Office Development (General) Technical xArticles

Using Microsoft OLE Automation Servers to Develop Solutions

Ken Lassesen
Microsoft Developer Network Technology Group

October 17, 1995

Sixteen sample applications are associated with this article. The buttons that access those samples are located throughout the article where each sample is discussed.

Abstract

This technical article shows how to use the many OLE Automation servers available in Microsoft® Office and other products to develop solutions to fulfill corporate development needs. Sixteen different servers are discussed, with examples of code provided for each.

Introduction

OLE Automation enables many Microsoft products to work together as a single entity. To appreciate the potential, consider the following scenario:

A travel industry customer service representative uses a Visual Basic® application as the ultimate business solution. When a customer calls, the application uses the Remote Data Object (RDO) to access data from Microsoft® SQL Server. The application inserts this customer data into Microsoft Excel and creates a variety of pivot tables that it presents to the representative. The application inserts selected data into Microsoft Access and prints for the customer a report of possible flights. The customer selects a list of cities to visit and the application launches Microsoft Project to create and print an itinerary. The application passes the new transaction to Microsoft Word to print a confirmation letter. From Microsoft SQL Server, the application pulls pictures and text and creates a PowerPoint® presentation of the trip for the customer. Entries are made into the representative Schedule+ file for follow-up phone calls. All of these reports and presentations are e-mailed or faxed to the customer using the Messaging API (MAPI), which saves mail, paper, and handling costs. Finally, all of the documents are bound into a unit, by using the Binder, and then saved.

What do the customer service representatives do now? They do what they did before—the *application*, not the *user*, controls all of the products. Representatives need no additional training time—they create no documents in Microsoft PowerPoint or Microsoft Project. They are ready to handle their next customer, and the customer receives a rich set of documents as an extra service.

Not all products that support OLE features support OLE Automation. For example, many products enable OLE linking and embedding, but not OLE Automation. Linking and embedding allow the user to access the object; OLE Automation allows one program (the controller) to control another program (the server).

This article serves as a jump start in using OLE Automation servers. There are many OLE Automation servers available—I will not explore any one OLE Automation server in depth. I will only cover the server sufficiently for you to write code that works, to show you what you need, and to send you in the right direction. To prevent version-specific complexities, I will concentrate on the Windows® 95 versions. I would recommend a brief reading of Kenneth Nilsen's "Using the OLE Automation Interface with Visual Basic" (MSDN Library Archive, Conference and Seminar Papers) as a general introduction to OLE and its concepts, or "Your Unofficial Guide to Using OLE Automation with Microsoft Office and Microsoft BackOffice" for a high-level overview.

Before we look at the servers, I will discuss the OLE Automation controllers and point out a few important general concepts.

OLE Automation Controllers

In this article I use the term *controller* for anything that controls an OLE Automation server. Other common names are *OLE Automation controller*, *OLE controller*, or incorrectly, *OLE Automation client*. The OLE Automation controller can connect to an OLE Automation server using one of several different methods. The method depends on the controller product and the version of the controller product.

A controller may connect to a server in one of three ways:

- Late binding
- Early binding
- OLE control binding

Once the OLE Automation controller establishes a connection, it controls the server with the same commands that the server uses. The OLE Automation commands require ordered arguments syntax in most earlier controller product versions, such as Visual Basic versions 2.0 or 3.0, or Microsoft Access version 20. All of the 32-bit versions of the controller products (for example, Windows 95 versions) support the preferred named arguments syntax.

The ABCs: Arguments, Bindings, and Controllers

Table 1 shows the OLE connection methods for all of the Microsoft products discussed in this technical article. All future products from Microsoft should support early binding and named arguments. If a product cannot function as an OLE Automation controller, "Controller Binding Allowed" in Table 1 is marked as None. (Note that version 7.0 of a product is the same as a "95" designation, for example, Microsoft Excel 95 is the same as Microsoft Excel version 7.0.)

Table 1. Relationship of Products, Binding, and Arguments

Official Name	Version	Controller Binding Allowed	Arguments
Microsoft Access	1.1, 2.0	Late, OLE Control	Ordered
Microsoft Access	7.0	Late, Early, OLE Control	Ordered, Named
Microsoft Excel	4.0	None	
Microsoft Excel	5.0	Late	Ordered, Named
Microsoft Excel	7.0	Late, Early	Ordered, Named
Microsoft Graph	1.0	None	
OLE Messaging	1.0	None	
Microsoft PowerPoint	4.0, 5.0, 7.0	None	
Microsoft Project	4.0, 7.0	Late, Early	Ordered, Named
Microsoft Schedule+	7.0	None	
Microsoft SQL Server	6.0	None	
Microsoft Word	2.0, 6.0, 7.0	None	
Visual Basic	4.0	Late, Early, OLE Control	Ordered, Named
Visual Basic	2.0, 3.0	Late, OLE Control	Ordered

A Rose by Any Other Name . . .

Depending on the products you are using, there are several different sets of vocabulary to

describe OLE components.

- In this article, I call the program that supplies services to an application a *server* and the application that uses the services a *controller*.
- A variable declared as an **Object** data type or as an *application-defined object type* ("server.object": **Excel.Wordsheet**, **DAO.DBEngine**) is called an *object*.
- The variable that has been initialized (bound to a server) is called an *instance*.
- The set of methods, properties, and objects that the server makes available through an instance is called the *class.*
- You can *create* and *destroy* an instance many times in a procedure. You can modify its properties repetitively.
- You can *declare* an object only once in a procedure. You cannot change or modify the class—it exists external to the controller.

Binding the Controller to the Server

The easiest way to explain the difference between bindings is to show you some code samples of the three ways of establishing the connection to the server. I will then explain what the differences mean to the developer. For a more technical discussion, see "Information for Visual Basic Programmers" in the "National Language Support Functions" Appendix under OLE Automation in the Win32® Software Development Kit (SDK) documentation.

Late Binding

Late binding declares a variable as an object or a variant. The variable is initialized by calling **GetObject** or **CreateObject** and naming the OLE Automation programmatic identifier (ProgID). For example, if the ProgID is "Mom.ApplePie," the code could appear like this:

```
Dim objPie As Object
Dim objSlice as variant
Set objPie = CreateObject("Mom.ApplePie")
Set objSlice = CreateObject("Mom.PieSlice")
```

Late binding was the first binding method implemented in controller products. Late binding is the friendly name for what C programmers call **IDispatch**-based binding. It uses a lot of overhead—it is faster than DDE, but slower than early binding. It is available in all products capable of being controllers. All OLE Automation servers support late binding.

Early Binding

Early binding declares a variable as an application-defined object type. Early binding is the friendly name for what C programmers call *virtual function table bindings* or *vtable binding*. Although some variables can be declared with **New** for some servers (this would initialize the variable automatically), avoid using it (see comments in the "Microsoft Schedule+" section later). The variable should be initialized using the **CreateObject** or **GetObject** commands. A type library, object library, or dynamic-link library is required to declare a variable as an application-defined object type. This library must be checked in the controller application's References dialog box. The OLE Messaging, Schedule+, and Microsoft Graph OLE Automation servers do not support early binding at present.

```
Dim objPie As New Mom.ApplePie 'Invalid use of the New Keyword common
```

Or:

```
Dim objPie As Mom.ApplePie
Set objPie = CreateObject("Mom.ApplePie")
```

OLE Control Binding

OLE control binding uses an OLE control to contain the OLE Automation server in a window belonging to the controller. This control is usually used for linking and embedding, but may

also be used for OLE Automation *if* the OLE product supports it. In the following code sample, the **ole1** variable is a MSOLE2 control:

```
ole1.Class = "Mom.ApplePie"
ole1.Action = VB.OLEContainerConstants.vbOLEEmbedded
ole1.Action = VB.OLEContainerConstants.vbOLEActivateAuto
ole1.Object.CrustThickness = 3
ole1.Object.Apples = "Granny Smith"
ole1.Object.Slices = 8
```

OLE control binding does late binding to the server.

Binding Factors

Some OLE Automation controller products have three options available for binding—which do you use? Early binding is preferred for several important reasons:

- Early binding checks all of the code against the syntax stored in the type library when you compile the executable or compile Visual Basic for Applications code. Syntax errors are detected during the compile. This is not true with late binding or OLE control binding.
- Early binding is insensitive to the localized version of the products you are using. Late binding and control binding are sensitive to the localized version. If the user is running the French version of Microsoft Word and you did *not* use a type library, the commands must be the French version's commands.
- Early binding performance is significantly faster than late or control binding.

I did some timings of early binding to in-process servers using Visual Basic 4.0. I found that early binding sometimes performed better than having the class instance within the executable.

Given these factors, I will assume in the remainder of this article that you are always using a library—a type library, an object library, or a DLL—and doing early binding (if it is available).

Performance Factors

OLE Automation servers contained in executables are called *out-of-process servers*. Servers contained in dynamic-link libraries (DLLs) are called *in-process servers*. If the controller is 16-bit, the in-process server must be a 16-bit DLL, and if the controller is 32-bit, the in-process server must be a 32-bit DLL. This requirement for in-process servers—that the controller and library both have the same "bitness"—results in data going directly between the DLL and the application with very, very little overhead—literally nothing!

Out-of-process servers move data indirectly between the controller executable and the server executable. This can result in many milliseconds to pass an integer, but 32-bit executables and 16-bit executables can talk to each other without thunking. This is an important issue if you are *writing* the server, but inasmuch as we are only talking about *using* servers, it is a moot issue.

If performance is critical to the controller, I do the following steps:

1. Count the number of communications between OLE Automation server and OLE Automation controller and try to reduce the steps by changing the commands or the algorithm.
2. Check to see if the server supports macros or procedures. If so, create the code as a macro or procedure in one or two communications and then call the macro or procedure.

These suggestions assume that I have already coded the commands as recommended in the next section. So let us look at coding commands in the controller application.

Writing Commands

There are several ways of writing commands in the controller code. The ways available in newer products such as Visual Basic 4.0 and Microsoft Access 7.0 are better than the ways required in the older products. Better? Yes, the code is cleaner, easier to understand, and runs faster. My basic rules for writing code in the newer products are:

- Use named arguments.
- Fully qualify objects and methods.
- Eliminate requalifying.

Use Named Arguments

Most developers have had the frustrating experience of misplacing an argument in a function call. With earlier versions of many products, you were required to give all arguments in the correct argument order for the function to work. At times, this programming approach can get very old—like the instructions from my old social studies teacher, Miss Schooly, who required us to put our name, class, date, and home room on the top of each page (including the back side of pages!).

The newer version gives the options of named arguments and optional arguments. An illustration may clarify the change.

```
A$ = SillyWalks(3, ra, cs, rs, 0) 'Ordered Arguments

A$ = SillyWalks(Actor:=rs, _
                    CrazySteps:=cs, _
                    RouteArray:=ra, _
                    Speed:=3) 'Named Arguments
'Note that 0 is dropped because this is the default value.
```

The named argument syntax above is easier to read and more meaningful—especially to anyone who inherits this code! There is no speed lost in the Windows 95 controller products when you use named arguments.

Fully Qualify Objects and Methods

Most developers are familiar with data access objects (DAOs) and will refer to them simply as **DBEngine** or **Database** in their code. Well, that's not cool anymore. This way will still work in many products (for backward compatibility), but you should qualify all application-defined object types by their library name (in this case, **DAO**). The new way is to use **DAO.DBEngine** or **DAO.Database** instead. Every object is qualified for some very significant reasons.

First, this approach allows the developer to know where this object is coming from—this will become a greater and greater problem as more OLE Automation servers are added to the marketplace. If I find a line of code declaring a variable as a **DocumentProperty** or **Permission**, can you tell me where to look for documentation on this object? Variables defined as **MicrosoftOffice.DocumentProperty** or **SQLOLE.Permission** clarify this quickly. I usually go the extra mile and fully qualify constants by including their object. Figure 1 shows an example of a fully qualified constant with the names of the components.

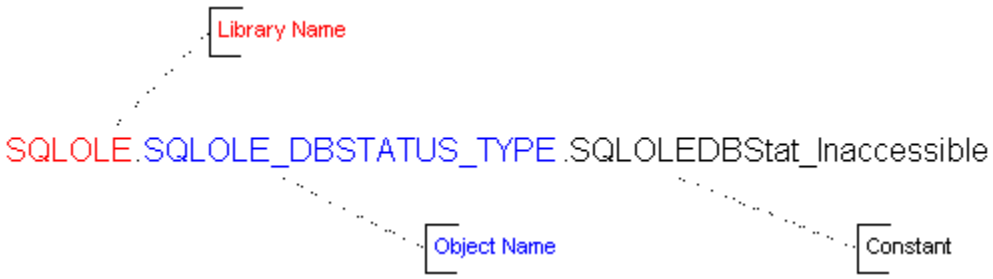

Figure 1. The structure of a full qualified constant

Second, this approach prevents ambiguous references. If two different OLE Automation servers have the same object name, you will be forced to qualify every instance later—it will happen. It *has* happened, for example with **SQLOLE.Databases** and **DAO.Databases** or with **SQLOLE.Properties**, **DAO.Properties**, and **Access.Properties**.

Third, if you do not qualify the object, the Visual Basic for Applications engine must search

through *all* of the references to find it. This slows down the application. In short, using qualified objects means faster execution!

Note Visual Basic for Applications reconciles an object first against its host and then against the type libraries in the order that they appear in the References dialog box.

Eliminate Requalifying

OLE Automation sends messages back and forth between controller and server. The older versions of controller products could not keep a connection open, so a new connection occurred each time a message needed to be sent, known as *requalifying* in the documentation. This connection can be kept open in the newer controller products if the **With** and **For ... With** commands are used. Some examples may clarify the difference in code.

```
'Reconnection occurs each time
MyObject.Height = 100
MyObject.Caption = "Hello World"
MyObject.Font.FontSize = 32
MyObject.Font.FontBold = True

'Only two connections
With MyObject
    .Height = 100
    .Caption = "Hello World"
    With .Font
        .FontSize = 32
        .FontBold = True
    End With
End With
```

I recommend always using the **With** command for early binding—the code is easier to understand and is faster. If you are using late binding, the **With** command will slow down execution and should be avoided. Table 2 shows the results on a sample automation I wrote to verify this information.

Table 2. Time to Execute an Automation Sequence with Different Options

Binding	Requalifying	Using "With"
Early	16.06 seconds	14.39 seconds
Late	25.78 seconds	28.06 seconds

Writing Code

"How do I write OLE Automation code? There is no documentation!" is a frequent complaint from developers new to OLE Automation. This is false—but people cannot see it because they expect a huge tome and get a few lines instead. Because the goal is to write code and not to read thousands of pages, I will start by looking at how to write code.

The three ways to write code are:

- Using a recorder
- Using the Object Browser
- Using the documentation

To be truthful, there is a fourth method: "Firing commands at random, blindfolded and drunk on fatigue." Since I've "been there, done that, ain't going back," I describe it simply as guessing object names, command names, and arguments until you get it right or give up in frustration. If you have been there and don't want to be there again, read the following carefully and get your wall clear for some big maps to these OLE Automation servers.

Using Recorders

My favorite trick is using a recorder to record a series of actions in the server application. I modify this recording to produce OLE Automation code for the controller. Recorders are not available in all products, but third-party developers will fill this void as the OLE Automation paradigm increases. The products with recorders built-in are shown in Table 3.

Table 3. Recorders Available in OLE Servers

Product	Version	Recorder Location	Comments
Microsoft Excel	4.0, 5.0, 7.0	Tools / Record Macro	Check that Options is set to Visual Basic. Code is stored in a module in the active workbook.
Microsoft Project	4.0, 7.0	Tools / Record Macro	Code is stored in module in the active project.
Microsoft Word	1.0, 2.0 ,6.0, 7.0	Tools / Macro / Record	Code is stored in a macro in the template files.

The Microsoft Excel recorder created the following code:

```
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "Name"
    Range("A1").Select
    With Selection.Font
        .Name = "Arial Black"
        .FontStyle = "Bold"
        .Size = 11
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlNone
        .ColorIndex = 3
    End With
```

You must make some changes for this code to work from a controller:

- Commands must be qualified by an instance.
- Commands (including constants!) must be prefixed with a dot (".").
- Arguments must be changed from "=" to ":=".

The resulting controller code becomes:

```
With ThisSpreadSheet
    .Range("A1").Select                '<== Dot added
    .ActiveCell.FormulaR1C1 = "Name"   '<== Dot added
    .Range("A1").Select                '<== Dot added
    With .Selection.Font               '<== Dot added
        .Name = "Arial Black"
        .FontStyle = "Bold"
        .Size = 11
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = Excel.xlNone     '<== Library name and dot added
        .ColorIndex = 3
    End With
End With
```

No major rocket science here—just monkey see, monkey do.

Using the Object Browser

The Object Browser is available in all of the Microsoft products that have Visual Basic for Applications built in. Visual Basic 3.0 (or earlier) and Microsoft Access 2.0 (or earlier) do not have Visual Basic for Applications built into them and do not have the Object Browser. (They also do not support named arguments.) When you are viewing a module or code window and you press F2, an Object Browser appears, similar to the one shown in Figure 2.

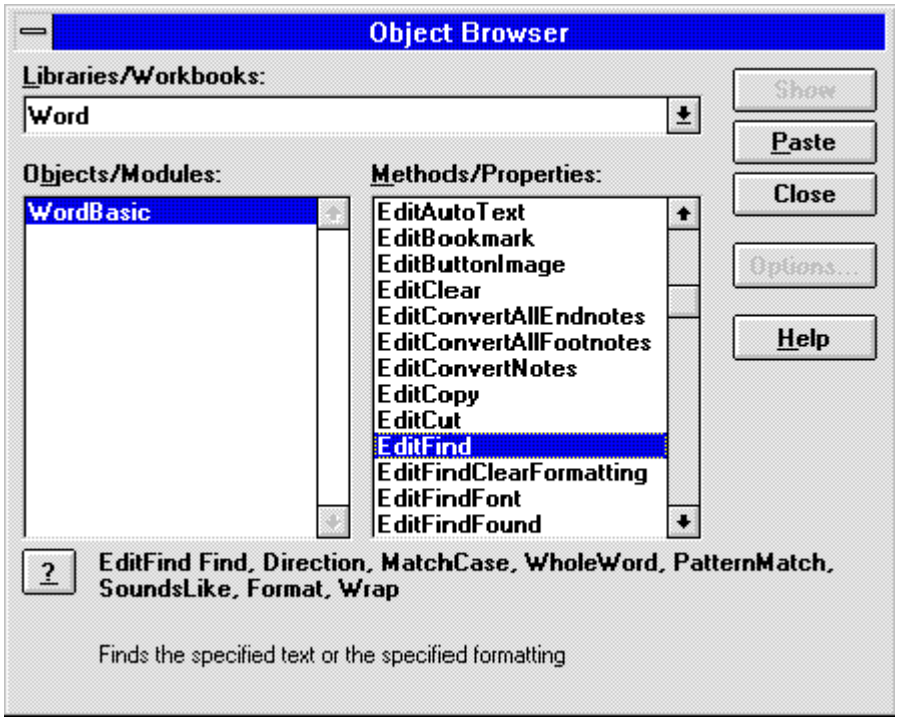

Figure 2. A sample Object Browser

The topmost drop-down list box shows all the available libraries from the References dialog box. These libraries allow you to "early bind" to OLE Automation servers. The left list box shows the objects in the library. The right list box shows the methods and properties (including constants). The bottom of the dialog box gives additional information about the item in the right list box, including a command button to jump to a Help topic (if available).

This dialog box makes life simpler. Object variables are declared by taking the first word in the Libraries/Workbooks drop-down list box, "Word," and then adding this word to the desired object in the left list box, "WordBasic," to get "Word.WordBasic." For example:

```
Dim MyWord as Word.WordBasic
```

To use a method or property, simply select the desired method or property, and click Paste to have a template pasted into your code. For example:

```
EditFind Find:=, Direction:=, MatchCase:=, WholeWord:=, PatternMatch:=,
SoundsLike:=, Format:=, Wrap:=
```

(In the above line of code, we added a hard return to make it visible on your screen. If you are pasting the code into your project, be sure to remove the hard return before "SoundsLike".)

You must then add the dot, qualify it by an instance, remove unneeded named arguments, and supply the appropriate argument values. For example:

```
With MyWord
    .EditFind Format:=1, Wrap:=1
    ' etc.
```

```
End With
```

A word of advice: Most OLE servers have their constant values defined in the library. ALWAYS (I do mean to shout!) use the built-in constants! The numeric values of some constants can change between product versions. Remember to qualify the constants with at least the library name. For example:

```
MyPreparedStatement.OpenResultset _
     (Type:=RDO.rdOpenForwardOnly, _
      LockType:=RDO.rdConcurRowver, _
      Options:=RDO.rdAsyncEnable)
```

If the type library has many objects that contain constants, include the object name as shown here:

```
MyPreparedStatement.OpenResultset( _
      Type:=RDO.ResultsetTypeConstants.rdOpenForwardOnly, _
      LockType:=RDO.LockTypeConstants.rdConcurRowver, _
      Options:=RDO.OptionConstants.rdAsyncEnable)
```

If the object browser does not list the OLE server you need, you must put a check mark by its library in the References dialog box, as shown in Figure 3. If the OLE Automation server does not appear in the list, you must add it by selecting the appropriate library file using the **Browse...** command button. Table 4 (below Figure 3) shows a list of these essential library files and their related products. Select the file listed under the heading Reference File. For more information, see "VBA Editing and Debugging Tools" in Chapter 2 of Eric Wells's *Developing Microsoft Excel 5 Solutions*.

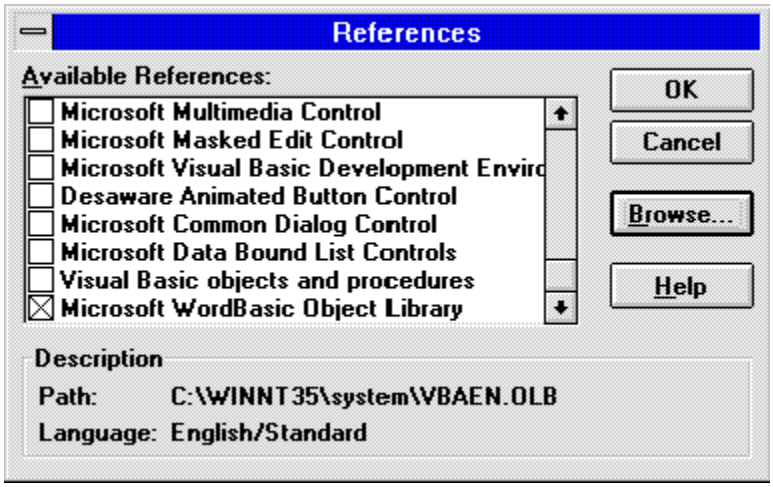

Figure 3. Selecting a reference

Table 4. Name of Product File Containing Object Browser Information

Product	Version	Reference File	Library Name	Reference Title
Microsoft Access	7.0	MSACCESS.TLB	Access	Microsoft Access for Windows 95
Binder	1	BINDER.TLB	OfficeBinder	Office Binder 1.0 Type Library
DAO	2.5/3.0	DAO2532.DLL	DAO	Microsoft DAO 2.5/3.0 Compatibility Library
DAO	2.5	DAO2516.DLL	DAO	Microsoft DAO 2.5 Object Library
DAO	3.0	DAO3032.DLL	DAO	Microsoft DAO 3.0 Object

				Library
Microsoft Excel	5.0	XL5EN32.OLB	Excel	Microsoft Excel 5.0 Object Library
MS Graph	5.0	GREN50.OLB.	Graph	Microsoft Graph 5.0 Object Library
MAPI	7.0	MDISP32.TLB MDISP.TLB	MAPI	OLE/Messaging 1.0 Object Library
Office	7.0	MSO50ENU.DLL	MicrosoftOffice	Microsoft Office 95 Object Library
PowerPoint	7.0	POWERPNT.TLB	PowerPoint	PowerPoint 7.0 Object Library
Microsoft Project	4.1	PJ4EN32.OLB	MSProject	Microsoft Project 4.1 Object Library
RDO	1.0	MSRDO32.DLL	RDO	Microsoft Remote Data Object 1.0
Schedule+	7.0	SP7EN32.OLB	SPL	Microsoft Schedule+ 7.0 Object Library
SQL-DMO	6.0	SQLOLE32.TLB	SQLOLE	Microsoft SQLOLE Object Library
Voice Command	1.0	VCAUTO.TLB	VCmdAuto	VoiceCommand 1.0 Type Library
Voice Text	1.0	VTXAUTO.TLB	VTxtAuto	VoiceText 1.0 Type Library
Word	6.0	WB60EN.TLB	Word	Microsoft WordBasic Object Library
Word	7.0	WB70EN32.TLB	Word	Microsoft WordBasic 95 Type Library
Word	7.0	WD95ACC.TLB	Word95ACC	Word 95 Objects for Access

The Confession

Some problems can frustrate your use of application-specific object types. These are not features, but bugs! They are fixed in the later version of Visual Basic for Applications shipped with Visual Basic 4.0 and Microsoft Access 7.0.

The Object Browser in Microsoft Excel 7.0 and Microsoft Project 4.1 will display hidden and disabled elements in some servers. Check the server's Help file or the extended map listed below to identify elements that you can use. A simple way to check that the Object Browser displays only enabled elements is to examine the "Microsoft DAO 2.5/3.0 Compatibility Library," found in Microsoft Access, Visual Basic, and some Microsoft Office products—the **IndexFields** object should *not* be visible in the Object Browser.

With the same products (Microsoft Excel 7.0 and Microsoft Project 4.1), you cannot successfully declare a variable as **Access.Application**. If you try, you will receive an "Object library feature not supported" message. Placing an underscore in front of the object and enclosing it in square brackets will generally resolve this problem. The variable should be declared as **Access.[_Application]**.

Further, in Microsoft Excel 7.0 and Microsoft Project 4.1, you cannot use the **New** keyword when you dimension the application-specific object types. One last confession—some of the

type libraries still have bugs in some of their commands; occasionally you will be forced to use late binding.

Using the Documentation

A properly constructed server comes with a type library that points to topics in a Help file. Dream on. The reality of the industry is that many type libraries and Help files often ship as afterthoughts because there was not enough time to complete them, test them, and still meet the product shipment date.

Microsoft Word is a good example. In Word version 2.0, the programming reference Help file did not ship with the product; it was a separate fulfillment product. In Word version 6.0 and Word 95, the Word type libraries did not ship with the product; they were distributed later for free. A few products succeed in delivering the dream—DAO, SQL Distributed Management Objects (SQL-DMO), and RDO shipped with polished type libraries and Help files.

You can use a programming reference Help file to write OLE Automation code. The Help file does describe the topics from the perspective of the OLE Automation application, so remember the following:

- Qualify every command with the appropriate object. Use the library name if you cannot identify an object.
- Always use named arguments. The arguments may not be shown in the correct ordered argument sequence.

A command from Microsoft Word will illustrate this. The following line from the WordBasic Reference gives the arguments for the **GetAddress$** command in Word 95 (a hard return was added before [SelectDialog] to make the line visible on your screen):

```
GetAddress$([Name$], [AddressProperties$], [UseAutoText], [DisplaySelectDialog],
[SelectDialog], [CheckNamesDialog], [MRUChoice], [UpdateMRU])
```

To use this command in Visual Basic, I qualify it with a Word object. For example:

```
Dim MyWord as Word.WordBasic
....
With MyWord
   letterAddress$ = .GetAddress$( _
      Name$ :="Ken Lassesen", _
      AddressProperties :="" _
      )
   .StartOfDocument
   .Insert letterAddress$
End With
```

Now that I have explained the general methods of creating OLE Automation code, it is time to look at each Microsoft OLE Automation server.

OLE Automation Servers

OLE Automation servers are like a family—each child inherits a mixture of characteristics that makes all the children look similar in some aspects, but each child has his or her own special characteristics. Different servers may contain very similar objects, commands, and behavior. Servers can also be very dissimilar—so much so, in fact, that a developer can become frustrated when one server does not behave like another server. Each server has its own personality.

In this section I will not contrast the many servers covered—that would be at least 240 sub-sections. I will, however, give:

- A thumbnail description of the server and some of its uses.
- A short tabular summary of necessary information about the server.
- A programmatic identifier (ProgID) and class identifier (CLSID) table for the server.

- An example of using the server with a very brief discussion of any important points.

An example of a tabular summary of information for a type library is shown in Table 5. The type library is used to do API calls. This API type library is not covered in this article, but it is shipped with the Microsoft Press® book *Hard Core Visual Basic* by Bruce McKinney.

Table 5. Quick Summary Example

Property	**Notes**
Reference File	WINAPI32.TLB, WINAPI16.TLB
Reference Title	Windows API Functions
Object Browser Library Name	Win
Object Browser Title	Windows API Functions
Programming Help File	Use the MSDN Library CD for the SDK documentation and Bruce McKinney's *Hard Core Visual Basic* book from Microsoft Press.
Extended Map	Not available at present
Redistribution Rights	See the McKinney book, or contact Microsoft Press for current rights.
Source Information	Available in *Hard Core Visual Basic* by Bruce McKinney, Microsoft Press
Externally Creatable (**New**)	False
Server Command	Multiple-system DLLs
CreateObject	Not used.
GetObject	Not used.
Terminate Object	Terminate with controller.

The programmatic identifier and class identifier table for Microsoft Project is shown in Table 6.

Table 6. Sample Identifier Table

ProgID	**CLSID**
MSProject.Application	{00020AFE-0000-0000-C000-000000000046}
MSProject.DocFile	{00020A00-0000-0000-C000-000000000046}
MSProject.Project	{00020A00-0000-0000-C000-000000000046}

Let us examine what these tables contain.

Reference File

This is the name of the file that must be registered in the References dialog box to use the Object Browser or early binding. If the server does not appear in the References dialog box, you must select the reference file name using the Browse command button. Use the Find command in Windows 95 to search your computer for this file.

The English version is listed first, before any foreign language versions. For example, the Visual Basic for Applications reference file may be one of the following:

VBAEN32.OLB
VBABRZ32.OLB

VBAFR32.OLB
VBADE32.OLB
VBAIBP32.OLB

Future versions of all products may have only an English reference file.

Reference Title

This is the name of the server as it appears in the References dialog box.

Object Browser Library Name

This is the name that qualifies all of the server's objects, methods, and constants.

Object Browser Title

This is the name of the server as it appears in the Object Browser dialog box. This name does not appear in the Microsoft Excel 7.0 or Microsoft Project 4.1 Object Browser. (See the earlier section "The Confession.") Although the Object Browser Title and the Reference Title *should* be the same, some older servers give them different names.

Programming Help File

This is the name of the file that contains detailed information about programming the server. I hate to admit it, but the shipped product may not include this essential documentation. If you must use the product before obtaining this documentation, use the Object Browser. It provides enough essential information to program the server.

Extended Map

An "extended map" is an exhaustive chart showing all of the objects, methods, properties, and lists in an object. It was an idea that I tossed out, and my fab boss, Nigel Thompson, said "Make it so!" A map is a good learning aid and a quick reference when developing an application. After producing a map, I found that it was easy to work with the object: I just post the appropriate map on my wall for quick reference, which is a lot faster than clicking objects in a Help file one by one to discover their properties and methods. Figure 4 shows an example of a simple extended map.

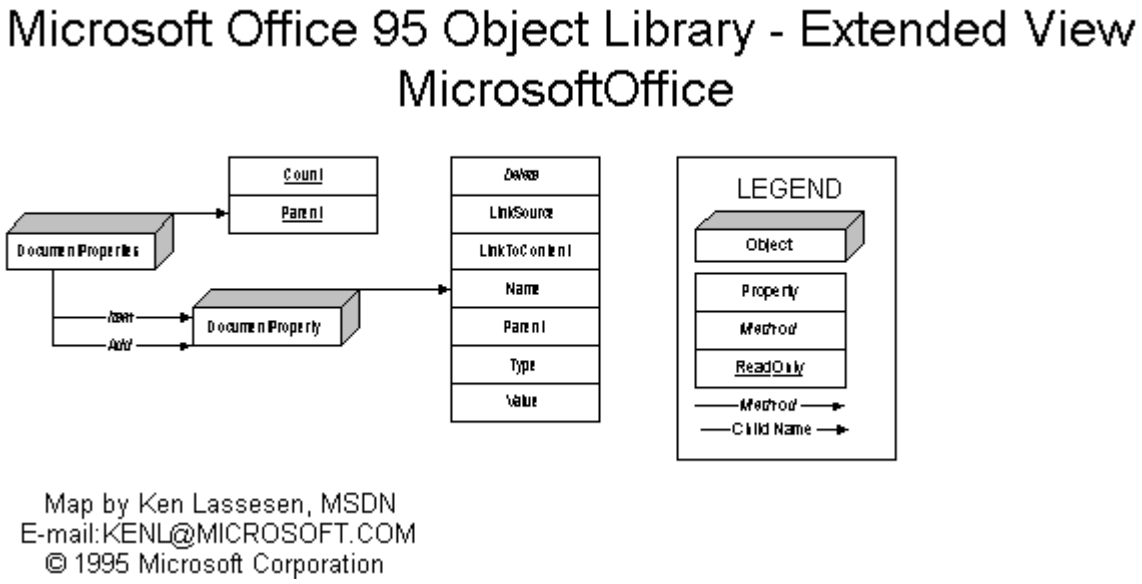

Figure 4. Extended map of the Microsoft Office 95 Object Library

Redistribution Rights

The Redistribution Rights entry gives you a short description of *some* restrictions that may apply. This summary is not a legal statement of the rights, nor is it complete. You'll find the legal statement regarding redistribution rights somewhere in the product's documentation or subsequent official notifications.

Source Information

When developers do not find the reference file on their computer, they start to search. They often spend a day reinstalling the product, calling friends, and checking computer bulletin board systems. Finally, they look through the file lists on all of the CD-ROM discs in the Development Platform (28 CD-ROMs for October 1995). Well, that gets old fast, so I include information on where to find the Reference File (or Programming Help File).

Externally Creatable

Can the topmost object be created using the **New** keyword? A server may contain both objects that *are* externally creatable and those that are *not* externally creatable. The name of the object according to the type library is shown. Some objects may be creatable only *in some circumstances*.

Server Command

This is the value of the **LocalServer** or **LocalServer32** property for the Class ID listed in the system registry (removing any path information). This is the file that supplies the server to the application. If the file is an executable (.EXE), the server is an out-of-process server. If the file is a dynamic-link library (.DLL), the server is an in-process server.

Some server executables require a command-line argument that includes the word *Automation*. These executables include MSGRAPH.EXE, EXCEL.EXE, WINWORD.EXE, SCHPLS32.EXE, MDISP32.EXE, and POWERPNT.EXE.

CreateObject

When the **CreateObject** function is called, one of three things could happen:

- A new instance is automatically created. The **Excel.Application** server behaves like this.
- A prior instance is used if available; otherwise, an instance is created. The **Word.WordBasic** server behaves like this.
- The call fails because the object is not creatable. The **MicrosoftOffice.Properties** server behaves like this.

The server's topmost object behavior is described *only* in this section. Each creatable object may display different behavior.

GetObject

When the **GetObject** function is called, the response could be one of the following:

- Succeed by creating a system instance.
- Succeed by seizing a prior system instance.
- Fail because there are no prior system instances loaded.
- Fail because it cannot be created.

The server's topmost object behavior is described here. Each creatable object may display different behavior.

TerminateObject

I would love to state that **Object.Quit** will terminate every server, but I can't. Different servers have many commands and conditions required before the server will terminate. The

requirements to terminate the object are listed here in this section.

ProgID

The programmatic identifier (ProgID) is the name of the object in the system registry. This string is used in **CreateObject** or **GetObject** to obtain an instance. In some cases, there may be a disagreement between the ProgID and the application-specific object type name shown in the library. The Office Binder shown below illustrates this.

```
Dim ThisBinder as OfficeBinder.Binder
Set ThisBinder = CreateObject("Office.Binder")
```

I can modify the registry to agree with the type library by creating a .REG file to modify the system registry. The lines below allow me to use **OfficeBinder.Binder** *in addition to* **Office.Binder**. Yes, I can use either, but I must remember to redistribute this file when I install an application that uses it.

```
REGEDIT
HKEY_CLASSES_ROOT\OfficeBinder.Binder = Microsoft Office Binder
HKEY_CLASSES_ROOT\OfficeBinder.Binder\CLSID =
  {59850400-6664-101B-B21C-00AA004BA90B}
HKEY_CLASSES_ROOT\OfficeBinder.Binder\CurVer = Office.Binder.95
```

For further information about this, see "Identifying and Registering an Object Class" in *The Component Object Model Specification* (MSDN Library, Specifications). If you distribute a type library with your application, you should include a .REG file to register it on other PCs.

CLSID

The class identifier (CLSID) is a unique identifier for the server. If several program IDs have the same class identifier, the programmatic identifiers are synonyms. In our sample above, MSProject.DocFile and MSProject.Project are the same.

A Word of Caution

The bad news is that the Quick Summary can be complex because *each* object in a server may have its own behavior. My advice is simple: Test each object and keep notes.

The Servers

This article covers more than OLE Automation servers—it covers the libraries available in the Object Browser. These libraries are described in three sections:

- Out-of-process servers
- Library encapsulations
- In-process servers

Out-of-Process Servers

Out-of-process servers are executables that allow controllers, whether 16-bit or 32-bit, to pass data between the controller and the server. The cost of this flexibility is slow data transfer. All the Microsoft Office products are out-of-process servers.

In general, these servers create a new instance for each **CreateObject** call or **New** declaration. The new instances may be system instances, as shown in this example:

```
Dim obj(0 To 5) As Object
For i = 0 To 5
    Set obj(i) = CreateObject("Excel.Application")
    obj(i).Visible = True
Next I
```

The code above will create six system instances of Microsoft Excel that will continue to exist after the controller closes. The new instances may be documents in a system instance instead.

For example.

```
Dim obj(0 To 5) As Object
For i = 0 To 5
    Set obj(i) = CreateObject("Excel.Sheet")
    obj(i).Visible = True
Next i
obj(0).Parent.Parent.Visible = True
```

This code will create six Microsoft Excel sheets in a single Microsoft Excel instance. The sheets will close as soon as their instance is set to **Nothing**. The Microsoft Excel application will continue to exist if it was visible when the sheets were set to **Nothing**.

Out-of-process servers will *usually* terminate when *all* instances are set to **Nothing**, the application is not visible, and the last instance is set to **Nothing**.

The out-of-process OLE Automation servers I will cover are:

- OLE Messaging
- Microsoft Schedule+
- Microsoft PowerPoint 7.0
- Microsoft Project 4.1
- Microsoft Excel 7.0
- Microsoft Binder 1.0
- Microsoft Graph 5.0
- Microsoft Access 7.0
- Microsoft Word 7.0
- Microsoft Voice Text 1.0
- Microsoft Voice Command 1.0
- Word 95 Objects for Access

OLE Messaging

The OLE Messaging server works using the Messaging API (MAPI). Table 7 shows a quick summary of the properties of OLE Messaging, which allows you to send, receive, and process electronic mail, including faxes. (Table 8 shows the MAPI ProgIDs.) You can write an obnoxious application to send e-mail to the world, asking everyone to send their VISA number, charge them $20 on their VISA card, automatically delete the flame mail, and process the donations. Of course, you can also do much more commendable work than this.

Table 7. Quick Summary for OLE Messaging

Property	Notes
Reference File	MDISP32.TLB, MDISP.TLB
Reference Title	OLE Messaging 1.0 Object Library
Object Browser Library Name	MAPI
Object Browser Title	OLE Messaging 1.0 Object Library
Programming Help File	OLEMSG.HLP
Extended Map	None
Redistribution Rights	Yes

Source Information	Backoffice SDK, Microsoft Solutions Development Kit 2.0, Win32 SDK
Externally Creatable (**New**)	False (Type library may not be used)
Server Command	MDISP32.EXE /Automation
CreateObject	Obtains running system instance, or starts a system instance if none exists.
GetObject	Obtains running system instance, or starts a system instance if none exists.
Terminate Object	Executes a **Mapi.Logoff**. Sets Objects to **Nothing.**

Table 8. Identifier Table for OLE Messaging

ProgID	CLSID
MAPI.Session	{3FA7DEB3-6438-101B-ACC1-00AA00423326}
MAPI.Message	{3FA7DEB4-6438-101B-ACC1-00AA00423326)
MAPI.Folder	{3FA7DEB5-6438-101B-ACC1-00AA00423326}

The following code sends an e-mail message to me, so I know that someone has executed the sample. Because I hope to get my boss to give me a $1 bonus for every e-mail I receive, give it a try!

```
Option Explicit
Sub Create_Message()
    Dim objSession As Object 'MAPI.Session
    Dim objMessage As Object 'MAPI.Message
    Dim  objRecip As Object 'MAPI.Recipient
    Set objSession = CreateObject("MAPI.SESSION")
    objSession.Logon
    Set objMessage = objSession.Outbox.Messages.Add
    objMessage.Subject = "Thank you for your article."
    Set objRecip = objMessage.Recipients.Add
    objRecip.Name = "Kenl@Microsoft.Com"
    objRecip.Type = 1     ' 1 is the value of the "mapiTo" constant.
    objMessage.Update
    objMessage.Send showDialog:=True
    objSession.Logoff
End Sub
```

At the time of writing, the server is still in its beta version and does not support early binding, so you must use late binding.

MAPI can become challenging because you may find two mail servers available on your PC: the Mapi.Session server described above and the MSMAPI.MAPISession that is part of MSMAPI.OCX. You can call **CreateObject** successfully with MSMAPI.MAPISession, but this instance is *very* different from a OLE Messaging instance.

Microsoft Schedule+

Schedule+ allows events to be scheduled on the user's calendar. Customer representatives can schedule call-backs or enter tasks from Microsoft Project or Microsoft Excel in their calendars, to cite just a few of the new possibilities with this server. It is assumed that the version of Schedule+ in the final version of the Microsoft Exchange Server SDK is installed.

Table 9. Quick Summary for Schedule+

Property	Notes
Reference File	Sp7en32.olb
Reference Title	Microsoft Schedule+ 7.0 Object Library
Object Browser Library Name	SPL
Object Browser Title	Microsoft Schedule+ 7.0 Object Library
Programming Help File	Microsoft Exchange Server SDK:*Microsoft Schedule+ Programmer's Guide*
Redistribution Rights	None
Source Information	The reference file may be obtained from the Microsoft Exchange Server SDK.
Externally Creatable (**New**)	Yes
Server Command	schdpl32.exe -Automation
CreateObject	Obtains running system instance or starts a system instance if none exists.
GetObject	Obtains running system instance or starts a system instance if none exists.
Terminate Object	Sets **Visible** to **False**. Set all instances to **Nothing**.

Table 10. Identifier Table for Microsoft Schedule+

ProgID	CLSID
SchedulePlus.Application	{0482E074-C5B7-101A-82E0-08002B36A333}
Schedule+.Application	{0482E074-C5B7-101A-82E0-08002B36A333}

The following code reminds you to be Santa Claus for your kids:

```
Dim appSchPlus As SPL.Application
Set appSchPlus = CreateObject("Schedule+.Application")
Visible = True
With appSchPlus
    .Logon
    .ScheduleSelected.Activate 'Makes visible
    Set MyAppt = .ScheduleSelected.singleappointments.New
        With MyAppt
            .SetProperties Text:="Santa Claus", _
                Start:=CVDate("12/24/95 23:30"), _
                End:=("12/25/95 00:30")
        End With
End With
```

This OLE Automation server has many unique characteristics; for a detailed introduction, see my article "An Extended Introduction to Schedule+ OLE Automation Programming."

Microsoft PowerPoint

The PowerPoint server allows you to automatically create or update PowerPoint presentations. For example, you could publish yesterday's sales figures each morning as part of an Executive Information System.

Table 11. Quick Summary for Microsoft PowerPoint

Property	Notes
Reference File	POWERPNT.TLB
Reference Title	PowerPoint 7.0 Object Library
Object Browser Library Name	PowerPoint
Object Browser Title	PowerPoint 7.0 Object Library
Programming Help File	VBA_PP.HLP
Extended Map	None
Redistribution Rights	None
Source Information	Part of PowerPoint
Externally Creatable (**New**)	False
Server Command	powerpnt.exe /AUTOMATION
CreateObject	Obtains running system instance, or starts a system instance if none exists.
GetObject	Obtains running system instance, or starts a system instance if none exists.
Terminate Object	Calls **Powerpoint.Quit.**

Table 12. Identifier Table for Microsoft PowerPoint

ProgID	CLSID
PowerPoint.Application	{81C3B541-2E17-101B-AF3C-00AA0038A98A}
PowerPoint.Show	{EA7BAE70-FB3B-11CD-A903-00AA00510EA3}
PowerPoint.Slide	{EA7BAE71-FB3B-11CD-A903-00AA00510EA3}
PowerPoint.Template	{EA7BAE71-FB3B-11CD-A903-00AA00510EA3}

The following code creates a simple slide containing a graphic:

```
Dim ThisPowerPnt As PowerPoint.Application
Dim ThisPresentation As PowerPoint.Presentation
Dim CurrentSlide As PowerPoint.Slide
 'PowerPoint 7 is the first version as an OLE Server.
Set ThisPowerPnt = CreateObject("PowerPoint.Application")
ThisPowerPnt.AppWindow.Visible = True

Set ThisPresentation = ThisPowerPnt.Presentations.Add( _
    WithWindow:=True _
    )
Set CurrentSlide = ThisPresentation.Slides.Add( _
    Index:=1, _
    Layout:=PowerPoint.SlideLayout.ppLayoutText _
    )
    With CurrentSlide
        'Note: Objects are "SlideObects"
        .Objects(1).Text = "MSDN PowerPoint Programmability"
        .Objects(2).Text = "Sixteen Point Star"
        .Objects.AddShape _
            type:=PowerPoint.ShapeType.ppShapeSixteenPointStar, _
            Left:=4800, _
            Top:=4300, _
```

```
            Width:=5000, _
            Height:=5000
        'Units of measurement are in TWIPS (like Visual Basic).
        .Objects(3).GraphicFormat.Fill.PresetTextured _
            PowerPoint.PresetTexture.ppPresetTextureWovenMat
    End With
```

The sample application changes the graphic's texture every tenth of a second as a special effect. Do not use numbers for constants or unqualified constants such as **ppShapeSixteenPointStar**; instead, use a fully qualified constant such as **PowerPoint.ShapeType.ppShapeSixteenPointStar**.

Microsoft Project

The Microsoft Project server allows the information on a project to be retrieved, added to, and updated. Using the MAPI server, reminder notices regarding tasks, deadlines, and milestones can be automatically mailed to members of a team. Reports can be generated for senior management. Meetings can be entered automatically into team members' Schedule+ calendars. A custom front end could be produced to simplify use of Microsoft Project.

Table 13. Quick Summary for Microsoft Project

Property	Notes
Reference File	PJ4EN32.OLB, PJ4ES32.OLB, PJ4DE32.OLB, PJ4FR32.OLB, PJ4SV32.OLB, PJ4IT32.OLB
Reference Title	Microsoft Project 4.1 Object Library
Object Browser Library Name	MSProject
Object Browser Title	Microsoft Project 4.1 Object Library
Programming Help File	VBA_PJ.HLP
Extended Map	None.
Redistribution Rights	None.
Source Information	Part of Microsoft Project 95
Externally Creatable (**New**)	False
Server Command	winproj.exe
CreateObject	Obtains running system instance, or starts a system instance if none exists.
GetObject	Obtains running system instance, or starts a system instance if none exists.
Terminate Object	Calls **MSProject.Quit**

Table 14. Identifier Table for Microsoft Project

ProgID	CLSID
MSProject.Application	{00020AFE-0000-0000-C000-000000000046}
MSProject.DocFile	{00020A00-0000-0000-C000-000000000046}
MSProject.Project	{00020A00-0000-0000-C000-000000000046}

The following example code adds a series of tasks for a new project:

```
Dim oProjApp As MSProject.Application
```

```
Dim oProjDoc As MSProject.Project
Dim  i As Integer
    Set oProjApp = CreateObject("MSProject.Application")
    oProjApp.Visible = True
    oProjApp.FileNew SummaryInfo:=False
    Set oProjDoc = oProjApp.ActiveProject
    For i = 1 To 10
        oProjDoc.tasks.Add Name:="Task" & i
    Next i
    oProjApp.fileSave
    oProjApp.Quit
```

Microsoft Excel

The Microsoft Excel server can be used to do complex mathematics, such as trend lines, can be used as a report engine, and can be used to manipulate data using pivot tables. Data may be passed to Microsoft Excel for statistical analysis.

Table 15. Quick Summary for Microsoft Excel

Property	**Notes**
Reference File	XL5EN32.OLB, XL5BRZ32.OLB, XL5FR32.OLB, XL5DE32.OLB, XL5IBP32.OLB
Reference Title	Microsoft Excel 5.0 Object Library
Object Browser Library Name	Excel
Object Browser Title	Microsoft Excel 5.0 Object Library
Programming Help File	VBA_XL.HLP
Extended Map	Not available at present
Redistribution Rights	None
Source Information	Part of Microsoft Excel 95 and Microsoft Excel 5
Externally Creatable (**New**)	False
Server Command	excel.exe /Automation
CreateObject	Creates a new system instance each time.
GetObject	Creates a new system instance, unless a system instance already exists.
Terminate Object	Calls **Excel.Quit**

Table 16. Identifier Table for Microsoft Excel

ProgID	**CLSID**
Excel.Addin	{00020810-0000-0000-C000-000000000046}
Excel.Application	{00020841-0000-0000-C000-000000000046}
Excel.Backup	{00020810-0000-0000-C000-000000000046}
Excel.Chart	{00020811-0000-0000-C000-000000000046}
Excel.CSV	{00020810-0000-0000-C000-000000000046}
Excel.Dialog	{00020810-0000-0000-C000-000000000046}

Excel.DIF	{00020810-0000-0000-C000-000000000046}
Excel.Macrosheet	{00020810-0000-0000-C000-000000000046}
Excel.Sheet	{00020810-0000-0000-C000-000000000046}
Excel.SLK	{00020810-0000-0000-C000-000000000046}
Excel.Template	{00020810-0000-0000-C000-000000000046}
Excel.VBAModule	{00020810-0000-0000-C000-000000000046}
Excel.Workspace	{00020810-0000-0000-C000-000000000046}
Excel.XLL	{00020810-0000-0000-C000-000000000046}

The example code below inserts an array of numbers into a worksheet and charts the information using the Chart object.

```
Dim ThisExcel As Excel.Application
Dim ThisChart As Excel.Chart
Dim TitleArray As Variant
Dim DataArray As Variant
TitleArray = Array("Dogs", "Cats", "Horses")
DataArray = Array(34, 53, 12)
Set ThisExcel = CreateObject("Excel.application")
With ThisExcel
    .Workbooks.Add
    .Range("A1:C1").Value = TitleArray
    .Range("A2:C2").Value = DataArray
    .Range("A1:C2").Select
    Set ThisChart = .Charts.Add()
    .Visible = True
End With
With ThisChart
    .Type = Excel.Constants.xl3DColumn
    .HasLegend = False
End With
```

The actual sample adds automatic turning of the three-dimensional (3-D) chart in Microsoft Excel to illustrate how a Microsoft Excel presentation can be enhanced. Microsoft Excel Chart is usually used instead of MSGraph because MSGraph lacks the ability to transfer data easily.

The reference file for Microsoft Excel 95 is the same as that for Microsoft Excel 5.0. (There is no XL7EN32.OLB.)

Microsoft Binder

The Microsoft Binder server allows some types of documents to be combined into a single document.

Table 17. Quick Summary for Microsoft Binder

Property	Notes
Reference File	BINDER.TLB
Reference Title	Office Binder 1.0 Type Library
Object Browser Library Name	OfficeBinder
Object Browser Title	Office Binder 1.0 Type Library
Programming Help File	VBA_BIN.HLP

Extended Map	Mapping the Office Binder: Binder 1.0
Redistribution Rights	None
Source Information	Part of Office 95
Externally Creatable (**New**)	False
Server Command	BINDER.EXE
CreateObject	Creates a new system instance each time.
GetObject	Creates a new system instance, unless specified file exists.
Terminate Object	Sets **Visible** to **False**; sets instance to **Nothing**.

Table 18. Identifier Table for Microsoft Binder

ProgID	CLSID
Office.Binder	{59850400-6664-101B-B21C-00AA004BA90B}

The following code creates a binder and adds two existing documents to it:

```
Sub Create_Binder()
    Dim objBinder As OfficeBinder.Binder, objWord As Object
    Set objBinder = CreateObject("Office.Binder")
    objBinder.Visible = True
    Set objWord = CreateObject("Word.Basic")
    With objWord
        .FileNewDefault
        .formatfont Points:=22, Bold:=True, Italic:=True
        .Insert "Russell says Binders are cool!"
    End With
    objWord.insertPara
    objWord.fileSaveAs "c:\Binder Summary.DOC"
    Set objWord = Nothing
    objBinder.Sections.Add filename:="c:\Binder Summary.doc"

    objBinder.Sections(1).Name = "Binder Summary"
    objBinder.SaveAs filename:="Mybinder.obd", saveOption:=3
    '3 is the value of the "bindDisplayDialog" constant
End Sub
```

Microsoft Graph

The MSGraph server allows data to be displayed on a graph. The functionality is similar to Charts in Microsoft Excel but requires fewer system resources to load.

Table 19. Quick Summary for Microsoft Graph

Property	Notes
Reference File	GREN50.OLB, GRBRZ32.OLB, GRFR32.OLB, GRDE32.OLB, GRIBP32.OLB
Reference Title	Microsoft Graph 5.0 Object Library
Object Browser Library Name	Graph
Object Browser Title	Microsoft Graph 5.0 Object Library
Programming Help File	VBA_GRP.HLP

Extended Map	Mapping Microsoft Graph 5.0: MSGraph, MSDN Library, October 1995
Redistribution Rights	None.
Source Information	Installed with several Microsoft Office applications.
Externally Creatable (**New**)	False
Server Command	Graph5.exe /Automation
CreateObject	Creates a new system instance each time.
GetObject	Creates a new system instance each time.
Terminate Object	Sets **Visible** to **False**; sets instances to **Nothing**.

Table 20. Identifier Table for Microsoft Graph

ProgID	CLSID
MSGraph.Application	{000208EC-0000-0000-C000-000000000046}
MSGraph.Chart	{00020801-0000-0000-C000-000000000046}

The following example code graphs some silly data using the MSGraph server (note the extra hard return before "vbTab"):

```
Dim ThisGraph As Object 'Graph.Application
MyData = "Cats" & vbTab & 20 & vbCr & "Dogs" & vbTab & "13" & vbCr & "Horses" &
vbTab & "4"
Clipboard.SetText MyData
Set ThisGraph = CreateObject("MSGraph.Application")
ThisGraph.Visible = True
SendKeys "%VD", True
SendKeys "%EP", True
ThisGraph.Chart.HasTitle = True
ThisGraph.Chart.ChartTitle.Caption = "Favorite Pets"
ThisGraph.Chart.AutoFormat Gallery:=xlColumn, Format:=2
```

The MSGraph server does not have a simple mechanism for inserting data series. To speed the entry of data, the data was copied to the Clipboard and then inserted into the data sheet using the **SendKeys** command. Using the Clipboard can greatly speed data transfers with out-of-process servers.

Microsoft Access Version 7.0

The Microsoft Access server makes Microsoft Access reports and forms available to the controller. Although DAO and Microsoft Access share some of the same functionality, the Microsoft Access server exposes the report and form objects that may be used an alternative to Crystal Reports and other products.

Table 21. Quick Summary for Microsoft Access

Property	Notes
Reference File	MSACCESS.TLB
Reference Title	Microsoft Access for Windows 95
Object Browser Library Name	Access
Object Browser Title	Microsoft Access for Windows 95
Programming Help File	VBA_ACC.HLP, VBAACCSP.HLP

Redistribution Rights	None
Source Information	Installs with Microsoft Access 7.0
Externally Creatable (**New**)	True (Access.Application)
Server Command	MSACCESS.EXE
CreateObject	Creates a new system instance each time.
GetObject	Creates a new system instance each time.
Terminate Object	Calls **Access.Quit**

Table 22. Identifier Table for Microsoft Access

ProgID	CLSID
Microsoft Access.Application	{B54DCF20-5F9C-101B-AF4E-00AA003F0F07}

The following code previews a report and opens a form using Microsoft Access:

```
Dim ThisAccess As New Access.Application
With ThisAccess
        .OpenCurrentDatabase "E:\Office95\Access\samples\NorthWind.mdb"
   With .DoCmd
       .OpenForm FormName:="Orders", _
         View:=Access.acNormal, _
         DataMode:=Access.acEdit, _
         WindowMode:=Access.acNormal
         'WARNING: "WindowMode:=Access.acDialog" will not return until
         ' user closes form!
       .OpenReport ReportName:="Invoice", _
        View:=Access.acPreview
   End With
End With
```

Although the Microsoft Access server exposes a DAO, it should not be used because Microsoft Access is an out-of-process server; instead, the in-process DAO server should be used. A controller should *never* invoke a method that does not return immediately; for example, opening a form with the **WindowMode** being **acDialog**. If the server does not return control to the controller in a reasonable time, an OLE time-out may occur, and the controller will appear to "hang" until the server returns.

Microsoft Word

The Microsoft Word server can be used as a form-letter generator, label generator, and report generator. It can also be used as a device for entering documents into databases.

Table 23. Quick Summary for Microsoft Word

Property	Notes
Reference File	WB70EN32.TLB
Reference Title	Microsoft WordBasic 95 Type Library
Object Browser Library Name	Word
Object Browser Title	Microsoft WordBasic 95 Type Library
Programming Help File	WRDBASIC.HLP
Extended Map	None

Redistribution Rights	Freely redistributable
Source Information	CompuServe®, WWW.Microsoft.Com
Externally Creatable (**New**)	False
Server Command	WINWORD.EXE /Automation [
CreateObject	Returns a system instance if available, otherwise creates a system instance.
GetObject	Returns a system instance if available, otherwise creates a system instance.
Terminate Object	Sets instance to **Nothing**. (Word may be visible.)

Table 24. Identifier Table for Microsoft Word

ProgID	CLSID
Word.Backup	{00020900-0000-0000-C000-000000000046}
Word.Bakup	{00020900-0000-0000-C000-000000000046}
Word.Basic	{000209FE-0000-0000-C000-000000000046}
Word.Document	{00020900-0000-0000-C000-000000000046}
Word.Picture	{00020901-0000-0000-C000-000000000046}
Word.RTF	{00020900-0000-0000-C000-000000000046}
Word.Template	{00020900-0000-0000-C000-000000000046}
Word.Wizard	{00020900-0000-0000-C000-000000000046}

The following code creates a document and records its creation date:

```
Dim objWord As Word.WordBasic
Set ThisWord = CreateObject("Word.Basic")
With ThisWord
    .AppMaximize
    .FileNewDefault
    .FormatFont _
        Points:=22, _
        Bold:=True, _
        Italic:=True
    .Insert "Welcome to Word OLE  Automation"
    .InsertPara
    .FormatFont _
        Points:=10, _
        Bold:=False, _
        Italic:=False
    .Insert "Report Created:"
    .InsertDateTime _
        DateTimePic:="YYYY MM DD HH:MM:SS", _
        InsertAsField:=False
    .InsertPara
End With
```

Microsoft Word is an arcane OLE server. It was the first OLE server and is inconsistent with later OLE servers. All commands are directed to the active document, which may be changed by the user or other applications. Always use named arguments with Microsoft Word; the ordered argument sequences are recorded incorrectly in some documentation, including the type library and what the recorder writes into macros.

Microsoft Voice Text

The Microsoft Voice Text Object (VTxtAuto) ships with the Microsoft Speech Software Development Kit (SDK). This object allows an application to speak any text aloud. The pronunciation engine is very advanced, and you can have it read a Microsoft Word document through a sound system, switching to Spanish or German when the text changes in the document. See the Speech API SDK documentation for further information (MSDN Library, Platform, SDK, and DDK Documentation), or search future issues of the MSDN Library.

Table 25. Quick Summary for Microsoft Voice Text

Property	**Notes**
Reference File	VTXTAUTO.TLB
Reference Title	VoiceText 1.0 Type Library
Object Browser Library Name	VTxtAuto
Object Browser Title	VoiceText 1.0 Type Library
Programming Help File	None at present
Redistribution Rights	See the Speech API SDK documentation.
Source Information	A component of the Microsoft Speech API SDK.
Behavior	Only a single instance may exist in the system.
Externally Creatable (**New**)	Yes (VTxtAuto.VTxtAuto)
Server Command	vcmd.exe
CreateObject	Returns a system instance if available; otherwise, creates a system instance.
GetObject	Returns a system instance if available; otherwise, creates a system instance.
Terminate Object	Sets instances to **Nothing**.

Table 26. Identifier Table for Microsoft Voice Text

ProgID	**CLSID**
Speech.VoiceText	{FF2C7A52-78F9-11ce-B762-00AA004CD65C}

The following sample code opens a Microsoft Word document and reads it aloud.

```
Dim Vtxt As New VTxtAuto.VTxtAuto
Dim Word As Word.WordBasic
Set Word = CreateObject("Word.Basic")
Vtxt.Register pszSite:="", pszApp:=App.Title 'Easy way in VB4
Word.FileOpen Name:= "My Sample.Doc"
While Not Word.AtEndOfDocument()
With Word
    .PageDown Count:=1, Select:=1 'Select next paragraph.
    .EditCopy  '.Selection is logical here BUT it gives errors on long text.
    A$ = Clipboard.GetText 'So we use the clipboard instead.
    If Len(A$) > 0 Then 'Make sure there is some text to read.
        Vtxt.Speak pszBuffer:=A$, dwFlags:=VTxtAuto.vtxtst_READING
    End If
    .CharRight Count:=1, Select:=0 'Move to start of next paragraph.
    While Vtxt.IsSpeaking() 'Wait until it is finished before going on.
        If fStop Then Vtxt.StopSpeaking: GoTo Stop_Now 'User terminate
```

```
        DoEvents 'Server runs independent of this application.
    Wend
End With
Wend
Stop_Now:
Word.FileClose
```

Although it is impolite to interrupt someone speaking, the module or global variable **fStop** allows the user to interrupt. The controller using the VtxtAuto server must register its name before the server can be used. The **Word.Selection** command often gives errors on long text selections, so I moved the text from Microsoft Word to the controller using the Clipboard instead. Having two sets of server commands mixed prevented me from using **With** to qualify both objects. I used **With** to qualify the most frequently used server, Microsoft Word, and requalify the less frequently used server, VTxtAuto.

Microsoft Voice Command

The Microsoft Voice Command object (**VCmdAuto**) ships with the Microsoft Speech API SDK. This object allows applications to understand spoken commands. See the Speech API SDK documentation for further information, or search future issues of the MSDN Library.

Table 27. Quick Summary for Microsoft Voice Command

Property	Notes
Reference File	VCAUTO.TLB
Reference Title	VoiceCommand 1.0 Type Library
Object Browser Library Name	VCmdAuto
Object Browser Title	VoiceCommand 1.0 Type Library
Programming Help File	None at present
Redistribution Rights	See the Speech API SDK documentation.
Source Information	A component of the Microsoft Speech API SDK.
Behavior	Only a single instance may exist in the system.
Externally Creatable (**New**)	Yes (VCmdAuto.VCmdAuto)
Server Command	vcmd.exe
CreateObject	Returns a system instance if available; otherwise, creates a system instance.
GetObject	Returns a system instance if available; otherwise, creates a system instance.
Terminate Object	Sets instances to **Nothing**.

Table 28. Identifier Table for Microsoft Voice Command

ProgID	CLSID
Speech.VoiceCommand	{A26D7620-6FA0-11ce-A166-00AA004CD65C}

The following Visual Basic code allows an application to respond to words listed in a list box. The words are in a list box called WordSpoken.

```
Dim VCmd As Object
Dim VMenu As Object
WordSpoken.AddItem "Stop"
```

```
WordSpoken.AddItem "Go"
WordSpoken.AddItem "Yield"
Set VCmd = CreateObject("Speech.VoiceCommand")
Call VCmd.Register("")
Set VMenu = VCmd.MenuCreate("VB Test Program", "TestMenu", 1033&, "", vcmdmc_CREAT
For i = 0 To WordSpoken.ListCount - 1
    Call VMenu.Add(i, WordSpoken.List(i))
Next i
VMenu.Active = True


Private Sub Timer1_Timer()
WordSpoken.ListIndex = VCmd.CommandSpoken
End Sub
```

The above sample code requires a timer to periodically check whether a command was spoken. Timers can be simulated with **DoEvents** loops in other products, as shown below.

```
Sub Checkforword(ByVal Seconds As Single)
endat = Seconds / (24# * 3600#) + Now
While Now < endat
    DoEvents
    WordSpoken.ListIndex = VCmd.CommandSpoken
Wend
End Sub
```

This is a fun OLE server to play with! If you use this OLE server with the Microsoft Voice Text server, you can spend hours talking to your computer.

"Word95 Objects for ACCESS"

"Word95 Objects for ACCESS" is a custom type library that exposes some parts of Microsoft Word that are useful from Microsoft Access. "Word95 Objects for ACCESS" enables the developer to manipulate Microsoft Word mail merge facilities. It is a subset of the Microsoft WordBasic 95 type library and contains only one object, despite the name claiming to have multiple objects.

Table 29. Quick Summary for "Word95 Objects for ACCESS"

Property	**Notes**
Reference File	WD95ACC.TLB
Reference Title	Word 95 Objects for Access
Object Browser Library Name	Word95ACC
Object Browser Title	Word 95 Objects for Access
Programming Help File	WRDBASIC.HLP
Redistribution Rights	None
Source Information	Is installed with Microsoft Access 95
Externally Creatable (**New**)	False
Externally Creatable (**New**)	False
Server Command	WINWORD.EXE /Automation
CreateObject	Returns a system instance if available; otherwise, creates a system instance.
GetObject	Returns a system instance if available; otherwise, creates a system instance.

Terminate Object	Set instance to **Nothing**. (May be visible.)

Table 30. Identifier Table for "Word 95 Objects for ACCESS"

ProgID	CLSID
Word.Basic	{000209FE-0000-0000-C000-000000000046}

The following code creates form letters from the first 10 customer records in the Northwind sample database shipped with Microsoft Access 95.

```
Dim ThisWord As wORD95aCC.Word95Access
Set ThisWord = CreateObject("Word.Basic")
ThisWord.FileOpen Name:=App.Path & "\MyMerge.Doc"
ThisWord.AppShow
With ThisWord
    .MailMergeOpenDataSource Name:="\Office95\ACCESS\Samples\Northwind.mdb", _
        LinkToSource:=1, _
        Connection:="TABLE Customers", _
        SQLStatement:="SELECT * FROM [Customers]"

    .MailMerge CheckErrors:=1, _
        Destination:=0, _
        MergeRecords:=1, _
        From:="1", _
        To:="10", _
        MailMerge:=1
End With
```

This type library is the same as the Microsoft Word type library except that only a few of the methods and properties are exposed in the library. When an instance of Word is created using the "Word95 Objects for ACCESS" library, this instance is created as not visible. (When the Word type library is used, it is created as visible.) The **.MailMerge** method is *not* exposed by the "Word95 Objects for ACCESS" library, but it is still available. This illustrates an important aspect of the libraries: They show what is recommended, not what is possible. This feature allows hidden methods and properties to be included for backward compatibility. Always write code according to the library model; it shows the recommended methods for the servers.

Library Encapsulations

Library encapsulations are libraries that are neither in-process nor out-of-process servers. They may use objects passed from other servers or be wrappers around API calls. They can simplify development for programmers by exposing unpublished API calls or encapsulating existing API calls in an object model.

There is only one type library server: the Microsoft Office server. The Windows API type library published by Bruce McKinney in his upcoming Microsoft Press book, *Hard Core Visual Basic*, is another important server, which I described in my article "Corporate Developer's Guide to Office 95 API Issues."

Microsoft Office 95 Object

The Microsoft Office 95 object is very badly misnamed; a more appropriate name would be the "Summary Information object." This object allows manipulation of the document properties of Microsoft Excel, Microsoft Project, and Binder documents. Document properties used to be called Summary Information. If you click the Summary tab in the Properties dialog box in Microsoft Excel (Figure 5), you will see what can be manipulated.

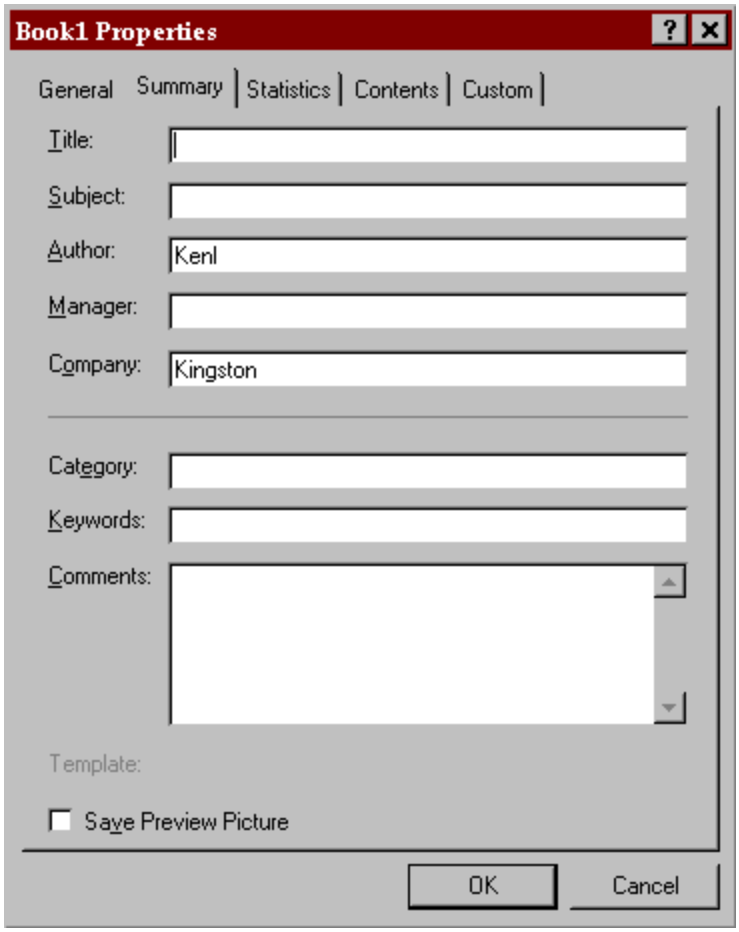

Figure 5. Properties dialog box showing the values that can be manipulated

Table 31. Quick Summary for Microsoft Office

Property	**Notes**
Reference File	Mso5enu.dll
Reference Title	Microsoft Office 95 Object Library
Object Browser Library Name	MicrosoftOffice
Object Browser Title	Microsoft Office 95 Object Library
Programming Help File	VBA_OFF.HLP
Redistribution Rights	None. User must own Microsoft Office.
Source Information	Microsoft Office 95
Externally Creatable (**New**)	False
Server Command	Mso5enu.dll
CreateObject	Invalid
GetObject	Invalid
Terminate Object	Invalid

Table 32. Identifier Table for Microsoft Office

ProgID	CLSID
There is no ProgID for MicrosoftOffice.	{2DF8D04C-5BFA-101B-BDE5-00AA0044DE52}

The following code sample receives an object, **ThisObject**, which has been set to an **Excel.Workbook**, **MSProject.Project**, or **Office.Binder**. The code displays all of the current values of the **BuiltinDocumentProperties** and **CustomDocumentProperties** in the document.

```
List1.Clear
List1.AddItem "Build in Properties"
On Error GoTo BadValue_err
For Each Prop In ThisObject.BuiltinDocumentProperties
    With Prop
        AName$ = .Name
        List1.AddItem .Name & ":= " & .Value
    End With
Next
List1.AddItem "Custom in Properties"
For Each Prop In ThisObject.CustomDocumentProperties
    With Prop
        AName$ = .Name
        List1.AddItem .Name & ":= " & .Value
    End With
Next
On Error GoTo 0
Exit Sub
BadValue_err:
List1.AddItem AName$ & ":= #NULL#"
Resume Next
```

The sample application demonstrates two important points:

- Some property values are not initialized, causing an error to occur.
- The built-in properties are not the same for the three different objects.

In-Process Servers

In-process servers require the controller to be the same "bitness" as the server—that is, if the controller is 16-bit, the server must also be 16-bit. This requirement allows data to be passed directly between the controller and the server, resulting in high-performance data transfer. In-process servers should always use early binding via a type library. All the database engines are in-process servers.

In general, these servers use a single instance for each application with implicit creation of the root instance. For example, if you open five databases and *then* create the application (or **DBEngine**) instance, you will find these five databases as children. If you create additional instances of the application (or **DBEngine**) instance, the first instance will always be returned.

In-process servers terminate when the calling application terminates because they are DLLs. Setting all of the instances to **Nothing** will not unload the DLL. The three Microsoft in-process OLE Automation servers are:

- Data Access Objects
- SQL Distributed Management Objects (SQL-DMO)
- Remote Data Objects

Data Access Objects

The Data Access Object server allows data to be accessed from Indexed Sequential Access Method (ISAM), Microsoft Access, and ODBC data sources. This server is the most used server of all the servers because data is the essence of business. It may access more different types

of databases than the other database servers in this article. This server supports remote OLE Automation—that is, the server may be running on a different computer.

Table 33. Quick Summary for Data Access Object

Property	**Notes**
Reference File	DAO2532.TLB, DAO2516.DLL, DAO3032.DLL
Reference Title	Microsoft DAO x.x Object Library
Object Browser Library Name	DAO
Object Browser Title	Microsoft DAO x.x Object Library
Programming Help File	DAO.HLP, DAOSDK.HLP
Redistribution Rights	See product documentation for Visual Basic or Microsoft Access.
Source Information	DAO.HLP ships with Visual Basic, Microsoft Access, and Microsoft Excel. DAOSDK.HLP ships with Visual C++.
Externally Creatable (**New**)	True (**DAO.DBEngine**). Not needed because **DBEngine** is created automatically when any component of DAO is referenced.
Server Command	DAO3032.DLL or DAO2516.DLL
CreateObject	Always returns same instance.
GetObject	Always returns same instance.
Terminate Object	Closes all DAO.Workspaces; sets DAO.DBEngine to **Nothing**.

Table 34. Identifier Table for Data Access Object

ProgID	**CLSID**
DAO.DBEngine	{00025e15-0000-0000-c000-000000000046}
DAO.Field	{00025e4c-0000-0000-c000-000000000046}
DAO.Group	{00025e5f-0000-0000-c000-000000000046}
DAO.Index	{00025e55-0000-0000-c000-000000000046}
DAO.PrivateDBEngine	{00025e19-0000-0000-c000-000000000046}
DAO.QueryDef	{00025e7a-0000-0000-c000-000000000046}
DAO.Relation	{00025e8b-0000-0000-c000-000000000046}
DAO.TableDef	{00025e43-0000-0000-c000-000000000046}
DAO.User	{00025e68-0000-0000-c000-000000000046}

The following code creates a database and then creates an ad-hoc table. The tables and their fields are then shown in a list box. (Note that you will have to remove the extra hard returns in the "Set MyDB" and "Query:" lines.)

```
Dim appDAO As New DAO.DBEngine
Dim MyDB As DAO.Database
Set MyDB = appDAO.CreateDatabase(Name:="MSDN1.MDB",
   Connect:=DAO.Constants.dbLangNorwDan, Option:=DAO.Constants.dbEncrypt)
With MyDB
    .Execute _
```

```
        Query:="Select 'Dr.Gui' As Name, 'MSDN@Microsoft' As Email, #1/1/52#
 as Birthdate into MyTables"
    For i% = 0 To .TableDefs.Count - 1
        List1.AddItem .TableDefs(i%).Name
        For j% = 0 To .TableDefs(i%).Fields.Count - 1
            List1.AddItem "  Field:" & .TableDefs(i%).Fields(j%).Name
        Next j%
    Next i%
End With
```

There are many other examples of using the DAO in samples shipped with various products, such as Visual Basic or Microsoft Access. Use **DAO.Database.Execute** to quickly create an ad-hoc table when the default data types assigned by the database engine are acceptable. This command can save many lines of code and can be easier to read. The Microsoft Access server is an out-of-process server; for this reason its DAO child object should be avoided by using the DAO server directly.

SQL Distributed Management Objects

The SQL Server object is a fair rose, officially named SQL Distributed Management Objects or SQL-DMO. It is called SQLOLE in the type library. (The poor developers could not change names to match the marketing manager's whims and still ship on time.) It exposes the complete Microsoft SQL Server as an object and allows the Microsoft SQL Server administration application to be easily managed. It should not be used for developing end-user applications.

Table 35. Quick Summary for SQL Distributed Management Objects

Property	**Notes**
Reference File	SQLOLE32.TLB
Reference Title	Microsoft SQLOLE Object Library
Object Browser Library Name	SQLOLE
Object Browser Title	Microsoft SQLOLE Object Library
Programming Help File	SQLBOOKS.MVB
Redistribution Rights	See documentation for Microsoft SQL Server.
Source Information	A component of SQL Server and SQL Server tools
Externally Creatable (**New**)	True (**SQLOLE.Application**). Not needed because the instance is created automatically when any component of SQLOLE is referenced.
Server Command	SQLOLE32.DLL
CreateObject	Always returns same instance.
GetObject	Always returns same instance.
Terminate Object	Closes application.

Table 36. Identifier Table for SQL Distributed Management Objects

ProgID	**CLSID**
SQLOLE.Alert	{00026bb0-0000-0000-C000-000000000046}
SQLOLE.Application	{00026ba0-0000-0000-C000-000000000046}
SQLOLE.Article	{00026ba0-0000-0000-C000-000000000046}

SQLOLE.Backup	{00026bb7-0000-0000-C000-000000000046}
SQLOLE.Check	{00026bbd-0000-0000-C000-000000000046}
SQLOLE.Column	{00026ba4-0000-0000-C000-000000000046}
SQLOLE.Database	{00026ba2-0000-0000-C000-000000000046}
SQLOLE.Default	{00026ba7-0000-0000-C000-000000000046}
SQLOLE.Device	{00026baf-0000-0000-C000-000000000046}
SQLOLE.Group	{00026baa-0000-0000-C000-000000000046}
SQLOLE.HistoryFilter	{00026bb8-0000-0000-C000-000000000046}
SQLOLE.Index	{00026bac-0000-0000-C000-000000000046}
SQLOLE.Key	{00026bad-0000-0000-C000-000000000046}
SQLOLE.Language	{00026bb2-0000-0000-C000-000000000046}
SQLOLE.Login	{00026bb1-0000-0000-C000-000000000046}
SQLOLE.Operator	{00026bb9-0000-0000-C000-000000000046}
SQLOLE.Publication	{00026bba-0000-0000-C000-000000000046}
SQLOLE.RemoteLogin	{00026bb4-0000-0000-C000-000000000046}
SQLOLE.RemoteServer	{00026bb3-0000-0000-C000-000000000046}
SQLOLE.Rule	{00026ba8-0000-0000-C000-000000000046}
SQLOLE.SQLServer	{00026ba1-0000-0000-C000-000000000046}
SQLOLE.StoredProcedure	{00026bab-0000-0000-C000-000000000046}
SQLOLE.Subscription	{00026bbc-0000-0000-C000-000000000046}
SQLOLE.Table	{00026ba3-0000-0000-C000-000000000046}
SQLOLE.Task	{00026bb5-0000-0000-C000-000000000046}
SQLOLE.Transfer	{00026bb6-0000-0000-C000-000000000046}
SQLOLE.Trigger	{00026bae-0000-0000-C000-000000000046}
SQLOLE.User	{00026ba9-0000-0000-C000-000000000046}
SQLOLE.UserDefinedDatatype	{00026ba6-0000-0000-C000-000000000046}
SQLOLE.View	{00026ba5-0000-0000-C000-000000000046}

The following code obtains information on the status of a series of databases on a SQL server.

```
Dim ThisSQLOLE As New SQLOLE.Application
Dim ThisSQLServer As New SQLOLE.SQLServer
Dim objSQLdb As SQLOLE.Database
ThisSQLServer.Connect ServerName:=ServerName$, Login:="sa", Password:=Null
List1.Clear
For Each objSQLdb In ThisSQLServer.Databases
    With objSQLdb
       If .Status <> SQLOLE.SQLOLE_DBSTATUS_TYPE.SQLOLEDBStat_Inaccessible Then
          List1.AddItem .Name & "[Size=" & Format$(.Size, "0.0") & " Mbytes, " _
          & " Used= " & Format$(.DataSpaceUsage, "0.0") _
          & ", Users=" & Format(.Users.Count, "0") & "]"
       End If
    End With
```

```
Next
```

The SQL-DMO server encapsulates the complete essence of SQL Server. The SQL Server Books Online gives many excellent examples of using this object. Kudos to my former boss, Casey Kiernan, for doing an excellent job, and to Ted Hart for creating one of the best examples of a type library!

Remote Data Object

The Remote Data Object (RDO) ships with Visual Basic 4.0 Enterprise Edition. It is a very thin layer over the ODBC API. The RDO does not use the Jet engine. It is fast and requires little memory to run.

Table 37. Quick Summary for Remote Data Object

Property	**Notes**
Reference File	MSRDO32.DLL
Reference Title	Microsoft Remote Data Object 1.0
Object Browser Library Name	RDO
Object Browser Title	Microsoft Remote Data Object 1.0
Programming Help File	ENTPRISE.HLP
Redistribution Rights	May be redistributed *only* with executables or DLLs generated by Visual Basic 4.0 Enterprise Edition. Not supported with other products.
Source Information	Visual Basic 4.0 Enterprise Edition
Externally Creatable (**New**)	True (**RDO.rdoEngine**). Not needed because the **rdoEngine** is created automatically when any component of RDO is referenced.
Server Command	MSRDO32.DLL
CreateObject	Always returns same instance.
GetObject	Always returns same instance.
Terminate Object	Closes application.

Table 38. Identifier Table for Remote Data Object

ProgID	**CLSID**
rdoEngine MicrosoftRDO.rdoEngine	{A93E470F-62D3-11CE-920A-08002B369A33}

The following sample code populates a list box with the results of an SQL query from an SQL server. It uses the sample "Pubs" database that is installed with Microsoft SQL Server.

```
Dim DB As rdo.rdoConnection
Dim RS As rdo.rdoResultset

Set DB = rdoEngine.rdoEnvironments(0).OpenConnection( _
    dsName:="Pubs", _
    Connect:="ODBC;Userid=sa;Password=;")
Set RS = DB.OpenResultset( _
    Name:="Select * from Authors;")
While Not RS.EOF
    List1.AddItem (RS(0))
```

```
    RS.MoveNext
Wend
RS.Close
```

This is similar to using the Data Access Object described above with the names changed to confuse the innocent.

Coding Tips

By this time, you should be pointed in the right direction. A few general words of advice may spare you some angst:

- Always fully qualify objects, properties, methods, and constants. Write your code for the person who is going to maintain it.
- Use the Clipboard to speed data transfer.
- Use macros or modules to speed execution.
- Use early binding whenever it is available.
- Use a splash screen when initializing an OLE Automation server and disable any forms until the OLE Automation server is loaded.

It is very reckless to assume anything about the window state of an OLE Automation server. It is important to remember that if you intend to make the OLE server visible to the user, you should include code to explicitly do the following:

- Make the server visible.
- Move the server on the screen.
- Restore the server from an icon and resize to an appropriate size.

Dream Sweet Dreams

This article covers a lot of territory in very few words. I smile when I look at these OLE Automation servers and controllers. This technology pushes the limit of practical corporate solutions to new levels. I am still trying to fully understand what it is possible to produce in one week of hard coding—it's awesome! As new servers are added, the potential grows and the labor hours decrease. Using OLE Automation servers and controllers represents a paradigm shift in how you code. Your learning curve will be steep for a while, but I hope this article flattens it for you. Try the examples, then dream, then code the dream.

Bibliography

Bienick, Paul. "Using OLE in Microsoft Visual FoxPro." (MSDN Library, Conference Papers)

Gilbert, Michael. "Building Custom Solutions with Schedule+." (MSDN Library Archive, Conference and Seminar Papers)

Hodges, Douglas. "Managing Object Lifetimes in OLE Automation." January 1995. (MSDN Library, Technical Articles)

Knowledge Base Q111311. "XL: CreateObject Function Starts Invisible Instance of Excel." (MSDN Library, Knowledge Base)

Knowledge Base Q112194. "How to Navigate Excel Objects from Visual Basic Version 3.0." (MSDN Library, Knowledge Base)

Knowledge Base Q114225. "XL5: OLE Automation Error Using Quit Method with GetObject." (MSDN Library, Knowledge Base)

Knowledge Base Q119469. "INF: How to Use OLE Automation to Modify MS Graph Object." (MSDN Library, Knowledge Base)

Knowledge Base Q120418. "INF: Using OLE Automation to Change a Graph's Type." (MSDN

Library, Knowledge Base)

Knowledge Base Q128994. "Behavior of GETOBJECT() with Excel and Word for Windows." (MSDN Library, Knowledge Base)

Knowledge Base Q129252. "XL7: Error Creating OLE Automation Object with Microsoft Excel." (MSDN Library, Knowledge Base)

Knowledge Base Q132535. "PRB: Releasing Object Variable Does Not Close Microsoft Excel." (MSDN Library, Knowledge Base)

Lassesen, Ken. "An Extended Introduction to Schedule+ OLE Automation Programming." (MSDN Library, Technical Articles)

Lassesen, Ken. "Mapping the Schedule+ Type Library: SPL 7.0." (MSDN Library, Technical Articles)

Microsoft Exchange Server SDK. *Microsoft Schedule+ Programmer's Guide*.

Microsoft Solutions Development Kit. "Your Unofficial Guide to Using OLE Automation with Microsoft Office and Microsoft BackOffice" 1995. (MSDN Library, Technical Articles)

Microsoft Visual FoxPro version 3.0 *Developers' Guide.* "Creating Objects with OLE Automation." 1995.

Microsoft Win32 Software Development Kit *OLE Programmer's Reference*. "OLE Registry Entries." 1995. (MSDN Library, Platform SDK)

Nilsen, Kenneth. "Using the OLE Automation Interface with Visual Basic." (MSDN Library Archive, Conference and Seminar Papers)

Wells, Eric. *Developing Microsoft Excel 95 Solutions*. Redmond, WA: Microsoft Press, 1995. Note especially "VBA Editing and Debugging Tools" in Chapter 2.
