Microsoft Visual Basic Version 4.0 ReadMe

Setup Information (ReadMe)

Miscellaneous Information (ReadMe)

Setup Toolkit and SetupWizard (ReadMe)

Data Access (ReadMe)

General

Connecting to ODBC Data Sources with no DSN
Microsoft SQL Server Stored Procedures
Orphaned Stored Procedures
Specifying Quoted Strings When Using ODBC Data Sources
Visigenic Oracle 32 bit Driver Installation Requirements

Microsoft Jet Data Access Objects (Professional and Enterprise Editions Only)

"Communication Link Failure" Error
Accessing SQL Server 6.0 Tables with Identity Columns
Can't Open FoxPro Table Contained in a Database Container
Cascades, Local Tables and Replication Don't Mix
Creating a **Recordset** Against an ODBC Data Source
DBEngine IniPath Now Uses Registry Entry
Miscellaneous Jet Database Replication Issues
Difference in Behavior of Error Message 'Data Has Changed'
Dynaset's Visibility of Extraneous Changes
GetRows Example Incorrect Code
Miscellaneous Jet Issues
Recordset Object Doesn't Support **CreateDynaset** Method
SystemDB Property of the **DBEngine** object
Use DAO to Access Local ISAM Databases
Using a Table Name When Addressing Fields in **Recordset** Objects
Using **CompactDatabase** with Microsoft Access Databases

Microsoft Remote Data Objects (RDO) (Enterprise Edition Only)

Creating Parameter Queries Example Incorrect
Executing RDO Queries Returning Multiple Resultsets
MaxRows Property (Remote Data)
ODBC Driver General Protection Faults When Given Incorrect Syntax
Remote Data Objects and Case-Sensitive Servers
Persistent Remote Data Object **rdoResultset** Objects
Providing Parameters to Stored Procedures
Remote Data Objects and **RemoteData** Control Miscellaneous Information
RemoteData Control **EOFAction** Property
SQL Server 6.0 Transactions With Server-Side Cursors
Syntax For the **OpenResultset** Method
Temporary Stored Procedures
Using a Forward-Only **rdoResultset**
Using Image or Text Data Types with the **RemoteData** Control or RDO
Using **SQLExecDirect**
Using the **RemoteData** control and the SQL Property
Working with BLOB Data Types and the ODBC Cursor Library

Crystal Reports (Professional and Enterprise Editions Only)

Paradox Engine Configuration Utility
Crystal VBX loses DataSource

OLE and Remote Automation (ReadMe)

Controls and Other Objects (ReadMe)

Miscellaneous (ReadMe)

Standard Controls (ReadMe)

Custom Controls (ReadMe)

CommonDialog Control

Changes to **CommonDialog** Flags for Windows 95 and Windows NT

DBCombo and DBList Controls

Area Parameter on Click and DblClick Events on **DBCombo** and **DBList** Controls

DBGrid Control

Cols property of the **DBGrid** Control
DBGrid Control Does Not Correctly Rebuild Columns
DBGrid Control Doesn't Repaint Border When Moved
HeadForeColor Property Applies To the **Column** Object
Properties for the **DBGrid** Control
RowLoaded Event Is Not Supported
Unbound Mode of the **DBGrid** Control

OLE Container Control

OLE Container Control: Pasting Objects From the Clipboard
SaveToFile vs. **SaveToOle1File**
Settings for the **Action** Property (MAPI Session Control)
Width and **Height** Property Values Change After Moving **OLE** Container control

(Professional and Enterprise Edition-Only Controls)

Gauge Control

Gauge Control Picture Property Also Accepts .WMF Files
NoDrop MousePointer is Nonfunctional in 16-bit Gauge and Key State Controls

Graph Control

GRAPH.VBX Control
Graph Control **ExtraData** Property Example
Unable to Access MAPI Functionality Under VB32.EXE

Masked Edit Control

Masked Edit Control: Mask Characters Incorrectly Listed in Custom Control Reference

Multimedia MCI Control

Multimedia Control **FileName** property

Windows 95 Controls (ReadMe)

These controls are available only in the Professional and Enterprise Editions of Visual Basic 4.0.

ImageList Control

ImageList Control Accepts More Than One Size Image
Picture Property is Read-only for **ImageList** ListImages

ListView Control

Error message 35604 - The first column in a **ListView** control must be left aligned

RichText Control

The **RichTextBox** Control Includes a **RightMargin** property

SSTab Control

Grouped **OptionButton** Controls on the **SSTab** Control
OCX files Installed When Custom Control Not Selected for Installation
TabStrip Control DblClick Event Not Supported

StatusBar Control

StatusBar Cannot Set **Left** Property of the **Panel** Object
The **StatusBar** Control Has a New **Style** Value

Toolbar Control

Constant Values in the Toolbar's **Value** Property (Custom Controls) Topic
Help Button on Customize Toolbar Dialog is Inactive
Toolbar Control: Button Objects with Placeholder Style Don't Wrap
Toolbar Control Example

TreeView Control

CreateDragImage Method Doesn't Display Text
HitTest Method Example has Faulty Code
TreeView Sorted Property Doesn't Sort Nodes Automatically

Language (ReadMe)

Error Messages (ReadMe)

User-defined types and fixed-length strings not allowed as the type of a public member
The topic does not exist. Contact your application vendor for an updated Help file.
Error message changes and additions

Functions (ReadMe)

Additional Information on VBA Registry Functions
Asc Function
Chr Function
FileAttr Function on 32-bit Operating Systems
InStr Function
Len and **LenB** Functions Return with User-defined Types
Shell Function Does Not Launch Documents with Associated Applications
Use Sleep API Instead of **DoEvents**
Useful Constants for **Chr**$ Function in Visual Basic for applications Type Library

Statements (ReadMe)

"For ... Next Statement" counter can be an element of a UDT
Additional Information on the **End** Statement
DeleteSetting Statement Differences Between Win16 and Win32
Functions as Arguments in **Print** Statements
Let and **Set** statements and **Property Let** and **Property Set** Procedures
Public Statement Cannot be Used in a Procedure
ReDim Statement as a Declaration
Using **Get** and **Put** with Arrays

Software Installation Information (ReadMe)

Welcome to Microsoft Visual Basic− the quickest and easiest way to create powerful applications for Microsoft Windows operating systems. Visual Basic helps you to be more productive by providing appropriate tools for the different aspects of Microsoft Windows application development.

The two Help topics listed below provide details on installing Visual Basic 4.0.

Setting Up From Floppy Disks

Setting Up From Compact Disc

Setting Up From Floppy Disks (ReadMe)

To set up Visual Basic from floppy disks

1. Insert Disk 1 in drive A.
2. Use the appropriate command in your operating environment to run the setup program.

- SETUP.EXE installs the 32-bit version of Visual Basic. The 32-bit version requires Microsoft Windows 95 or later, or Microsoft Windows NT 3.51 or later. The 32-bit version will compile 32-bit applications only.
- SETUP16.EXE installs the 16-bit version of Visual Basic. The 16-bit version requires Microsoft Windows 3.1 or later. The 16-bit version can also be installed and run on Windows 95 and Windows NT 3.51. The 16-bit version will compile 16-bit applications only.

3. Follow the setup instructions on the screen.

Setting Up From Compact Disc (ReadMe)

Uninstall (ReadMe)

If you want to uninstall Visual Basic 4.0, run Setup again from your original source and then follow instructions.

Setup Instructions (ReadMe)

Professional Edition

To set up the Professional Edition from compact disc

1. Insert the compact disc in the CD-ROM drive.
2. Use the appropriate command in your operating environment to run the setup program, which is available in the root directory on the compact disc. The setup application offers three installation options. You can do any of the following:

- Install the 32-bit version of Visual Basic. The 32-bit version of Visual Basic can be run only on 32-bit operating systems such as Microsoft NT 3.51 or higher, and Microsoft Windows 95.
- Install the 16-bit version of Visual Basic. The 16-bit version of Visual Basic can be used on 32-bit operating systems, as well as 16-bit environments like Windows 3.1 and Windows for Workgroups 3.11.
- Install Microsoft Developer Network (MSDN).

You may choose to install the 16-bit and 32-bit versions of Visual Basic in a single directory, or you can install them in separate directories. Installing them in the same directory allows you to save disk space by avoiding duplicating files used by both versions. In either case you should install the 16-bit version first.

3. Follow the Setup instructions on the screen

If you choose to not install the Help files, the CD must be in the CD-ROM drive for Help and Books Online to work correctly.

If you run into problems during Setup, contact your product support provider.

Enterprise Edition

To set up the Enterprise Edition from compact disc

1. Insert the compact disc in the CD-ROM drive.
2. Use the appropriate command in your operating environment to run the setup program, which is available in the root directory on the compact disc. The setup application offers four options. You can do any of the following:

- Install the 32-bit version of Visual Basic. The 32-bit version of Visual Basic can be run only on 32-bit operating systems like Microsoft NT 3.51 or higher, and Microsoft Windows 95.
- Install the 16-bit version of Visual Basic. The 16-bit version of Visual Basic can be used on 32-bit operating systems, as well as 16-bit environments like Windows 3.1 and Windows for Workgroups 3.11.
- Install Microsoft Visual SourceSafe 4.0.
- Install MSDN.

You may choose to install the 16-bit and 32-bit versions of Visual Basic in a single directory, or you can install them in separate directories. Installing them in the same directory allows you to save disk space by avoiding duplicating files used by both versions.

3. Follow the Setup instructions on the screen.

If you run into problems during Setup, contact your product support provider.

Contents of Your Compact Disc (ReadMe)

Root Directory

Contains SETUP.EXE, the master setup application file.

\VB Directory

Contains Visual Basic Books Online, and directories containing uncompressed, uninstalled copies of all of the files contained in your Visual Basic application. These files may be useful if you have to call product support. These files are not installed on your system. You can't run Visual Basic from this directory.

\MSDN Directory

Contains the Microsoft Developer Network (MSDN) Starter Kit. The Microsoft Developer Network for Visual Basic Users provides technical information and development toolkits for all developers who write applications for Microsoft operating systems. MSDN members receive a quarterly CD-ROM disk and a bimonthly newsletter. The CD contains code samples, technical articles, development tools, and the Microsoft Knowledge Base. Printed material included in your Visual Basic package provides information on starting a subscription to MSDN.

\TOOLS Directory

Contains various tools and accessories. These tools are intentionally not installed on your hard disk. For more information, see the readme or Help file provided in each directory.

Directory	Description
DATAEX32	Contains a tool that allows you to explore all of the data access options available for use with Visual Basic 4.0.
IMAGEDIT	Contains a tool that lets you create and edit bitmaps, cursors, and icons.
PSS	Contains a number of analytical programs to help identify problems with Visual Basic, OLE, and your application before you call for technical support. PSS engineers will often refer to these tools while working with you on an issue.
RESOURCE	Contains the Resource Compiler (RC), which compiles the resource-definition file and the resource files (such as icon and wave files) into a binary resource (.RES) file.
SYSINFO	Contains the 32-bit SysInfo control, which allows you to respond to certain system messages sent to all applications by the operating system. Your application can then adapt to changes in the operating system if necessary.
VBCP	Contains the Visual Basic Code Profiler, which is a Visual Basic add-in used to determine what code is being executed in an application, how many times it gets executed, and how long it takes to execute. It performs this analysis at the function level or line level.

\SRCSAFE Directory

Contains Microsoft Visual SourceSafe 4.0, an easy-to-use tool for team development of software, publications, manufacturing procedures, and any work that benefits from source control. Not available with the Standard or Professional Edition.

\SETUP Directory

Contains supplementary setup applications.

- SETUP.EXE installs the 32-bit version of Visual Basic. The 32-bit version requires Microsoft Windows 95 or later, or Microsoft Windows NT 3.51 or later. The 32-bit version compiles only 32-bit applications.
- SETUP16.EXE installs the 16-bit version of Visual Basic. The 16-bit version requires Microsoft Windows 3.1 or later. The 16-bit version can also be installed and run on Windows 95 and Windows NT 3.51. The 16-bit version compiles 16-bit applications only.

Using Disks with Distribution Media Format (DMF) (ReadMe)

With the exception of the Setup disk (Disk 1), your Microsoft Visual Basic disks use a new format called DMF (Distribution Media Format). DMF increases the capacity of a 3.5-inch floppy disk, reducing the number of disks needed to install your application.

Because DMF is a new format, many existing utilities such as Norton Disk Doctor, Microsoft ScanDisk, MS-DOS DiskCopy, and Microsoft Windows Copy Disk do not support DMF. You should not use disk utilities to examine a DMF formatted disk, as these utilities can corrupt the DMF disk. MS-DOS DiskCopy or Microsoft Windows Copy Disk cannot be used to copy DMF disks.

Operating systems other than Windows 3.1 (or later), Windows NT 3.5 (or later), or Windows 95 may not have the correct files to support DMF. For users with Windows NT 3.51, if you updated FLOPPY.SYS or have installed Microsoft Windows NT Service Pack 3, you should be able to install Microsoft Visual Basic. If you did not install these files or are unsure, there are two things you can do to correct this:

1. Update your operating system.
2. Contact Microsoft Product Support Services or use CompuServe to obtain the correct system files.

Copying Files from DMF Disks (ReadMe)

Operating systems prior to Microsoft Windows 95 or Microsoft Windows NT 3.5 cannot read files directly from DMF diskettes.

If you need to copy the Microsoft Visual Basic disks onto a network server or other fixed disk, you may use the copy switch (**/C**) with the EXTRACT.EXE utility on Disk 1 to copy the Microsoft Visual Basic installation files to the target location. For example, after creating a directory called C:\DISKS on your hard disk for the Microsoft Visual Basic files, copy all the files on Disk 1 to that directory. You can use the standard MS-DOS Copy command with Disk 1 because it does not use DMF:

```
COPY A:\*.* C:\DISKS
```

Switch to drive A and type the following command to copy the rest of the disks to the directory C:\DISKS:

```
FOR %I IN (*.*) DO C:\DISKS\EXTRACT /C A:\%I C:\DISKS\%I
```

A cabinet (.CAB) file includes many files stored as a single file. If for some reason you need only a single file that is contained in one of the cabinet files, you may search for it using the **/D** switch with EXTRACT.EXE. Once you find the file, you can use EXTRACT.EXE again to copy the file to the desired location. You can use the PACKING.LST file (located in the Visual Basic setup directory) to determine which cabinet files contain the Visual Basic files you want to extract. You can also type **EXTRACT /?** to get help on the EXTRACT command options.

Here are some examples of how to use the EXTRACT command to find files.

To list all files in a cabinet file:

EXTRACT /D A:*cabinetfilename*

To list all .EXE files in a cabinet file:

EXTRACT /D A: *cabinetfilename* ***.EXE**

Here are some examples of how to use EXTRACT to copy a single file out of a cabinet file.

To extract ANY.EXE to the current directory:

EXTRACT A:*cabinetfilename* **ANY.EXE**

To extract ANY.EXE to C:\VB:

EXTRACT A:*cabinetfilename* **/L C:\VB ANY.EXE**

Scroll Bars on TextBox Controls (ReadMe)

With the **Multiline** property set to **True** on a **TextBox** and the **ScrollBars** property set to anything except None (0), scroll bars will always appear on the **TextBox**.

File Sharing for OLE and Data Access (ReadMe)

If you are using applications that support OLE, you must run either SHARE.EXE or VSHARE.386. VSHARE.386 eliminates the need for SHARE.EXE when you run Windows 3.1 or Windows for Workgroups in 386 enhanced mode. If you run Windows 3.1 in standard mode, you still need to run SHARE.EXE. If you run applications that are not compatible with SHARE.EXE, and you run Windows 3.1 in 386 enhanced mode, you may be able to use VSHARE.386 instead of SHARE.EXE. If you are running Windows for Workgroups 3.1 or 3.11 in 386 enhanced mode, you are already using VSHARE.386. If you are running Windows 3.1, follow this procedure to use VSHARE.386.

To use VSHARE.386

1. Make sure VSHARE.386 is in the Windows \SYSTEM subdirectory.
2. Using a text editor (such as MS-DOS Editor), edit your AUTOEXEC.BAT file and remove the command for SHARE.EXE.
3. Edit the [386Enh] section in your SYSTEM.INI file to add the following line:

   ```
   device=vshare.386
   ```

Array Behavior Changes (ReadMe)

Because arrays are reallocated differently in Visual Basic 4.0, some code that worked in earlier versions may no longer work. Visual Basic now temporarily locks an array when any element of the array is passed by reference to another procedure. This means that, during the lifetime of the procedure that receives the element by reference, the array cannot be resized. The array is unlocked when there are no further references to array elements passed by reference.

Paradox Engine Configuration Utility (ReadMe)

In the Crystal Reports for Visual Basic Help file, the topic, "Paradox Engine Configuration Utility" refers to a file named PXENGCFG.EXE, which is used to configure the Paradox engine for a network. This file is not included with Visual Basic because the Microsoft Jet database engine handles all connections to Paradox data. For more information, search Help for *accessing external databases*.

Visual Basic 4.0 Does Not Support Out-of-Process OLE Controls (ReadMe)

An out-of-process OLE control is one running in a separate address space from the current instance of Visual Basic 4.0. If you attempt to load a 16-bit out-of-process OLE control on a 32-bit system, or vice versa, you'll get a clear and appropriate error message.

However, in the case of trying to load a 16-bit out-of-process OLE control on a 16-bit system, or a 32-bit out-of-process control on a 32-bit system, the error message may not be totally informative. Understand in this context that Visual Basic 4.0 does not support out-of-process OLE controls.

Width and Height Property Values Change After Moving OLE Container Control (ReadMe)

When a user moves an **OLE** container control on a form, the **Height** and **Width** property values of the object may be slightly different after the move. The parameters to OLE_ObjectMove() are pixel values converted to the current form's scaling mode. The conversion from pixels to twips and back doesn't result in identical values.

With...End With and the Line, Print, Circle, and PSet Methods (ReadMe)

The **Line**, **Print**, **Circle**, and **PSet** methods cannot be used in a **With...End With** block.

Recordset Object Doesn't Support CreateDynaset Method (ReadMe)

The **Data** control in Visual Basic 4.0 creates a **Recordset** object, whereas the **Data** control in Visual Basic 3.0 created a **Dynaset** object. The **CreateDynaset** method does not exist on the **Recordset** object. To verify this, follow this procedure.

1. Start Visual Basic; Form1 is created
2. Add a **Data** control (Data1) and a **TextBox** (Text1).
3. Set the following control properties in the Properties window:

Control	Property	Value
Data1	**DatabaseName**	"BIBLIO.MDB"
Data1	**RecordSource**	"Authors"
Data1	**RecordsetType**	1 - Dynaset
Text1	**DataSource**	Data1
Text1	**DataField**	"Author"

4. Add a **CommandButton** (Command1) to Form1. Add the following code to the Click event of Command1:

```
Private Sub Command1_Click()
   Dim ds As Dynaset

   Set ds = Data1.Recordset.CreateDynaset()
   ds.Close
   Set ds = Nothing
 End Sub
```

5. Press F5 and click the command button; error 3251 generated.

To avoid the error, change this code to `Data1.Recordset.OpenRecordset()` instead of `Data1.Recordset.CreateDynaset()`.

Changes to CommonDialog flags for Windows 95 and Windows NT (ReadMe)

The File Open/Save dialog box structure supports three new flags:

Flag	Value	Description
cdlOFNExplorer	0x00080000	Use the Explorer-like Open A File dialog box template. Common dialogs that use this flag do not work under Windows NT using the Windows 95 shell.
cdlOFNNoDereferenceLinks	0x00100000	Do not dereference shell links (also known as shortcuts). By default, choosing a shell link causes it to be dereferenced by the shell.
cdlOFNLongNames	0x00200000	Use long filenames.

The **cdlOFNExplorer** and **cdlOFNNoDereferenceLinks** flags work only under Windows 95. Multiselect common dialogs under Windows 95 using **cdlOFNExplorer** use null characters for delimiters, but under Windows NT or Win16, the multiselect uses spaces for delimiters (thus no support for long filenames). Of course when Windows NT gets the Windows 95 shell you will have to treat Windows NT and Windows 95 the same.

Multiselect Issues

Under both Windows NT and Windows 95 if you do not choose the **cdlOFNAllowMultiselect** flag, then both the **cdlOFNExplorer** and **cdlOFNLongNames** flags have no effect and are essentially the default. Under Windows NT, you will get long filenames, and under Windows 95 you will get the new style dialog and long filenames.

If you use the **cdlOFNAllowMultiselect** flag by itself under both Windows NT and Windows 95, you will not have support for long filenames. This is because the multiple filenames come back space delimited and long filenames could include spaces. Until Windows NT gets the Windows 95 shell, you cannot avoid this behavior. If you use **cdlOFNAllowMultiselect**, you cannot see long filenames. If you add the **cdlOFNExplorer** flag under Windows 95, you will be able to both multiselect and see long filenames. But the filenames come back null character delimited and not space delimited. Thus, using **cdlOFNAllowMultiselect** with **cdlOFNExplorer** will require different parsing of the filename result under Windows 95 and Windows NT.

Errors loading Visual Basic 3.0 Binary Projects from a Network (ReadMe)

When loading a Visual Basic 3.0 project using a UNC pathname, you may get the following message: `The basic code in <file name> was corrupt, and could not be loaded.`

To safely load a Visual Basic 3.0 project from a network share, copy the files to a local directory and load the project from that location.

Mapping Untrapped Errors in OLE Automation Objects (ReadMe)

The text following Figure 21.4 in Chapter 21, "Handling Run-Time Errors" in the *Programmer's Guide* states, "Visual Basic 4.0 automatically maps untrapped errors arising in objects outside of Visual Basic as error code 440." This is false. If an error is raised−or if you raise an error

−in an external object, and it is untrapped, it will be raised in the procedure that called the external object. It will not be mapped to error code 440.

Debugging Tips (ReadMe)

- Before beginning any debugging session, you should open the Options dialog box (available from the Tools menu), select the Advanced tab, and set the Error Trapping option. The setting for these options is not saved with the project; Visual Basic will use the last setting entered, even if the setting was entered for another project.

  The three Error Trapping options on the Advanced tab of the Options dialog box (available from the Tools menu) allow you to determine how errors are handled in the Visual Basic development environment. If you run your application and you get thrown into break mode unexpectedly, you can easily reset this option. During break mode, you can click the right mouse button to display a context menu that includes the following selections:

  Cut
  Copy
  Paste

  Toggle Breakpoint
  Step to Cursor
  Instant Watch
  Set Next Statement

  Break on All Errors
  Break in Class Module
  Break on Unhandled Errors

  Procedure Definition

  You can use the Break on All Errors, Break in Class Module, or Break on Unhandled Errors menu item to reset the Error Trapping option.

- CTRL+HOME and CTRL+END will take you to the top or bottom of the Immediate pane.

Use Sleep API Instead of DoEvents (ReadMe)

With the 32-bit version of Visual Basic, using the Sleep API function is more appropriate for 'waiting' in code than using **DoEvents**. The declare for this function is:

```
Declare Sub Sleep Lib "kernel32" Alias "Sleep" (ByVal dwMilliseconds _
As Long)
```

To call the Sleep function, you could use code like the following:

```
Call Sleep (1000)
```

Limitation on Number of Files on Project Load (ReadMe)

You can only open 384 files on project load.

New Icons (ReadMe)

In Appendix B , Icon Library in the *Programmer's Guide*, there are four message box icons listed in the 'Computers' section: MSGBOX01 through MSGBOX04. These icons allow you to create more elaborate message boxes than those available with the standard **MsgBox** function. The icons that Windows 95 displays using the **MsgBox** function are different from those displayed under Windows 3.1. Both sets are included in the \ICONS\COMPUTER directory. The new Windows 95 icons are:

W95MBX01.ICO
W95MBX02.ICO
W95MBX03.ICO
W95MBX04.ICO

Four additional icons that are not listed in Appendix B are included with Visual Basic 4.0:

Path	Description
\FLAGS\FLGRSA.ICO	Flag for the Republic of South Africa
\FLAGS\FLGBELG.ICO	Flag for Belgium
\WRITING\NOTE10B.ICO	Note (Italian)
\WRITING\NOTE10C.ICO	Note (Japanese)

Difference in Passing Controls Between Visual Basic 3.0 and Visual Basic 4.0 (ReadMe)

If you are calling a **Function** or **Sub** procedure in a .DLL or .VBX that was written for Visual Basic 3.0, and the **Declare** statement for the **Function** or **Sub** has a parameter that is defined **As Control**, then the correct Visual Basic 4.0 parameter should have a **ByVal** preceding the parameter name. This is because in Visual Basic 3.0 a parameter defined **As Control** incorrectly passed an hCtl (the handle to the control) rather than a pointer to the hCtl. Because all other parameters passed by reference (they had no **ByVal** preceding them) were passed as pointers to the parameters, the **As Control** parameters were an exception■which has been corrected in Visual Basic 4.0. For example, a DLL function such as:

```
HWND FAR PASCAL GetControlHwnd(HCTL hCtl)
{
   return VBGetControlHwnd(hCtl);
}
```

would have a Visual Basic 3.0 declaration of:

```
Declare Function GetControlHwnd Lib libname.dll (hCtl As Control) _
As Integer
```

and should have a Visual Basic 4.0 declaration of:

```
Declare Function GetControlHwnd Lib libname.dll (ByVal hCtl As _
Control) As Integer
```

Visual Basic 4.0 resolves this problem in the vast majority of the cases, and can correctly identify the proper value to pass to Visual Basic APIs. However, to make absolutely sure that the program using this .VBX or .DLL will be compatible with future releases of Visual Basic, you should follow the correct methodology in setting parameters.

Using Compile on Demand (ReadMe)

If you select the Background Compile or Compile on Demand option on the Advanced tab on the Options dialog box and then choose Start With Full Compile from the Run menu, Visual Basic overrides the check box settings on the Advanced tab and performs a full compile.

In Chapter 20, "Debugging," in the *Programmer's Guide*, the section with this title contains text recommending that you flush out hidden errors by turning Compile on Demand off and then running the application. This may require you to reset Compile on Demand after running the application. A better option may be to use CTRL+F5, Start after Full Compile (Run menu), because that will leave the setting of Compile On Demand alone.

Development Environment Tips (ReadMe)

- CTRL+TAB and SHIFT+CTRL+TAB move focus between the windows in the Visual Basic 4.0 development environment.
- You can select multiple controls on a form and then set the value of common properties in the Properties window. To select multiple controls, you can either press the left mouse button and select each control by dragging, or press SHIFT while clicking each of the controls you want to include in the selection. The Properties window then displays only properties that are common to all the controls you selected. Values you enter in the Properties window apply to all the selected controls. When you select a group of **TextBox** controls however, the **Text** property is not available.
- Double-clicking the left margin of the Code window selects an entire procedure, pressing CTRL and clicking the left margin of the Code window selects all, and a single-clicking selects a line.
- CTRL+SHIFT+F2 takes the caret back to its previous position. This is useful after using SHIFT+F2 to examine a procedure.

DeleteSetting Statement Differences Between Win16 and Win32 (ReadMe)

The **DeleteSetting** statement, which deletes initialization information, should operate the same on Win16 and Win32 operating systems (for instance, it should raise run-time errors in the same way). There is one minor difference. On Windows NT, this code runs without a generating a run-time error:

```
SaveSetting "foo", "sect", "key", "val"
DeleteSetting "foo", "sect", "key" ' Assume "key" is last key in "sect"
DeleteSetting "foo", "sect"
```

On Win16, the final **DeleteSetting** statement raises an `Illegal Function Call` error.

On Win16, **DeleteSetting** cannot tell the difference between an empty section (a section with no keys), and a section which is not present in the .INI file. Deleting a key followed by the whole section doesn't make a lot of sense.

To keep the .INI on Win16 consistent, **DeleteSetting** on Win16 will remove a section name when the last key in the section is deleted. This makes it easier for the user to see if a section doesn't exist.

Note that the **DeleteSetting** statement stores information differently on Win16 and Win32 operating systems: On Win16, **SaveSetting**, **GetSetting**, **GetAllSetting**, and **DeleteSetting** operate on an .INI file; on Win32, these operate on the Windows Registry.

CUR, ICO, and ANI Files (ReadMe)

Visual Basic 4.0 does not support color cursor files (.CUR). Color cursor files such as those shipped with Windows NT 3.51, will be displayed in black and white. To display a color cursor, use a color icon file (.ICO).

You can use the **MouseIcon** property to load either cursor or icon files. This provides your program with easy access to custom cursors of any size, with any desired hot spot location.

The 32-bit version of Visual Basic does not support loading of animated cursor (.ANI) files, even though they are supported by 32-bit Windows operating systems.

Multimedia Control FileName Property (ReadMe)

The Multimedia MCI control has a **FileName** property, which allows you to specify which file to play. For example, to play a .WAV file, you could specify a particular file to play. However, to change the **FileName** property, you must close and reopen the Multimedia MCI control.

For example:

```
' Place a MCI control and a FileListBox on a form, and paste
' this code into the form.

Private Sub File1_Click()
   ' The code enumerates through the FileListBox's items.
   ' When the code finds a selected item, it closes MMControl1,
   ' resets the device type, resets the FileName property with
   ' the selected item, then reopens MMControl1.
   Dim i As Integer
   For i = 0 To File1.ListCount - 1
      If File1.Selected(i) = True Then
         MMControl1.Command = "close"
         MMControl1.DeviceType = "WaveAudio"
         MMControl1.FileName = "c:\windows\" & File1.List(i)
         MMControl1.Command = "open"
      End If
   Next i
End Sub

Private Sub Form_Load()
   ' Set properties needed by MMControl1 to open.
   With MMControl1
      .Notify = False
      .Wait = True
      .Shareable = False
      .DeviceType = "WaveAudio"
      ' Set the file to be opened before opening the MMControl.
      .FileName = "C:\WINDOWS\chimes.WAV"
      ' Open the MCI WaveAudio device.
      .Command = "Open"
   End With

   ' Set parameters for the FileListBox.
   File1.Path = "c:\windows"
   File1.Pattern = "*.wav"   ' Only display .WAV files.

End Sub

Private Sub Form_Unload(Cancel As Integer)
   ' Close the control to free resources.
   MMControl1.Command = "close"
End Sub
```

Guide to Data Access Objects Changes (ReadMe)

Under "Functionality Supported by Some Servers" in Chapter 9, "Developing Client/Server Applications," the **CVDate** function is now the **CDate** function.

The RichTextBox Control Includes a RightMargin Property (ReadMe)

Returns or sets the right margin for the text in a **RichTextBox** control.

Syntax

object.**RightMargin** [= *value*]

The **RightMargin** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a **RichTextBox** control.
value	An **Integer** that determines the indent in twips from the right edge of the text to the right edge of the control.

Remarks

The **RichTextBox** control also uses the **RightMargin** property to determine how to word wrap.

Limitations on TabIndex Property (ReadMe)

The **Menu**, **Timer**, **CommonDialog**, **Data**, **Image**, **Line**, and **Shape** controls are not included in the tab order in Visual Basic. To verify this, place a **TextBox** control on a form and then in order, a **CommonDialog**, **Data**, **Image**, **Line**, **Shape**, **Timer**, and second **TextBox** control. Check the **TabIndex** properties of the two **TextBox** controls. Notice they are 0 and 1. The other controls are not included in the tab order.

Using a Table Name When Addressing Fields in Recordset Objects (ReadMe)

In Visual Basic version 3.0, you could reference the fields in a **Recordset** using a variety of syntax. For example, all of the following code used to reference the Title field of a **Dynaset** object created against the Titles table are valid:

```
Dim db As Database, ds As Dynaset
Set db = OpenDatabase("biblio.mdb")
Set ds = db.CreateDynaset("Select * from Titles")

Print ds(0)
Print ds!Title
Print ds("title")
Print ds.Fields("title").Value
Print ds("titles.title")
```

In Visual Basic 4.0, using the Table name in the string when the **Dynaset**, **Snapshot**, or dynaset-type or snapshot-type **Recordset** is generated from an SQL string is no longer valid.

SetupWizard Dependency File Notes for Microsoft Visual Basic (ReadMe)

The following topics describe the SetupWizard's SWDEPEND.INI file and provide information about which Visual Basic files can be freely distributed with your Visual Basic applications in accordance with the other requirements in the license agreement.

SWDEPEND.INI (ReadMe)

Depending on the operations your application performs and the custom controls it uses, you may need to distribute separate files containing the objects used by your application. These other files are called dependencies and are identified in the SWDEPEND.INI file (located in the \WINDOWS directory), which is a standard Windows .INI file that can be modified with a text editor.

SWDEPEND.INI lists the file dependencies for the controls (.OCX and .VBX), dynamic link libraries (DLLs), and references shipped with Visual Basic. When a reference to a custom control (.OCX or .VBX), DLL, or .EXE file in your application's .VBP file is detected by the SetupWizard file (Step 5), the SWDEPEND.INI file will be consulted to determine which other files should be distributed with your application.

SETUP.LST (required by SETUP.EXE) depends on information supplied in the SWDEPEND.INI file, and is created automatically by the SetupWizard. SETUP.LST is a text file that lists all the files to be installed on your user's machine, including the disk on which they reside, where they will be installed, whether and how they are to be registered in the system registry, and so forth.

Each section of the SWDEPEND.INI file will have one or more entries representing:

- The individual files that your Visual Basic application depends on.
- Other entire sections that Visual Basic depends on.
- The destination path where the files associated with that dependency will be installed.
- How the dependency is registered when it is installed.

Note If Visual Basic version 3.0 was installed on your machine, relevant information from the Visual Basic 3.0 SETUPWIZ.INI file will be copied to SWDEPEND.INI the first time you use the Visual Basic 4.0 SetupWizard.

Syntax in SWDEPEND.INI (ReadMe)

Each section in SWDEPEND.INI uses the following syntax:

[*DEPENDENCYNAME*]
Dest=*DESTINATIONDIRECTORY*
Uses*N***=***FILENAME.EXT*
Register=*REGISTERKEY*

For example, the following is the section for [ODBC.DLL], so if your application uses ODBC, then you will need to distribute all of the files listed under [ODBC.DLL].

```
[ODBC.DLL]
Dest=$(WinSysPath)
Uses1=ODBC.DLL
Uses2=ODBCADM.EXE
Uses3=ODBCCURS.DLL
Uses4=ODBCINST.DLL
Uses5=CTL3DV2.DLL
```

[DEPENDENCYNAME]

DEPENDENCYNAME is the name of the custom control (.OCX or .VBX) or DLL file that is a dependency file, such as OLE2.DLL or SPIN16.OCX. Each control should have a separate section for the 16-bit and 32-bit files that may be used.

If a section does not use an actual filename as the *DEPENDENCYNAME*, Visual Basic will append "-32" to a *DEPENDENCYNAME* when it is called from a 32-bit version of Visual Basic (for instance, [SetupWiz-32] instead of [SetupWiz]).

Dest=DESTINATIONDIRECTORY

The optional *DESTINATIONDIRECTORY* argument specifies where the *DEPENDENCYNAME* file should be installed. *DESTINATIONDIRECTORY* can take the following values:

Value	Description
$(WinSysPath)	Installs the file into the user's \WINDOWS\SYSTEM directory.
$(WinPath)	Installs the file into the user's \WINDOWS directory.
$(AppPath)	Installs the file into the application's root directory, which is specified by the user during setup.
c:\path	Installs the file into the specified path (not recommended).
$(AppPath)\SAMPLES	Installs the file into the \SAMPLES subdirectory, just below the application's root directory.

If no *DESTINATIONDIRECTORY* is supplied, the SetupWizard will determine the destination directory, based on the file's extension. All .DLL, .OCX, and .VBX files will be installed into the \WINDOWS\SYSTEM directory, and all other files will be installed into the application's root directory.

If no *DESTINATIONDIRECTORY* is supplied for a *DEPENDENCYNAME*, but that dependency is a file used by another dependency section and that second dependency section contains *DESTINATIONDIRECTORY* information, then the first dependency will be installed to the location specified under the second dependency. For example, although no *DESTINATIONDIRECTORY* is specified under the section for DEPEND1.DLL, DEPEND1.DLL will be copied to the \WINDOWS\SYSTEM directory, as specified under DEPEND2.DLL:

```
[DEPEND1.DLL]
Uses1=SUPPORT.DLL
Register=$(DLLSelfRegister)

[DEPEND2.DLL]
Uses1=SUPPRT2.EXE
Uses2=DEPEND1.DLL
```

```
Dest=$(WinSysPath)
Register=$(DLLSelfRegister)
```

Note Filenames ending with ":1" will be installed in the \WINDOWS\SYSTEM directory, to be backward compatible with the Visual Basic 3.0 SETUPWIZ.INI file.

UsesN=FILENAME.EXT

In the **Uses***N* syntax, *N* starts at 1 and is incremented for each dependent file or group. *FILENAME.EXT* lists files, and the other sections that Visual Basic 4.0 depends on for the *DEPENDENCYNAME* dependency. Filenames ending with ":0" will be copied to disks but not compressed, and installed "as is" on your user's machine.

Register=REGISTERKEY

REGISTERKEY indicates how an object can get registered. *REGISTERKEY* can take the following values:

Value	Description
$(DLLSelfRegister)	The component is a DLL file that includes self-registering information that should be called during the installation. This also applies to OLE controls (.OCX).
$(EXESelfRegister)	The component is an .EXE that can be invoked with the /**REGSERVER** command-line argument during installation. This applies to OLE servers created with Visual Basic.
FILENAME.REG	The component uses a .REG file to get registered, and REGEDIT.EXE should be called with *FILENAME*.REG.

Third-Party Controls: Adding Dependency Information to SWDEPEND.INI (ReadMe)

If you are developing and distributing your own custom controls, you will need to add dependency information to the SWDEPEND.INI file. Also, if you are using third-party custom controls, you may need to add dependency information to the SWDEPEND.INI file for those custom controls (if the installation program for those controls does not do this for you).

The following code snippet (which you could add to your setup program) adds a section named "OCXNAME.OCX" (the name of the control) and creates the line "Uses1=SUPPORT.DLL" (SUPPORT.DLL might be the name of a support DLL for the .OCX). The second entry specifies the line "Register=$(DLLSelfRegister)" because .OCX controls typically contain a DLLSelfRegister function.

```
fOk = WritePrivateProfileString("OCXNAME.OCX", "Uses1", "SUPPORT.DLL",
"SWDEPEND.INI");

fOk = WritePrivateProfileString("OCXNAME.OCX", "Register",
"$(DLLSelfRegister)", "SWDEPEND.INI");
```

The following information would then be added to the SWDEPEND.INI file:

```
[OCXNAME.OCX]
Uses1=SUPPORT.DLL
Register=$(DLLSelfRegister)
```

Other Redistributable Visual Basic Files (ReadMe)

The files in the following directories can be freely distributed with Visual Basic applications:

\BITMAPS
\ICONS
\METAFILE
\SAMPLES

\ODBC or \ODBC32 (Professional Edition only)

Also, the following files included with Visual Basic 4.0, Enterprise Edition are distributable royalty free according to the terms of the Microsoft Visual Basic 4.0 license agreement.

Distributable with 16-bit client projects

AUTPRX16.DLL
AUTPRX.DLL
SECURITY.DLL
RPCRT1.DLL
RPC16C1.DLL
RPC16C5.DLL
RPC16C6.DLL
RPC16DG6.DLL
RPC16C4.DLL
RPC16DG3.DLL
RPC16C3.DLL
RPCREG.DAT

Distributable with 32-bit client projects

AUTPRX32.DLL
AUTMGR32.EXE

Distributable with 32-bit server projects

AUTREG32.DLL
RACMGR32.EXE
AUTMGR32.EXE
AUTPRX32.DLL

Print Method and the AutoRedraw Property (ReadMe)

If you have **AutoRedraw** set to **False**, the **Print** method prints on top of graphical controls such as the **Image** and **Shape** controls.

Printing on Forms, Picture Boxes, and the Debug window (ReadMe)

If you use a comma to separate items in the outputlist, Visual Basic separates these items with a tab when printing. For example:

```
Debug.Print "hello", "hello"
```

will print:

```
hello     hello
```

Testing for Multiple Buttons (ReadMe)

When the code example on page 341 under "Testing for Multiple Buttons'" in Chapter 12, "Responding to Mouse Events," of the *Programmer's Guide* is run and both buttons are pressed, all three messages in the section's MouseMove procedure will be displayed.

Selecting Drag Icons (ReadMe)

When you set the **DragIcon** property at run time by assigning the **Picture** property of one control to the **DragIcon** property of another control, the **Picture** property must contain an .ICO file, not a .BMP file.

The StatusBar Control Has a New Style Value (ReadMe)

The **StatusBar** control has a new **Style** property value of 7 (Kana lock key). This value displays the letters "KANA" in bold when scroll lock is enabled, and dimmed when disabled.

StatusBar is a Windows 95 control, and is not available in the Standard Edition of Visual Basic.

Unexpected Triggering of the Deactivate and LostFocus Events (ReadMe)

If an .EXE file built by Visual Basic displays a dialog box created by a .DLL also built in Visual Basic, the .EXE file's form will get Deactivate and LostFocus events. This may be unexpected, because you should not get the Deactivate event:

- If the server is an out-of-process server.
- If the server isn't written in Visual Basic.
- In the development environment when calling a DLL built in Visual Basic.

Adding Remote Server Support Files in Setup (ReadMe)

The check box in Step 6 of the SetupWizard for determining whether or not to add remote server support files will add not only AUTMGR32.DLL and AUTMGR32.EXE to the distribution files, but also RACMGR32.EXE, GAUGE32.OCX, TABCTL32.OCX, RACREG32.DLL, and ODKOB32.DLL.

This clarifies what is stated just before Figure 30.8 in the Chapter 30, "Distributing Your Applications," of the *Programmer's Guide*.

Some Differences Between Windows 95, Windows NT and Win16 (ReadMe)

- Context menus should be triggered under the MouseUp event on Windows 95 and should always use the **vbPopupMenuRightButton** flag to act like Windows 95 context menus. Under Windows NT and Win16 they should be triggered on the MouseDown event. When Windows NT gets the Windows 95 shell, you must treat Windows NT and Windows 95 the same.
- Windows 95 common controls are not available on 16-bit versions of Visual Basic.
- OLE server options are not identical between 16-bit and 32-bit versions of Visual Basic.

Avoid Hiding and Showing a Modal Form in the Same Event (ReadMe)

You should avoid hiding and showing a modal form in the same event for reasons described here. The results you encounter

- either an `Out of stack space` error or no event generation
- are not what you would expect, but it is by design.

The problem is related to a low stack space condition, though you do not actually run out of stack space. The reason the low stack space condition occurs is due to a limitation in how Visual Basic handles the showing and hiding of modal forms within the same event. Before an event is called, Visual Basic sets a status flag of the modal state of the active form. Visual Basic doesn't reset this flag until the event has completed. So if you hide the active (modal) form within an event, Visual Basic will still treat the form as modal even though the act of hiding it should make it non-modal.

When you hide a form, you would expect the code immediately after the `FormX.Show 1` statement to be executed, but this does not happen. Visual Basic executes the remaining code in the event, then after the event is completed, checks the modal state of the form and then executes the code following the `FormX.Show 1`. It will execute the code only if no other modal forms are showing. Below is a code sample assuming you have two forms, Form1 and Form2, in your project.

```
' Code for Form1.
Private Sub Form_Click ()
   Form2.Show 1
   Debug.Print "Form2 is non-modal"
End Sub

' Code for Form2.
Private Sub Form_Click ()
   Debug.Print "Hiding form2"
   Form2.Hide
   Debug.Print "Form2 is hidden"
End Sub
```

Run the above code, click Form1 and then click Form2. In the Debug window you will see:

```
Hiding form2
Form2 is hidden
<The Form_Click event of Form2 completes>
Form2 is non-modal
<The Form_Click event of Form1 completes>
```

This scenario demonstrates that Visual Basic will execute the remaining code in Form_Click of Form2 before executing the code after the `Form2.Show 1` statement. If indeed the form was considered non-modal immediately after the `Form2.Hide` statement, you would expect to see:

```
Hiding form2
Form2 is non-modal
<The Form_Click event of Form1 completes>
Form2 is hidden
<The Form_Click event of Form2 completes>
```

In this case, Visual Basic will always show the next modal form before it considers the current form to be non-modal. Whenever you click the command button, a modal form is always showing, therefore Visual Basic has no opportunity to complete the event where the form was shown modally. Each click adds another call to an event that cannot complete and you end up with recursive calls to each event.

To work around this behavior, you need to separate the `FormX.Hide` for the current form and the `FormX.Show 1` for the new form into separate events. For example, you can hide

the current form in the Form_Click event (as your already doing), enable a timer, and then show the new modal form from within the Timer event. Using this scheme leads to the following events:

1. The current form is hidden.
2. The timer is enabled for 1 millisecond.
3. The Click event terminates. Visual Basic now recognizes the form is no longer modal.
4. The code after `FormX.Show 1` for the current form completes, thus the event where the form was shown completes, avoiding recursion.
5. The Timer event is triggered.
6. The new form is shown modally.
7. The timer is disabled to avoid additional events.

OLE Container Control: Pasting Objects from the Clipboard (ReadMe)

Applications that provide objects behave differently when an object is deleted. When you delete an OLE object (using the **Delete** method), the objects application may or may not close. If the application does close, any objects on the Clipboard associated with that application may also be closed. Because of this, you may not be able to cut an object (copy, then delete), because deleting the object may also cause the data on the Clipboard to be deleted.

Another instance of this behavior is when you try to copy an object, and then paste the object back onto itself. This action may cause an error, because in order to paste over an existing object, the existing object is first deleted. If the application associated with the object closes, and subsequently deletes any objects it has on the Clipboard, the Clipboard no longer contains an object to paste.

Height and Width Range for Grid Cells on the Form (ReadMe)

The range for both height and width of grid cells on the form at design time is 24 to 1188 twips.

This is a correction to information in "Environment Options" in Chapter 4, "Managing Projects," of the *Programmer's Guide*, and to the Help topic 'Environment Options Tab.'

Use of REGSVR and REGOCX (ReadMe)

REGSVR.EXE and REGSVR32.EXE

You can use the REGSVR.EXE and REGSVR32.EXE utilities to manually register and unregister OLE servers, OLE DLLs, and OLE controls (.OCX). REGSVR.EXE and REGSVR32.EXE are available in the \TOOLS directory.

Syntax

REGSVR[**32**] [**/u**] *filename*

The REGSVR[32] syntax has these parts:

Part	Description
/u	Unregisters an OLE server, OLE DLL, or an OLE control.
filename	The name of the file you want to register or unregister.

Remarks

The utility will display either a success or failure dialog box upon completion.

REGOCX16.EXE and REGOCX32.EXE

You can use the REGOCX16.EXE and REGOCX32.EXE utilities to manually register and unregister OLE controls.

Syntax

REGOCX[**16**|**32**] [**/u**] *filename*

The REGOCX[16|32] syntax has these parts:

Part	Description
/u	Unregisters an OLE control.
filename	The name of the file you want to register or unregister.

Remarks

The utility does not return success or failure upon completion. To determine if the operation succeeded, use REGSVR[32].EXE, which does return a result. REGOCX16.EXE and REGOCX32.EXE are available in the \TOOLS directory.

Creating Picture and Font Objects (ReadMe)

If you set a reference to Standard OLE Types using the References dialog box, you can use the StdFont and StdPicture classes to create your own font types. If you view the Object Browser, you will notice that there are StdFont, StdPicture, Font, and Picture classes. The Font and Picture classes are derived from the StdFont and StdPicture base classes and are supported by all controls.

You can use the following syntax:

```
Dim MyFont As Font
```

But, you cannot use:

```
Dim MyFont As New Font
```

Instead, to create your own font or picture types, use code like the following:

```
Dim MyFont As New StdFont
With MyFont
   .Bold = True
   .Name = "Arial"
End With
Set Text1.Font = MyFont
```

Returning an Error Value from a DLL (ReadMe)

To return an error value from a dynamic link library (DLL) procedure, the C language prototype must be coded so that the return value is an HRESULT. Refer to the Microsoft Press *OLE 2 Programmers Reference*, *Volume 2* for more information on how to do this.

Functions as Arguments in Print Statements (ReadMe)

Visual Basic hoists function calls when they are specified as in an argument expression for a **Print** statement. Visual Basic first evaluates all functions that are arguments of a **Print** statement, and then prints the resulting argument return values. For example, the following code would display different values in the Debug window for Visual Basic 3.0 and Visual Basic 4.0:

```
Function F(N)
   N = N + 1
   F = N
End Function

Sub Test ()
    N = 3
   Debug.Print N, F (N), N
End Sub
```

Visual Basic 3.0 will print the following in the Debug window:

```
3     4     4
```

Visual Basic 4.0 will print the following in the Debug window:

```
4     4     4
```

GRAPH.VBX Control (ReadMe)

Some of the properties of the Visual Basic 3.0 GRAPH.VBX control will not be saved properly in Visual Basic 4.0, and this may result in loss of data. If you are using the **Graph** control extensively, you should upgrade to the GRAPH16.OCX or GRAPH32.OCX control.

Sample Code for Font dialog box (ReadMe)

In the section, "Using the Font Dialog Box" in Chapter 11, "Dialogs," of the *Programmer's Guide*, the sample code on one line should be:

```
Text1.Font.Strikethru = CMDialog1.FontStrikethru
```

and not:

```
Text1.FontStrikethru = CMDialog1.FontStrikethru
```

Append Behavior Changes (ReadMe)

In earlier versions of Visual Basic, when a file is opened for **Append**, Visual Basic sets the next write position to the position of the first of any CTRL+Z characters (ASCII 26) in the file. Visual Basic now sets the write position after the last character in the file, whether or not it contains embedded CTRL+Z characters.

DBEngine IniPath Now Uses Registry Entry (ReadMe)

The **DBEngine**.**IniPath** property on 32-bit systems now uses a Windows Registry entry, not an .INI file. This is an update to the Help topics 'Customizing Data Access INI Settings' and 'Managing Connections to ODBC Data Sources.'

For example, to store an applications setting, you can use the following code:

```
SaveSettting "AppName", "Engines\Jet", "System\DB", "C:\DATA\SYSTEM.MDA"
```

To retrieve an application setting, you can use the following code:

```
DBEngine.IniPath = "HKEY_CURRENT_USER\Software\VB and VBA Program Settings\
appname"
```

Additional information on the End Statement (ReadMe)

The **End** Statement topic states, "End■Terminates execution. Never required by itself but may be placed anywhere in a procedure to close files opened with the **Open** statement and to clear variables."

Understand that the **End** statement stops execution abruptly, without invoking the Unload, QueryUnload, or Terminate event. The **End** statement terminates execution immediately, without executing any further Visual Basic code. Code you have placed in the Unload, QueryUnload, and Terminate events of forms and class modules will not be executed. Objects created from class modules will be destroyed, files opened using the **Open** statement will be closed, and memory used by your program will be freed. Object references held by other programs will be invalidated.

The **End** statement provides a way to force your program to halt. For normal termination of a Visual Basic program, it is recommended that you unload all forms. Your program will then close as soon as there are no other programs holding references to objects created from your public class modules, and no code executing.

The Ending Execution topic states, "To end execution, choose End from the Run menu, or click the End button on the toolbar. You can also use the **End** statement in code."

Understand in addition that all of the above terminate execution immediately, without executing code you have place in the Unload, QueryUnload, and Terminate events of forms and class modules. Search Visual Basic Help for *End statement* for more information.

SetupWizard Macros Listing (ReadMe)

In Step 7 of the SetupWizard, a File Details Destination Directory drop-down combo box lists available macros. The list is incomplete. The following are macros that are supported but do not appear in the drop-down list box:

- $(CommonFiles)
- $(WinSysPathSysFile)
- $(MsAppsPath)
- $(ProgramFiles)

$(CommonFiles) is documented, but it is not recommended for use without a derived subdirectory.

$(ProgramFiles) is used for the default application directory (which the user can then modify).

"Communication Link Failure" Error (ReadMe)

If you run Visual Basic 3.0 applications after installing Visual Basic 4.0, you may receive a `Communication Link Failure` error when executing queries with Microsoft Jet against a Microsoft or Sybase SQL Server. You can retry the operation with asynchronous execution disabled. To do this, add the following entry to your VB.INI file at design time. Also, at run time, add this to the *appname*.INI file indicated by the **IniPath** property:

```
[Debug]
RmtTrace=16
```

Visual Basic will continue to run synchronously until this line is removed from VB.INI.

No Click Event on Right Mouse Button for ListBox (ReadMe)

The Click event Help topic states:

"Occurs when the user presses and then releases a mouse button over an object. It can also occur when the value of a control is changed.

For a **Form** object, this event occurs when the user clicks either a blank area or a disabled control. For a control, this event occurs when the user:

Clicks a control with the left or right mouse button. With a **CheckBox**, **CommandButton**, or **OptionButton** control, the Click event occurs only when the user clicks the left mouse button."

This list of controls should also include the **ListBox** control. A right mouse button click will not invoke a Click event on a **ListBox**.

ImageList Control Accepts More Than One Size Image (ReadMe)

Images of many different sizes can be added to an **ImageList** control. It stretches the new images as necessary. The resolution of the images is determined by either of the following:

- The setting of **ImageWidth** and **ImageHeight** properties before any images are added.
- The dimensions of the first image added.

For example, if the first image added is a 16 x 16 icon, and then a 32 x 32 icon is added, they will both be displayed as 16 x 16 images.

Add ListBox to Objects in PG Chapter 7 Example (ReadMe)

The table of objects and settings for the example following the heading "Public Collection Example: The House of Straw" is missing an entry for the list box. This is in Chapter 7, "Introduction to Objects," of the *Programmer's Guide*.

Object	Property	Value
List box	Name	lstEmployees

Area Parameter on Click and DblClick Events on DBCombo and DBList Controls (ReadMe)

The **DBCombo** and **DBList** controls have a parameter named *area* in their Click and DblClick events.

Syntax

Private Sub DBCombo_Click (*area* **As Integer)**

Private Sub DBList_Click (*area* **As Integer)**

Private Sub DBCombo_DblClick (*area* **As Integer)**

Private Sub DBList_DblClick (*area* **As Integer)**

The *area* argument identifies what area the click is made on. If you need to test for the value of the *area* argument, you can use constants listed in the Microsoft Data Bound List Controls (MSDBCtls) object library in the Object Browser.

Constant	Value	Description
dbcAreaButton	0	Button area
dbcAreaEdit	1	Edit area
dbcAreaList	2	List area

Explanation of the Behavior of Null String Pointers in Visual Basic 4.0 (ReadMe)

Visual Basic 4.0 recognizes two very different kinds of strings that look the same, but act different. One is a **Null** pointer string. The other is an empty string. Here's how you might code them in a public module:

```
Public Const sEmpty = ""
Public sNull As String
```

If you look at these two strings in the Watch pane, they look exactly the same (they are both displayed as ""), and you can use them in almost the same contexts. However, internally they are very different.

Internally, `sEmpty` is a pointer to an empty string. It is a valid pointer to some memory location. In C this would be coded as:

```
const char sEmpty[] = "";
```

Whereas internally, `sNull` is a **Null** pointer. This does not point to any memory location and has a value of zero. In C this would be coded as:

```
const char *sNull = NULL;
```

All Basic variables are initialized to 0 until initialized. In previous versions of Visual Basic, uninitialized variable-length strings were automatically initialized to an empty string (""). Hence, for compatibility with previous versions, string variables must be initialized to empty strings. But in Visual Basic 4.0, strings are in the BSTR format. In the BSTR format, a null pointer is defined to behave exactly the same as an empty string. So, Visual Basic 4.0 can just leave the initial zero value of uninitialized strings and get the same behavior as for empty strings. This means that `sNull` can now be passed to any Windows API function that takes a **Null** pointer. This is something that was not possible in previous versions of Visual Basic. For example, it can be passed to FindWindow, which gets the handle of a window, given either its class name or its title, or both. However, `sNull` must be passed **ByVal** for this to work.

In general, for FindWindow (or any other Windows API) to work, the **Declare** statement must be written to pass the string **ByVal As String** or **ByVal As Any**. If they are passed by reference, a pointer to a BSTR would then be passed, which is nothing but a pointer to a pointer to char. This will not work because Windows APIs expect strings that are pointers to char.

In Visual Basic 4.0, `sNull` might also be expected to be equivalent to:

```
Public Const sNull As String = 0&
```

However, Visual Basic does automatic numeric conversion on this and converts it to "0", which is neither an empty string nor a null string pointer.

Step-by-Step Example

1. Start a new project in Visual Basic. Form1 is created by default.
2. Add the following code to the general Declarations section of Form1:

   ```
   Const sEmpty = ""
   Dim sNull As String

   Private Declare Function FindWindow Lib "user32" Alias _
   "FindWindowA" (ByVal lpClassName As Any, ByVal _
   lpWindowName As Any) As Long
   ```

3. In the Form_Click event for Form1, add the following code:

   ```
   Shell "Calc.exe", 1
   DoEvents
   x& = FindWindow(sNull, "Calculator")
   ' x& = FindWindow(sEmpty, "Calculator")
   Debug.Print x&
   ```

4. Press F5 to run the program. Click Form1 and view the Debug window. A nonzero value will be printed. This is the handle of the Calculator programs Window. If the line:

   ```
   x& = FindWindow(sNull, "Calculator")
   ```

   is commented and the line:

   ```
   ' x& = FindWindow(sEmpty, "Calculator")
   ```

   is uncommented and then the program is run again, a value of 0 will be printed in the Debug window, indicating that FindWindow failed. This happens because `sEmpty` is not a **Null** string pointer.

Difference in Behavior of Error Message 'Data Has Changed' (ReadMe)

There is a difference in the behavior between Jet 1.1 and Jet 2.0-3.0. This difference is in how it processes the `Data Has Changed` error message. In Jet 1.1, the error was triggered by default to warn users that data had changed in their tables. In Jet 2.0-3.0, this error does not get triggered. To cause the database engine to trigger the error message, you need to set the **Options** property for the **Recordset** to **dbSeeChanges** (512).

To change the **Options** property on a **Data** control to the proper value there are two methods you can use:

- In the design environment, make sure the **Data** control is the selected item on your form and go to the Properties window. Set the **Options** property to 512.
- In the run-time environment, use the following line of code to set the **Options** property of a **Data** control named `data1`.

```
data1.Options = dbSeeChanges
```

To create a **Recordset** with the **Options** property set to **dbSeeChanges**, you can use the following code segment:

```
Dim db As Database
Dim rs As Recordset

Set db = DBEngine.WorkSpaces(0).OpenDatabase("biblio.mdb")
Set rs = db.OpenRecordset("authors", dbOpenDynaset, dbSeeChanges)
```

After the **Options** property of the **Recordset** is set, you can trigger the error after the following sequence of events.

1. Two programs have the same record open in an edit mode.
2. Both programs make changes to the record.
3. One program saves its changes.
4. When the second program attempts to save its changes, the error message is triggered.

Asc Function (ReadMe)

In addition to the **Asc** function and the **AscB** function described in the **Asc** Function Help topic, the **AscW** function is provided for use with 32-bit Windows operating systems. In 32-bit Windows, **AscW** accepts a character and returns the corresponding character code that is native to the operating system. In 16-bit Windows, **AscW** behaves exactly the same as the **Asc** function; in 32-bit Windows, it returns a Unicode character code.

Chr Function (ReadMe)

In addition to the **Chr** function and the **ChrB** function described in the **Chr** Function Help topic, the **ChrW** function is provided for use with 32-bit Windows operating systems. In 32-bit Windows, **ChrW** accepts a character code and returns the corresponding character that is native to the operating system. In 16-bit Windows, **ChrW** behaves exactly the same as the **Chr** function; in 32-bit Windows, it returns a Unicode character.

Len and LenB Functions Return with User-Defined Types (ReadMe)

The **Len** function returns the size of a user-defined type as it will be when written to a file. By contrast, **LenB** returns the in-memory size of a user-defined type, including any padding between elements.

InStr Function (ReadMe)

The **InStr** function does not use named arguments as shown in the syntax picture of the **Instr** function Help topic. It uses conventional, position-dependent arguments in the order shown in the syntax picture.

Unicode (ReadMe)

Unicode is a character standard developed by the International Standards Organization (ISO) to compensate for the 256-character limitation of the ANSI and Extended ASCII character standards. While the ANSI and Extended ASCII standards both use 8-bit (1 byte) character codes that limit the number of unique characters to 256, Unicode uses a 16-bit (2-byte) coding scheme that allows for 65,536 distinct character spaces, with approximately 34,000 currently mapped characters. This accommodates all the characters and alphabets used in the world today, including several antiquated languages such as Sanskrit and Egyptian hieroglyphs. Unicode also includes representations for punctuation marks, mathematical symbols, and dingbats, with substantial room for future expansion. The complete standard is documented in *The Unicode Standard, Worldwide Character Encoding* by the Unicode Consortium Staff (2 vols., Addison-Wesley).

Using Get and Put with Arrays (ReadMe)

Earlier versions of Visual Basic did not permit the use of arrays in **Get** and **Put** statements. This limitation has been removed. If a variable is permitted in a **Get** or **Put** statement, an array of that type is also permitted.

Illegal Names in Classes (ReadMe)

Using certain names in Visual Basic is illegal because they conflict with OLE symbols. These names include:

- QueryInterface
- AddRef
- Release
- GetTypeInfoCount
- GetTypeInfo
- GetIDsOfNames
- Invoke

For example, using these names as the name of a procedure causes the following error: `Member already exists in this form`.

Error Message Changes and Additions (ReadMe)

The following error messages have been changed:

User-defined types, private classes, and form modules not allowed as the type of a public member of a public class

This error message now reads:

User-defined types and fixed-length strings not allowed as the type of a public member of a class or form; private classes and form modules not allowed as the type of a public member of a public class.

Bad DLL calling convention (Error 49)

In 32-bit Windows, this error occurs if you use any calling convention other than StdCall.

Bad File Name or Number (Error 52)

The information is correct, but applies only to MS-DOS-based systems (Microsoft Windows versions through 3.11 and Windows for Workgroups). File specification rules in Windows 95 and Windows NT are much more liberal. For example, a filename can be 255 characters and can include almost any characters, including blank spaces. Some restrictions are still imposed by the operating system. For example, in Windows 95 this error is caused by specifying a filename beginning with a period, double slash or double backslash. See you system documentation for more details on file naming.

The following error messages have been added:

An unrecognized error occurred during compilation

An error code was returned, but no detailed information is available for the error.

Functionality not supported in DLL

Some Visual Basic for applications statements and functions cannot appear in a dynamic-link library (DLL). For example, a **Stop** statement cannot be used in a DLL.

This edit requires a Reset

Generally, Visual Basic for applications permits you to edit suspended code, then continue running the code. But some edits, such as changes to declarations of static variables, preclude continuation of code execution. In these cases, Visual Basic for applications must reset all variables and begin code execution from the beginning.

Useful Constants for Chr$ Function in Visual Basic for applications Type Library (ReadMe)

The VBA type library now contains constants for the following:

- Carriage-return/linefeed (**vbCrLf** = Chr$(13)+Chr$(10))
- Null character (**vbNullChar** = Chr$(0))
- Carriage return (**vbCr** = Chr$(13))
- Linefeed (**vbLf** = Chr$(10))
- Backspace (**vbBack** = Chr$(8))
- Tab (**vbTab** = Chr$(9))
- Vertical tab (**vbVerticalTab** = Chr$(11))
- Form feed (**vbFormFeed** = Chr$(12)

You can use any of these constants in code rather than making the equivalent **Chr$** function call. However, only those for the carriage return, linefeed, tab and backspace are meaningful in Microsoft Windows.

Microsoft Access Version Number (ReadMe)

Occasionally Microsoft Access is referred to as Version 3.0. This should be Version 7.0, which is also known as Microsoft Access for Windows 95.

ReDim Statement as a Declaration (ReadMe)

The **ReDim** statement acts as a declarative statement if the variable it declares does not exist at module- or procedure-level. If another variable with the same name is created later, even if in a wider scope, **ReDim** will refer to the later variable and will not necessarily cause a compilation error, even if **Option Explicit** is in effect. To avoid such conflicts, **ReDim** should not be used as a declarative statement, but simply for redimensioning arrays.

Coercion of Byte and String Types (ReadMe)

You can assign strings to resizable arrays of bytes. An array of bytes can also be assigned to a variable-length string. This coercion also occurs in passing arguments to **ByVal** parameters. Be careful with this however, because it can be expensive because of the necessity to create temporary variables and arrays during the call. Similarly, be aware that the number of bytes in a string varies among platforms. On Unicode platforms the same string contains twice as many bytes as it does on a non-Unicode platform.

FileAttr Function on 32-bit Operating Systems (ReadMe)

On 32-bit operating systems, **FileAttr** can return the file mode, but causes an error if you specify the operating system file handle as the return. Therefore, the second argument can only be 1 with 32-bit operating systems.

Additional Information on VBA Registry Functions (ReadMe)

Each of these functions (**DeleteSetting**, **GetSetting**, **GetAllSettings**, **SaveSetting**) takes some combination of an *appName*, *section*, *key*, and *value*. Using these names as an example, on 16-bit Windows, the file *appName*.INI would exist in the Windows directory with the entries:

```
[section]
key=value
```

On 32-bit Windows, the registry entry HKEY_CURRENT_USER\Software\VB and VBA Program Settings\appName\section would contain an entry: "key: REG_SZ: value".

Limitations

appName on Win16 must be a file pathname (<255 bytes, MAX_PATH). You may leave off the path, and the Windows directory is the default. Also, the ".INI" extension may be left off; it is assumed if not present.

"Software\VB and VBA Program Settings\" + *appName* + *section* must be less than 260 characters on Win32 (260 = MAX_PATH).

On Win16, the entire .INI file is cached by the operating system in a 64K segment, so the file cannot be bigger than 64K.

On Win16, *section*, *key* and *value* are limited to 64K strings. However, in practice having a section or key over 32K has caused variable behavior, so it is recommended to keep them less than 32K. You may have problems replacing a value where the length of the old value + length of the new value is greater than 64K.

On Win32, *key* and *value* have no known size issues.

On Win32, entries of types other than "REG_SZ" are not recognized.

On Win16 DBCS, there are no known problems with DBCS characters in .INI files, although Win16 documentation specifically recommends against this. A user may not be able to read an .INI file containing DBCS characters using "type" at the MS-DOS prompt. Also note that Win16 length limits are bytes limits; DBCS characters will consume two bytes. (Win32 limits are characters limits).

On Windows 95, strings are converted from ANSI to UNICODE and vice versa. They are stored in ANSI by the operating system and manipulated in UNICODE by Visual Basic.

Unbound Mode of the DBGrid Control (ReadMe)

In the Add and Write events, the **RowBuffer** that is passed in is not fully populated, but contains entries only for those cells that were modified. This behavior is inherited from bound mode, which does not fetch every column unless it absolutely has to, and hence builds a sparse update list. Therefore, these event handlers need to test each row buffer entry for null support before updating the **Recordset**.

The Read event allows you to inform the **DBGrid** of BOF/EOF and error conditions by setting the **RowBuffer** object's **Count** property to 0. The Add and Write events support the same mechanism so that the **DBGrid** can gracefully handle data conversion and insufficient permission errors.

Similarly, you can fail the Delete event by setting **Bookmark** to **Null**, because this event does not use a **RowBuffer**.

Remote Data Objects and RemoteData Control Miscellaneous Information (ReadMe)

The following information includes tips, techniques, and suggestions that are not included in the product documentation.

- When using the **Bookmark** property, be sure to save bookmark values in variables declared as **Variant**, not as **String**.
- The **StillExecuting** property also applies to the **RemoteData** control.
- The **Version** property also applies to the **RemoteData** control.
- The **EditMode** property applies to the **rdoResultset** object, even though it is listed as only applying to the **RemoteData** control in Help.

Properties for the DBGrid Control (ReadMe)

The following properties do not appear in the list of properties in the **DBGrid** Control topic in online Help:

- **Row**
- **Col**
- **FirstRow**

Also, **TopRow** is incorrectly listed as a property of the **DBGrid** control.

Remote Automation Limitation on 16-bit Applications (ReadMe)

The maximum amount of data transferred across a single Remote Automation call to or from a 16-bit application must not exceed 64K bytes. If it does, an `Out of Memory` error will occur (E_OUTOFMEMORY = 0x80000002L), and the transfer will not be completed. This is a per call limit and therefore is based on the sum data transferred from a property read/write or the sum of all variables and results being transferred to or from a method invocation. This applies to 16-bit applications regardless of operating environment.

This limitation does not apply to 32-bit applications.

Jump Paths for Help Components in 16-bit Visual Basic (ReadMe)

These are the jump paths that are hard-coded in the Supplemental Help Files topic for 16-bit Visual Basic Help. If the (default) directories below are not used when installing the components, some of the jumps in Help may not work.

Component	Directory
Visual Basic	(VB root) VB.HLP
Biblio	(VB root)\SAMPLES\BIBLIO.HLP
Data Manager	(VB root) DATAMGR.HLP
Learning Microsoft Visual Basic	(VB root) VBCBT.HLP
Product Support Services	(VB root) PSS.HLP
Samples	(VB root)\SAMPLES\SAMPLES.HLP
Setup Wizard	(VB root)\SETUPKIT\SETUPWIZ.HLP
Custom Control Reference	(VB root) CTRLREF.HLP
Crystal Reports	(VB root)\REPORT\CRW.HLP
Hotspot Editor	(VB root)\HC\SHED.EXE
ODBC Installation	\WINDOWS\SYSTEM\ODBCINST.HLP
SQL Server ODBC Driver	\WINDOWS\SYSTEM\DRVSSRVR.HLP
VisData	(VB root)\SAMPLES\VISDATA.HLP
Enterprise	(VB root) ENTPRISE.HLP
Visual SourceSafe Administrator	\SS4\SSUSADM.HLP
Visual SourceSafe Explorer	\SS4\SSUSEXP.HLP

StatusBar Cannot Set Left Property of the Panel Object (ReadMe)

Add a **StatusBar** control to a form, and then add the following code to the Code window:

```
Private Sub Form_Load()
   StatusBar1.Panels.Item(1).Left = 100
End Sub
```

Run the application. Visual Basic generates run-time error 383 `Property is read-only`. This is contrary to the description in the Help topic "Left, Top Properties (Custom Controls)."

Creating a Recordset Against an ODBC Data Source (ReadMe)

When creating a **Recordset** against an ODBC data source with Microsoft Jet data access objects (DAO), you must move to the last record before any additional **Recordset** objects can be created. Using the **MoveLast** method against the **Recordset** fully populates the result set and frees the connection for additional operations. If you open an additional **Database** against the same data source, Jet attempts to share the first connection so you still cannot create additional **Recordset** objects until the first **Recordset** is fully populated.

SystemDB Property of the DBEngine Object (ReadMe)

Returns or sets the full path and filename of the System Database (.MDA) file.

Syntax

object.**SystemDB** [= *value*]

The **SystemDB** property syntax has these parts:

Part	Description
object	An object expression that evaluates to a **DBEngine** object.
value	A string expression that points to a system database file (typically named "SYSTEM.MDA"). This is also known as a "workgroup file."

Remarks

Jet provides the ability for you to define a workgroup and give varying permissions to each object in the database to different users in the workgroup. The workgroup is defined by the workgroup file, typically called "SYSTEM.MDA".

For your users to gain access to the secured objects in your database, DAO needs the location of the workgroup file that specifies the database. This can be set either by specifying it in the Windows Registry or by using the **SystemDB** property.

This property is only available in the 32-bit version of Visual Basic.

MaxRows Property (Remote Data) (ReadMe)

When the SQL_MAX_ROWS ODBC statement option is set to a nonzero value, the maximum number of rows processed by Microsoft SQL Server is limited to *n* rows. This means that only *n* rows are returned by a query, or only *n* rows are inserted, updated, or deleted by an action query. SQL_MAX_ROWS is set indirectly by using the **rdoPreparedStatement** object's **MaxRows** property. If you share the **hStmt** created for an **rdoPreparedStatement** that has **MaxRows** set, the operations executed against the **hStmt** are also affected by the limitation imposed by SQL_MAX_ROWS on both the number of rows returned from a query and the number of rows processed in an action query.

In addition, if you reuse an **rdoPreparedStatement** that has **MaxRows** set, the number of rows affected by any update, delete, or insert action query will be limited to *n* rows.

Using a Forward-Only rdoResultset (ReadMe)

When using a forward-only **rdoResultset**, the you can reposition the current row only by using the **MoveNext** method. You cannot use the **MoveLast**, **MovePrevious**, **MoveFirst**, or **Move** method, or the **PercentPosition** or **AbsolutePosition** property, to reposition the current row pointer.

Error message 35604 - The first column in a ListView control must be left aligned (ReadMe)

When the **ListView** control's **View** property is set to 3 (Report), the left-most column (column 1) can only be left aligned. Any attempt to set the alignment to another value will result in error 35604 `The first column in a ListView control must be left aligned`.

This control is 32-bit only.

The topic does not exist. Contact your application vendor for an updated Help file. (ReadMe)

If you see this error message when you are working in Visual Basic, open Help and search for the appropriate topic. Browse through the contents or enter the appropriate words in the search dialog box, such as a programming keyword or the name of a dialog box. On Windows 95, you can also make a full-text search using the Find tab.

CreateDragImage Method Doesn't Display Text (ReadMe)

Contrary to what is written in the Help topic, the image being created is not composed of both the image and the text, but only of the image.

Phone Information for UK Developer Support (ReadMe)

The phone number is (01734) 271414

ByVal and ByRef Keywords (ReadMe)

If you don't specify either the **ByVal** or **ByRef** keyword when passing an argument, your value is sent by reference. The default is **ByRef**.

Crystal VBX loses DataSource (ReadMe)

When a project containing the Crystal data-bound .VBX control is converted to Visual Basic 4.0 format and the control is converted to the .OCX version of the control (either 16-bit or 32-bit), the **DataSource** property value is lost in the conversion. The result in this case is that the **DataSource** property is empty. You must manually set the **DataSource** back to the appropriate **Data** control.

Should Select Startup Module with Code Profiler (ReadMe)

When you load an application into the Code Profiler, you get a multiselect list of all the modules and forms. This way you can choose to profile selected modules. If you don't select the module that contains the startup point for the application, the Code Profiler will never get started when you run the application.

Providing Parameters to Stored Procedures (ReadMe)

This topic applies to Remote Data Objects (RDO), not Microsoft Jet Data Access Objects (DAO).

When providing parameters to stored procedures when using ODBC-connected data sources, stored procedures can be written to accept parameters that are passed to the procedure when it is executed. These parameters can be passed positionally or by name. If you pass parameters positionally, or if you use the **rdoParameters** collection associated with an **rdoPreparedStatement**, you must provide parameters from left to right. That is, even though some parameters might have defaults, you must provide the first parameters in the positional list. If you want all parameters to take their default values, you should provide no parameters. For example, to pass parameters to a procedure that can accept three parameters you can use the following code:

```
My_sp  param1, param2
```

In this case, *param3* is missing, so it is set to its default value as determined by the stored procedure declaration. You cannot however, leave off *param1* and still pass *param2* because the ODBC driver does not accept comma placeholders for parameters.

This same rule applies when you pass parameters using **rdoParameter** objects. For example, if you provide `rdoParameters(1)` (the second parameter), you must also provide `rdoParameters(0)`.

You can provide parameters yourself by using named arguments. Because each parameter in the stored procedure declaration is named, you can use this name in the statement that executes the procedure. For example, to execute a procedure whose second parameter is "@Age", you can write the following code:

```
Execute My_sp  @Age = 10
```

In this case, the remaining parameters take on their default values. However, you cannot use this syntax with the **rdoParameters** collection.

Microsoft SQL Server Stored Procedures (ReadMe)

When working with Remote Data Objects (RDO), Data Access Objects (DAO), VBSQL, or the ODBC API and Microsoft SQL Server stored procedures, you can create procedures that contain one or more SELECT statements or one or more SELECT statements in combination with one or more UPDATE, INSERT or DELETE statements. If you submit these queries in batches without benefit of a stored procedure

- each statement returns a result set, and the non-SELECT statements return the number of rows affected. If, however, you create a stored procedure that contains UPDATE, DELETE, or INSERT statements, these statements do not return result sets when the stored procedure is executed. In this case you cannot determine how many rows are returned through conventional means
- as with the **RecordsAffected** or **RowsAffected** properties with DAO or RDO, respectively.

Just to be clear, however, you can execute batch SQL queries and run stored procedures that contain a mix of select and action statements using RDO. The action statements execute and if they fail, an error is returned to RDO which causes a trappable Visual Basic run-time error which you can trap using **On Error** syntax. However, the actual number of rows affected by the action statement is not available because this information is not returned to ODBC from the SQL Server.

If any portion of the query fails, including the action query, a trappable error is generated, so for most situations, you don't need to know the actual number of rows affected by the statement. However, if you need to know the actual number of rows affected, you can add:

```
Select @@ROWCOUNT
```

after any UPDATE, INSERT, or DELETE statements in your stored procedure. This produces a one-column, one-row result set containing the number of rows affected by the action statement.

Releasing Pointers with Visual Basic Add-Ins (ReadMe)

When using C++ to develop add-ins, every time you get a pointer from Visual Basic using OLE Automation, you must call "release" when you are done with it. While Visual Basic follows the standard OLE reference counting rules, it isn't always clear when a pointer needs to be released. For example, when you get a pointer out of Visual Basic by calling _NewEnum, you must call release through that pointer. Also, when you use:

```
pDispatch = m_ourMenuItems.Item(pszCaption);
```

you must also use:

```
pDispatch->Release();
```

OCX Files Installed When Custom Control Not Selected for Installation (ReadMe)

If you select Custom and remove the check mark for custom controls when installing Visual Basic, a few .OCX files are still installed with the main Visual Basic setup option. If you don't want these files loaded when you start Visual Basic, you'll need to remove the .OCX files from AUTO16LD.VBP and/or AUTO32LD.VBP.

Questions and Answers About Microsoft Visual Basic for Windows Version 4.0 (ReadMe)

Will the 16-bit and 32-bit versions of Visual Basic 4.0 run on Windows 95?

The 16-bit version of Visual Basic for Windows version 4.0 will run on Windows, Windows for Workgroups, Windows NT, and Windows 95. Under Windows NT and Windows 95, it is handled as any other 16-bit application. The 32-bit version will run only on Windows 95 and Windows NT version 3.51 or greater. On Windows NT, your application will run in a 16-bit environment provided by the Windows On Windows (WOW) layer, which allows Windows NT to run 16-bit applications in a protected environment.

Can I load a VBX control in the 32-bit version of Visual Basic?

VBX custom controls are limited to 16-bit and are fully supported only by Visual Basic 3.0 and the 16-bit version of Visual Basic 4.0. Visual Basic 4.0 uses the new OLE control model as its main control architecture. Visual Basic has moved to this new, open control model to support controls in both 16- and 32-bit environments. The OLE Control architecture merges all of the benefits of the VBX custom control with OLE. The new OLE Controls can be used in any OLE client application and are available on 16-bit as well as 32-bit platforms.

Visual Basic 4.0 includes OLE controls (.OCX) that are upgrades to the VBX custom controls that shipped in previous versions. Automatic conversion is provided to replace the VBX references in projects with references to OLE controls. Visual Basic 4.0 will ship with OLE equivalents of all of the controls shipped in Visual Basic 3.0, in addition to several brand new OLE controls in both 16- and 32-bit versions. Visual Basic will continue to support VBXs in the 16-bit version only.

How can I write an application for 16-bit and 32-bit systems at the same time?

Microsoft has taken a number of steps to assure source code compatibility between 16-bit and 32-bit applications:

- All Visual Basic-provided language is portable.
- All custom controls in the product have identical object models.
- Basic file I/O enables files to be shared by applications running on different platforms.

Visual Basic 4.0, Professional and Enterprise Editions include both 16-bit and 32-bit versions of VB.EXE. Using a single source code tree, you can create 16-bit applications that run on Microsoft Windows 3.X, 32-bit applications that run on Microsoft Windows NT 3.X (Intel), and 32-bit Windows 95-based applications. By taking advantage of the new conditional compilation switches in Visual Basic 4.0, you can quickly recompile the same source code to target and exploit the capabilities of different Windows platforms. Conditional compilation switches allow you to easily target different platforms or languages using a single source code tree and a simple recompile.

Visual Basic provides both the language and command-line support for conditionally compiling declarations and procedural code into an application. Both the Project Options dialog and the Visual Basic 4.0 command line allow the setting of constants that are subsequently used in evaluating **#If...#Else...#End If** structures.

```
#If Win16 Then ' Use Win16 calls.
   Declare Function GetWindow Lib "User" (ByVal hWnd As _
       Integer, ByVal wCmd As Integer) As Integer
   #Const ANSI=True
#Else       ' Use Win32 calls.
   Declare Function GetWindow Lib "User32" (ByVal hWnd As _
       Long, ByVal wCmd As Integer) As Long
   #Const ANSI=False4
#End If
```

Can a Visual Basic 4.0 32-bit application run on Win32s?

No. Win32s is a subsystem of DLLs which extends the Windows 3.1 16-bit operating system by translating 32-bit calls to the underlying 16-bit operating system. Programs

written with the 32-bit version of Visual Basic running on Windows 3.1 with Win32s would generally run slower (due to the extra memory and overhead of the translation layer) than the same program created with the 16-bit version of Visual Basic. Instead of using the more limited Win32s features, Microsoft chose to have the 32-bit version exploit the more advanced 32-bit features available only on Windows 95 and Windows NT.

Can a developer create DLL files with Visual Basic 4.0?

Visual Basic version 4.0 includes an option to make OLE DLL files. These files behave similarly to standard DLLs except that they use OLE as an interface to the objects described by the DLL. With Visual Basic 4.0, you can create reusable business objects that can be shared across applications. Rather than having to create DLLs using a C compiler or other language, you can now easily create reusable objects entirely from within Visual Basic.

These OLE components accomplish the reusability of DLLs and are easier to reuse because they are essentially self-documenting.

At design time you can browse the components with the Object Browser and examine what methods and properties the OLE objects expose. Similar to the way in which VBX vendors encapsulated their expertise into controls, Visual Basic programmers can now rapidly package their business rules, commonly used code libraries, as well as any legacy code into reusable, programmable objects. Once created, these libraries of reusable objects, complete with custom properties and methods, can be used by any application capable that supports OLE Automation.

A few examples of applications that can reuse these objects are Visual Basic 4.0 (which can both use and create OLE objects), Microsoft Excel 5.0, Microsoft Access 2.0, and Microsoft Project.

Future versions of Microsoft SQL Server, Microsoft FoxPro, Microsoft Word For Windows, and Windows will also take advantage of these objects.

Does this mean that I can use a Visual Basic created DLL from other languages?

Yes, you can use the OLE DLLs from other languages (such as Microsoft Visual C++), but you do not declare the functions in the DLL like you would a Windows DLL function. Instead you access the functions via the OLE interface.

Will Visual Basic 4.0 exist for MS-DOS and Macintosh platforms?

Visual Basic 4.0 incorporates the portable Visual Basic, Applications Edition language engine. Microsoft Excel 5.0 was recently released for the Macintosh and along with it the Visual Basic for applications component of Microsoft Excel, so the language engine is there. There are no commitments at this time for a version of Visual Basic for the Macintosh or for MS-DOS, but consideration is one based strictly on a good business case.

Does a specific version of 32-bit Visual Basic exist for Windows NT on MIPS, ALPHA, PPC, platforms?

Preliminary discussions have occurred concerning all other major architectures that support Windows NT, and although no commitments have been made, this is certainly a possibility for the future.

The 16-bit version of Visual Basic 4.0 will run on any machine which can run Windows NT. Windows NT includes a Windows On Windows (WOW) emulation layer which can run 16-bit applications.

Does the 32-bit version of Visual Basic provide statements and functions for multitasking?

Currently Visual Basic 4.0 does not support threads on Windows NT 3.5 or Windows 95. Research indicates that the main reasons Visual Basic programmers would want to use threads in a Visual Basic application would be for asynchronous database queries, printing, or file I/O operations. Microsoft is expecting providers of data sourcing OLE controls to provide the former, and will consider the latter for future versions.

Does Visual Basic 4.0 use the new controls of Windows 95?

Visual Basic does use the new Windows 95 controls, but because these are 32-bit controls for a 32-bit operating system, they are only available in the 32-bit version of Visual Basic.

Do you know if the third party vendors will be selling OLE Controls soon?

There a large number of control developers who are migrating existing VBX controls to the OLE control model as well as those who are writing new controls. There should be a large number of controls available immediately after Visual Basic releases.

Do I need to change my Visual Basic 3.0 applications to compile with the 32-bit version of Visual Basic 4.0?

There are two significant changes that need to be addressed to re-compile a Visual Basic 3.0 application to Visual Basic 4.0 32-bit.

First, if API calls are being used by the application, you must change your API calls to the appropriate target environment. Win16 and Win32 provide APIs with similar functions but different **Declare** statements (to take into account the size of the data types). Visual Basic 3.0 typically uses the 16-bit declarations. The 32-bit version of Visual Basic 4.0 must use the 32-bit API declarations. Rather than simply replacing the existing API calls, you can use Visual Basic 4.0's conditional compilation feature to target both 16- and 32-bit platforms from a single source code tree.

Second, the 32-bit version of Visual Basic 4.0 makes use of the newer OLE Control architecture exclusively. VBXs are not supported for the 32-bit version. This is not a problem for VBX controls shipped with previous versions of Visual Basic. All of these controls have OLE Control counterparts (in Visual Basic 4.0, these files all end in .OCX). You will be prompted to upgrade these controls the first time you load your project into Visual Basic 4.0. However, you will need to acquire OLE Control upgrades to any third-party controls used in your applications. While Microsoft has made the technology available for control developers to easily port their existing VBX controls to the new OLE Control technology, there is no guarantee that all of the available VBX controls will be available as OLE Controls. Contact the control vendor for upgrade information.

The 'Basic Language' code of both the 16-bit and 32-bit versions of Visual Basic are fully compatible. Creating a 32-bit application that runs on either Windows NT 3.5 or Windows 95 from a Visual Basic 3.0 (or 16-Bit Visual Basic 4.0) application is a simple matter of loading an existing 16-bit Visual Basic application into the 32-bit version of Visual Basic and choosing the Make EXE File command from the File menu.

Do I need to change my Visual Basic 3.0 application to compile with the 16-bit version of Visual Basic?

No. To run a Visual Basic 3.0 application with the 16-bit version of Visual Basic, load the source code into the 16-bit version of Visual Basic 4.0 running under Microsoft Windows 3.x, Microsoft Windows 3.x, Windows NT 3.51, or Windows 95, and make an .EXE file.

Does Visual Basic 4.0 support DBCS?

All versions of Visual Basic 4.0 are Double-Byte Character Set (DBCS) enabled, in that all dialogs accept DBCS characters, all edit controls can accept DBCS text, and strings specified in code can contain DBCS characters.

In what languages is Visual Basic 4.0 available?

Visual Basic for Windows version 4.0 is slated for release in French, German, Italian, Spanish, Chinese, and Japanese.

What is the difference between Standard and Professional versions ?

Feature Highlights	Standard	Professional and Enterprise
Visual development environment for fast assembly through drag-and-drop of pre-built components and controls	Yes	Yes
New extensible design environment lets you add third party products such as source code control and CASE tools as well as wizards you create yourself in Visual Basic (Professional and Enterprise only) to further customize your design environment	Yes	Yes

Support for the widest use of pre-built component types for rapid development: OLE controls, VBX controls (16-bit only),OLE objects, and DLLs	Yes	Yes
Advanced structured programming language common across many Microsoft applications. Now includes object-oriented constructs **For Each** and **With**	Yes	Yes
Color-coding editor and incremental compiler with syntax checking "on-the-fly" and new line continuation character	Yes	Yes
Full-featured debugging with watch variables, breakpoints, and procedure stack	Yes	Yes
Rich set of standard controls: **Grid**, **Label**, **Frame**, **CheckBox**, **ComboBox**, **CommandButton**, **DirListBox**, **Line**, **Shape**, **DriveListBox**, **FileListBox**, **OptionButton**, **PictureBox**, **HScrollBar**, **VScrollBar**, **TextBox**, **Timer**, and **CommonDialog**	Yes	Yes
Conditional Compilation for advanced debugging or targeting of both 16- and 32-bit operating systems from a single source code tree	Yes	Yes
Support for Windows Resource files provide no-recompile customization	Yes	Yes
Professional controls: six more 3-D Interface Controls, Animated Button, Gauge, Graph, Key State, Bitmap Clipping, Messaging API (2 MAPI controls), Multimedia MCI, Communications, Outline, Masked Edit, and Spin Button	No	Yes
Support for Windows 3.X, Windows NT 3.51 and Windows 95	No	Yes
Enhanced Jet database engine (includes query optimizer, transaction processing, multiuser support, cascading updates and deletes, optimistic and pessimistic locking, and more)	Yes	Yes
Data exchange with the Microsoft Access, Visual FoxPro, dBASE, Paradox, and Btrieve (16-bit only) databases as well as Microsoft Excel and Text formats	Yes	Yes
Data Manager, for creating and manipulating databases, tables, and indexes	Yes	Yes
Data control, to access databases without writing code, can now supply dynasets, tables, and snapshots	Yes	Yes
Many data-aware controls, for building database front ends without writing code: **TextBox**, **CheckBox**, **Label**, **PictureBox**, **Image**, **DBGrid**, **DBList**, **DBCombo**, Masked Edit, 3D Panel, and 3D Check Box	Yes	Yes
Support for data sourcing OLE Controls for direct access and control binding to remote data	Yes	Yes
Full ODBC 2.0 support, including scrollable cursors	No	Yes
Microsoft SQL Server driver	No	Yes
Full programmatic data-access layer for fine-tuned control of the integrated Jet database engine through data access objects	No	Yes
OLE container control for using OLE components in your application dynamically at run time	Yes	Yes
OLE Automation support for controlling OLE components	Yes	Yes

OLE Object Browser provides fast navigation of OLE objects and their syntax as well as providing direct cut-and-paste capability into Code window	Yes	Yes
OLE Compound Document objects on the Toolbox for quick drag-and-drop access to pre-built components such as the Microsoft Excel charting object with full in-place editing and toolbar and menu negotiation	Yes	Yes
OLE Automation server creation allows the building of reusable libraries of your own objects that can be used by any tool with OLE Automation support (Excel, Project, Access, Visual C++)	No	Yes
Context sensitive Help puts the information you need at your finger tips	Yes	Yes
Improved Setup Toolkit and SetupWizard for creating custom setup programs and automating the process of distributing applications	Yes	Yes
Sample projects and hundreds of usable code clips in online Help	Yes	Yes
Icon Library with over 450 icons to enhance your applications	Yes	Yes
New Crystal Reports for Visual Basic 4.0, including improved database engine integration, fine-tuning during print preview, integrated e-mail support for report distribution, two-pass reporting, grouping, mailing labels, and drag and drop■all with no run-time fee	No	Yes
Crystal Reports Control allows easy report runs from an application	No	Yes
Online Windows 3.1 API reference (Win32 API reference on the CD-ROM)	No	Yes
Over 250 assorted bitmaps and metafiles for use in your applications	No	Yes
Help Compiler for authoring Help files for Windows-based applications	No	Yes

What does the Enterprise Edition add to this?

The Enterprise Edition contains all of the features listed above for the Professional Edition. In addition, the Enterprise Edition adds the following features.

Remote Data Objects (RDO) and **RemoteData** control (RDC)

Visual Basic 4.0, Enterprise Edition is bringing an entirely new paradigm to front-end developers. RDO has a set of data access objects not unlike Jet but tuned and optimized specifically for SQL Server. It will also run against Oracle. This means that it can gracefully and efficiently deal with server-side cursors, asynchronous queries, multiple result sets, input, output, and return value parameters from stored procedures and much, much more ■ but using the same programming paradigm that Jet uses. RDO can create a "cursorless" result set and three other types of cursors (static, dynamic, and keyset). It promises to become the new standard for accessing SQL Server from Visual Basic.

The **RemoteData** control is a replacement for the Jet **Data** control. It can connect bound controls to a selected ODBC data source. Neither the **RemoteData** control nor RDO require any vestige of Jet, which gives it a much smaller footprint (about 250K).

Visual SourceSafe

The Enterprise Edition of Visual Basic includes Visual SourceSafe, a fully integrated source code control system. It is an easy-to-use tool for team software development. Visual SourceSafe tracks changes to files and stores the changes so that files, such as code

modules, can be easily and economically reused.

Remote Automation

OLE Automation servers are an ideal mechanism for providing business, data, and application services. Once a service has been implemented as an OLE server, it can be used as an application component by developers throughout the enterprise. Such services can be implemented very rapidly using Visual Basic. Remote Automation provides the infrastructure for easily extending the OLE client/server relationship across networks. It allows an application's OLE client components to access services provided by OLE servers anywhere on the network.

What is new in Visual Basic 4.0?

Visual Basic, Applications Edition

Visual Basic, Applications Edition, version 2.0 (VBA) is added as the language engine in Visual Basic. This version of VBA is fully backward-compatible with earlier versions of the stand-alone Visual Basic product, and with Visual Basic, Applications Edition, version 1.0, which is included in Microsoft Excel version 5.0 and Microsoft Project version 4.0. The insertion of VBA in Visual Basic 4.0 makes it easier to program OLE Automation objects, and means that Visual Basic code can be easily moved between modules in the applications that support OLE Automation.

Creating Custom Objects and Collections

You can create custom objects, with their own properties and methods, and assemble them into an object model. The definitions of these objects are called classes, and are contained in Visual Basic's new class modules. Your object model can include custom collections, built using Visual Basic's new **Collection** object.

OLE Automation Servers

Using OLE Automation in Visual Basic, you can borrow the functionality of other applications by controlling their objects from within your Visual Basic application. Visual Basic 4.0 also gives you the capability of creating your own OLE Automation servers. You can use Visual Basic 4.0, Professional Edition to create applications which expose objects with an interface of your own design, and are callable from applications supporting OLE Automation, including Visual Basic, Visual C++, Microsoft Project, Microsoft Access, and Microsoft Excel.

Property Procedures

Property procedures allow you to add custom properties to form modules, standard modules, and class modules, and to execute code when the property is set or retrieved. For example, you could add an Inverted property to a form. When Inverted is set to True, the code in the associated property procedure invokes an API to invert a bitmap on the form.

32-Bit Support and Conditional Compilation

The 32-bit version of Visual Basic is designed to let you create versions of your programs to run on 32-bit systems, using the same source code as for 16-bit versions. The 32-bit version of Visual Basic supports long filenames in all project components and relaxation of most of the 64K capacity constraints for properties and the Visual Basic language. With conditional compilation, you can embed platform-specific code segments in **#If...Then** statements, and selectively build versions of your application for different platforms.

Jet 3.0

Many additional features of the Jet 3.0 database engine are available in Visual Basic 4.0, Professional Edition, including the ability to create new databases, change database structure, and the ability to manipulate database security and referential integrity.

In addition, Visual Basic 4.0 adds the following features:

- A 32-bit development tool for creating 32-bit applications
- Migration to OLE controls
- OLE insertable objects in the Toolbox
- Open IDE extensibility
- Source code management (SourceSafe)

- Improved Crystal report writer
- New data-aware controls
- Line continuation character, a much requested feature
- Commands to allow use of Resource files

How do I apply for a Microsoft beta program?

You can request to become a beta site by writing to:

Microsoft Corporation
Attn: XXXX Beta Test Administrator
One Microsoft Way
Redmond, WA 98052-6399

where XXXX is the product you want to apply for. Various products within Microsoft have varying capacity to respond to those who are not selected. Unfortunately, due to the volume of requests, the Visual Basic group is not able to inform those who were not selected for the beta.

Where can I find out more about Visual Basic certification?

For information regarding the Microsoft Certified Professional program, the developer certification, or the Visual Basic or Microsoft Access exams, please refer to the Microsoft Education & Certification Roadmap. This can be downloaded from CompuServe (GO MSEDCERT, Library #5, filename E&CMAP.ZIP) or can be ordered direct from Microsoft by calling 800-636-7544.

How do I send suggestions for product features/improvements to Microsoft?

Contact the Microsoft Wish Line at (206) 936-WISH [936-9474]. If it takes more than two minutes to describe, you can do one of the following:

- Fax it to us at 206-936-7329
- Write to us at:

Attn: Microsoft Wish
One Microsoft Way
Redmond WA, 98052

How can I find out more about calling the Windows API?

The Windows SDK Help file discusses Windows API general topics, functions, structures and messages. Its companion Help file, Windows 3.1 API Help, offers Help on the Visual Basic Declare Statements, Type Declarations and Global Constants used to access much of the Windows API. In addition, there are the following resources:

Chapter 26, "Calling Procedures in DLLs," and Chapter 28, "Programming for 16-Bit Systems," of the *Programmer's Guide* are good places to start looking at DLL and Visual Basic issues.

Article Q106553 (How to write C DLL's and call them from Visual Basic) continues this discussion and comments on some memory management issues.

Article Q110219 (LONG: How to call windows API from Visual Basic - General Guidelines) may prove a useful reference. Visual Basic and C data types do not always translate 1:1, and looking at how an API call handles a given data type may give you the type declaration you will need for your DLL's.

Article Q109290 (Popular Windows API Functions Used from Visual Basic 3.0) discusses the more commonly used API calls by Visual Basic programmers.

How do I create an OLE Control?

Visual C++ 2.0 comes with the OLE Control Development Kit, which will enable you to create an OLE control.

Article Q113895 (Intro to Microsoft OLE Custom Control Architecture & Tools) discusses OLE controls and their future within Visual Basic and Windows.

Where can I place an order or get upgrade and pricing information about Microsoft Visual Basic for Windows?

For information regarding product updates, prices, and sales, please call the Microsoft Sales

Information Center (MSIC) at the following number. Note that no technical support is provided on this line.

- In the United States, call 800-426-9400.
- In the United Kingdom, call 0734-270000.

Where can I get the latest updates for Visual Basic files?

You can identify available updates by searching for the pointer article in the Knowledge Base (described elsewhere in this document). Search on the keyword UPD or SOFTLIB. To get the latest release of Visual Basic updated files, download the appropriate file (the updates are all stored as self-extracting files *.EXE) from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for <filename>.EXE
  Display results and download
- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download <filename>.EXE
- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get <filename>.EXE

What can you tell me about the next version of ...

Stop! The answer is, Microsoft can tell you nothing. It is standard Microsoft policy to not discuss unannounced products. Individuals involved in Beta testing Microsoft products are bound by a similar non-disclosure agreement which requires that they not discuss the product.

What do I do if I have a problem with Visual Basic?

You have a number of options for assistance from Microsoft as well as other developers. They include: Telephone support, CompuServe, Internet, and Microsoft Solutions Providers. For details, you can look in Visual Basic 4.0 online Help. Choose Obtaining Technical Support from the Visual Basic Help menu for more information. You can also download Knowledge Base article Q108102 which details the same support options and article Q80850 for support phone numbers.

What should I do before I ask for help?

Read the manuals. The Visual Basic manuals cover every keyword and most common programming situations. This information is also available in the online Help files.

Look in the Knowledge Base. The Microsoft Developer Knowledge Base (GO MDKB) is a tremendous resource for dealing with Microsoft products. Developer Support Engineers at Microsoft create solutions and explain problems or techniques that come up in the course of using Microsoft products. These discussions are written up into articles and placed into the Knowledge Base (KB) which is periodically updated on CompuServe. The Visual Basic Knowledge Base (VBKB_FT.EXE) is a downloadable subset of the Knowledge Base which contains articles relevant to Visual Basic. It contains a full text search engine to help you locate the information you want. This file is available in the Microsoft Software Library (GO MSL). CAUTION: This file is over four megabytes in compressed form; it will take a while to download.

Look at the FAQ list. FAQ: A Frequently Asked Questions list is currently available on the Internet (not an MS document) and Microsoft will have a FAQ on CompuServe in the near future.

Isolate the problem. Isolating the problem often leads to the solution. Also, if you can't accurately describe how and when the problem occurs, then any support engineer would have to take you back through your code to attempt to isolate the problem.

What are some things I can do to isolate the problem?

First and foremost, make a backup. This is the point at which even experienced programmers frequently lose many hours of work. When you experiment, it is far too easy to accidentally overwrite or delete necessary sections of code.

Use the debugging facilities built in to Visual Basic 4.0. See Chapter 20, "Debugging," in the *Programmer's Guide*. Attempt to identify the line or lines of code generating the error. Isolate the code. If you can isolate the problem to one block of code, try to reproduce the same problem with this block of code separated from the rest of your program. Select the code, copy it, start a new project, paste the code into the new project, run the new project, and see if the error still occurs.

Create a log file. If you cannot isolate the code or if the problem is erratic or if the problem only happens when compiled, then the debugging facility of Visual Basic will be less effective. In these situations you can create a log file which records the activity of your program. This will allow you to progressively isolate the location of the suspect code. Call the following procedure from various points in your program. You should pass in a string of text which indicates the current location of the code executing in your program.

```
Sub LogFile (Message As String)
   Dim LogFile As Integer
   LogFile = FreeFile
   Open "C:\VB\LogFile.Log" For Append As #LogFile
   Print #LogFile, Message
   Close #LogFile
End Sub

Sub Sub1 ()
   '...
   Call LogFile("Here I am in Sub1")
   '...
End Sub
```

Simplify the problem. If possible, remove any third party controls and custom controls from your project. Replace them with Visual Basic standard controls. Eliminate any code that does not seem to relate to the problem.

Reduce the search space. If you cannot resolve the problem with any of the above methods, then it is time to eliminate all other non-Visual Basic causes from the problem search space. Copy your AUTOEXEC.BAT and CONFIG.SYS files to backup files. Comment out any and all drivers and programs from these two files that are not absolutely essential to running your program under Windows. Change your Windows video driver to the standard Windows VGA driver. Shut down Windows and reboot your machine. This will eliminate the possibility that there is some other program or driver which is interfering with your program.

If you cannot locate a solution and are unable to isolate or resolve the problem with any of these methods, it's time to look for help. See the online Help under "Technical Support" or check into the other resources mentioned in this file.

How can I get Knowledge Base articles?

You can request individual articles to be faxed to you via the FastTips service. Microsoft FastTips is an automated fax and voice service that provides technical product support information at no charge through 800 numbers available 24 hours a day, 7 days a week (including holidays). You can use FastTips services with your touch-tone phone. Each of the specific FastTips services listed at the end of this article includes the following:

- A comprehensive map for easy navigation.
- A Microsoft FastTips Technical Library catalog, from which you can obtain current technical support information by ordering Microsoft Knowledge Base articles and Application Notes.
- Recorded questions and answers for the most frequently asked questions about Microsoft products. Faxes of these questions and answers are also available with the

FastTips services.

Microsoft Desktop Applications:	(800) 936-4100
Microsoft Personal Operating Systems:	(800) 936-4200
Microsoft Development Tools:	(800) 936-4300
Microsoft Advanced Systems:	(800) 936-4400

For best use of the FastTips services, request a map and catalog during your first call.

If you cannot determine which article is best for you, you can get assistance by calling the Product Support numbers. Engineers have access to the complete Microsoft Knowledge Base and can fax or mail the individual articles.

Can I get a copy of the entire Knowledge Base?

Due to its size, the Microsoft Knowledge Base is made available to the public by the individual product groups in separate pieces. The Visual Basic Knowledge Base (VBKB_FT.EXE) is a downloadable subset of the Knowledge Base which contains articles relevant to Visual Basic. It contains a full text search engine to help you locate the information you want.

There are over 600 categorized articles in the Visual Basic for Windows collection. The two Help files that hold these articles have been placed in a self-extracting file that you can download from several different places (listed later in this section). Choose to download either VBKB_FT.EXE (the full-text search version) or VBKB.EXE (the version without full-text search).

The Help files in VBKB_FT.EXE have an additional Find button that allows full-text search. The Help files in VBKB.EXE do not have the Find button and do not allow full-text search. The technical content in VBKB.EXE is identical to that in VBKB_FT.EXE. VBKB.EXE is 1.5 megabytes in size while VBKB_FT.EXE is approximately 4 megabytes. VBKB_FT.EXE is larger because it includes index and .DLL files needed for full-text search.

To obtain the Help files, download VBKB.EXE or VBKB_FT.EXE. Then run it in an empty directory to extract the files.

- Download VBKB_FT.EXE if you want to use full-text search to query the Microsoft Visual Basic Knowledge Base. The Help files in this package include a Find button that allows you to search the Microsoft Knowledge Base for any word you choose.
- Download VBKB.EXE if you want a smaller package and don't need full-text search. The Help files in this package have only the Search button, which allows you to search for article Q numbers (the number that identifies each Microsoft Knowledge Base article).

Where to Find VBKB.EXE and VBKB_FT.EXE

Download either VBKB.EXE or VBKB_FT.EXE (both are self-extracting files) from the Microsoft Software Library (MSL) on the following services:

- CompuServe
  GO MSL
  Search for and download VBKB.EXE
  -or-
  Search for and download VBKB_FT.EXE
- Microsoft Download Service (MSDL)
  Dial (206) 936-6735 to connect to MSDL
  Download VBKB.EXE or VBKB_FT.EXE
- Internet (anonymous FTP)
  ftp ftp.microsoft.com
  Change to the \softlib\mslfiles directory
  Get VBKB.EXE
  -or-
  Get VBKB_FT.EXE

After downloading either VBKB.EXE or VBKB_FT.EXE, run it to extract the files it contains.

What are some books that might be of use to a Visual Basic developer?

Title: PC Magazine: Visual Basic Programmer's Guide to the Windows API
Author: Appleman
ISBN: 1-56276-073-4
Publisher: Ziff-Davis Press

Title: Windows 3.1 Programming for Mere Mortals
Author: Woody Leonhard
ISBN: 0-201-60832-4
Publisher: Addison-Wesley

Title: Database Developer's Guide with Visual Basic 3
Author: Roger Jennings
ISBN: 0-672-30440-6
Publisher: Sams Publishing

Title: Running Visual Basic for Windows (2nd Ed.) [covers 3.0]
Author: Ross Nelson
ISBN: 1-55615-564-6
Publisher: Microsoft Press Div. of Microsoft Corp.

Title: Guide to the SQL Standard, A: A User's Guide to the Standard Relational Language SQL
Author: C.J. Date
ISBN: 201502097
Publisher: Addison-Wesley Publishing Co. Inc.

Title: Inside Windows 95
Author: Adrian King
ISBN: 1-55615-626-X
Publisher: Microsoft Press

Title: Programming Windows 3.1, Third Edition
Author: Charles Petzold
ISBN: 1-55615-395-3
Publisher: Microsoft Press

In addition, there is a reading list of books, periodicals, and other information in the Knowledge Base in article number Q118782.

When I load an existing application into Visual Basic 4.0, it says it's going to 'upgrade' my controls. Is it actually changing my controls and should I back them up?

No. Your controls are fine. Visual Basic 4.0 contains newer, backward-compatible controls that replace controls shipped with previous versions of Visual Basic. Your project's .VBP file will be updated to use these newer controls if you click Yes. It is recommended that you use the latest controls so that you can take advantage of any new features that have been implemented.

Why are the distributables so large?

Visual Basic 4.0 uses the OLE Control architecture on a scale unmatched by any other in the industry. This means that until the OLE Control architecture becomes the industry standard and part of the Windows operating system, Visual Basic programs have to include the support files which enable this powerful new technology. It is expected that as the OLE

model becomes integrated at the operating system level, this requirement will fall away and the size of the distributable will drop dramatically.

GetRows Example Incorrect Code (ReadMe)

The example provided with the **GetRows** method is incorrect. It uses the **CurrentDB()** function, which is not supported in Visual Basic 4.0. This function returns the "current" **Database** but is only applicable in Microsoft Access. The code also incorrectly codes the arguments to the **OpenRecordset** method. Instead of a "+" between the **dbOpenSnapshot** and **dbForwardOnly** arguments, you should code a ",". The example also uses the NorthWind example database as supplied with Microsoft Access. The corrected code is shown below:

```
Dim dbsCurrent As Database, rstTitle As Recordset
Dim avarRecords As Variant
Dim intFields As Integer, intRows As Integer

Set dbsCurrent = OpenDatabase("biblio.mdb")

Set rstTitle = dbsCurrent.OpenRecordset("Select * from titles", _
dbOpenSnapshot, dbForwardOnly)

avarRecords = rstTitle.GetRows(50)

intField = 0
intRecord = 0


   ' Use intField and intRecord as array indexes to enumerate
   ' the returned records, locate particular values, and so on.
```

Enterprise AutoLoad File Contains a Different Data Access Reference in 32-bit Versions (ReadMe)

The 32-bit Enterprise Edition AutoLoad File (AUTO32EN.VBP) contains a reference to Remote Data Objects (MSRDO32.DLL) instead of Data Access Objects (DAO3032.DLL).

DBGrid Control Doesn't Repaint Border When Moved (ReadMe)

When using the **Move** method of the **DBGrid** control, the grid will not repaint its border correctly. To solve this problem, issue a **Refresh** immediately after moving the grid. For example:

```
DBGrid1.Move 5,5
DBGrid1.Refresh   ' This will cause the grid to repaint itself.
```

NoDrop MousePointer is Nonfunctional in 16-bit Gauge and Key State Controls (ReadMe)

If you set the **MousePointer** property of the 16-bit **Gauge** or **Key State** controls to 12 - No Drop, then run the application and locate the mouse cursor so that it is over the control, you'll see that the shape of the cursor is still the default.

The solution is to set the **MousePointer** property to 99 - Custom and set the **MouseIcon** property to your own NoDrop icon or some other custom icon.

Component Manager Displays Hidden Coclasses and Interfaces (ReadMe)

This topic applies only to the Enterprise Edition of Visual Basic 4.0.

The Add OLE Components dialog box improperly lists hidden properties and methods of OLE servers.

Toolbar Control: Button Objects with Placeholder Style Don't Wrap (ReadMe)

If a **Button** object in a **Toolbar** control is the last button on the toolbar, and has the **Style** property set to **tbrPlaceholder** (4), the button does not wrap properly.

To avoid this, make sure a button with the **Style** property set to **tbrPlaceholder** (4) is not the last button on the toolbar. Instead place an invisible, disabled button with the **Style** property set to **tbrDefault** (0) at the end.

Note This may not always work, as after a resizing the Toolbar control, a button with the **Style** property set to **tbrPlaceholder** (4) could still become the last button, and thus not wrap properly.

Masked Edit Control: Mask Characters Incorrectly Listed in Custom Control Reference (ReadMe)

In the **Mask** property topic, the printed *Custom Control Reference* lists '0' and 'L' as mask characters. Actually, they are literals, not mask characters. Online Help contains the correct list of mask characters for the **Mask** Property. The **Mask** Property applies to the Masked Edit control.

Cols Property of the DBGrid Control (ReadMe)

The **VisibleCols** property topic incorrectly states that the **Cols** property applies to the **DBGrid** control. The number of columns in a **DBGrid** control is actually determined by the **Count** property of the **Columns** collection.

DBGrid columns are added or removed at run time by manipulating the **Columns** collection through use of the **Add** and **Remove** methods.

Help Button on Customize Toolbar Dialog is Inactive (ReadMe)

The Help button on the Toolbar's Customize Toolbar dialog box is inactive. To display this dialog box, place a **Toolbar** control on a form, set the **AllowCustomize** property of the **Toolbar** to **True**, run the application, and then double-click a blank portion of the **Toolbar** control. The Help button that appears on the dialog is inactive. That is, no Help topic is displayed when the button is clicked.

Grouped OptionButton Controls on the SSTab Control (ReadMe)

If you intend to put **OptionButton** controls on more than one tab of the **SSTab** control, be sure to place them on separate **Frame** controls. Otherwise, the **OptionButton** controls will act as if they are grouped together.

HitTest Method Example has Faulty Code (ReadMe)

The **HitTest** method example contains the following code:

```
Private Sub TreeView1_MouseMove(Button As Integer, Shift As Integer, _
x As Single, y As Single)

   If Button = vbLeftButton Then
      ' Set DropHighlight to the mouse's coordinates.
      Set TreeView1.DropHighlight = TreeView1.HitTest(x,y)
   End If
End Sub
```

This code will not work. A simple edit is necessary. Remove the following statements:

```
If Button = vbLeftButton Then
.
.
End If
```

This results in the code below, which runs correctly:

```
Private Sub TreeView1_MouseMove(Button As Integer, Shift As Integer, _
x As Single, y As Single)

   ' Set DropHighlight to the mouse's coordinates.
   Set TreeView1.DropHighlight = TreeView1.HitTest(x,y)
End Sub
```

Remote Procedure Call (RPC) Files (ReadMe)

Chapter 30, "Distributing Your Applications," of *the Programmer's Guide,* incorrectly states that when you use the SetupWizard to create the distribution media for your 16-bit application, and specify a .VBR file (Remote Support file) by using the Add OLE Servers button, the RPC files are not included for distribution.

In fact, when you specify a .VBR file by using the Add OLE Servers button, the RPC files are included for distribution. You are not required to install the RPC components using a separate RPC setup program.

Visigenic Oracle 32-bit Driver Installation Requirements (ReadMe)

The Oracle driver is installed automatically if the following three drivers are present on the user's machine:

ORANT71.DLL

SQLNTTT.DLL

CORENT23.DLL

If a user does not have these files present when Visual Basic is installed, install them and then rerun Visual Basic setup.

HeadForeColor Property Applies To the Column Object (ReadMe)

Contrary to what is written in the **HeadForeColor** property topic in online Help, it applies to the **Column** object, not the **DBGrid** control.

Picture Property is Read-only for ImageList ListImages (ReadMe)

The note at the bottom of the **Picture** Property (Custom Controls) online Help topic states you can assign the graphic returned by the **LoadPicture** function to the **Picture** property. This is incorrect, since the **ListImages** property is read-only at run time. If you try to assign to it a graphic returned by the **LoadPicture** function, you will get run-time error 383 (`Property is read-only`).

Miscellaneous Jet Issues (ReadMe)

The following list describes miscellaneous Jet issues.

- The Data Access Objects (DAO) **BeginTrans** method (on the **Database** or **Workspace** objects) allows at most five levels of nested transactions.
- Jet (Access) 2.0 databases containing more than 32 relationships cannot be converted to Jet 3.0 format and will result in the error `Couldn't create index; too many indexes already defined`. The solution is to remove enough indexes or relationships to allow the conversion to proceed. The maximum number of indexes is 32, and in Jet 3.0 both sides of relationships use an index.
- Multiple users attempting to access Paradox tables over a network may see the error: `Invalid file format`. This error is caused by not having registry entries for ParadoxNetPath that point to identical drive letters and net files on both users machines.
- For Access 2.0 databases with multiple sessions running on one machine (using optimistic concurrency), if two sessions try to update the same record, the second session will fail (correct behavior), but the error message will be `Write conflict error` which may be unclear.

Attempts to import extremely small values from text files (less than 2.225E-308) into IEEE double fields will not succeed and will return the error `Conversion failure`.

- If a user exceeds the server lock count while running a Windows 95 client against a Novell NetWare server, the users application will not complete properly. Workarounds include:
- Increase the number of locks on the server.
- Open the table as 'exclusive', which should prevent locking.

SaveToFile vs. SaveToOle1File (ReadMe)

The Remarks section of the **SaveToFile** method Help topic reads: "Use this method to save OLE objects. To save an OLE object, use the **SaveToOle1File** method."

The **SaveToOle1File** method Help topic reads: "Saves an object in the OLE file format." Then, in the Remarks section, it reads "If the object you want to save is an OLE object, use the **SaveToFile** method."

The **SaveToFile** method topic should read: "Use this method to save OLE objects. To save an OLE 1.0 object, use the **SaveToOle1File** method."

The **SaveToOle1File** method topic should read: "Saves an object in the OLE 1.0 file format. If the object you want to save is an OLE 2.0 object, use the **SaveToFile** method."

Using the RemoteData Control and the SQL Property (ReadMe)

The **RemoteData** control behaves like the Jet-driven **Data** control in most respects. The following guidelines illustrate a few differences when setting the **SQL** property.

The **RemoteData** control has no **RecordSource** property. You can treat the **RemoteData** control's **SQL** property like the **Data** control's **RecordSource** property except that it cannot accept the name of a table by itself unless you populate the **rdoTables** collection first. Generally, the **SQL** property specifies an SQL query. For example, instead of just "Authors", you would code "SELECT * FROM AUTHORS" which provides the same functionality.

The result set created by the **RemoteData** control might not be in the same order as the **Recordset** created by the **Data** control. For example, if the **Data** control's **RecordSource** property is set to "Authors" and the **RemoteData** control's **SQL** property is set to "SELECT * FROM AUTHORS", the first record returned by Jet to the **Data** control is based on the first available index on the Authors table. The **RemoteData** control, however, returns the first row returned by the remote database engine based on the physical sequence of the rows in the database, regardless of any indexes. In some cases the order of the records could be identical, but not always.

This difference in behavior can affect how bound controls handle the resultant rows -- especially multiple-row bound controls like the **DBGrid** control.

Shell Function Does Not Launch Documents with Associated Applications (ReadMe)

The **Shell** Function Help topic implies in the description of the 'pathname' argument that it will launch documents with their associated applications:

"pathname Name of the program to execute and any required arguments or command line switches; may include directory and drive. May also be the name of a document that has been associated with an executable program."

In fact, **Shell** does not have the capability of launching files based on their file association. In order to launch a document using its file association use the ShellExecute Windows API call. For example, the following code will launch the file TEST.DOC using the file association for .DOC files:

```
#If Win32 Then
Private Declare Function ShellExecute Lib _
   "shell32.dll" Alias "ShellExecuteA" _
   (ByVal hwnd As Long, _
   ByVal lpOperation As String, _
   ByVal lpFile As String, _
   ByVal lpParameters As String, _
   ByVal lpDirectory As String, _
   ByVal nShowCmd As Long) As Long

#Else
   Private Declare Function ShellExecute Lib _
   "shell.dll" Alias "ShellExecute" _
   (ByVal hwnd As Integer, _
   ByVal lpOperation As String, _
   ByVal lpFile As String, _
   ByVal lpParameters As String, _
   ByVal lpDirectory As String, _
   ByVal nShowCmd As Integer) As Integer
#End If

Private Const SW_SHOWNORMAL = 1

Private Sub Command1_Click()
   Dim iret As Long
   iret = ShellExecute(Me.hwnd, _
      vbNullString, _
      "c:\test.doc", _
      vbNullString, _
      "c:\", _
      SW_SHOWNORMAL)
End Sub
```

Also note that the **Shell** function now returns a **Long**. This is not stated in the Help topic.

Public Statement Cannot be Used in a Procedure (ReadMe)

The **Public** statement Help topic contains a Tip that indicates **Public** can be used in a procedure. In fact, it cannot be used in any type of procedure.

Unable to Start DDE Communication with Program Manager (ReadMe)

If you receive this error while installing Visual Basic 4.0, try the following procedure before restarting installation.

1. Close all open applications and folders
2. Restart your machine.

Orphaned Stored Procedures (ReadMe)

When using the ODBC API, Remote Data Objects, or the Microsoft Jet database engine to connect to an ODBC data source, the SQL Server driver might not remove temporary stored procedures created by the ODBC driver in the course of working on your application. This is most likely to occur when working in design mode and constantly starting, stopping, debugging and restarting applications. Loading new applications does not clear unused stored procedures. After working for some time, the TempDB database might become full. In this case, try ending Visual Basic or terminating the SQL Server connection. This might free sufficient space to continue working.

If this does not work, you must restart the SQL Server to release these resources. Note that this situation is exacerbated in development environments where a number of users are accessing the same SQL Server, each creating their own temporary stored procedures. This problem does not affect ODBC executables as their temporary stored procedures are released when the application ends.

RowLoaded Event Is Not Supported (ReadMe)

Contrary to what is included in Help, the **DBGrid** control doesn't support the RowLoaded event.

Specifying Quoted Strings When Using ODBC Data Sources (ReadMe)

When passing quoted strings as arguments in SQL statements, you can no longer use double quotes to frame strings; you must use single (') quotes. For example, the following SQL query is acceptable in SQL Server version 4.2 when accessed by Visual Basic Version 3.0:

```
Select *  From titles where title like "Computer%"
```

This SQL syntax is no longer supported by the ODBC drivers included with Visual Basic version 4.0 and must be coded as follows:

```
Select * From titles where title like 'Computer%'
```

In cases where strings include embedded single quotes, you must edit your code to accommodate this change.

Directories Added to the SETUPKIT Directory (ReadMe)

Two additional directories have been added to the VB\SETUPKIT directory. These are:

\SETUPKIT\KITFILES\SYS16 (16-bit only)

\SETUPKIT\KITFIL32\SYS32 (32-bit only)

When the SetupWizard looks for system files to copy to the distribution disk, it first looks in the corresponding \KITFILES directory rather than the computer's system directories. The purpose of this is to allow the SetupWizard to run from any 16-bit or 32-bit operating system and yet copy system files to the distribution disks that are appropriate for the application's target platform. The \KITFILES\SYS16 directory currently contains mostly 16-bit OLE files, and the \KITFIL32\SYS32 directory currently contains the Windows NT version of CTL3D32.DLL, which the Setup Toolkit never installs under Windows 95.

Graph Control ExtraData Property Example (ReadMe)

The example code provided with the Help topic for the **ExtraData** property of the Graph control doesn't work correctly. It first sets **DrawMode** = 2, which draws the picture and then sets **GraphType** = 2, which should draw a 3D pie chart. Change the last two lines as follows:

```
Graph1.DrawMode = 2
Graph1.GraphType = 2
```

to:

```
Graph1.GraphType = 2
Graph1.DrawMode = 2
```

Creating Example SETUP.LST Files (ReadMe)

To create the example SETUP.LST file for the Setup Toolkit, run the SetupWizard on a sample application like CALC.VBP.

1. Start the SetupWizard.
2. Point it to \SAMPLES\CALC.VBP.
3. Point to some temporary directory on your hard drive.
4. At step 7, accept all defaults and click Finish.
5. The SETUP.LST created on your hard disk is the file that you want. You can discard all others.
6. Repeat for the 16-bit or 32-bit versions if needed.

Context-Sensitive Help Disabled on Some Dialog Boxes (ReadMe)

Due to the way that Windows 95 supplies context sensitive help for all dialog boxes based on the system-supplied common dialogs, pressing the F1 key in many of the dialogs in Visual Basic 4.0 while running under Windows 95 will only bring up generic operating system-supplied Help windows. To obtain specific Help for a dialog box, search in Help using the title of the dialog box. For example: to search for more information on the Make EXE File dialog box search for "Make EXE File." The topic in this case appears under "Make EXE file command."

Setup Does Not Launch Correctly From Drive B (ReadMe)

Attempting to launch Visual Basic Master Setup from a network drive that is logically mapped to 'B:' fails. To prevent this problem from occurring, map your network drive to a letter other than 'B'.

Temporary Stored Procedures (ReadMe)

When the ODBC interface executes an SQL statement, it creates one or more stored procedures on the server. These procedures contain the SQL statement specified in the **rdoPreparedStatement** object or the **OpenResultset** method and are designed to accept any parameters that might be specified for the statement. Depending on the version of the server, these procedures are either created in the current database or in the TempDB database. In some cases, several stored procedures can be created for a single statement. Generally, these procedures are not released until you close the connection or end the application. Ending the application in design mode does not clear these statements. In this case, only ending Visual Basic clears these temporary procedures.

To avoid the creation of these procedures in the first place, specify the **rdExecDirect** option when using the **OpenResultset** method. For example,

```
Set rs = cn.OpenResultset("Select * from Authors", rdOpenStatic, _
rdConcurValues, rdExecDirect)
```

By using the **rdExecDirect** option, the ODBC interface does not create a procedure which is used to subsequently run the SQL statement. In some cases, this can be somewhat faster to execute, but only if the statement is used infrequently.

ODBC Driver Produces General Protection Faults When Given Incorrect Syntax (ReadMe)

In some cases, when you use the **OpenResultset** method and the SQL statement specified contains invalid SQL syntax, the ODBC driver fails with an untrappable GPF. Not all syntax errors cause this. The following SQL query causes this type of GPF:

```
" Select * from authors where name '%' (?) '%'  "
```

To avoid this problem, verify all SQL statements for accuracy and correct syntax before using them in your application. In addition, do not permit users to enter SQL statements directly, as these might be prone to failure.

TabStrip Control DblClick Event Not Supported (ReadMe)

The **TabStrip** control topic in online Help incorrectly lists this as one of the events for the control. The **TabStrip** control does not support the DblClick event.

Master Setup Won't Run from a UNC Path (ReadMe)

If you run MSETUP from a UNC path, ie: "\\visualbasic\vb4\help\readme" then it cannot shell to the actual Visual Basic setup. On the other hand, the Visual Basic setup itself will run from a UNC path.

Working with BLOB Data Types and the ODBC Cursor Library (ReadMe)

Many DBMS's, including SQL Server, support Binary Large Object (BLOB) data types. These types are most often used to store large amounts of text or image data. Due to limitations in the ODBC cursor library, special rules apply to the use of these kinds of data types when using ODBC cursors.

The **ColumnSize** property on the **rdoColumn** object represents the actual length of the data in a BLOB column. When using the ODBC cursor library, this value will always be -1, indicating that the data length is not available. When using server-side cursors, the **ColumnSize** property will always return the actual data length of a BLOB column.

To get the data from a BLOB column, the user must use the **GetChunk** methods, which take a number of bytes to retrieve at a time. When using server-side cursors, the user can pass the value of the **ColumnSize** property as the number of bytes to retrieve to get all the data at once. Since the **ColumnSize** property is not available when using the ODBC cursor library, the user should call **GetChunk** repeatedly until no more data is returned. Below is a code sample that shows how to do this:

```
Dim s As String
Dim sTemp As String
Dim lColSize As Long

lColSize = MyResultset!MyBLOBColumn.ColumnSize
If lColSize = -1 Then
   ' Column size is not available.
   ' Loop getting chunks until no more data.
   sTemp = MyResultset!MyBLOBColumn.GetChunk(50)
   Do
      s = s & sTemp
      sTemp = MyResultset!MyBLOBColumn.GetChunk(50)
   Loop While Len(sTemp) > 0
Else
' Get all of it.
   If lColSize > 0 Then
      s = MyResultset!MyBLOBColumn.GetChunk(lColSize)
   End If
End If
```

In addition, when using ODBC cursor library and BLOB data types, the user must select at least one non-BLOB column in their result set so that RDO can use SQLExtendedFetch to retrieve the data. This would be the common case anyway, since you need to include a key field in the result set if you want to update the data.

SQL Server 6.0 Transactions With Server-Side Cursors (ReadMe)

Due to a bug in the SQL Server ODBC driver, calling the **BeginTrans** method of the **rdoConnection** object after the result set has been opened will not actually begin a transaction on the server, and the user is still in auto-commit mode. This bug does not affect transactions when using the ODBC cursor library, only when using SQL Server 6.0 server-side cursors.

To work around this problem, the user can either begin the transaction before the result set is opened, or execute Transact SQL statements to begin, commit and roll back transactions. The following sample code shows how to use Transact SQL statements to do transactions:

```
Dim cn as rdoConnection
Dim rs as rdoResultset

Set cn = rdoEnvironments(0).OpenConnection("")
Set rs = cn.OpenResultset("select * from authors", rdOpenKeyset,
rdConcurValues)
...
' Do a transaction and roll it back.
cn.Execute "BeginTrans"
rs.Edit
rs(0) = "Some Value"
rs.Update
If fAllIsOK Then
   cn.Execute "CommitTrans"
Else
   cn.Execute "RollbackTrans"
End if
```

Use DAO to Access Local ISAM Databases (ReadMe)

Remote Data Objects (RDO) were designed to access *remote* databases and not local ISAMs. Data Access Objects (DAO) is the best choice to use when working with ISAM data because it's built with the semantics used with ISAM data (find methods, opening tables rather than executing queries). There is also no benefit to using RDO when talking to ISAM data, since the Microsoft Access ODBC driver will still load the Jet engine.

"For...Next Statement" Counter Can Be An Element Of a UDT (ReadMe)

In the Help topic "For...Next Statement", the description of the syntax element "counter" incorrectly states: "The variable can't be an element of a user-defined type." In fact, the counter can be an element of a user-defined type (UDT). This was not allowed in Visual Basic 3.0, but is a feature of Visual Basic 4.0.

For example, add this declaration to a standard module:

```
Type x
   i As Integer
End Type
```

Then , insert the following code in the Form_Load procedure:

```
Private Sub Form_Load()

   Dim z As x
   For z.i = 1 To 10
      Debug.Print z.i;
   Next
   Debug.Print
   With z
      For .i = 1 To 5
         Debug.Print .i;
      Next
   End With
   Debug.Print

End Sub
```

Memory Leaks with Remote Automation Applications on Windows NT Version 3.51 (ReadMe)

Visual Basic applications using machines acting as Remote Automation servers (in other words, running AUTMGR32.EXE) may consume resources until they finally exhaust all available resources and the machine hangs. This typically occurs when memory is not cleaned up after objects are created and then released by the operating system. This problem is fixed by NT 3.51 Service Pack 2.

Obtaining a Windows NT Service Pack

Obtaining a Windows NT Service Pack (ReadMe)

You can obtain a Windows NT Service Pack by the following methods:

For CompuServe (i386 only):

Log onto CIS and type the following:
Go microsoft
7 (Microsoft Support Forums and Services)
1 (US Product Support)
8 (Microsoft Operating Systems)
6 (Microsoft Windows NT Service Pack Download Area)
Download latest Service Pack for NT 3.51

For Internet Access:

ftp ftp.microsoft.com
logon anonymous
cd bussys/winnt/winnt-public/fixes/nt351
bin
get <Latest Service Pack for NT 3.51>

Insufficient Disk Space to Complete This Operation (ReadMe)

This error is caused by too many network protocols loaded on a machine. To avoid this error, you can set Registry keys to prevent the loading of specific protocols on any given machine. Locate these keys at the following location in the Registry:

```
HKEY_LOCAL_MACHINE/Software/Microsoft/Automation Manager
```

The keys beginning "nca" indicate which network protocols are loaded. In each case the value is a DWORD. Set the value to zero for protocols you don't want loaded.

You can check the NT Event Log to determine which network protocols are loaded on a machine.

This problem is fixed by NT 3.51 Service Pack 2.

Obtaining a Windows NT Service Pack

Call was Rejected by Callee (ReadMe)

This error occurs when outbound methods from Remote Automation applications cross each other, or when server applications are extremely busy. To avoid this error, you can change the timeout setting on the server machine. Locate the appropriate key at the following location in the Registry:

```
HKEY_LOCAL_MACHINE/Software/Microsoft/Automation Manager
```

The timeout setting is the value for the key "OLEServerBusyTimeout". The value is a DWORD. Increase this value as needed to avoid the error.

Gauge Control Picture Property Also Accepts .WMF Files (ReadMe)

The Help topic for the Gauge Control's **Picture** property states that the control accepts .BMP and .ICO files. In addition, the **Picture** property will accept .WMF files.

User-Defined Types And Fixed-Length Strings Not Allowed As The Type Of A Public Member (ReadMe)

The full text of the error is "User-defined types and fixed-length strings not allowed as the type of a public member of a class or form; private classes and form modules not allowed as the type of a public member or a public class." The Help topic associated with this error is titled "User-defined types not allowed as the type of a public member of a class"

The Help topic should also say that you can't pass user-defined types through a public member of a class. The reason is that OLE Automation doesn't support user-defined types. Consequently you can't have a public member in a class module or form module which has a user-defined type parameter. The member must be private (in other words, not exposed).

For example, create a new project including a form module with a **Label** placed on it, a class module, and a standard module. Insert the following code in the form:

```
Dim x as New Class1

Private Sub Form_Load()
   x.StartIt
End Sub
```

Insert this declaration in the standard module:

```
Type TestUDT
   i As Integer
End Type
```

Insert these two procedures in the class module:

```
Sub StartIt ()
   Dim udtX as TestUDT
   udtX.i = 45
   TryThis udtX
End Sub

Private Sub TryThis (y As TestUDT)
   Form1.label1.Caption = "hello"
End Sub
```

Run the application. The code runs with the TryThis procedure declared as a private procedure. If you delete the **Private** keyword in the TryThis procedure declaration and then run the application, you get the error discussed in this topic.

Product Support Phone Number in Japan (ReadMe)

The product support phone number for Japan listed in the Programmer's Guide and online Help is incorrect. The correct phone number is:

(81) (424) 41-8700

This number is used for free support up to 90 days after purchase of Visual Basic 4.0.

Constant Values in the Value Property (Custom Controls) Topic (ReadMe)

In the online Help topic Value Property (Custom Controls), the values for the **Toolbar** control constants **tbrPressed** and **tbrUnpressed** are reversed. The correct values for these constants are:

tbrPressed	1
tbrUnpressed	0

Persistent Remote Data Object rdoResultset Objects (ReadMe)

When you use the **OpenResultset** method against an **rdoConnection** or **rdoPreparedStatement**, and assign the result to an existing **rdoResultset** object, the existing object is maintained and a new **rdoResultset** object is appended to the **rdoResultsets** collection. When performing similar operations using the Microsoft Jet database engine, existing recordsets are automatically closed when the variable is assigned ,and no two **Recordsets** collection members can have the same name. For example, using RDO:

```
Dim rs as rdoResultset
Dim cn as rdoConnection

Set cn = OpenConnection....
Set rs = cn.OpenResultset("Select * from Authors", rdOpenStatic)
Set rs = cn.OpenResultset("Select * from Titles", rdOpenDynamic)
```

This code opens two separate **rdoResultset** objects; both are stored in the **rdoResultsets** collection. After this code runs the second query, which is stored in rdoResultsets(1), is assigned to the **rdoResultset** variable `rs`. The first query is available and its cursor is still available by referencing rdoResultsets(0). Because of this implementation, more than one member of the **rdoResultsets** collection can have the same name.

This behavior permits you to maintain existing **rdoResultset** objects, which are maintained in the **rdoResultsets** collection, or close them as needed. In other words, you must explicitly close any **rdoResultset** objects that are no longer needed. Simply assigning another **rdoResultset** to a rdoResultset-type variable has no affect on the existing **rdoResultset** formerly referenced by the variable. Note that the procedures and other temporary objects created to manage the **rdoResultset** are maintained on the remote server as long as the **rdoResultset** remains open.

If you write an application that does not close each **rdoResultset** before opening additional **rdoResultset** objects, the number of procedures maintained in TempDB or elsewhere on the server increase each time another **rdoResultset** object is opened. Over time, this behavior can overflow the capacity of the server or workstation resources.

NewEnum Property Not Supported By SelectedComponents or ControlTemplates Collection (ReadMe)

The **SelectedComponents** Collection (Add-In) topic in online Help incorrectly lists the **NewEnum** property as a property of the collection, and the **NewEnum** property topic incorrectly lists the **SelectedComponents** collection as an object the property applies to.

Similarly, the ControlTemplate Object, ControlTemplates Collection (Add-In) topic in online Help incorrectly lists the **NewEnum** property as a property of the collection, and the **NewEnum** property topic incorrectly lists the **ControlTemplate** object and **ControlTemplates** collections as objects the property applies to.

Appearance Property Not Supported by HScrollBar and VScrollBar Controls (ReadMe)

The online Help topic for the **HScrollBar** and **VScrollBar** controls incorrectly lists the **Appearance** property as a property of these controls.

DBGrid Control Does Not Correctly Rebuild Columns (ReadMe)

When you use the **Remove** method against the **DBGrid** control, existing columns of the grid are removed. However, when you subsequently use the **Add** method to add new columns, column data from the previous columns re-appear. Whenever you remove existing columns or add new columns to the **DBGrid** control, use the **ReBind** and **Refresh** methods once all changes have been made. This instructs the **DBGrid** control to rebuild its internal column layout matrix to correctly reflect the true status of the control.

Executing RDO Queries Returning Multiple Resultsets (ReadMe)

When executing Remote Data Object (RDO) queries that return more than one set of results, you can use only the ODBC cursor drivers. The Microsoft SQL Server server-side cursors do not support result sets that return more than a single set of results. To enable the ODBC cursor driver, set the **rdoEnvironment** object's **CursorDriver** property to **rdUseODBC** before creating the cursor.

Settings for the Action Property (MAPI Session Control) (ReadMe)

The settings as listed in the **Action** Property (MAPI Session Control) online Help topic are incorrect. The correct settings are:

mapSignOff

mapSignOn

Remote Automation Connection Registry APIs (ReadMe)

Part of the Visual Basic 4.0 support for Remote Automation is a Connection Registry utility, (RACREG32.DLL for 32-bit applications and RACREG16.EXE for 16-bit applications), which allows applications to programatically read and set the Remote Automation registry information for OLE objects. This allows applications to control which server machine on which they want to run an OLE object, assuming it has been previously installed on the specified servers, including running the object locally or remotely or changing the location of an object from remote server1 to remote server2. The Connection Registry utility has two APIs (methods), **SetAutoServerSettings** and **GetAutoServerSettings**.

Note RACREG32.DLL and RACREG16.EXE may be freely distributed with any application developed with Visual Basic, Enterprise Edition.

SetAutoServerSettings Method
GetAutoServerSettings Function

SetAutoServerSettings Method (ReadMe)

Sets the Remote Automation registry values to meet OLE and Remote Automation requirements, including configuration settings for remote server access.

Syntax

object.**SetAutoServerSettings** (*Remote*, [*ProgID*], [*CLSID*], [*ServerName*], [*Protocol*], [*Authentication*])

The **SetAutoServerSettings** method syntax has these parts:

Part	Type	Description
object	**RegClass**	An object variable of type **RegClass**.
Remote	**Boolean**	**True** if the server is remote, **False** if local.
ProgID	**Variant**	The ProgID for the server.
CLSID	**Variant**	The CLSID for the server.
ServerName	**Variant**	The name of the server machine.
Protocol	**Variant**	The RPC name of the protocol to be used.
Authentication	**Variant**	The RPC authentication level.

Return Values

The method returns the following error codes:

Value	Description
0	No error.
1	Unknown run time error occurred.
2	No protocol was specified.
3	No server machine name was specified.
4	An error occurred reading from the registry.
5	An error occurred writing to the registry.
6	Both the *ProgID* and *CLSID* parameters were missing.
7	There is no local server (either in-process or cross-process, 16-bit or 32-bit).
8	There was an error looking for the Proxy DLLs, check that they were installed properly.

Remarks

The **SetAutoServerSettings** method will take either a CLSID or a ProgID and set the registry information to local or remote depending on the value of the *Remote* parameter. If a *CLSID* and a *ProgID* are passed to the method, the *CLSID* will take precedence.

Example

This example switches the Hello server from local registration to remote, and back. See the \SAMPLES\REMAUTO\HELLO directory for the code for the Hello World sample project.

```
Sub SwitchHello()
   Dim oRegClass As New RegClass
   ' Register Hello to run remotely on a machine called Server1.
   oRegClass.SetAutoServerSettings True, "HelloProj.HelloClass",1 _
   ServerName:="Server1", Protocol:="ncacn_ip_tcp"
   ' Register Hello to run locally again.
   oRegClass.SetAutoServerSettings False, "HelloProj.HelloClass"
End Sub
```

GetAutoServerSettings Function (ReadMe)

Returns information about the state of an OLE object's registration.

Syntax

object.**GetAutoServerSettings** ([*ProgID*], [*CLSID*])

The **GetAutoServerSettings** method has these parts

Parameter	Type	Description
object	**RegClass**	An object variable of type **RegClass**.
ProgID	**Variant**	The ProgID of the OLE class.
CLSID	**Variant**	The CLSID of the OLE class.

Return Value

The **GetAutoServerSettings** method returns a **Variant** that contains an array of values about the given OLE class. The index values and descriptions shown in the following table.

Value	Description
1	**True** if the server is registered remotely.
2	Remote machine name.
3	RPC network protocol name.
4	RPC authentication level.

If a value is missing or not available, the value will be an empty string. If there is an error during the method, then the return value will be a **Variant** of type **Empty**.

Example

This example retrieves information about a remotely registered Hello object.

```
Sub ViewHello()
   Dim oRegClass As New RegClass
   Dim vRC As Variant
   vRC = oRegClass.GetAutoServerSettings("HelloProj.HelloClass")
   If Not(IsEmpty(vRC)) Then
      If vRC(1) Then
         MsgBox "Hello is registered remotely on a " _
         & "server named: " & vRC(1)
      Else
         MsgBox "Hello is registered locally."
      End If
   End if
End Sub
```

Cascades, Local Tables and Replication Don't Mix (ReadMe)

Cascade updates and deletes are not supported between local tables in a replicated database. If you have local tables in a replicated database, and you have cascades turned on for these local tables, then updates and deletes at the primary table are not supported.

Note It is unusual to use cascade updates on local tables. Local tables are expected to be used for simple tasks, and cascades is an advanced feature.

Creating Parameter Queries Example Incorrect (ReadMe)

This topic applies only to the Enterprise Edition of Visual Basic 4.0.

The final example in the online Help topic Creating Parameter Queries is coded incorrectly. A correct example is shown below. Note that you use the variable set to the created **rdoPreparedStatement** to create the **rdoResultset,** which is not shown correctly in the Help example. This example executes a stored procedure that expects two input parameters and returns two output parameters along with a return value argument.

```
Dim SQL As String, MyOutputVal1 As Variant
Dim MyOutputVal2 As Variant, MyRetVal As Variant
Dim cn As rdoConnection, rs As rdoResultset

rdoEnvironments(0).CursorDriver = rdUseOdbc
' To permit execution on SQL Server 6.0 Set
' cn=rdoEnvironments(0).OpenConnection(dsname:="MyDSN",
' Prompt:=rdDriverNoPrompt)

' Use ODBC parameter argument syntax.
SQL = "{ ? = call MyProcName (?, ?,?,?) }"

Dim Ps As rdoPreparedStatement
' Create reusable rdoPreparedStatement.
Set Ps = cn.CreatePreparedStatement("PsTest",SQL)

' Set Parameter "direction" types for each parameter,
' both input and output.
Ps(0).Direction = rdParamReturnValue
Ps(1).Direction = rdParamInput
Ps(2).Direction = rdParamInput
Ps(3).Direction = rdParamOutput
Ps(4).Direction = rdParamOutput

' Set the input argument values.
Ps.rdoParameters(1) = "Test%"
Ps.rdoParameters(2) = 1


' Create the result set and populate the Ps values.
Set rs = Ps.OpenResultset(rdOpenStatic)

MyRetVal = Ps(0)        ' Contains the return value argument.
MyOutputVal1 = Ps(3)    ' Contains the first output parameter.
MyOutputVal2 = Ps(4)    ' Contains the second output parameter.
```

Using SQLExecDirect (ReadMe)

The **rdExecDirect** option is available when executing queries that do not require the creation of temporary stored procedures or where creation of these procedures affects performance. This option uses the ODBC API **SQLExecDirect** function to execute the query when invoking the RDO **OpenResultset** or **Execute** methods. If you do not use this option, the ODBC driver creates a temporary stored procedure that is used to execute the actual query. In cases where you execute the same query repeatedly, this method provides better performance. However, if you must use SQL syntax that is unacceptable to the ODBC layer but recognized by the remote database engine, use the **rdExecDirect** option to bypass the creation of the temporary stored procedures.

In some situations, temporary stored procedures are not removed until the connection to the remote database engine is closed. Using the **SQLExecDirect** option can eliminate this problem as the procedures are not created.

Connecting to ODBC Data Sources with no DSN (ReadMe)

It is possible to open a connection to a remote database server without first creating a permanent DSN entry by specifying additional information in the **Connect** property. Basically, the connect string must provide the minimum information contained in the DSN: Driver, Server and usually Database. The following example shows how to open an RDO connection to Microsoft SQL Server without using a DSN:

```
Cnct$ = "Driver={SQL Server};Server=MySQLServer;Database=MyDB"
Set cn = rdoEnvironments(0).OpenConnection(dsname:="",Connect:=Cnct$,
prompt:=rdDriverNoPrompt)
```

This strategy does not work when opening ODBC connections using the Microsoft Jet database engine with the **Data** control, or Data Access Objects. However, it does work when using the **RemoteData** control.

Caution Do not attempt to use this feature with Jet ODBC connections as it causes a General Protection Fault error.

Syntax For the OpenResultset Method (ReadMe)

The syntax for the **OpenResultset** method currently states:
Set *variable* = *connection* **.OpenResultset**(*source* [, *type* [, *locktype* [, *options*]]])
Set *variable* = *object* **.OpenResultset**([*type* [, *locktype* [, *options*]]])

The *source* argument should be *name*, and the *options* argument should be *option*. The correct syntax is as follows:
Set *variable* = *connection* **.OpenResultset**(***name*** [, ***type*** [, ***locktype*** [, ***option***]]])
Set *variable* = *object* **.OpenResultset**([***type*** [, ***locktype*** [, ***option***]]])

Using CompactDatabase with Microsoft Access Databases (ReadMe)

You should not use the **CompactDatabase** method when converting databases created or maintained with Microsoft Access if you expect to subsequently use them with Microsoft Access version 7.0. To convert Microsoft Access databases from one version to another, use the Compact Database menu command within Microsoft Access version 7.0.

If you use the **CompactDatabase** method from Visual Basic version 4.0, the database cannot be opened by Microsoft Access version 7.0, and you must revert to a backup copy and use Microsoft Access version 7.0 to perform the conversion.

Databases converted by Microsoft Access version 7.0 can be opened and manipulated by Visual Basic 4.0.

Unable to Access MAPI Functionality Under VB32.EXE (ReadMe)

If you attempt to run a program under VB32.EXE that accesses MAPI functionality, such as the VBMAIL sample distributed with Visual Basic, you may be unable to perform MAPI functions such as **SignOn**. The reason for this may be that your operating environment does not have 32-bit MAPI DLLs installed properly.

For example, on Windows 95, you must install Mail during the operating system setup, or install it separately from the control panel to correctly use MAPI functions or MAPI custom controls from Visual Basic.

Let and Set statements and Property Let and Property Set Procedures (ReadMe)

Two Visual Basic statements are used in combination with the assignment operator (=). The **Let** statement, although usually implicit, is used for assigning values. The **Set** statement, which must always be explicit, is used for assigning object references. If you use **Let** instead of **Set** when assigning an object reference, you will generally end up assigning the value of the object's default property. Attempting to use the resulting variable as an object reference will usually result in an error, such as Error 424 `Object required`.

Property procedures allow you to execute code, in addition to simple **Let** and **Set** statements and assignments. Such code might validate ranges for a property or type of object reference. To create a procedure that executes code whenever a user assigns a value to a property, use a **Property Let** procedure. To create a procedure that executes code whenever a user sets an object reference, use a **Property Set** procedure. To create a procedure that executes code whenever a user obtains either a value or an object reference, use a **Property Get** procedure.

Coercion of Hexadecimal Values (ReadMe)

You can use the type-declaration character to prevent sign extension when using hexadecimal values. For example:

```
Print &H82A0      ' Prints -32096
Print &H82A0&     ' Prints  33440
```

Administration of Component Catalogs (ReadMe)

A bug in the Component Manager can create Component Catalog database errors if two users try to update the contents of a remote or shared Component Catalog at the same time. To avoid this problem, only one user at a time (the designated administrator) should be given write permission to a Component Catalog database. Your ODBC database administration tools should be used to assign user read/write permissions to the Catalog ODBC database. The Component Manager will then use these permissions to disable update commands in the Component Manager's user interface.

Samples Help File Refers to Wrong Chapter For BIBLIO.VBP (ReadMe)

The description of the BIBLIO.VBP sample application in the SAMPLES.HLP file contains the following text:

This sample allows you to browse the BIBLIO.MDB database, which is located in the main Visual Basic directory (\VB). The sample demonstrates the Data control. For more information, see Chapter 12, "Accessing Databases with the Data control," in the Programmer's Guide

The chapter information is incorrect. Instead of Chapter 12, it should be Chapter 22.

Books Online Path (ReadMe)

If you have a previous installation of Visual Basic and you install it a second time to a new location, Books Online will fail to run. In this case, setup doesn't update the BooksExePath and/or BooksExePath16 entries in the VB.INI file with the new location for Books Online. To correct the problem, enter the correct location for Books Online in the BooksExePath and BooksExePath16 entries in VB.INI.

Version Compatibility Feature Does Not Work For Some OLE Servers (ReadMe)

The Version Compatibility feature for OLE servers is described in the section "Version Compatibility" of Chapter 2, Building OLE Servers, in the Professional Features book, *Creating OLE Servers.* In brief, this feature allows a developer to enhance an OLE server by adding new objects and methods, while maintaining backward compatibility for OLE client applications compiled with older versions of the OLE server.

Due to last-minute changes in the product, the Version Compatibility feature does not work under the following conditions: (1) an OLE client application uses early binding to refer to objects provided by an OLE server; (2) the OLE server runs out-of-process with respect to the OLE client; (3) the new version of the OLE server is only *version compatible,* rather than *version identical*.

When these conditions are met, as explained below, the OLE client application *cannot* use a new version of the OLE server that is produced according to the rules for version compatibility. If the existing client makes OLE calls to the new version of the OLE server, the calls will fail with OLE Automation error -2147319765, (&H8002802B), `Element not found`.

Early binding, referred to in condition (1), means that the OLE client contains variables declared using the names of **Public** class modules exposed by the OLE server. For example, the following code from an OLE client shows early binding to a Sprocket object exposed by the Widget server:

```
Private Sub Command1_Click()
   Dim s As Widget.Sprocket        ' Use early binding.
   Set s = New Widget.Sprocket
   s.Spin      ' Call the Spin method of the Sprocket object.
End Sub
```

By contrast, if the OLE client uses variables declared **As Object**, the objects are *late bound,* and the client can use compatible versions of the OLE server without causing errors. However, this method of accessing objects is slower than early binding, and the **CreateObject** function must be used instead of the **New** operator.

Out-of-process, referred to in condition (2), means that the objects early bound by the OLE client are provided by an OLE server running in a separate process space. An OLE server that is compiled as an .EXE file is always an out-of-process OLE server.

By contrast, an *in-process* OLE server is compiled as a .DLL file, and when it is used directly by an OLE client (that is, the DLL is loaded into the OLE client's process space), the version compatibility feature works as described in *Creating OLE Servers.*

Any object obtained from an OLE server running on another computer, using the Remote Automation feature of Visual Basic Enterprise Edition, is out-of-process with respect to the OLE client, regardless of the type of OLE server in which it originated.

Whenever a reference to an object is passed across process boundaries, even between two OLE client applications running on the same computer, the object is out-of-process with respect to the second OLE client.

Version compatible, referred to in condition (3), means that an OLE server has been enhanced by the addition of new classes, or new methods to existing classes. If the only changes you have made to an OLE server involve the internal implementation details of existing methods, then the new version is classified as *version identical,* and all OLE clients compiled using previous versions will continue to work correctly.

If you require version compatibility for an OLE server, that is, if client applications compiled using one version of an OLE server must work with all subsequent versions of the OLE server, you can follow these guidelines for *OLE client* applications: (1) If the objects provided by the OLE server will be used out-of-process, especially between different computers, make sure that OLE clients use late binding (`Dim objWidget As Object`) for

those objects. (2) If the OLE server is compiled as an OLE DLL, and used in-process by an OLE client, the OLE client can use early binding.

Restrictions on Creating Version Incompatible OLE Servers (ReadMe)

The Version Compatibility feature for OLE servers is described in the section "Version Compatibility" of Chapter 2, Building OLE Servers, in the Professional Features book, *Creating OLE Servers.* In brief, this feature allows a developer to enhance an OLE server by adding new objects and methods, while maintaining backward compatibility for OLE client applications compiled with older versions of the OLE server.

This item adds a restriction not found in the documentation.

You may find it necessary to enhance an existing OLE server so that it is *version incompatible* with previously distributed OLE clients, as defined in the reference cited above. For example, changing the number of parameters required by an existing method will cause the new version to be incompatible with previous versions. Visual Basic marks the server so that previously compiled OLE clients cannot use it.

To ensure that OLE clients compiled with earlier versions of the server will continue to work, there are three things you must do when compiling a version incompatible OLE server:

- Use the Make EXE File or the Make OLE DLL File dialog box to enter a new name for the .EXE or .DLL file, to prevent the incompatible version from replacing older versions already installed on client computers.
- On the Project tab of the Options dialog box, enter a new Project Name for the OLE server, to prevent the incompatible version from replacing Windows Registry information for objects supplied by older versions.
- On the Project tab of the Options dialog box, clear the Compatible OLE Server field, to ensure that the incompatible version does not use the same type library identifier as older versions.

The first two items in the list are mentioned in the documentation cited earlier, and in warning messages provided during the compilation process. The last item is an additional restriction not found in the documentation or compilation warnings.

Dynaset's Visibility of Extraneous Changes (ReadMe)

In Visual Basic 3.0, the user always saw extraneous changes made to underlying data when viewing the data through a **Dynaset** object. This was because Visual Basic 3.0 would call Jet's **Idle** method every time it entered the idle loop. In Visual Basic 4.0 you cannot call the Jet idle loop. Because of this, you will not see those changes unless you force Jet to do its idle processing. To force this, you should call **DBEngine**.**Idle** at points in your code where you would look at underlying data of dynaset-type **Recordset** objects.

Using Image or Text Data Types with the RemoteData Control or RDO (ReadMe)

A known bug prevents complete use of **Image** or **PictureBox** controls with the **RemoteData** control. While it is possible to read TEXT and IMAGE datatypes using the RDC, it is not possible to update this type of column. It is possible to update these columns using RDO code or with the **Data** control.

Note that graphics columns created by Microsoft Access must be accessed with OLE controls, not **Picture** or **Image** controls.

Accessing SQL Server 6.0 Tables with Identity Columns (ReadMe)

If you attempt to access a SQL Server 6.0 table that includes an identity column, you can trigger an erroneous 3622 error. To prevent this problem, use the **dbSeeChanges** option when using the **OpenRecordset** method or 512 in the **Options** property of the **Data** control. This applies to attached SQL Server tables as well as those opened directly.

Toolbar Control Example (ReadMe)

When you run the **Toolbar** control example, the toolbar buttons are not in their proper positions. To see all buttons on the **Toolbar** example, manually resize the form.

Can't Open FoxPro Table Contained in a Database Container (ReadMe)

Jet cannot open a FoxPro 3.0 table that is contained in a FoxPro 3.0 database container. Tables that are not contained in a database will open correctly. If you attempt to attach to one of these tables, you will get a blank dialog box.

RemoteData Control EOFAction Property (ReadMe)

When accessing an empty, updatable results et with the **RemoteData** (RDC) control, you cannot depend on the **EOFAction** property to force the control to switch to **AddNew** mode. To add the first record to an empty result set, you must use the **AddNew** method against the **RemoteData Resultset** property. This will be corrected in a later release of the control.

TreeView Sorted Property Doesn't Sort Nodes Automatically (ReadMe)

The **Sorted** Property (**TreeView** Control) topic has this statement:

"In either case, setting the **Sorted** property to **True** mans any new **Node** objects added to a **Node** or **TreeView** control will be sorted automatically."

This is not true. After nodes are added, you must reset the **Sorted** property to **True** to sort the newly added nodes.

Miscellaneous Jet Database Replication Issues (ReadMe)

Update Conflicts Should be Resolved Periodically

For databases containing autogenerate (random autonumber) columns, update conflicts should be resolved periodically, to prevent sync failures. For a table containing an autogenerate (random autonumber) column, if two replicas update the same record, cause a conflict, and therefore create an entry in the side-table

AND, before the conflict is cleared, the replicas attempt to update the same two records again, and the same replica "loses" (does not succeed in updating the record),

THEN, when the system attempts to update the side-table with this new conflict, the synchronization will fail. The workaround is to clear the initial conflict before having the same replicas sync. This could be accomplished by several different methods:

a) delete the specific rows in the conflict table,

b) delete the whole conflict table,

c) using DAO (see the Help topic "Resolving Synchronization Conflicts").

In Access, you can also use the Conflict Resolution Wizard.

Cascade Updates and Deletes

Cascade updates and deletes are not supported between local tables in a replicated database. If you have local tables in a replicated database, and you have cascades turned on for these local tables, then updates and deletes at the primary table are not supported.

Excess Lock Conflicts

Two clients programs on one Windows 95 machine accessing the same remote database may experience excess lock conflicts. This issue is particularly noticeable when running replication and the transporter.

Remote Data Objects and Case-Sensitive Servers (ReadMe)

When using remote data objects against a SQL Server that has case-sensitivity enabled, if your table name contains any upper-case letters the **Update** method will fail with an 'invalid object <table name>' error. This will only occur when using the ODBC Cursor Library against a case-sensitive SQL Server.

To work around this problem you have several options:

- Remove case-sensitivity from the SQL Server
- Make your table names all lower-case
- Use server-side cursors

If the above options are not available you can issue update SQL statements to the server using the **Execute** method.

This does not affect Oracle servers.