID (Globally-Unique IDentifier) that uniquely identifies each OLE 2 server. To check the registration data for an OLE server, launch the Registry Editor and choose ~~E~~dit, ~~F~~ind to open the Find dialog. Type the executable file name, including the extension, in the Find What text box and click the Find Next button. Press F3 to bypass file extension association entries until Regedit reaches the ...\CLSID hive. Figure 7.1 shows the primary LocalServer32 entry for Excel 95 (version 7.0), Microsoft Excel Sheet. If the entry does not point to the current location of the server, double-click the string ("AB") icon to open the Edit String dialog. Edit the Value Data string as required to specify the new location for both the 32-bit LocalServer32 and 16-bit LocalServer values, unless you want to use Excel 5.0 for testing 16-bit container applications.

In addition to the CLSID for Excel 95's Sheet object (ProgID = Excel.Sheet.5), there are successive CLSID entries for Excel Chart (ProgID = Excel.Chart.5), and Application objects, each of which points to *d*:\path\excel.exe. ProgID is an abbreviation for Programmatic ID, the name of the object used with Visual Basic's **CreateObject**() function for creating an instance of an Automation server object. Launching Excel 95 as an Automation server requires adding the /Auto command-line parameter to preclude its window from appearing and to prevent addition of the Microsoft Excel item to the taskbar. If you change the location of a server, you need to alter each incidence of the server location string in the Registry. Unfortunately, the Registry Editor doesn't have an Edit, Replace menu choice.

Note

> Windows 3.1+'s REGEDIT.EXE (opened with the /V parameter for verbose mode) and Windows NT's 3.5's Regedt32.exe use similar entries to specify the location of OLE servers. Only Windows 95's Regedit.exe, however, includes the full-featured Edit, Find feature described in this section. You must manually locate entries for server registration in Windows 3.1+ and Windows NT 3.5+. The hierarchy of Registry hives in Windows NT is quite similar to that of Windows 95.

Regsvr32.exe, which is included with Visual Basic 4, is a command-line application that adds, updates, or deletes registry entries for OLE servers, including OLE Controls. You must use Regsvr32.exe to install OLE Controls that don't come with an installation application; many shareware and freeware OLE Controls don't include a setup feature. To register an OLE server, use the following syntax in the Open text box of Windows 95's Run dialog:

```
regsvr32.exe d:\path\oleserver.ext
```

The preceding syntax example assumes that Regsvr32.exe is located in the \Windows, \Windows\System, or another folder on the current DOS path. Figure 7.2 shows the message box that appears when registration of an OLE Control, in this case the Lenel MediaDeveloper L_dvid32.ocx described in the preceding chapter, is successful. The conventional location for Regsvr32.exe, OLE Controls, and Automation servers you create with Visual Basic 4 is the \Windows\System folder.

If you want to remove an OLE 2x server or intend to move an OLE 2x server to a new location, take advantage of Regsvr32.exe's capability to unregister a server. The server must be present for unregistration to work, so don't delete or move the server's file until you execute the following command line:

```
regsvr32.exe /u d:\path\oleserver.ext
```

Figure 7.3 shows the somewhat strange message you receive when unregistering a server. You can repeatedly register and/or unregister servers with no ill effects. Unregistering a server you move to a new location is useful to ensure that all the registration data is updated.

Using Visual Basic 4 to Create OLE Container Applications

Like Visual Basic 3, Visual Basic 4 includes an OLE container control. Unlike Visual Basic 3's MSOLE2.VBX, Visual Basic 4's OLE Control is intrinsic; it automatically appears in the toolbox when you launch Visual Basic 4. The OLE container control is a "wrapper" object that provides a set of properties, methods, and events, which are applicable to a wide variety of embedded and linked objects. The following sections describe the features of the OLE container control and how to use the OLE container control to embed or link a variety of OLE objects.

Working with the OLE Container Control

Visual Basic 4's OLE container control is capable of displaying and manipulating documents created by any OLE full server or mini-server, including OLE 1.0 servers. Adding an OLE container control to a Visual Basic 4 form is similar to creating a compound document with any other OLE container application, such as Microsoft Excel or Word. In the case of Visual Basic 4, the form, rather than an Excel worksheet or Word document, is the container document. An Access 2.0/95 form with an unbound object frame control also is a container document. When you add an OLE container control to a Visual Basic form, the standard OLE 2.0 Insert Object dialog appears, as shown in figure 7.4. The Object Type list displays each OLE local server registered by your computer. The following two sections describe how to embed or link documents with the OLE container control.

Embedding Objects in the OLE Container Control

To create a new (empty) embedded instance of an object, with the Create New option selected, pick the server to create the object from the Object Type list and click OK. Alternatively, you can double-click the Object Type item. An empty Excel 95 worksheet embedded in a form with the OLE container control initially appears, as shown in figure 7.5. The empty worksheet object is activated for in-place editing, superimposed over the *presentation* of the object. The presentation of an OLE object is a static Windows metafile that displays an object's data when the object is not activated. In design mode, the embedded object grafts all or a part of its menu structure to the form; embedded objects don't display the server's ~~F~~ile menu choice. Menus grafted from OLE servers don't appear in run mode unless you add a menu bar to your form and set the NegotiateMenus property of the form to **True** (the default).

To activate the worksheet object, double-click the presentation surface. Alternatively, with the mouse pointer over the presentation surface, click the right mouse button to open a pop-up menu that displays the verbs applicable to the object. Most servers offer ~~E~~dit and ~~O~~pen choices; ~~E~~dit activates the object in place, and ~~O~~pen opens the server's window. Figure 7.6 illustrates an Excel worksheet activated in run mode. A menu bar with a single ~~F~~ile menu choice is added to the form to enable grafted menu negotiation. (The "Lenel MediaDeveloper Controls" section of chapter 6, ”Using Visual Basic as a Multimedia Front-End," describes the menu grafting and negotiation process for OLE Controls. Excel's row and column headers, scroll bars, and sheet tab(s) appear as frame adornments. You must provide a margin at the top and to the left of the OLE container control to accommodate Excel's frame adornments. You can enter data in the empty activated worksheet and use many of Excel's grafted menu choices to manipulate the embedded object. The eight sizing handles allow you to adjust the size of the activated object within the limit of the size of the OLE container control. Pressing Esc deactivates the object and displays its presentation in the OLE container control (see fig. 7.7).

Tip

> If you want to activate the OLE object when your form opens, apply the appropriate DoVerb (vbOLE*Constant*) method to the object in the Form_Activate event handler. The DoVerb method is one of the subjects of the "Properties and Methods of the OLE Container Control" section, later in this chapter. Activating the control from the Form_Activate event, rather than the Form_Load event, displays the form and the presentation of the object while the server loads. Depending on the speed of the user's PC and free system resources available, opening the object's server can take an appreciable amount of time.

Note

> The OLE 2x specification's state diagram for embedded objects requires that a single click outside the surface of the activated object must deactivate the object. Clicking the surface of the form (outside the OLE container control) doesn't deactivate the object in run mode. If your form contains one or more controls in addition to the OLE container control, you can use the Form_Click event handler to apply the SetFocus method to another control. If you don't want another control to appear on your form, add to the form a small PictureBox control with no border and a ForeColor value that matches the ForeColor property value of your form. Apply the SetFocus method to the PictureBox and then to the OLE container control, as in the following example:

You also can embed an object based on data contained in a server file. When the Insert Object dialog appears upon adding an OLE container control to a form, click the Create from File option button to display the file version of the Insert Object dialog. Click the Browse button to select in the Browse dialog the file that contains the data. The file extension association contained in the HKEY_LOCAL_MACHINE\ROOT hive of the Registry identifies the appropriate server for the document. Figure 7.8 illustrates selection of a Word 95 document. (Although Word 95 is version 7.0, Word objects are classified as Microsoft Word 6.0 documents to maintain backward compatibility with 16-bit Word 6.0 and 32-bit Word for Microsoft NT.) When you click OK, the upper-left corner of your document appears in Word's Page Layout presentation. (Normal and Outline view are not available when embedding Word documents.)

When you activate the Word object, either by double-clicking the OLE container or by a Form_Activate event handler, Word's toolbars appear as sizable floating toolbars and the Word menu is grafted to your form's menu, as shown in figure 7.9. Unlike Excel, Word 95 doesn't provide frame adornments for navigating the document; you must use the cursor positioning keys to maneuver to a specific location in the Word document.

Note

> The OLE container control exhibits several anomalies with embedded Excel 95 worksheets. For example, Word's toolbars appear as floating toolbars after you activate an embedded Word 95 document object, but Excel 95's toolbars do not appear on activation. Although the Formula Bar choice appears when you open Excel's ~~V~~iew menu, the Formula Bar does not appear, except for a brief flash below the menu bar. Activating embedded Excel 95 worksheets created from .xls files results in peculiar appearance in the OLE container, especially if Excel's AutoFormat method has been applied to the worksheet. One of the problems with activating in Visual Basic 4 Excel 95 worksheets based on files is described in the Microsoft Knowledge Base document Q129793, "BUG: Excel Displays Only First Column in Embedded Worksheet."

The extent to which you can manipulate activated embedded document objects within the OLE container control is dependent on capabilities contributed by the server. Each OLE 2x server has its own set of features and idiosyncrasies when used to embed objects in compound documents. Idiosyncrasies, in particular, confuse users who expect the embedded object to behave in an OLE container control exactly as the object behaves when opened in its server application or when inserted as an object in a conventional container document. Thus you're likely to find that opening the server's window, rather than in-place activation, is the better design approach, especially when dealing with documents based on files. Substitute the vbOLEOpen constant for vbOLEUIActivate as the argument of the DoVerb method in your Form_Activate event handler to open the server's window.

Linking Objects to the OLE Container Control

When you embed an object with an OLE container control, the object's data is contained in the form. Other applications cannot access the embedded data, other than by Clipboard operations with an active instance of the object. When you compile a project, the embedded data is incorporated in the project's .exe file. Very large embedded objects, such as spreadsheets with many rows and column, full-screen color bitmaps, or multimedia (.wav and .avi) files, can have a profound effect on the .exe file's size. The solution to these problems is to link the file to the OLE container object. Linking only embeds a pointer file; like embedding an object from a file, the file extension determines the OLE server that is used to open the file. The linked file may be located on a server, and you can create a link to a part of a file, such as a range of cells in an Excel worksheet, a paragraph of a Word document, or a particular segment of a multimedia file.

Creating an object linked to an OLE container control is similar to embedding an object from a file. Select the Create from File option, select the file that contains the data you want to link, and mark the Link check box (see fig. 7.10.) Click OK to create the link. When you run your form, the presentation appears as shown in figure 7.11. Double-click the surface of the OLE container control to open Excel 95's window, or use the right mouse button to display the pop-up verb menu; choosing either ~~E~~dit or ~~O~~pen from the verb menu launches Excel 95 and displays Excel's window.

By default, the server opens in a normal window of modest dimensions (see fig. 7.12). All the server's menu choices are available when you open the server's main window, including the ~~F~~ile menu, so you don't need to include a menu bar on your form.

When you create a link to a file, the path to and name of the file appears as the value of the SourceDoc property. 32-bit Visual Basic 4 supports long file names and Uniform Naming Convention (UNC) for files located on servers. Thus you can use \\*ServerName**ShareName**FileName.ext* as the value of the SourceDoc property. UNC file names eliminate Invalid link messages when server shares are mapped to workstation drive letters, which may change from user to user.

You can specify a subset of the data in the linked file to appear in the presentation window by specifying a data identifier in the SourceItem property text box. For Excel spreadsheets, you use row-column (R#C#:R#C#) syntax or a named range; Word documents use Bookmarks to specify a particular block of text. An alternative method is to open the file in the server's window, select the cells or text you want to link, and copy the selection to the Clipboard. Right-click the OLE container control, and choose Delete Object from the pop-up menu; then open the pop-up menu again, and choose Paste Special to open the Paste Special dialog. Select the Paste Link option (see fig. 7.13,) and click the OK button to re-create the link. When you open the server's window, the selected cells or text is selected automatically (see fig. 7.14).

Note

> When you change a link by typing a new value for the SourceDoc property or alter the value of the SourceItem property, a Delete Current Link message box appears. Click Yes to delete the existing link; then right-click the container control to open the pop-up menu. Choose Create Link to re-create the link to the new file and/or data item. If you want to change only the SourceItem property, delete the data item extension (such as !R1C1:R12:C6) from the SourceDoc value; then retype the data item identifier in the SourceItem property text box.

Properties and Methods of the OLE Container Control

The preceding sections describe the most commonly used properties and methods of the OLE container control. Online help for the OLE container control provides (in most cases) the information you need to set property values and apply methods programmatically. Following are some of the changes to OLE container control methods and properties between Visual Basic 3 and 4:

- The Action property of the Visual Basic 3 OLE container custom control is replaced by methods of the Visual Basic 4 OLE container control. The Action property is retained for backward compatibility; new OLE applications should use the methods that correspond to the 14 Action constants. (Search online help for the Action property of the OLE container control for the methods.)

- The Verb property of Visual Basic 3 is replaced with the DoVerb method, although you can still use the Verb property and its numeric arguments with existing code. The syntax of the DoVerb method is ole*Name*.DoVerb(vbOLE*Constant*), where vbOLE*Constant* is one of seven intrinsic constants whose value ranges from 0 to [nd]6. The vbOLEHide constant ([nd]3) hides the server's window after you choose the Open option of an embedded control or apply the DoVerb method with the vbOLEOpen ([nd]2) constant. When the server is hidden, the presentation of the object is hatched with diagonal lines. Using the vbOLEShow constant ([nd]1) activates an embedded object in-place, as does vbOLEUIActivate ([nd]4).

- The Zoom and Stretch values of the SizeMode property seldom are useful in real-world applications other than displaying bitmapped and vector images. Large spreadsheets or other documents are illegible when Zoomed or Stretched. Using the AutoSize value, which causes the control to fit its content, often makes the size of the OLE container control larger than the form.

- OLEDropAllowed is a new property that, when set to **True**, enables the OLE container control to act as a target for OLE drag-and-drop operations. OLE drag and drop is the subject of the next section.

Implementing OLE Drag and Drop with the OLE Container Control

To obtain the right to apply Microsoft's "Designed for Windows 95" logo to an application that uses files, the application must support the OLE drag-and-drop feature of Windows 95. OLE drag and drop lets you drag the icon of a file that's supported by a local OLE server from My Computer or Explorer and drop it on the surface of an OLE container control. The new object created by the drag-and-drop process replaces the object, if any, in the OLE container control. If you drag a desktop shortcut to a file supported by an OLE server, an icon representing the shortcut appears in the OLE container control. You double-click the shortcut to open the server's window to edit the shortcut's file.

To enable an OLE container object as an OLE drag-and-drop target in run mode, set the value of the OLEDropAllowed property to **True** (the default is **False**) and set the value of the OLETypeAllowed property to Linked (only), Embedded (only), or Either (0, 1, or 2, respectively). In design mode, you can use the OLE container control as an OLE drop target without setting OLEDropAllowed to **True**. Figure 7.15 shows a linked Excel 95 worksheet created by dragging the Dtv_toc.xls file icon from Windows' Explorer to an OLE Control container in run mode.

Note

> The presentation of a link that you create in run mode does not appear when you return to design mode. To make the presentation appear in design mode, right-click the OLE container control and choose Create Link from the pop-up menu.

Binding the OLE Container Control to an OLE Object Field

Like many of Visual Basic 4's other OLE Controls, the OLE container control is data-enabled, letting you display OLE objects linked to or embedded in OLE Object fields of Access tables. Visual Basic 3's OLE Control was not data-enabled, so displaying objects contained in Access OLE Object fields involved a substantial amount of code to deal with Access's "OLE wrapper." When you bind the OLE container control to an OLE Object field, the control behaves much the same as Access's bound object frame control. For most Visual Basic 4 developers, displaying OLE Objects in tables of .mdb files is the most common application for the OLE container control.

To bind a Visual Basic 4 OLE container control to a field of Access's OLE Object type, add a Data control to a form, set the value of the DatabaseName property to point to an Access .mdb file, such as Northwind.mdb, that contains a table with an OLE Object field, and then choose the table with the OLE Object field as the RecordSource of the Data control. Add an OLE container control to the form; then click Cancel when the Insert Object dialog appears to create an empty control.

Connect the DataSource property of the OLE container control to the Data control, and select the OLE Object field for the DataField property. Figure 7.16 shows a simple form that displays the contents of the Photo, Notes, LastName, and FirstName fields of Northwind.mdb's Employees table. No code is required to create the form shown in figure 7.16. Double-clicking the bitmapped presentation opens the Windows Paint application to let you edit the image.

Writing and Reading OLE Data Files

Embedded OLE objects you create or modify in OLE container controls in run mode are not persistent; the content is lost when you return to design mode or close your executable application. You can save the presentation and data of an embedded OLE object or the presentation and link pointer of a linked OLE object with the OLE container's SaveToFile method. The following code saves an embedded OLE object in an OLE container named oleExcelSheet to the Sheet1.ole file when you choose ~~F~~ile, ~~S~~ave from the form's menu:

```
Private Sub mnuFileSave_Click()
   Dim intFileNum As Integer
   intFileNum = FreeFile
   Open "Sheet1.ole" For Binary As #intFileNum
   oleExcelSheet.SaveToFile intFileNum
   Close #intFileNum
End Sub
```

The file is saved in OLE stream format. Reading the data from an OLE file created by the preceding subprocedure and updating the presentation requires use of the ReadFromFile method, as in the following example:

```
Private Sub mnuFileOpen_Click()
   Dim intFileNum As Integer
   intFileNum = FreeFile
   'Open the file
   Open "Sheet1.ole" For Binary As #intFileNum
   oleExcelSheet.ReadFromFile intFileNum
   Close #intFileNum
   'Update the presentation
   oleExcelSheet.DoVerb (vbOLEUIActivate)
   picEmpty.SetFocus
   oleExcelSheet.SetFocus
End Sub
```

The last three lines of code update the static presentation to display the data from the file by momentarily activating the embedded Excel worksheet object. The picEmpty picture box control serves as a control to which focus is set temporarily to deactivate the OLE container control.

Insertable Objects

As noted in the "Working with the OLE Container Control" section earlier in the chapter, Visual Basic 4 forms are OLE containers. You can prove this by embedding an *insertable object* directly in a form, rather than into an OLE Control. Visual Basic 4's online help defines an insertable object as "[a]n object of an application, such as a Microsoft Excel Worksheet, that is a type of custom control." Visual Basic 4 forms are simple OLE containers that support operations such as grafted menus (menu negotiation) and in-place activation, but most insertable objects don't provide a window to display the object's presentation.

You can add buttons representing insertable objects to Visual Basic 4's toolbox from the Custom Controls dialog. Marking only the Insertable Objects check box results in a list of objects identical to those in the Object Type list of the Insert Object dialog (see fig. 7.17). Like OLE Controls, the items you check are appended to the toolbox when you click the OK button. The default name of an insertable object is derived from its object type: Excel objects are Sheet1 or Chart1, and a Media Player object is named mplayer1.

Insertable objects that support in-place activation, such as Excel Worksheet and Chart objects, graft their menus to Visual Basic 4 menu bars when activated. Figure 7.18 shows insertable Media Player and Excel Chart objects added to a form; the Chart object is activated for editing. Unlike OLE Controls, insertable objects have only a generic property set and a few events that are common to all control objects. Insertable objects do not support methods, except two generic methods, SetFocus and Move. Insertable objects created by OLE Automation servers, however, have an Object property that you can use to program the insertable object. Thus the primary application for today's insertable objects is to provide a visual presentation for programmable objects. Programming objects exposed by OLE Automation servers is the subject of the remaining sections of this chapter.

Creating Integrated Applications with OLE Automation Servers

OLE 2.0 introduced the concept of OLE servers that expose programmable objects for manipulation by OLE Automation client applications. An OLE Automation client need not be a container application, because an invisible instance of the OLE Automation server handles manipulation of its programmable objects. Any application that incorporates VBA is an OLE Automation client, and all members of the Professional Edition of Office 95 are OLE Automation servers. (Microsoft Word 6.0 and Word 95 are not full-fledged OLE Automation servers, but the Word.Basic object exposed by these versions of Word lets you manipulate Word documents with Word Basic commands.) An OLE Automation server need not be an OLE full server; Access 95 is an example of an OLE Automation server that cannot contribute objects to a compound document created by an OLE container application because Access 95 is not an OLE 2x full server or mini-server.

Excel 5.0 was the first product to incorporate VBA as an application programming language and to expose objects as an OLE Automation out-of-process server. Visual Basic 3, which predated the retail release of Excel 5.0, was the first OLE Automation client. Today, most of Microsoft's software offerings to the business community are OLE Automation servers or are in the process of being updated to expose programmable objects. Microsoft's competitors, such as Novell and Lotus, were adding OLE Automation to their forthcoming 32-bit productivity software suites when this book went to press. When Microsoft releases the next major upgrade to Windows NT 3.51+, presently code-named Cairo, 32-bit OLE Automation will become a basic component of the Windows operating system.

The following sections discuss the basics of programming predefined hierarchies of objects exposed by commercial full-server applications. As noted in the "Defining OLE 2.1 Terms" section near the beginning of this chapter, Visual Basic 4 lets you create your own in-process and out-of-process local Automation servers, plus remote Automation servers. Chapter 19, "Remote Automation," discusses out-of-process servers that you can access over a network.

Exposing Object Properties and Methods with Type Libraries and References

The Microsoft OLE 2.0 *Programmer's Reference* defines properties of programmable objects as "member function pairs that that set or return information about the state of an object, such as whether or not an object is visible." Methods are single member functions "that perform an action on an object, such as resizing it or causing it to evaluate member data." OLE Automation interfaces provide access to the collection of member functions through the IDispatch, IEnumVariant, ITypeLib, and ITypeInfo interfaces. The IDispatch and IEnumVariant interfaces access to individual objects and object collections, respectively. ITypeLib supplies information about the objects exposed by the Automation server, and ITypeInfo provides details about the properties and methods of each object. The purpose of ITypeLib and ITypeInfo interfaces, which are not mandatory, is to assist object programmers by providing information about the exposed objects with object browsers.

VBA introduced the use of references to type libraries supplied with OLE Automation servers. Type libraries for out-of-process servers use .olb and .tlb extensions; many in-process OLE DLLs, such as DAO3032.dll, have self-contained type libraries. (Most OLE Controls use .oca type libraries to speed loading of the control.) VBA also adds an object browser that lists the exposed objects, plus the properties and methods of each object. Another advantage of using a type library is that the Visual Basic interpreter/compiler checks the syntax of your OLE Automation code in design mode. Visual Basic 3 and Access 2.0 detect errors in OLE Automation source code only when the code is executed in run mode.

To make a type library visible in Visual Basic 4's object browser and accessible by the VBA interpreter/compiler, you create a reference to the type library in the References dialog that opens when you choose ~~T~~ools, ~~R~~eferences. The names and locations of type libraries appear in Registry entries. Figure 7.19 shows the References dialog listing type libraries in the Available References list. To make a type library accessible to Visual Basic 4 and the object browser, you must mark the check box to the left of the library name and then click OK to close the References dialog.

Visual Basic creates some references, such as those for insertable object controls, automatically. You must add references to object libraries of applications such as Microsoft Excel, Access, Word, and Project manually. (Word 95's Word95 Objects for ACCESS type library exposes only the few methods of the Word95Access object needed for mail-merge applications.) Once you've added references to the server applications you plan to use, information contained in the referenced type libraries is accessible from the object browser. Objects contained in your current Visual Basic 4 project appear as the default entry in the Libraries/Modules drop-down list of the object browser. To display the objects, methods, properties, and constants defined in an OLE Automation type library, open the Libraries/Modules list and pick the appropriate library. Figure 7.20 shows the extraordinary syntax for the ChartWizard method of the Chart object of the MSGraph5 OLE Automation mini-server. Clicking the ? button displays the help file for object, method, or property. (The ChartWizard method, which is used by Access 95 and Excel 95, is not documented in the online help file for Microsoft Graph 5.0, but The Excel 95 VBA help file has a topic that covers the ChartWizard method).

Note

> Special type libraries for insertable objects used as controls contained in Visual Basic 4 forms are temporary type library files named VB####.tmp, which are stored in your \Windows\Temp folder. #### represents an arbitrarily assigned four-character hexadecimal number. As an example, the type library for an insertable Graph object (named MSGraphCtrl) might be VB4290.tmp. Type libraries for insertable objects expose the standard set of properties and methods applicable to all Visual Basic 4 controls that are created from insertable objects. A .tmp control type library does not enumerate the object(s) exposed by the OLE Automation server.

Unlike OLE Controls, programmable OLE objects do not expose events. If you link or embed a programmable object in an OLE container control, events such as Updated are triggered by the control, not by the server object contained in the control. All arguments and return values of methods and properties for exposed objects are of the **Variant** data type for compatibility with the IEnumVariant interface.

Creating and Manipulating Programmable Objects

The easiest way to experiment with programmable OLE objects is to add an insertable object or OLE container control to a form and then write code in the debug window to manipulate the control's Object property. An Excel Sheet object linked to an OLE container control or an MSChartCtl insertable object control is a good starting point for learning the basics of OLE Automation client programming. Figure 7.21 illustrates use of the debug window to modify the default chart that appears when you add an MSChartCtl control to a form. You gain access to the programmable aspects of the control through the Chart1.Object.Application.Chart object. The presentation of the insertable chart object changes as you alter the values of the Chart object's properties in the debug window.

To minimize the amount of typing necessary to refer to the Chart object, create a pointer to the Chart object by declaring an object variable in the declarations section of your form and then setting a pointer to the Chart object in the Form_Load event handler, as in the following example:

```
Option Explicit
Private chtTest As Object

Private Sub Form_Load()
   Set chtTest = Chart1.Object.Application.Chart
End Sub
```

After creating a pointer to the Chart object, you use the new object variable in statements such as those as shown in figure 7.22. The last five statements in the debug window of figure 7.22 are equivalent to the **With...End With** structure in the following code:

```
Private Sub Form_Load()
   Set chtTest = Chart1.Object.Application.Chart
   With chtTest
      .HasTitle = True
      .ChartTitle.Text = "Sales by Territory"
      .ChartTitle.Font.Size = 18
      .ChartTitle.Font.Italic = True
      .Type = xlBar
   End With
End Sub
```

Note

Documentation for the objects, methods, and functions of Graph5.exe in the MSGraph.hlp file is limited, to be charitable. Excel 95's online help has substantially more information on the object hierarchy of MSGraph5, plus the properties and methods of MSGraph5 objects. Type **Chart Object** in the Index page of Help Topics: Microsoft Excel dialog to display the starting point for Chart objects. MSGraph5 is of limited utility as a Visual Basic 4 graphing tool because you cannot gain direct access to the Datasheet object that stores data values for the Chart object. Graph5.exe expects source data from a specified range of an Excel worksheet or from an application, such as Access, in Excel's native BIFF format.

Getting and setting the cell values of an Excel Sheet object linked to an OLE container control uses a similar approach. Setting a pointer to a Sheet object exposed by Excel requires the following code:

```
Option Explicit
Dim xlsTOC As Object

Private Sub Form_Load()
    Set xlsTOC = _
        oleExcelLink.Object.Application.Workbooks(1).Sheets(1)
End Sub
```

Examples of getting and setting the properties of a Sheet object in the debug window appear as shown in figure 7.22. Changes to cell values are reflected in the presentation. To close the invisible instance of Excel, you apply the Quit method to the Application object, which causes a Save changes? message box to appear, if you make changes to the Sheet object. You can bypass saving changes and the message box by executing xlsTOC.Parent.Saved **= True** to clear the dirty flag prior to executing xlsTOC.Application.Quit.

Using VBA's *CreateObject* and *GetObject* Functions

If you don't want to display an object's presentation on your form, you can create object variables with the **CreateObject** or **GetObject** function. To use either of these functions, first declare an object variable in the declarations section of a form or module. **CreateObject** is related to inserting an empty object of a type chosen in the Insert Object dialog, and **GetObject** parallels the Insert from File option of the Insert Object dialog. The generalized syntax of these two functions is:

```
Set objVariable = CreateObject("AppName.ObjType")
Set objVariable = GetObject(["d:\path\filename.ext"], ["AppName.ObjType"])
```

The *AppName.ObjType* argument consists of the application name and the object type you want to create or open, such as Word.Basic for Word 6.0/95 or Excel.Sheet for Excel 5.0/95. The *AppName.ObjType* argument is optional for the **GetObject** function if the file extension specifies, through an association entry in the Registry, the particular object type you want. In most cases, you don't need to use the second object. If you omit **GetObject**'s *d:\path\filename.ext* argument, you must provide the *AppName.ObjType* argument for an open instance of the application and object type.

The following code example populates a list box, lstFeeds, with the first 100 unduplicated items from a list of satellite wildfeeds (unscheduled programming) contained in a Wildfeed.xls file. The program names appear in column A of the worksheet, beginning at row 2.

```
Option Explicit
Private xlsFeeds As Object

Private Sub Form_Activate()
    Dim intCtr As Integer
    Dim strProgram As String

    lstPrograms.Clear
    Set xlsFeeds = GetObject("e:\vb4\wildfeed.xls")
    For intCtr = 2 To 101
        strProgram = xlsFeeds.Range("A" & CStr(intCtr)).Value
        If lstPrograms.ListCount = 0 Then
            lstPrograms.AddItem strProgram
        ElseIf lstPrograms.List(lstPrograms.ListCount - 1) _
                <> strProgram Then
            lstPrograms.AddItem strProgram
        End If
        lstPrograms.Refresh
    Next intCtr
    xlsFeeds.Parent.Saved = True
    xlsFeeds.Application.Quit
    Set xlsFeeds = Nothing
End Sub
```

Adding the preceding code to the Form_Activate event handler and executing the lstPrograms.Refresh statement in the **For...Next** loop lets you watch the list box fill with the entries as they are retrieved from the Wildfeeds worksheet. Setting xlsFeeds to **Nothing** frees resources associated with the xlsFeeds object.

Tip

> If you're performing complex operations on another application's programmable objects and the application supports VBA, you can speed operation of your client application by writing a VBA function in the server application and calling the function from your Visual Basic 4 code. Communication with out-of-process servers is by Lightweight Remote Procedure Calls (LRPCs), which are very slow in comparison to execution of VBA code contained in a server application module. Writing as much VBA code as possible in the server application makes the testing process easier and faster, too. The Run method of Excel's Application object lets you execute from Visual Basic 4 a function written in Excel VBA. For more information on the Run method, search Excel 95's online help.

Expanding Your Use of Visual Basic 4 to Create OLE Automation Client Applications

Getting the most out of integrated applications requires familiarity with the hierarchies of objects exposed by OLE Automation servers. Excel 95 exposes more than 100 objects and object collections. The Word.Basic pseudo-object lets you use a multitude of sometimes-arcane Word Basic commands. (VBA's capability to use named arguments makes dealing with Word Basic commands much easier than Visual Basic 3 code.) Publishing limitations preclude adding code examples to this chapter for all of today's OLE Automation servers, but the preceding examples should serve to get you started writing VBA code to create large-scale integrated productivity applications.

Summary

Microsoft is betting the future of its Microsoft Office productivity software suite, the BackOffice server suite, and upgrades to the Windows 95 and Windows NT operating systems on the Component Object Model and 32-bit OLE 2.1+. This chapter emphasizes use of Visual Basic 4's 32-bit OLE container control in conjunction with OLE Automation objects exposed by Excel 95 and Word 95. As Windows 95 and Windows NT 3.51+ gain momentum, virtually all mainstream Windows applications will include OLE full server and Automation features. Today's OLE mega-servers are heavy-hitters in the resource department; future versions are likely to be replaced by fully-componentized applications that let you pick the feature set you need for your Visual Basic 4+ OLE client applications. When members of Microsoft Office 95, the Blackbird authoring tools for MSN and the Internet, and other best-selling softwar are upgraded to include OLE Control container features, the market for 32-bit OLE Controls will expand by at least an order of magnitude. As this chapter demonstrates, however, you can use today's OLE full server, mini-servers, and Automation severs, both in-process and out-of-process, to create useful 32-bit OLE 2.1 component applications with Visual Basic 4.