#Const Directive Example

This example uses the **#Const** directive to declare conditional compiler constants for use in **#If...#Else...#End If** constructs.

```
#Const DebugVersion = 1 ' Will evaluate true in #If block.
```

#If...Then...#Else Directive Example

This example references conditional compiler constants in an **#If...Then...#Else** construct to determine whether to compile certain statements.

```
' If Mac evaluates as true, do the statements following the #If.
#If Mac Then
    '. Place exclusively Mac  statements here.
    '.
    '.
' Otherwise, if it is a 32-bit Windows program, do this:
#ElseIf Win32 Then
    '. Place exclusively 32-bit Windows statements here.
    '.
    '.
' Otherwise, if it is neither, do this:
#Else
    '. Place other platform statements here.
    '.
    '.
#End If
```

#Const Directive

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadirConstC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadirConstX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadirConstS"}

Used to define conditional compiler constants for Visual Basic.

Syntax

#Const *constname* = *expression*

The **#Const** compiler directive syntax has these parts:

Part	Description
constname	Required; **Variant** (**String**). Name of the constant; follows standard variable naming conventions.
expression	Required. Literal, other conditional compiler constant, or any combination that includes any or all arithmetic or logical operators except **Is**.

Remarks

Conditional compiler constants are always **Private** to the module in which they appear. It is not possible to create **Public** compiler constants using the **#Const** directive. **Public** compiler constants can only be created in the user interface.

Only conditional compiler constants and literals can be used in *expression*. Using a standard constant defined with **Const**, or using a constant that is undefined, causes an error to occur. Conversely, constants defined using the **#Const** keyword can only be used for conditional compilation.

Conditional compiler constants are always evaluated at the module level, regardless of their placement in code.

#If...Then...#Else Directive

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadirIfC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadirIfX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadirIfS"}

Conditionally compiles selected blocks of Visual Basic code.

Syntax

#If *expression* **Then**
    statements
[**#ElseIf** *expression-n* **Then**
    [*elseifstatements*]]
[**#Else**
    [*elsestatements*]]
#End If

The **#If...Then...#Else** directive syntax has these parts:

Part	Description
expression	Required. Any expression, consisting exclusively of one or more conditional compiler constants, literals, and operators, that evaluates to **True** or **False**.
statements	Required. Visual Basic program lines or compiler directives that are evaluated if the associated expression is **True**.
expression-n	Optional. Any expression, consisting exclusively of one or more conditional compiler constants, literals, and operators, that evaluates to **True** or **False**.
elseifstatements	Optional. One or more program lines or compiler directives that are evaluated if *expression-n* is **True**.
elsestatements	Optional. One or more program lines or compiler directives that are evaluated if no previous *expression* or *expression-n* is **True**.

Remarks

The behavior of the **#If...Then...#Else** directive is the same as the **If...Then...Else** statement, except that there is no single-line form of the **#If**, **#Else**, **#ElseIf**, and **#End If** directives; that is, no other code can appear on the same line as any of the directives. Conditional compilation is typically used to compile the same program for different platforms. It is also used to prevent debugging code from appearing in an executable file. Code excluded during conditional compilation is completely omitted from the final executable file, so it has no size or performance effect.

Regardless of the outcome of any evaluation, all expressions are evaluated. Therefore, all constants used in expressions must be defined—any undefined constant evaluates as **Empty**.

Note The **Option Compare** statement does not affect expressions in **#If** and **#ElseIf** statements. Expressions in a conditional-compiler directive are always evaluated with **Option Compare Text**.

Visual Basic Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscTypeLibConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscTypeLibConstantsS"}

Visual Basic for Applications defines constants to simplify your programming. The following constants can be used anywhere in your code in place of the actual values:

Calendar Constants

Color Constants

Compiler Constants

Date Constants

Dir, **GetAttr**, and **SetAttr** Constants

IMEStatus Constants

Instr, **StrComp** Constants

Keycode Constants

Miscellaneous Constants

MsgBox Constants

Shell Constants

StrConv Constants

System Color Constants

VarType Constants

Calendar Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscCalendarConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscCalendarConstantsS"}

The following constants can be used anywhere in your code in place of the actual values:

Constant	Value	Description
vbCalGreg	**0**	Indicates that the Gregorian calendar is used.
vbCalHijri	**1**	Indicates that the Hijri calendar is used.

Compiler Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscCompilerConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscCompilerConstantsS"}

Visual Basic for Applications defines constants for exclusive use with the **#If...Then...#Else** directive. These constants are functionally equivalent to constants defined with the **#If...Then...#Else** directive except that they are global in scope; that is, they apply everywhere in a project.

On 16-bit development platforms, the compiler constants are defined as follows:

Constant	Value	Description
Win16	**True**	Indicates development environment is 16-bit.
Win32	**False**	Indicates that the development environment is not 32-bit.

On 32-bit development platforms, the compiler constants are defined as follows:

Constant	Value	Description
Win16	**False**	Indicates that the development environment is not 16-bit.
Win32	**True**	Indicates that the development environment is 32-bit.

Note These constants are provided by Visual Basic, so you cannot define your own constants with these same names at any level.

Date Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscDateConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscDateConstantsS"}

The following constants can be used anywhere in your code in place of the actual values:

Argument Values

The *firstdayofweek* argument has the following values:

Constant	Value	Description
vbUseSystem	0	Use NLS API setting.
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

The *firstdayofyear* argument has the following values:

Constant	Value	Description
vbUseSystem	0	Use NLS API setting.
VbFirstJan1	1	Start with week in which January 1 occurs (default).
vbFirstFourDays	2	Start with the first week that has at least four days in the new year.
vbFirstFullWeek	3	Start with the first full week of the year.

Return Values

Constant	Value	Description
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Dir, GetAttr, and SetAttr Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscDirGetAttrSetAttrConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscDirGetAttrSetAttrConstantsS"}

The following constants can be used anywhere in your code in place of the actual values:

Constant	Value	Description
vbNormal	0	Normal (default for **Dir** and **SetAttr**)
vbReadOnly	1	Read-only
vbHidden	2	Hidden
vbSystem	4	System file
vbVolume	8	Volume label
vbDirectory	16	Directory or folder
vbArchive	32	File has changed since last backup

IMEStatus Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsclIMEStatusConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsclIMEStatusConstantsS"}

The following constants can be used anywhere in your code in place of the actual values.

The constants for the Japanese locale are as follows:

Constant	Value	Description
vbIMENoOp	0	No IME installed
vbIMEOn	1	IME on
vbIMEOff	2	IME off
vbIMEDisable	3	IME disabled
vbIMEHiragana	4	Hiragana double-byte characters (DBC)
vbIMEKatakanaDbl	5	Katakana DBC
vbIMEKatakanaSng	6	Katakana single-byte characters (SBC)
vbIMEAlphaDbl	7	Alphanumeric DBC
vbIMEAlphaSng	8	Alphanumeric SBC

The constants for the Chinese (traditional and simplified) locale are as follows:

Constant	Value	Description
vbIMENoOp	0	No IME installed
vbIMEOn	1	IME on
vbIMEOff	2	IME off

For the Korean locale, the first five bits of the return value are set as follows:

Bit	Value	Description	Value	Description
0	0	No IME installed	1	IME installed
1	0	IME disabled	1	IME enabled
2	0	IME English mode	1	Hangeul mode
3	0	Banja mode (SBC)	1	Junja mode (DBC)
4	0	Normal mode	1	Hanja conversion mode

Instr, StrComp Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscInstrStrCompConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscInstrStrCompConstantsS"}

The following constants are defined in the Visual Basic for Applications type library and can be used anywhere in your code in place of the actual values:

Constant	Value	Description
vbBinaryCompare	0	Perform a binary comparison
vbTextCompare	1	Perform a textual comparison
vbDatabaseCompare	2	For Microsoft Access, perform a comparison based on information contained in your database.

Miscellaneous Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscMiscellaneousConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscMiscellaneousConstantsS"}

The following constants are defined in the Visual Basic for Applications type library and can be used anywhere in your code in place of the actual values:

Constant	Equivalent	Description
vbCrLf	**Chr(**13**) + Chr(**10**)**	Carriage return–linefeed combination
vbCr	**Chr(**13**)**	Carriage return character
vbLf	**Chr(**10**)**	Linefeed character
vbNewLine	**Chr(**13**) + Chr(**10**)** or **Chr(**13**)**	Platform-specific new line character; whichever is appropriate for current platform
vbNullChar	**Chr(**0**)**	Character having value 0
vbNullString	String having value 0	Not the same as a zero-length string (""); used for calling external procedures
vbTab	**Chr(**9**)**	Tab character
vbBack	**Chr(**8**)**	Backspace character
vbFormFeed	**Chr(**12**)**	Not useful in Microsoft Windows
vbVerticalTab	**Chr(**11**)**	Not useful in Microsoft Windows

MsgBox Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscmsgboxConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscMsgBoxConstantsS"}

The following constants can be used anywhere in your code in place of the actual values:

MsgBox Arguments

Constant	Value	Description
vbOKOnly	0	**OK** button only (default)
vbOKCancel	1	**OK** and **Cancel** buttons
vbAbortRetryIgnore	2	**Abort**, **Retry**, and **Ignore** buttons
vbYesNoCancel	3	**Yes**, **No**, and **Cancel** buttons
vbYesNo	4	**Yes** and **No** buttons
vbRetryCancel	5	**Retry** and **Cancel** buttons
vbCritical	16	Critical message
vbQuestion	32	Warning query
vbExclamation	48	Warning message
vbInformation	64	Information message
vbDefaultButton1	0	First button is default (default)
vbDefaultButton2	256	Second button is default
vbDefaultButton3	512	Third button is default
vbDefaultButton4	768	Fourth button is default
vbApplicationModal	0	Application modal message box (default)
vbSystemModal	4096	System modal message box
vbMsgBoxHelpButton	16384	Adds Help button to the message box
VbMsgBoxSetForeground	65536	Specifies the message box window as the foreground window
vbMsgBoxRight	524288	Text is right aligned
vbMsgBoxRtlReading	1048576	Specifies text should appear as right-to-left reading on Hebrew and Arabic systems

MsgBox Return Values

Constant	Value	Description
vbOK	1	**OK** button pressed
vbCancel	2	**Cancel** button pressed
vbAbort	3	**Abort** button pressed
vbRetry	4	**Retry** button pressed
vbIgnore	5	**Ignore** button pressed
vbYes	6	**Yes** button pressed
vbNo	7	**No** button pressed

Shell Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscShellConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscShellConstantsS"}

The following constants can be used anywhere in your code in place of the actual values:

Constant	Value	Description
vbHide	0	Window is hidden and focus is passed to the hidden window.
vbNormalFocus	1	Window has focus and is restored to its original size and position.
vbMinimizedFocus	2	Window is displayed as an icon with focus.
vbMaximizedFocus	3	Window is maximized with focus.
vbNormalNoFocus	4	Window is restored to its most recent size and position. The currently active window remains active.
vbMinimizeNoFocus	6	Window is displayed as an icon. The currently active window remains active.

StrConv Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscStrConvConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscStrConvConstantsS"}

The following constants can be used anywhere in your code in place of the actual values:

Constant	Value	Description
vbUpperCase	1	Converts the string to uppercase characters.
vbLowerCase	2	Converts the string to lowercase characters.
vbProperCase	3	Converts the first letter of every word in string to uppercase.
vbWide	4	Converts narrow (single-byte) characters in string to wide (double-byte) characters. Applies to Far East locales.
vbNarrow	8	Converts wide (double-byte) characters in string to narrow (single-byte) characters. Applies to Far East locales.
vbKatakana	16	Converts Hiragana characters in string to Katakana characters. Applies to Japan only.
vbHiragana	32	Converts Katakana characters in string to Hiragana characters. Applies to Japan only.
vbUnicode	64	Converts the string to Unicode using the default code page of the system.
vbFromUnicode	128	Converts the string from Unicode to the default code page of the system.

VarType Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVarTypeConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVarTypeConstantsS"}

The following constants can be used anywhere in your code in place of the actual values:

Constant	Value	Description
vbEmpty	0	Uninitialized (default)
vbNull	1	Contains no valid data
vbInteger	2	**Integer**
vbLong	3	Long integer
vbSingle	4	Single-precision floating-point number
vbDouble	5	Double-precision floating-point number
vbCurrency	6	**Currency**
vbDate	7	**Date**
vbString	8	**String**
vbObject	9	Object
vbError	10	Error
vbBoolean	11	**Boolean**
vbVariant	12	**Variant** (used only for arrays of variants)
vbDataObject	13	Data access object
vbDecimal	14	**Decimal**
vbByte	17	**Byte**
vbArray	8192	Array

Color Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscColorConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscColorConstantsS"}

The following constants can be used anywhere in your code in place of the actual values:

Constant	Value	Description
vbBlack	0x0	Black
vbRed	0xFF	Red
vbGreen	0xFF00	Green
vbYellow	0xFFFF	Yellow
vbBlue	0xFF0000	Blue
vbMagenta	0xFF00FF	Magenta
vbCyan	0xFFFF00	Cyan
vbWhite	0xFFFFFF	White

Keycode Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscKeycodeConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscKeycodeConstantsS"}

The following constants can be used anywhere in your code in place of the actual values:

Constant	Value	Description
vbKeyLButton	0x1	Left mouse button
vbKeyRButton	0x2	Right mouse button
vbKeyCancel	0x3	CANCEL key
vbKeyMButton	0x4	Middle mouse button
vbKeyBack	0x8	BACKSPACE key
vbKeyTab	0x9	TAB key
vbKeyClear	0xC	CLEAR key
vbKeyReturn	0xD	ENTER key
vbKeyShift	0x10	SHIFT key
vbKeyControl	0x11	CTRL key
vbKeyMenu	0x12	MENU key
vbKeyPause	0x13	PAUSE key
vbKeyCapital	0x14	CAPS LOCK key
vbKeyEscape	0x1B	ESC key
vbKeySpace	0x20	SPACEBAR key
vbKeyPageUp	0x21	PAGE UP key
vbKeyPageDown	0x22	PAGE DOWN key
vbKeyEnd	0x23	END key
vbKeyHome	0x24	HOME key
vbKeyLeft	0x25	LEFT ARROW key
vbKeyUp	0x26	UP ARROW key
vbKeyRight	0x27	RIGHT ARROW key
vbKeyDown	0x28	DOWN ARROW key
vbKeySelect	0x29	SELECT key
vbKeyPrint	0x2A	PRINT SCREEN key
vbKeyExecute	0x2B	EXECUTE key
vbKeySnapshot	0x2C	SNAPSHOT key
vbKeyInsert	0x2D	INSERT key
vbKeyDelete	0x2E	DELETE key
vbKeyHelp	0x2F	HELP key
vbKeyNumlock	0x90	NUM LOCK key

The A key through the Z key are the same as the ASCII equivalents A – Z:

Constant	Value	Description
vbKeyA	65	A key
vbKeyB	66	B key
vbKeyC	67	C key
vbKeyD	68	D key
vbKeyE	69	E key
vbKeyF	70	F key

vbKeyG	71	G key
vbKeyH	72	H key
vbKeyI	73	I key
vbKeyJ	74	J key
vbKeyK	75	K key
vbKeyL	76	L key
vbKeyM	77	M key
vbKeyN	78	N key
vbKeyO	79	O key
vbKeyP	80	P key
vbKeyQ	81	Q key
vbKeyR	82	R key
vbKeyS	83	S key
vbKeyT	84	T key
vbKeyU	85	U key
vbKeyV	86	V key
vbKeyW	87	W key
vbKeyX	88	X key
vbKeyY	89	Y key
vbKeyZ	90	Z key

The 0 key through 9 key are the same as their ASCII equivalents 0 – 9:

Constant	Value	Description
vbKey0	48	0 key
vbKey1	49	1 key
vbKey2	50	2 key
vbKey3	51	3 key
vbKey4	52	4 key
vbKey5	53	5 key
vbKey6	54	6 key
vbKey7	55	7 key
vbKey8	56	8 key
vbKey9	57	9 key

The following constants represent keys on the numeric keypad:

Constant	Value	Description
vbKeyNumpad0	0x60	0 key
vbKeyNumpad1	0x61	1 key
vbKeyNumpad2	0x62	2 key
vbKeyNumpad3	0x63	3 key
vbKeyNumpad4	0x64	4 key
vbKeyNumpad5	0x65	5 key
vbKeyNumpad6	0x66	6 key
vbKeyNumpad7	0x67	7 key
vbKeyNumpad8	0x68	8 key

vbKeyNumpad9	0x69	9 key
vbKeyMultiply	0x6A	MULTIPLICATION SIGN (*) key
vbKeyAdd	0x6B	PLUS SIGN (+) key
vbKeySeparator	0x6C	ENTER key
vbKeySubtract	0x6D	MINUS SIGN (–) key
vbKeyDecimal	0x6E	DECIMAL POINT (.) key
vbKeyDivide	0x6F	DIVISION SIGN (/) key

The following constants represent function keys:

Constant	Value	Description
vbKeyF1	0x70	F1 key
vbKeyF2	0x71	F2 key
vbKeyF3	0x72	F3 key
vbKeyF4	0x73	F4 key
vbKeyF5	0x74	F5 key
vbKeyF6	0x75	F6 key
vbKeyF7	0x76	F7 key
vbKeyF8	0x77	F8 key
vbKeyF9	0x78	F9 key
vbKeyF10	0x79	F10 key
vbKeyF11	0x7A	F11 key
vbKeyF12	0x7B	F12 key
vbKeyF13	0x7C	F13 key
vbKeyF14	0x7D	F14 key
vbKeyF15	0x7E	F15 key
vbKeyF16	0x7F	F16 key

System Color Constants

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscSystemColorConstantsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscSystemColorConstantsS"}

The following constants can be used anywhere in your code in place of the actual values:

Constant	Value	Description
vbScrollBars	0x80000000	Scroll bar color
vbDesktop	0x80000001	Desktop color
vbActiveTitleBar	0x80000002	Color of the title bar for the active window
vbInactiveTitleBar	0x80000003	Color of the title bar for the inactive window
vbMenuBar	0x80000004	Menu background color
vbWindowBackground	0x80000005	Window background color
vbWindowFrame	0x80000006	Window frame color
vbMenuText	0x80000007	Color of text on menus
vbWindowText	0x80000008	Color of text in windows
vbTitleBarText	0x80000009	Color of text in caption, size box, and scroll arrow
vbActiveBorder	0x8000000A	Border color of active window
vbInactiveBorder	0x8000000B	Border color of inactive window
vbApplicationWorkspace	0x8000000C	Background color of multiple-document interface (MDI) applications
vbHighlight	0x8000000D	Background color of items selected in a control
vbHighlightText	0x8000000E	Text color of items selected in a control
vbButtonFace	0x8000000F	Color of shading on the face of command buttons
vbButtonShadow	0x80000010	Color of shading on the edge of command buttons
vbGrayText	0x80000011	Grayed (disabled) text
vbButtonText	0x80000012	Text color on push buttons
vbInactiveCaptionText	0x80000013	Color of text in an inactive caption
vb3DHighlight	0x80000014	Highlight color for 3-D display elements
vb3DDKShadow	0x80000015	Darkest shadow color for 3-D display elements
vb3DLight	0x80000016	Second lightest 3-D color

		after **vb3DHighlight**
vbInfoText	0x80000017	Color of text in ToolTips
vbInfoBackground	0x80000018	Background color of ToolTips

Call Statement Example

This example illustrates how the **Call** statement is used to transfer control to a **Sub** procedure, an intrinsic function, and a dynamic-link library (DLL) procedure.

```
' Call a Sub procedure.
Call PrintToDebugWindow("Hello World")
' The above statement causes control to be passed to the following
' Sub procedure.
Sub PrintToDebugWindow(AnyString)
   Debug.Print AnyString   ' Print to Debug window.
End Sub

' Call an intrinsic function. The return value of the function is
' discarded.
Call Shell(AppName, 1)  ' AppName contains the path of the
      ' executable file.

' Call a Microsoft Windows DLL procedure. The Declare statement must be
' Private in a Class Module, but not in a standard Module.
Private Declare Sub MessageBeep Lib "User" (ByVal N As Integer)
Sub CallMyDll()
   Call MessageBeep(0)  ' Call Windows DLL procedure.
   MessageBeep 0  ' Call again without Call keyword.
End Sub
```

Choose Function Example

This example uses the **Choose** function to display a name in response to an index passed into the procedure in the `Ind` parameter.

```
Function GetChoice(Ind As Integer)
   GetChoice = Choose(Ind, "Speedy", "United", "Federal")
End Function
```

DoEvents Function Example

This example uses the **DoEvents** function to cause execution to yield to the operating system once every 1000 iterations of the loop. **DoEvents** returns the number of open Visual Basic forms, but only when the host application is Visual Basic.

```
' Create a variable to hold number of Visual Basic forms loaded
' and visible.
Dim I, OpenForms
For I = 1 To 150000  ' Start loop.
   If I Mod 1000 = 0 Then  ' If loop has repeated 1000 times.
      OpenForms = DoEvents ' Yield to operating system.
   End If
Next I' Increment loop counter.
```

Do...Loop Statement Example

This example shows how **Do...Loop** statements can be used. The inner **Do...Loop** statement loops 10 times, sets the value of the flag to **False**, and exits prematurely using the **Exit Do** statement. The outer loop exits immediately upon checking the value of the flag.

```
Dim Check, Counter
Check = True: Counter = 0   ' Initialize variables.
Do ' Outer loop.
   Do While Counter < 20    ' Inner loop.
      Counter = Counter + 1  ' Increment Counter.
      If Counter = 10 Then ' If condition is True.
         Check = False  ' Set value of flag to False.
         Exit Do  ' Exit inner loop.
      End If
   Loop
Loop Until Check = False' Exit outer loop immediately.
```

End Statement Example

This example uses the **End** Statement to end code execution if the user enters an invalid password.

```
Sub Form_Load
   Dim Password, Pword
   PassWord = "Swordfish"
   Pword = InputBox("Type in your password")
   If Pword <> PassWord Then
      MsgBox "Sorry, incorrect password"
      End
   End If
End Sub
```

Exit Statement Example

This example uses the **Exit** statement to exit a **For...Next** loop, a **Do...Loop**, and a **Sub** procedure.

```
Sub ExitStatementDemo()
Dim I, MyNum
   Do          ' Set up infinite loop.
      For I = 1 To 1000 ' Loop 1000 times.
         MyNum = Int(Rnd * 1000)   ' Generate random numbers.
         Select Case MyNum ' Evaluate random number.
            Case 7: Exit For ' If 7, exit For...Next.
            Case 29: Exit Do ' If 29, exit Do...Loop.
            Case 54: Exit Sub      ' If 54, exit Sub procedure.
         End Select
      Next I
   Loop
End Sub
```

For Each...Next Statement Example

This example uses the **For Each...Next** statement to search the **Text** property of all elements in a collection for the existence of the string "Hello". In the example, `MyObject` is a text-related object and is an element of the collection `MyCollection`. Both are generic names used for illustration purposes only.

```
Dim Found, MyObject, MyCollection
Found = False  ' Initialize variable.
For Each MyObject In MyCollection  ' Iterate through each element.
   If MyObject.Text = "Hello" Then ' If Text equals "Hello".
      Found = True   ' Set Found to True.
      Exit For ' Exit loop.
   End If
Next
```

For...Next Statement Example

This example uses the **For...Next** statement to create a string that contains 10 instances of the numbers 0 through 9, each string separated from the other by a single space. The outer loop uses a loop counter variable that is decremented each time through the loop.

```
Dim Words, Chars, MyString
For Words = 10 To 1 Step -1  ' Set up 10 repetitions.
   For Chars = 0 To 9   ' Set up 10 repetitions.
      MyString = MyString & Chars  ' Append number to string.
   Next Chars  ' Increment counter
   MyString = MyString & " " ' Append a space.
Next Words
```

GoSub...Return Statement Example

This example uses **GoSub** to call a subroutine within a **Sub** procedure. The **Return** statement causes the execution to resume at the statement immediately following the **GoSub** statement. The **Exit Sub** statement is used to prevent control from accidentally flowing into the subroutine.

```
Sub GosubDemo()
Dim Num
' Solicit a number from the user.
   Num = InputBox("Enter a positive number to be divided by 2.")
' Only use routine if user enters a positive number.
   If Num > 0 Then GoSub MyRoutine
   Debug.Print Num
   Exit Sub ' Use Exit to prevent an error.
MyRoutine:
   Num = Num/2 ' Perform the division.
   Return   ' Return control to statement.
End Sub  ' following the GoSub statement.
```

GoTo Statement Example

This example uses the **GoTo** statement to branch to line labels within a procedure.

```
Sub GotoStatementDemo()
Dim Number, MyString
   Number = 1   ' Initialize variable.
   ' Evaluate Number and branch to appropriate label.
   If Number = 1 Then GoTo Line1 Else GoTo Line2

Line1:
   MyString = "Number equals 1"
   GoTo LastLine  ' Go to LastLine.
Line2:
   ' The following statement never gets executed.
   MyString = "Number equals 2"
LastLine:
   Debug.Print MyString ' Print "Number equals 1" in
      ' Debug window.
End Sub
```

If...Then...Else Statement Example

This example shows both the block and single-line forms of the **If...Then...Else** statement. It also illustrates the use of **If TypeOf...Then...Else**.

```
Dim Number, Digits, MyString
Number = 53 ' Initialize variable.
If Number < 10 Then
   Digits = 1
ElseIf Number < 100 Then
' Condition evaluates to True so the next statement is executed.
   Digits = 2
Else
   Digits = 3
End If

' Assign a value using the single-line form of syntax.
If Digits = 1 Then MyString = "One" Else MyString = "More than one"
```

Use **If TypeOf** construct to determine whether the Control passed into a procedure is a text box.

```
Sub ControlProcessor(MyControl As Control)
   If TypeOf MyControl Is CommandButton Then
      Debug.Print "You passed in a " & TypeName(MyControl)
   ElseIf TypeOf MyControl Is CheckBox Then
      Debug.Print "You passed in a " & TypeName(MyControl)
   ElseIf TypeOf MyControl Is TextBox Then
      Debug.Print "You passed in a " & TypeName(MyControl)
   End If
End Sub
```

IIf Function Example

This example uses the **IIf** function to evaluate the `TestMe` parameter of the `CheckIt` procedure and returns the word "Large" if the amount is greater than 1000; otherwise, it returns the word "Small".

```
Function CheckIt (TestMe As Integer)
   CheckIt = IIf(TestMe > 1000, "Large", "Small")
End Function
```

On...GoSub, On...GoTo Statements Example

This example uses the **On...GoSub** and **On...GoTo** statements to branch to subroutines and line labels, respectively.

```
Sub OnGosubGotoDemo()
Dim Number, MyString
   Number = 2  ' Initialize variable.
   ' Branch to Sub2.
   On Number GoSub Sub1, Sub2      ' Execution resumes here after
      ' On...GoSub.
   On Number GoTo Line1, Line2     ' Branch to Line2.
   ' Execution does not resume here after On...GoTo.
   Exit Sub
Sub1:
   MyString = "In Sub1" : Return
Sub2:
   MyString = "In Sub2" : Return
Line1:
   MyString = "In Line1"
Line2:
   MyString = "In Line2"
End Sub
```

Partition Function Example

This example assumes you have an Orders table that contains a Freight field. It creates a select procedure that counts the number of orders for which freight cost falls into each of several ranges. The **Partition** function is used first to establish these ranges, then the SQL Count function counts the number of orders in each range. In this example, the arguments to the **Partition** function are *start* = 0, *stop* = 500, *interval* = 50. The first range would therefore be 0:49, and so on up to 500.

```
SELECT DISTINCTROW Partition([freight],0, 500, 50) AS Range,
Count(Orders.Freight) AS Count
FROM Orders
GROUP BY Partition([freight],0,500,50);
```

Select Case Statement Example

This example uses the **Select Case** statement to evaluate the value of a variable. The second **Case** clause contains the value of the variable being evaluated, and therefore only the statement associated with it is executed.

```
Dim Number
Number = 8  ' Initialize variable.
Select Case Number' Evaluate Number.
Case 1 To 5 ' Number between 1 and 5.
   Debug.Print "Between 1 and 5"
' The following is the only Case clause that evaluates to True.
Case 6, 7, 8' Number between 6 and 8.
   Debug.Print "Between 6 and 8"
Case Is > 8 And Number < 11' Number is 9 or 10.
Debug.Print "Greater than 8"
Case Else' Other values.
   Debug.Print "Not between 1 and 10"
End Select
```

Shell Function Example

This example uses the **Shell** function to run an application specified by the user.

```
' Specifying 1 as the second argument opens the application in
' normal size and gives it the focus.
Dim RetVal
RetVal = Shell("C:\WINDOWS\CALC.EXE", 1) ' Run Calculator.
```

Stop Statement Example

This example uses the **Stop** statement to suspend execution for each iteration through the **For...Next** loop.

```
Dim I
For I = 1 To 10' Start For...Next loop.
   Debug.Print I  ' Print I to Debug window.
   Stop  ' Stop during each iteration.
Next I
```

Switch Function Example

This example uses the **Switch** function to return the name of a language that matches the name of a city.

```
Function MatchUp (CityName As String)
   Matchup = Switch(CityName = "London", "English", CityName _
               = "Rome", "Italian", CityName = "Paris", "French")
End Function
```

While...Wend Statement Example

This example uses the **While...Wend** statement to increment a counter variable. The statements in the loop are executed as long as the condition evaluates to **True**.

```
Dim Counter
Counter = 0 ' Initialize variable.
While Counter < 20' Test value of Counter.
   Counter = Counter + 1    ' Increment Counter.
Wend  ' End While loop when Counter > 19.
Debug.Print Counter  ' Prints 20 in Debug window.
```

With Statement Example

This example uses the **With** statement to execute a series of statements on a single object. The object `MyObject` and its properties are generic names used for illustration purposes only.

```
With MyObject
   .Height = 100     ' Same as MyObject.Height = 100.
   .Caption = "Hello World"  ' Same as MyObject.Caption = "Hello World".
   With .Font
      .Color = Red      ' Same as MyObject.Font.Color = Red.
      .Bold = True      ' Same as MyObject.Font.Bold = True.
   End With
End With
```

Call Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmCallC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmCallX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmCallS"}

Transfers control to a **Sub** procedure, **Function** procedure, or dynamic-link library (DLL) procedure.

Syntax

[**Call**] *name* [*argumentlist*]

The **Call** statement syntax has these parts:

Part	Description
Call	Optional; keyword. If specified, you must enclose *argumentlist* in parentheses. For example: `Call MyProc(0)`
name	Required. Name of the procedure to call.
argumentlist	Optional. Comma-delimited list of variables, arrays, or expressions to pass to the procedure. Components of *argumentlist* may include the keywords **ByVal** or **ByRef** to describe how the arguments are treated by the called procedure. However, **ByVal** and **ByRef** can be used with **Call** only when calling a DLL procedure.

Remarks

You are not required to use the **Call** keyword when calling a procedure. However, if you use the **Call** keyword to call a procedure that requires arguments, *argumentlist* must be enclosed in parentheses. If you omit the **Call** keyword, you also must omit the parentheses around *argumentlist*. If you use either **Call** syntax to call any intrinsic or user-defined function, the function's return value is discarded.

To pass a whole array to a procedure, use the array name followed by empty parentheses.

Choose Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctChooseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctChooseX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctChooseS"}

Selects and returns a value from a list of arguments.

Syntax

Choose(*index***,** *choice-1*[**,** *choice-2*, *...* [**,** *choice-n*]]**)**

The **Choose** function syntax has these parts:

Part	Description
index	Required. Numeric expression or field that results in a value between 1 and the number of available choices.
choice	Required. **Variant** expression containing one of the possible choices.

Remarks

Choose returns a value from the list of choices based on the value of *index*. If *index* is 1, **Choose** returns the first choice in the list; if *index* is 2, it returns the second choice, and so on.

You can use **Choose** to look up a value in a list of possibilities. For example, if *index* evaluates to 3 and *choice-1* = "one", *choice-2* = "two", and *choice-3* = "three", **Choose** returns "three". This capability is particularly useful if *index* represents the value in an option group.

Choose evaluates every choice in the list, even though it returns only one. For this reason, you should watch for undesirable side effects. For example, if you use the **MsgBox** function as part of an expression in all the choices, a message box will be displayed for each choice as it is evaluated, even though **Choose** returns the value of only one of them.

The **Choose** function returns a **Null** if *index* is less than 1 or greater than the number of choices listed.

If *index* is not a whole number, it is rounded to the nearest whole number before being evaluated.

DoEvents Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctDoEventsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctDoEventsX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctDoEventsS"}

Yields execution so that the operating system can process other events.

Syntax

DoEvents( )

Remarks

The **DoEvents** function returns an **Integer** representing the number of open forms in stand-alone versions of Visual Basic, such as Visual Basic, Professional Edition. **DoEvents** returns zero in all other applications.

DoEvents passes control to the operating system. Control is returned after the operating system has finished processing the events in its queue and all keys in the **SendKeys** queue have been sent.

DoEvents is most useful for simple things like allowing a user to cancel a process after it has started, for example a search for a file. For long-running processes, yielding the processor is better accomplished by using a Timer or delegating the task to an ActiveX EXE component. In the latter case, the task can continue completely independent of your application, and the operating system takes case of multitasking and time slicing.

Caution Any time you temporarily yield the processor within an event procedure, make sure the procedure is not executed again from a different part of your code before the first call returns; this could cause unpredictable results. In addition, do not use **DoEvents** if other applications could possibly interact with your procedure in unforeseen ways during the time you have yielded control.

Do...Loop Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmDoC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmDoX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmDoS"}

Repeats a block of statements while a condition is **True** or until a condition becomes **True**.

Syntax

Do [{**While** | **Until**} *condition*]
  [*statements*]
  [**Exit Do**]
  [*statements*]
Loop

Or, you can use this syntax:

Do
  [*statements*]
  [**Exit Do**]
  [*statements*]
Loop [{**While** | **Until**} *condition*]

The **Do Loop** statement syntax has these parts:

Part	Description
condition	Optional. Numeric expression or string expression that is **True** or **False**. If *condition* is **Null**, *condition* is treated as **False**.
statements	One or more statements that are repeated while, or until, *condition* is **True**.

Remarks

Any number of **Exit Do** statements may be placed anywhere in the **Do…Loop** as an alternate way to exit a **Do…Loop**. **Exit Do** is often used after evaluating some condition, for example, **If…Then**, in which case the **Exit Do** statement transfers control to the statement immediately following the **Loop**.

When used within nested **Do…Loop** statements, **Exit Do** transfers control to the loop that is one nested level above the loop where **Exit Do** occurs.

End Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmEndC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmEndX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmEndS"}

Ends a procedure or block.

Syntax

End
End Function
End If
End Property
End Select
End Sub
End Type
End With

The **End** statement syntax has these forms:

Statement	Description
End	Terminates execution immediately. Never required by itself but may be placed anywhere in a procedure to end code execution, close files opened with the **Open** statement and to clear variables.
End Function	Required to end a **Function** statement.
End If	Required to end a block **If…Then…Else** statement.
End Property	Required to end a **Property Let**, **Property Get**, or **Property Set** procedure.
End Select	Required to end a **Select Case** statement.
End Sub	Required to end a **Sub** statement.
End Type	Required to end a user-defined type definition (**Type** statement).
End With	Required to end a **With** statement.

Remarks

When executed, the **End** statement resets all module-level variables and all static local variables in all modules. To preserve the value of these variables, use the **Stop** statement instead. You can then resume execution while preserving the value of those variables.

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

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

Exit Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmExitC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmExitX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmExitS"}

Exits a block of **Do…Loop**, **For...Next**, **Function**, **Sub**, or **Property** code.

Syntax

Exit Do
Exit For
Exit Function
Exit Property
Exit Sub

The **Exit** statement syntax has these forms:

Statement	Description
Exit Do	Provides a way to exit a **Do...Loop** statement. It can be used only inside a **Do...Loop** statement. **Exit Do** transfers control to the statement following the **Loop** statement. When used within nested **Do...Loop** statements, **Exit Do** transfers control to the loop that is one nested level above the loop where **Exit Do** occurs.
Exit For	Provides a way to exit a **For** loop. It can be used only in a **For...Next** or **For Each...Next** loop. **Exit For** transfers control to the statement following the **Next** statement. When used within nested **For** loops, **Exit For** transfers control to the loop that is one nested level above the loop where **Exit For** occurs.
Exit Function	Immediately exits the **Function** procedure in which it appears. Execution continues with the statement following the statement that called the **Function**.
Exit Property	Immediately exits the **Property** procedure in which it appears. Execution continues with the statement following the statement that called the **Property** procedure.
Exit Sub	Immediately exits the **Sub** procedure in which it appears. Execution continues with the statement following the statement that called the **Sub** procedure.

Remarks

Do not confuse **Exit** statements with **End** statements. **Exit** does not define the end of a structure.

For Each...Next Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmForEachC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmForEachX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmForEachS"}

Repeats a group of statements for each element in an array or collection.

Syntax

For Each *element* **In** *group*
  [*statements*]
  [**Exit For**]
  [*statements*]
Next [*element*]

The **For...Each...Next** statement syntax has these parts:

Part	Description
element	Required. Variable used to iterate through the elements of the collection or array. For collections, *element* can only be a **Variant** variable, a generic object variable, or any specific object variable. For arrays, *element* can only be a **Variant** variable.
group	Required. Name of an object collection or array (except an array of user-defined types).
statements	Optional. One or more statements that are executed on each item in *group*.

Remarks

The **For...Each** block is entered if there is at least one element in *group*. Once the loop has been entered, all the statements in the loop are executed for the first element in *group*. If there are more elements in *group*, the statements in the loop continue to execute for each element. When there are no more elements in *group*, the loop is exited and execution continues with the statement following the **Next** statement.

Any number of **Exit For** statements may be placed anywhere in the loop as an alternative way to exit. **Exit For** is often used after evaluating some condition, for example **If…Then**, and transfers control to the statement immediately following **Next**.

You can nest **For...Each...Next** loops by placing one **For...Each...Next** loop within another. However, each loop *element* must be unique.

Note If you omit *element* in a **Next** statement, execution continues as if *element* is included. If a **Next** statement is encountered before its corresponding **For** statement, an error occurs.

You can't use the **For...Each...Next** statement with an array of user-defined types because a **Variant** can't contain a user-defined type.

For...Next Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmForC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmForX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmForS"}

Repeats a group of statements a specified number of times.

Syntax

For *counter* **=** *start* **To** *end* [**Step** *step*]
   [*statements*]
   [**Exit For**]
   [*statements*]
Next [*counter*]

The **For…Next** statement syntax has these parts:

Part	Description
counter	Required. Numeric variable used as a loop counter. The variable can't be a **Boolean** or an array element.
start	Required. Initial value of *counter*.
end	Required. Final value of *counter*.
step	Optional. Amount *counter* is changed each time through the loop. If not specified, *step* defaults to one.
statements	Optional. One or more statements between **For** and **Next** that are executed the specified number of times.

Remarks

The *step* argument can be either positive or negative. The value of the *step* argument determines loop processing as follows:

Value	Loop executes if
Positive or 0	*counter* <= *end*
Negative	*counter* >= *end*

After all statements in the loop have executed, *step* is added to *counter*. At this point, either the statements in the loop execute again (based on the same test that caused the loop to execute initially), or the loop is exited and execution continues with the statement following the **Next** statement.

Tip Changing the value of *counter* while inside a loop can make it more difficult to read and debug your code.

Any number of **Exit For** statements may be placed anywhere in the loop as an alternate way to exit. **Exit For** is often used after evaluating of some condition, for example **If...Then**, and transfers control to the statement immediately following **Next**.

You can nest **For...Next** loops by placing one **For...Next** loop within another. Give each loop a unique variable name as its *counter*. The following construction is correct:

```
For I = 1 To 10
   For J = 1 To 10
      For K = 1 To 10
         ...
      Next K
   Next J
Next I
```

Note If you omit *counter* in a **Next** statement, execution continues as if *counter* is included. If a

Next statement is encountered before its corresponding **For** statement, an error occurs.

GoSub...Return Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmGoSubC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmGoSubX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmGoSubS"}

Branches to and returns from a subroutine within a procedure.

Syntax

GoSub *line*
. . .
line
. . .
Return

The *line* argument can be any line label or line number.

Remarks

You can use **GoSub** and **Return** anywhere in a procedure, but **GoSub** and the corresponding **Return** statement must be in the same procedure. A subroutine can contain more than one **Return** statement, but the first **Return** statement encountered causes the flow of execution to branch back to the statement immediately following the most recently executed **GoSub** statement.

Note You can't enter or exit **Sub** procedures with **GoSub...Return**.

Tip Creating separate procedures that you can call may provide a more structured alternative to using **GoSub...Return**.

GoTo Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmGoToC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmGoToX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmGoToS"}

Branches unconditionally to a specified line within a procedure.

Syntax

GoTo *line*

The required *line* argument can be any line label or line number.

Remarks

GoTo can branch only to lines within the procedure where it appears.

Note Too many **GoTo** statements can make code difficult to read and debug. Use structured control statements (**Do...Loop**, **For...Next**, **If...Then...Else**, **Select Case**) whenever possible.

If...Then...Else Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmIfC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmIfX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmIfS"}

Conditionally executes a group of statements, depending on the value of an expression.

Syntax

If *condition* **Then** [*statements*] [**Else** *elsestatements*]

Or, you can use the block form syntax:

If *condition* **Then**
  [*statements*]
[**ElseIf** *condition-n* **Then**
  [*elseifstatements*] . . .
[**Else**
  [*elsestatements*]]
End If

The **If...Then...Else** statement syntax has these parts:

Part	Description
condition	Required. One or more of the following two types of expressions: A numeric expression or string expression that evaluates to **True** or **False**. If *condition* is **Null**, *condition* is treated as **False**. An expression of the form **TypeOf** *objectname* **Is** *objecttype*. The *objectname* is any object reference and *objecttype* is any valid object type. The expression is **True** if *objectname* is of the object type specified by *objecttype*; otherwise it is **False**.
statements	Optional in block form; required in single-line form that has no **Else** clause. One or more statements separated by colons; executed if *condition* is **True**.
condition-n	Optional. Same as *condition*.
elseifstatements	Optional. One or more statements executed if associated *condition-n* is **True**.
elsestatements	Optional. One or more statements executed if no previous *condition* or *condition-n* expression is **True**.

Remarks

You can use the single-line form (first syntax) for short, simple tests. However, the block form (second syntax) provides more structure and flexibility than the single-line form and is usually easier to read, maintain, and debug.

Note With the single-line form, it is possible to have multiple statements executed as the result of an **If...Then** decision. All statements must be on the same line and separated by colons, as in the following statement:

```
If A > 10 Then A = A + 1 : B = B + A : C = C + B
```

A block form **If** statement must be the first statement on a line. The **Else**, **ElseIf**, and **End If** parts of the statement can have only a line number or line label preceding them. The block **If** must end with an **End If** statement.

To determine whether or not a statement is a block **If**, examine what follows the **Then** keyword. If anything other than a comment appears after **Then** on the same line, the statement is treated as a single-line **If** statement.

The **Else** and **ElseIf** clauses are both optional. You can have as many **ElseIf** clauses as you want in a block **If**, but none can appear after an **Else** clause. Block **If** statements can be nested; that is, contained within one another.

When executing a block **If** (second syntax), *condition* is tested. If *condition* is **True**, the statements following **Then** are executed. If *condition* is **False**, each **ElseIf** condition (if any) is evaluated in turn. When a **True** condition is found, the statements immediately following the associated **Then** are executed. If none of the **ElseIf** conditions are **True** (or if there are no **ElseIf** clauses), the statements following **Else** are executed. After executing the statements following **Then** or **Else**, execution continues with the statement following **End If**.

Tip **Select Case** may be more useful when evaluating a single expression that has several possible actions. However, the **TypeOf** *objectname* **Is** *objecttype* clause can't be used with the **Select Case** statement.

IIf Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIIfC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIIfX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIIFS"}

Returns one of two parts, depending on the evaluation of an expression.

Syntax

IIf(*expr*, *truepart*, *falsepart*)

The **IIf** function syntax has these named arguments:

Part	Description
expr	Required. Expression you want to evaluate.
truepart	Required. Value or expression returned if ***expr*** is **True**.
falsepart	Required. Value or expression returned if ***expr*** is **False**.

Remarks

IIf always evaluates both ***truepart*** and ***falsepart***, even though it returns only one of them. Because of this, you should watch for undesirable side effects. For example, if evaluating ***falsepart*** results in a division by zero error, an error occurs even if ***expr*** is **True**.

On...GoSub, On...GoTo Statements

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmOnGoSubC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmOnGoSubX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmOnGoSubS"}

Branch to one of several specified lines, depending on the value of an expression.

Syntax

On *expression* **GoSub** *destinationlist*
On *expression* **GoTo** *destinationlist*

The **On...GoSub** and **On...GoTo** statement syntax has these parts:

Part	Description
expression	Required. Any numeric expression that evaluates to a whole number between 0 and 255, inclusive. If *expression* is any number other than a whole number, it is rounded before it is evaluated.
destinationlist	Required. List of line numbers or line labels separated by commas.

Remarks

The value of *expression* determines which line is branched to in *destinationlist*. If the value of *expression* is less than 1 or greater than the number of items in the list, one of the following results occurs:

If *expression* is	Then
Equal to 0	Control drops to the statement following **On...GoSub** or **On...GoTo**.
Greater than number of items in list	Control drops to the statement following **On...GoSub** or **On...GoTo**.
Negative	An error occurs.
Greater than 255	An error occurs.

You can mix line numbers and line labels in the same list. You can use as many line labels and line numbers as you like with **On...GoSub** and **On...GoTo**. However, if you use more labels or numbers than fit on a single line, you must use the line-continuation character to continue the logical line onto the next physical line.

Tip **Select Case** provides a more structured and flexible way to perform multiple branching.

Partition Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctPartitionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctPartitionX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctPartitionS"}

Returns a **Variant** (**String**) indicating where a number occurs within a calculated series of ranges.

Syntax

Partition(*number*, *start*, *stop*, *interval*)

The **Partition** function syntax has these named arguments:

Part	Description
number	Required. Whole number that you want to evaluate against the ranges.
start	Required. Whole number that is the start of the overall range of numbers. The number can't be less than 0.
stop	Required. Whole number that is the end of the overall range of numbers. The number can't be equal to or less than ***start***.
interval	Required. Whole number that is the interval spanned by each range in the series from ***start*** to ***stop***. The number can't be less than 1.

Remarks

The **Partition** function identifies the particular range in which ***number*** falls and returns a **Variant** (**String**) describing that range. The **Partition** function is most useful in queries. You can create a select query that shows how many orders fall within various ranges, for example, order values from 1 to 1000, 1001 to 2000, and so on.

The following table shows how the ranges are determined using three sets of ***start***, ***stop***, and ***interval*** parts. The First Range and Last Range columns show what **Partition** returns. The ranges are represented by *lowervalue*:*uppervalue*, where the low end (*lowervalue*) of the range is separated from the high end (*uppervalue*) of the range with a colon (**:**).

start	***stop***	***interval***	**Before First**	**First Range**	**Last Range**	**After Last**
0	99	5	" :-1"	" 0: 4"	" 95: 99"	" 100: "
20	199	10	" : 19"	" 20: 29"	" 190: 199"	" 200: "
100	1010	20	" : 99"	" 100: 119"	" 1000: 1010"	" 1011: "

In the table shown above, the third line shows the result when ***start*** and ***stop*** define a set of numbers that can't be evenly divided by ***interval***. The last range extends to ***stop*** (11 numbers) even though ***interval*** is 20.

If necessary, **Partition** returns a range with enough leading spaces so that there are the same number of characters to the left and right of the colon as there are characters in ***stop***, plus one. This ensures that if you use **Partition** with other numbers, the resulting text will be handled properly during any subsequent sort operation.

If ***interval*** is 1, the range is ***number:number***, regardless of the ***start*** and ***stop*** arguments. For example, if ***interval*** is 1, ***number*** is 100 and ***stop*** is 1000, **Partition** returns " 100: 100".

If any of the parts is **Null**, **Partition** returns a **Null**.

Select Case Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmSelectCaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmSelectCaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmSelectCaseS"}

Executes one of several groups of statements, depending on the value of an expression.

Syntax

Select Case *testexpression*
  [**Case** *expressionlist-n*
    [*statements-n*]] . . .
  [**Case Else**
    [*elsestatements*]]
End Select

The **Select Case** statement syntax has these parts:

Part	Description
testexpression	Required. Any numeric expression or string expression.
expressionlist-n	Required if a **Case** appears. Delimited list of one or more of the following forms: *expression*, *expression* **To** *expression*, **Is** *comparisonoperator expression*. The **To** keyword specifies a range of values. If you use the **To** keyword, the smaller value must appear before **To**. Use the **Is** keyword with comparison operators (except **Is** and **Like**) to specify a range of values. If not supplied, the **Is** keyword is automatically inserted.
statements-n	Optional. One or more statements executed if *testexpression* matches any part of *expressionlist-n.*
elsestatements	Optional. One or more statements executed if *testexpression* doesn't match any of the **Case** clause.

Remarks

If *testexpression* matches any **Case** *expressionlist* expression, the *statements* following that **Case** clause are executed up to the next **Case** clause, or, for the last clause, up to **End Select**. Control then passes to the statement following **End Select**. If *testexpression* matches an *expressionlist* expression in more than one **Case** clause, only the statements following the first match are executed.

The **Case Else** clause is used to indicate the *elsestatements* to be executed if no match is found between the *testexpression* and an *expressionlist* in any of the other **Case** selections. Although not required, it is a good idea to have a **Case Else** statement in your **Select Case** block to handle unforeseen *testexpression* values. If no **Case** *expressionlist* matches *testexpression* and there is no **Case Else** statement, execution continues at the statement following **End Select**.

You can use multiple expressions or ranges in each **Case** clause. For example, the following line is valid:

```
Case 1 To 4, 7 To 9, 11, 13, Is > MaxNumber
```

Note The **Is** comparison operator is not the same as the **Is** keyword used in the **Select Case** statement.

You also can specify ranges and multiple expressions for character strings. In the following example, **Case** matches strings that are exactly equal to `everything`, strings that fall between `nuts` and `soup` in alphabetic order, and the current value of `TestItem`:

```
Case "everything", "nuts" To "soup", TestItem
```

Select Case statements can be nested. Each nested **Select Case** statement must have a matching **End Select** statement.

Shell Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctShellC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctShellX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctShellS"}

Runs an executable program and returns a **Variant** (**Double**) representing the program's task ID if successful, otherwise it returns zero.

Syntax

Shell(*pathname*[,*windowstyle*])

The **Shell** function syntax has these named arguments:

Part	Description
pathname	Required; **Variant** (**String**). Name of the program to execute and any required arguments or command-line switches; may include directory or folder and drive.
windowstyle	Optional. **Variant** (**Integer**) corresponding to the style of the window in which the program is to be run. If ***windowstyle*** is omitted, the program is started minimized with focus.

The ***windowstyle*** named argument has these values:

Constant	Value	Description
vbHide	0	Window is hidden and focus is passed to the hidden window.
vbNormalFocus	1	Window has focus and is restored to its original size and position.
vbMinimizedFocus	2	Window is displayed as an icon with focus.
vbMaximizedFocus	3	Window is maximized with focus.
vbNormalNoFocus	4	Window is restored to its most recent size and position. The currently active window remains active.
vbMinimizedNoFocus	6	Window is displayed as an icon. The currently active window remains active.

Remarks

If the **Shell** function successfully executes the named file, it returns the task ID of the started program. The task ID is a unique number that identifies the running program. If the **Shell** function can't start the named program, an error occurs.

Note The **Shell** function runs other programs asynchronously. This means that a program started with **Shell** might not finish executing before the statements following the **Shell** function are executed.

Stop Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmStopC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmStopX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmStopS"}

Suspends execution.

Syntax

Stop

Remarks

You can place **Stop** statements anywhere in procedures to suspend execution. Using the **Stop** statement is similar to setting a breakpoint in the code.

The **Stop** statement suspends execution, but unlike **End**, it doesn't close any files or clear variables, unless it is in a compiled executable (.exe) file.

Switch Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctSwitchC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctSwitchX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctSwitchS"}

Evaluates a list of expressions and returns a **Variant** value or an expression associated with the first expression in the list that is **True**.

Syntax

Switch(*expr-1***,** *value-1***[,** *expr-2***,** *value-2* **…** [**,** *expr-n***,***value-n*]**])**

The **Switch** function syntax has these parts:

Part	Description
expr	Required. **Variant** expression you want to evaluate.
value	Required. Value or expression to be returned if the corresponding expression is **True**.

Remarks

The **Switch** function argument list consists of pairs of expressions and values. The expressions are evaluated from left to right, and the value associated with the first expression to evaluate to **True** is returned. If the parts aren't properly paired, a run-time error occurs. For example, if *expr-1* is **True**, **Switch** returns *value-1*. If *expr-1* is **False**, but *expr-2* is **True**, **Switch** returns *value-2*, and so on.

Switch returns a **Null** value if:

- None of the expressions is **True**.
- The first **True** expression has a corresponding value that is **Null**.

Switch evaluates all of the expressions, even though it returns only one of them. For this reason, you should watch for undesirable side effects. For example, if the evaluation of any expression results in a division by zero error, an error occurs.

While...Wend Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmWhileC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmWhileX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmWhileS"}

Executes a series of statements as long as a given condition is **True**.

Syntax

While *condition*
[*statements*]
Wend

The **While...Wend** statement syntax has these parts:

Part	Description
condition	Required. Numeric expression or string expression that evaluates to **True** or **False**. If *condition* is **Null**, *condition* is treated as **False**.
statements	Optional. One or more statements executed while condition is **True**.

Remarks

If *condition* is **True**, all *statements* are executed until the **Wend** statement is encountered. Control then returns to the **While** statement and *condition* is again checked. If *condition* is still **True**, the process is repeated. If it is not **True**, execution resumes with the statement following the **Wend** statement.

While...Wend loops can be nested to any level. Each **Wend** matches the most recent **While**.

Tip The **Do...Loop** statement provides a more structured and flexible way to perform looping.

With Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmWithC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmWithX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmWithS"}

Executes a series of statements on a single object or a user-defined type.

Syntax

With *object*
[*statements*]
End With

The **With** statement syntax has these parts:

Part	Description
object	Required. Name of an object or a user-defined type.
statements	Optional. One or more statements to be executed on *object*.

Remarks

The **With** statement allows you to perform a series of statements on a specified object without requalifying the name of the object. For example, to change a number of different properties on a single object, place the property assignment statements within the **With** control structure, referring to the object once instead of referring to it with each property assignment. The following example illustrates use of the **With** statement to assign values to several properties of the same object.

```
With MyLabel
   .Height = 2000
   .Width = 2000
   .Caption = "This is MyLabel"
End With
```

Note Once a **With** block is entered, *object* can't be changed. As a result, you can't use a single **With** statement to affect a number of different objects.

You can nest **With** statements by placing one **With** block within another. However, because members of outer **With** blocks are masked within the inner **With** blocks, you must provide a fully qualified object reference in an inner **With** block to any member of an object in an outer **With** block.

Important Do not jump into or out of **With** blocks. If statements in a **With** block are executed, but either the **With** or **End With** statement is not executed, you may get errors or unpredictable behavior.

Asc Function Example

This example uses the **Asc** function to return a character code corresponding to the first letter in the string.

```
Dim MyNumber
MyNumber = Asc("A")   ' Returns 65.
MyNumber = Asc("a")   ' Returns 97.
MyNumber = Asc("Apple") ' Returns 65.
```

CBool Function Example

This example uses the **CBool** function to convert an expression to a **Boolean**. If the expression evaluates to a nonzero value, **CBool** returns **True**; otherwise, it returns **False**.

```
Dim A, B, Check
A = 5: B = 5' Initialize variables.
Check = CBool(A = B)  ' Check contains True.

A = 0 ' Define variable.
Check = CBool(A)   ' Check contains False.
```

CByte Function Example

This example uses the **CByte** function to convert an expression to a **Byte**.

```
Dim MyDouble, MyByte
MyDouble = 125.5678  ' MyDouble is a Double.
MyByte = CByte(MyDouble)' MyByte contains 126.
```

CDate Function Example

This example uses the **CDate** function to convert a string to a **Date**. In general, hard-coding dates and times as strings (as shown in this example) is not recommended. Use date literals and time literals, such as #2/12/1969# and #4:45:23 PM#, instead.

```
Dim MyDate, MyShortDate, MyTime, MyShortTime
MyDate = "February 12, 1969" ' Define date.
MyShortDate = CDate(MyDate)' Convert to Date data type.

MyTime = "4:35:47 PM"' Define time.
MyShortTime = CDate(MyTime)' Convert to Date data type.
```

CCur Function Example

This example uses the **CCur** function to convert an expression to a **Currency**.

```
Dim MyDouble, MyCurr
MyDouble = 543.214588' MyDouble is a Double.
MyCurr = CCur(MyDouble * 2)' Convert result of MyDouble * 2
      ' (1086.429176) to a
      ' Currency (1086.4292).
```

CDbl Function Example

This example uses the **CDbl** function to convert an expression to a **Double**.

```
Dim MyCurr, MyDouble
MyCurr = CCur(234.456784)  ' MyCurr is a Currency.
MyDouble = CDbl(MyCurr * 8.2 * 0.01)     ' Convert result to a Double.
```

CInt Function Example

This example uses the **CInt** function to convert a value to an **Integer**.

```
Dim MyDouble, MyInt
MyDouble = 2345.5678 ' MyDouble is a Double.
MyInt = CInt(MyDouble)   ' MyInt contains 2346.
```

CLng Function Example

This example uses the **CLng** function to convert a value to a **Long**.

```
Dim MyVal1, MyVal2, MyLong1, MyLong2
MyVal1 = 25427.45: MyVal2 = 25427.55     ' MyVal1, MyVal2 are Doubles.
MyLong1 = CLng(MyVal1)   ' MyLong1 contains 25427.
MyLong2 = CLng(MyVal2)   ' MyLong2 contains 25428.
```

CSng Function Example

This example uses the **CSng** function to convert a value to a **Single**.

```
Dim MyDouble1, MyDouble2, MySingle1, MySingle2
' MyDouble1, MyDouble2 are Doubles.
MyDouble1 = 75.3421115: MyDouble2 = 75.3421555
MySingle1 = CSng(MyDouble1)' MySingle1 contains 75.34211.
MySingle2 = CSng(MyDouble2)' MySingle2 contains 75.34216.
```

CStr Function Example

This example uses the **CStr** function to convert a numeric value to a **String**.

```
Dim MyDouble, MyString
MyDouble = 437.324' MyDouble is a Double.
MyString = CStr(MyDouble)   ' MyString contains "437.324".
```

CVar Function Example

This example uses the **CVar** function to convert an expression to a **Variant**.

```
Dim MyInt, MyVar
MyInt = 4534' MyInt is an Integer.
MyVar = CVar(MyInt & "000")' MyVar contains the string
      ' 4534000.
```

CVErr Function Example

This example uses the **CVErr** function to return a **Variant** whose **VarType** is **vbError** (10). The user-defined function `CalculateDouble` returns an error if the argument passed to it isn't a number. You can use **CVErr** to return user-defined errors from user-defined procedures or to defer handling of a run-time error. Use the **IsError** function to test if the value represents an error.

```
' Call CalculateDouble with an error-producing argument.
Sub Test()
   Debug.Print CalculateDouble("345.45robert")
End Sub
' Define CalculateDouble Function procedure.
Function CalculateDouble(Number)
   If IsNumeric(Number) Then
      CalculateDouble = Number * 2 ' Return result.
   Else
      CalculateDouble = CVErr(2001)      ' Return a user-defined error
   End If   ' number.
End Function
```

Val Function Example

This example uses the **Val** function to return the numbers contained in a string.

```
Dim MyValue
MyValue = Val("2457")' Returns 2457.
MyValue = Val(" 2 45 7")' Returns 2457.
MyValue = Val("24 and 57") ' Returns 24.
```

Conversion Functions

Asc Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctAscC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctAscX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctAscS"}

Returns an **Integer** representing the character code corresponding to the first letter in a string.

Syntax

Asc(*string***)**

The required *string* argument is any valid string expression. If the *string* contains no characters, a run-time error occurs.

Remarks

The range for returns is 0 – 255 on non-DBCS systems, but -32768 – 32767 on DBCS systems.

Note The **AscB** function is used with byte data contained in a string. Instead of returning the character code for the first character, **AscB** returns the first byte. The **AscW** function returns the Unicode character code except on platforms where Unicode is not supported, in which case, the behavior is identical to the **Asc** function.

CVErr Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctCVErrC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctCVErrX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctCVErrS"}

Returns a **Variant** of subtype **Error** containing an error number specified by the user.

Syntax

CVErr(*errornumber***)**

The required *errornumber* argument is any valid error number.

Remarks

Use the **CVErr** function to create user-defined errors in user-created procedures. For example, if you create a function that accepts several arguments and normally returns a string, you can have your function evaluate the input arguments to ensure they are within acceptable range. If they are not, it is likely your function will not return what you expect. In this event, **CVErr** allows you to return an error number that tells you what action to take.

Note that implicit conversion of an **Error** is not allowed. For example, you can't directly assign the return value of **CVErr** to a variable that is not a **Variant**. However, you can perform an explicit conversion (using **CInt**, **CDbl**, and so on) of the value returned by **CVErr** and assign that to a variable of the appropriate data type.

Val Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctValC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctValX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctValS"}

Returns the numbers contained in a string as a numeric value of appropriate type.

Syntax

Val(*string***)**

The required *string* argument is any valid string expression.

Remarks

The **Val** function stops reading the string at the first character it can't recognize as part of a number. Symbols and characters that are often considered parts of numeric values, such as dollar signs and commas, are not recognized. However, the function recognizes the radix prefixes `&O` (for octal) and `&H` (for hexadecimal). Blanks, tabs, and linefeed characters are stripped from the argument.

The following returns the value 1615198:

```
Val("    1615 198th Street N.E.")
```

In the code below, **Val** returns the decimal value -1 for the hexadecimal value shown:

```
Val("&HFFFF")
```

Note The **Val** function recognizes only the period (**.**) as a valid decimal separator. When different decimal separators can be used, for example, in international applications, use **CDbl** instead to convert a string to a number.

Type Conversion Functions

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaGrpTypeConversionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaGrpTypeConversionX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vagrpTypeConversionS"}

Each function coerces an expression to a specific data type.

Syntax

CBool(*expression***)**

CByte(*expression***)**

CCur(*expression***)**

CDate(*expression***)**

CDbl(*expression***)**

CDec(*expression***)**

CInt(*expression***)**

CLng(*expression***)**

CSng(*expression***)**

CVar(*expression***)**

CStr(*expression***)**

The required *expression* argument is any string expression or numeric expression.

Return Types

The function name determines the return type as shown in the following:

Function	Return Type	Range for *expression* argument
CBool	**Boolean**	Any valid string or numeric expression.
CByte	**Byte**	0 to 255.
CCur	**Currency**	-922,337,203,685,477.5808 to 922,337,203,685,477.5807.
CDate	**Date**	Any valid date expression.
CDbl	**Double**	-1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values.
CDec	**Decimal**	+/-79,228,162,514,264,337,593,543,950,335 for zero-scaled numbers, that is, numbers with no decimal places. For numbers with 28 decimal places, the range is +/-7.9228162514264337593543950335. The smallest possible non-zero number is 0.0000000000000000000000000001.
CInt	**Integer**	-32,768 to 32,767; fractions are rounded.
CLng	**Long**	-2,147,483,648 to 2,147,483,647; fractions are rounded.
CSng	**Single**	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values.

CVar	**Variant**	Same range as **Double** for numerics. Same range as **String** for non-numerics.
CStr	**String**	Returns for CStr depend on the *expression* argument.

Remarks

If the *expression* passed to the function is outside the range of the data type being converted to, an error occurs.

In general, you can document your code using the data-type conversion functions to show that the result of some operation should be expressed as a particular data type rather than the default data type. For example, use **CCur** to force currency arithmetic in cases where single-precision, double-precision, or integer arithmetic normally would occur.

You should use the data-type conversion functions instead of **Val** to provide internationally aware conversions from one data type to another. For example, when you use **CCur**, different decimal separators, different thousand separators, and various currency options are properly recognized depending on the locale setting of your computer.

When the fractional part is exactly 0.5, **CInt** and **CLng** always round it to the nearest even number. For example, 0.5 rounds to 0, and 1.5 rounds to 2. **CInt** and **CLng** differ from the **Fix** and **Int** functions, which truncate, rather than round, the fractional part of a number. Also, **Fix** and **Int** always return a value of the same type as is passed in.

Use the **IsDate** function to determine if *date* can be converted to a date or time. **CDate** recognizes date literals and time literals as well as some numbers that fall within the range of acceptable dates. When converting a number to a date, the whole number portion is converted to a date. Any fractional part of the number is converted to a time of day, starting at midnight.

CDate recognizes date formats according to the locale setting of your system. The correct order of day, month, and year may not be determined if it is provided in a format other than one of the recognized date settings. In addition, a long date format is not recognized if it also contains the day-of-the-week string.

A **CVDate** function is also provided for compatibility with previous versions of Visual Basic. The syntax of the **CVDate** function is identical to the **CDate** function, however, **CVDate** returns a **Variant** whose subtype is **Date** instead of an actual **Date** type. Since there is now an intrinsic **Date** type, there is no further need for **CVDate**. The same effect can be achieved by converting an expression to a **Date**, and then assigning it to a **Variant**. This technique is consistent with the conversion of all other intrinsic types to their equivalent **Variant** subtypes.

Note The **CDec** function does not return a discrete data type; instead, it always returns a **Variant** whose value has been converted to a **Decimal** subtype.

Returns for CStr

If *expression* is	CStr returns
Boolean	A string containing **True** or **False**
Date	A string containing a date in the short date format of your system
Null	A run-time error
Empty	A zero-length string ("")
Error	A string containing the word **Error** followed by the error number
Other numeric	A string containing the number

Boolean Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatBooleanC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatBooleanX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatBooleanS"}

Boolean variables are stored as 16-bit (2-byte) numbers, but they can only be **True** or **False**. **Boolean** variables display as either `True` or `False` (when **Print** is used) or `#TRUE#` or `#FALSE#` (when **Write #** is used). Use the keywords **True** and **False** to assign one of the two states to **Boolean** variables.

When other numeric types are converted to **Boolean** values, 0 becomes **False** and all other values become **True**. When **Boolean** values are converted to other data types, **False** becomes 0 and **True** becomes -1.

Byte Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatByteC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatByteX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatByteS"}

Byte variables are stored as single, unsigned, 8-bit (1-byte) numbers ranging in value from 0–255.

The **Byte** data type is useful for containing binary data.

Currency Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatCurrencyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatCurrencyX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatCurrencyS"}

Currency variables are stored as 64-bit (8-byte) numbers in an integer format, scaled by 10,000 to give a fixed-point number with 15 digits to the left of the decimal point and 4 digits to the right. This representation provides a range of -922,337,203,685,477.5808 to 922,337,203,685,477.5807. The type-declaration character for **Currency** is the at sign (**@**).

The **Currency** data type is useful for calculations involving money and for fixed-point calculations in which accuracy is particularly important.

Data Type Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vagrpDataTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vagrpDataTypeS"}

The following table shows the supported data types, including storage sizes and ranges.

Data type	Storage size	Range
Byte	1 byte	0 to 255
Boolean	2 bytes	**True** or **False**
Integer	2 bytes	-32,768 to 32,767
Long (long integer)	4 bytes	-2,147,483,648 to 2,147,483,647
Single (single-precision floating-point)	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
Double (double-precision floating-point)	8 bytes	-1.79769313486232E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values
Currency (scaled integer)	8 bytes	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	14 bytes	+/-79,228,162,514,264,337,593,543,950,335 with no decimal point; +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest non-zero number is +/-0.0000000000000000000000000001
Date	8 bytes	January 1, 100 to December 31, 9999
Object	4 bytes	Any **Object** reference
String (variable-length)	10 bytes + string length	0 to approximately 2 billion
String (fixed-length)	Length of string	1 to approximately 65,400
Variant (with numbers)	16 bytes	Any numeric value up to the range of a **Double**
Variant (with characters)	22 bytes + string length	Same range as for variable-length **String**
User-defined (using **Type**)	Number required by elements	The range of each element is the same as the range of its data type.

Note Arrays of any data type require 20 bytes of memory plus 4 bytes for each array dimension plus the number of bytes occupied by the data itself. The memory occupied by the data can be calculated by multiplying the number of data elements by the size of each element. For example, the data in a single-dimension array consisting of 4 **Integer** data elements of 2 bytes each occupies 8 bytes. The 8 bytes required for the data plus the 24 bytes of overhead brings the total memory requirement for the array to 32 bytes.

A **Variant** containing an array requires 12 bytes more than the array alone.

Date Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatDateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatDateX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatDateS"}

Date variables are stored as IEEE 64-bit (8-byte) floating-point numbers that represent dates ranging from 1 January 100 to 31 December 9999 and times from 0:00:00 to 23:59:59. Any recognizable literal date values can be assigned to **Date** variables. Date literals must be enclosed within number signs (**#**), for example, `#January 1, 1993#` or `#1 Jan 93#`.

Date variables display dates according to the short date format recognized by your computer. Times display according to the time format (either 12-hour or 24-hour) recognized by your computer.

When other numeric types are converted to **Date**, values to the left of the decimal represent date information while values to the right of the decimal represent time. Midnight is 0 and midday is 0.5. Negative whole numbers represent dates before 30 December 1899.

Decimal Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatDecimalC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatDecimalX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatDecimalS"}

Decimal variables are stored as 96-bit (12-byte) unsigned integers scaled by a variable power of 10. The power of 10 scaling factor specifies the number of digits to the right of the decimal point, and ranges from 0 to 28. With a scale of 0 (no decimal places), the largest possible value is +/-79,228,162,514,264,337,593,543,950,335. With a 28 decimal places, the largest value is +/-7.9228162514264337593543950335 and the smallest, non-zero value is +/-0.0000000000000000000000000001.

Note At this time the **Decimal** data type can only be used within a Variant, that is, you cannot declare a variable to be of type **Decimal**. You can, however, create a **Variant** whose subtype is **Decimal** using the **CDec** function.

Double Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatDoubleC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatDoubleX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatDoubleS"}

Double (double-precision floating-point) variables are stored as IEEE 64-bit (8-byte) floating-point numbers ranging in value from -1.79769313486232E308 to -4.94065645841247E-324 for negative values and from 4.94065645841247E-324 to 1.79769313486232E308 for positive values. The type-declaration character for **Double** is the number sign (**#**).

Integer Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatIntegerC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatIntegerX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatIntegerS"}

Integer variables are stored as 16-bit (2-byte) numbers ranging in value from -32,768 to 32,767. The type-declaration character for **Integer** is the percent sign (**%**).

You can also use **Integer** variables to represent enumerated values. An enumerated value can contain a finite set of unique whole numbers, each of which has special meaning in the context in which it is used. Enumerated values provide a convenient way to select among a known number of choices, for example, black = 0, white = 1, and so on. It is good programming practice to define constants using the **Const** statement for each enumerated value.

Long Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatLongC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatLongX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatLongS"}

Long (long integer) variables are stored as signed 32-bit (4-byte) numbers ranging in value from -2,147,483,648 to 2,147,483,647. The type-declaration character for **Long** is the ampersand (**&**).

Object Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatObjectC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatObjectX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatObjectS"}

Object variables are stored as 32-bit (4-byte) addresses that refer to objects. Using the **Set** statement, a variable declared as an **Object** can have any object reference assigned to it.

Note Although a variable declared with **Object** type is flexible enough to contain a reference to any object, binding to the object referenced by that variable is always late (run-time binding). To force early binding (compile-time binding), assign the object reference to a variable declared with a specific class name.

Single Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatSingleC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatSingleX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatSingleS"}

Single (single-precision floating-point) variables are stored as IEEE 32-bit (4-byte) floating-point numbers, ranging in value from -3.402823E38 to -1.401298E-45 for negative values and from 1.401298E-45 to 3.402823E38 for positive values. The type-declaration character for **Single** is the exclamation point (**!**).

String Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatStringC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatStringX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatStringS"}

There are two kinds of strings: variable-length and fixed-length strings.

- A variable-length string can contain up to approximately 2 billion (2^31) characters.
- A fixed-length string can contain 1 to approximately 64K (2^16) characters.

Note A **Public** fixed-length string can't be used in a class module.

The codes for **String** characters range from 0–255. The first 128 characters (0–127) of the character set correspond to the letters and symbols on a standard U.S. keyboard. These first 128 characters are the same as those defined by the ASCII character set. The second 128 characters (128–255) represent special characters, such as letters in international alphabets, accents, currency symbols, and fractions.The type-declaration character for **String** is the dollar sign (**$**).

User-Defined Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatUserDefinedC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatUserDefinedX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatUserDefinedS"}

Any data type you define using the **Type** statement. User-defined data types can contain one or more elements of a data type, an array, or a previously defined user-defined type. For example:

```
Type MyType
   MyName As String  ' String variable stores a name.
   MyBirthDate As Date  ' Date variable stores a birthdate.
   MySex As Integer  ' Integer variable stores sex (0 for
End Type          ' female, 1 for male).
```

Variant Data Type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vadatVariantC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vadatVariantX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vadatVariantS"}

The **Variant** data type is the data type for all variables that are not explicitly declared as some other type (using statements such as **Dim**, **Private**, **Public**, or **Static**). The **Variant** data type has no type-declaration character.

A **Variant** is a special data type that can contain any kind of data except fixed-length **String** data and user-defined types. A **Variant** can also contain the special values **Empty**, **Error**, **Nothing**, and **Null**. You can determine how the data in a **Variant** is treated using the **VarType** function or **TypeName** function.

Numeric data can be any integer or real number value ranging from -1.797693134862315E308 to -4.94066E-324 for negative values and from 4.94066E-324 to 1.797693134862315E308 for positive values. Generally, numeric **Variant** data is maintained in its original data type within the **Variant**. For example, if you assign an **Integer** to a **Variant**, subsequent operations treat the **Variant** as an **Integer**. However, if an arithmetic operation is performed on a **Variant** containing a **Byte**, an **Integer**, a **Long**, or a **Single**, and the result exceeds the normal range for the original data type, the result is promoted within the **Variant** to the next larger data type. A **Byte** is promoted to an **Integer**, an **Integer** is promoted to a **Long**, and a **Long** and a **Single** are promoted to a **Double**. An error occurs when **Variant** variables containing **Currency**, **Decimal**, and **Double** values exceed their respective ranges.

You can use the **Variant** data type in place of any data type to work with data in a more flexible way. If the contents of a **Variant** variable are digits, they may be either the string representation of the digits or their actual value, depending on the context. For example:

```
Dim MyVar As Variant
MyVar = 98052
```

In the preceding example, `MyVar` contains a numeric representation—the actual value `98052`. Arithmetic operators work as expected on **Variant** variables that contain numeric values or string data that can be interpreted as numbers. If you use the **+** operator to add `MyVar` to another **Variant** containing a number or to a variable of a numeric type, the result is an arithmetic sum.

The value **Empty** denotes a **Variant** variable that hasn't been initialized (assigned an initial value). A **Variant** containing **Empty** is 0 if it is used in a numeric context and a zero-length string ("") if it is used in a string context.

Don't confuse **Empty** with **Null**. **Null** indicates that the **Variant** variable intentionally contains no valid data.

In a **Variant**, **Error** is a special value used to indicate that an error condition has occurred in a procedure. However, unlike for other kinds of errors, normal application-level error handling does not occur. This allows you, or the application itself, to take some alternative action based on the error value. **Error** values are created by converting real numbers to error values using the **CVErr** function.

Date Function Example

This example uses the **Date** function to return the current system date.

```
Dim MyDate
MyDate = Date   ' MyDate contains the current system date.
```

Date Statement Example

This example uses the **Date** statement to set the computer system date. In the development environment, the date literal is displayed in short date format using the locale settings of your code.

```
Dim MyDate
MyDate = #February 12, 1985# ' Assign a date.
Date = MyDate  ' Change system date.
```

DateAdd Function Example

This example takes a date and, using the **DateAdd** function, displays a corresponding date a specified number of months in the future.

```
Dim FirstDate As Date' Declare variables.
Dim IntervalType As String
Dim Number As Integer
Dim Msg
IntervalType = "m"' "m" specifies months as interval.
FirstDate = InputBox("Enter a date")
Number = InputBox("Enter number of months to add")
Msg = "New date: " & DateAdd(IntervalType, Number, FirstDate)
MsgBox Msg
```

DateDiff Function Example

This example uses the **DateDiff** function to display the number of days between a given date and today.

```
Dim TheDate As Date  ' Declare variables.
Dim Msg
TheDate = InputBox("Enter a date")
Msg = "Days from today: " & DateDiff("d", Now, TheDate)
MsgBox Msg
```

DatePart Function Example

This example takes a date and, using the **DatePart** function, displays the quarter of the year in which it occurs.

```
Dim TheDate As Date  ' Declare variables.
Dim Msg
TheDate = InputBox("Enter a date:")
Msg = "Quarter: " & DatePart("q", TheDate)
MsgBox Msg
```

DateSerial Function Example

This example uses the **DateSerial** function to return the date for the specified year, month, and day.

```
Dim MyDate
' MyDate contains the date for February 12, 1969.
MyDate = DateSerial(1969, 2, 12)    ' Return a date.
```

DateValue Function Example

This example uses the **DateValue** function to convert a string to a date. You can also use date literals to directly assign a date to a **Variant** or **Date** variable, for example, MyDate = #2/12/69#.

```
Dim MyDate
MyDate = DateValue("February 12, 1969")   ' Return a date.
```

Day Function Example

This example uses the **Day** function to obtain the day of the month from a specified date. In the development environment, the date literal is displayed in short format using the locale settings of your code.

```
Dim MyDate, MyDay
MyDate = #February 12, 1969# ' Assign a date.
MyDay = Day(MyDate)   ' MyDay contains 12.
```

Hour Function Example

This example uses the **Hour** function to obtain the hour from a specified time. In the development environment, the time literal is displayed in short time format using the locale settings of your code.

```
Dim MyTime, MyHour
MyTime = #4:35:17 PM#' Assign a time.
MyHour = Hour(MyTime)' MyHour contains 16.
```

Minute Function Example

This example uses the **Minute** function to obtain the minute of the hour from a specified time. In the development environment, the time literal is displayed in short time format using the locale settings of your code.

```
Dim MyTime, MyMinute
MyTime = #4:35:17 PM#' Assign a time.
MyMinute = Minute(MyTime)   ' MyMinute contains 35.
```

Month Function Example

This example uses the **Month** function to obtain the month from a specified date. In the development environment, the date literal is displayed in short date format using the locale settings of your code.

```
Dim MyDate, MyMonth
MyDate = #February 12, 1969# ' Assign a date.
MyMonth = Month(MyDate) ' MyMonth contains 2.
```

Now Function Example

This example uses the **Now** function to return the current system date and time.

```
Dim Today
Today = Now ' Assign current system date and time.
```

Second Function Example

This example uses the **Second** function to obtain the second of the minute from a specified time. In the development environment, the time literal is displayed in short time format using the locale settings of your code.

```
Dim MyTime, MySecond
MyTime = #4:35:17 PM#' Assign a time.
MySecond = Second(MyTime)   ' MySecond contains 17.
```

Time Function Example

This example uses the **Time** function to return the current system time.

```
Dim MyTime
MyTime = Time   ' Return current system time.
```

Time Statement Example

This example uses the **Time** statement to set the computer system time to a user-defined time.

```
Dim MyTime
MyTime = #4:35:17 PM#' Assign a time.
Time = MyTime   ' Set system time to MyTime.
```

Timer Function Example

This example uses the **Timer** function to pause the application. The example also uses **DoEvents** to yield to other processes during the pause.

```
Dim PauseTime, Start, Finish, TotalTime
If (MsgBox("Press Yes to pause for 5 seconds", 4)) = vbYes Then
   PauseTime = 5  ' Set duration.
   Start = Timer  ' Set start time.
   Do While Timer < Start + PauseTime
      DoEvents ' Yield to other processes.
   Loop
   Finish = Timer ' Set end time.
   TotalTime = Finish - Start      ' Calculate total time.
   MsgBox "Paused for " & TotalTime & " seconds"
Else
   End
End If
```

TimeSerial Function Example

This example uses the **TimeSerial** function to return a time for the specified hour, minute, and second.

```
Dim MyTime
MyTime = TimeSerial(16, 35, 17)     ' MyTime contains serial
    ' representation of 4:35:17 PM.
```

TimeValue Function Example

This example uses the **TimeValue** function to convert a string to a time. You can also use date literals to directly assign a time to a **Variant** or **Date** variable, for example, MyTime = #4:35:17 PM#.

```
Dim MyTime
MyTime = TimeValue("4:35:17 PM")    ' Return a time.
```

Weekday Function Example

This example uses the **Weekday** function to obtain the day of the week from a specified date.

```
Dim MyDate, MyWeekDay
MyDate = #February 12, 1969# ' Assign a date.
MyWeekDay = Weekday(MyDate)' MyWeekDay contains 4 because
    ' MyDate represents a Wednesday.
```

Year Function Example

This example uses the **Year** function to obtain the year from a specified date. In the development environment, the date literal is displayed in short date format using the locale settings of your code.

```
Dim MyDate, MyYear
MyDate = #February 12, 1969# ' Assign a date.
MyYear = Year(MyDate)' MyYear contains 1969.
```

Date Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctDateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctDateX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctDateS"}

Returns a **Variant** (**Date**) containing the current system date.

Syntax

Date

Remarks

To set the system date, use the **Date** statement.

Date Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmDateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmDateX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmDateS"}

Sets the current system date.

Syntax

Date = *date*

For systems running Microsoft Windows 95, the required *date* specification must be a date from January 1, 1980 through December 31, 2099. For systems running Microsoft Windows NT, *date* must be a date from January 1, 1980 through December 31, 2079.

DateAdd Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctDateAddC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctDateAddX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctDateAddS"}

Returns a **Variant** (**Date**) containing a date to which a specified time interval has been added.

Syntax

DateAdd(*interval, number, date*)

The **DateAdd** function syntax has these named arguments:

Part	Description
interval	Required. String expression that is the interval of time you want to add.
number	Required. Numeric expression that is the number of intervals you want to add. It can be positive (to get dates in the future) or negative (to get dates in the past).
date	Required. **Variant** (**Date**) or literal representing date to which the interval is added.

Settings

The ***interval*** argument has these settings:

Setting	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week
h	Hour
n	Minute
s	Second

Remarks

You can use the **DateAdd** function to add or subtract a specified time interval from a date. For example, you can use **DateAdd** to calculate a date 30 days from today or a time 45 minutes from now.

To add days to ***date***, you can use Day of Year ("y"), Day ("d"), or Weekday ("w").

The **DateAdd** function won't return an invalid date. The following example adds one month to January 31:

```
DateAdd("m", 1, "31-Jan-95")
```

In this case, **DateAdd** returns 28-Feb-95, not 31-Feb-95. If ***date*** is 31-Jan-96, it returns 29-Feb-96 because 1996 is a leap year.

If the calculated date would precede the year 100 (that is, you subtract more years than are in ***date***), an error occurs.

If ***number*** isn't a **Long** value, it is rounded to the nearest whole number before being evaluated.

DateDiff Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctDateDiffC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctDateDiffX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctDateDiffS"}

Returns a **Variant** (**Long**) specifying the number of time intervals between two specified dates.

Syntax

DateDiff(*interval, date1, date2*[*, firstdayofweek*[*, firstweekofyear*]])

The **DateDiff** function syntax has these named arguments:

Part	Description
interval	Required. String expression that is the interval of time you use to calculate the difference between ***date1*** and ***date2***.
date1, ***date2***	Required; **Variant** (**Date**). Two dates you want to use in the calculation.
firstdayofweek	Optional. A constant that specifies the first day of the week. If not specified, Sunday is assumed.
firstweekofyear	Optional. A constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs.

Settings

The ***interval*** argument has these settings:

Setting	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week
h	Hour
n	Minute
s	Second

The ***firstdayofweek*** argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use the NLS API setting.
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

The ***firstweekofyear*** argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use the NLS API setting.
vbFirstJan1	1	Start with week in which January 1 occurs (default).
vbFirstFourDays	2	Start with the first week that has at least four days in the new year.
vbFirstFullWeek	3	Start with first full week of the year.

Remarks

You can use the **DateDiff** function to determine how many specified time intervals exist between two dates. For example, you might use **DateDiff** to calculate the number of days between two dates, or the number of weeks between today and the end of the year.

To calculate the number of days between ***date1*** and ***date2***, you can use either Day of year ("y") or Day ("d"). When ***interval*** is Weekday ("w"), **DateDiff** returns the number of weeks between the two dates. If ***date1*** falls on a Monday, **DateDiff** counts the number of Mondays until ***date2***. It counts ***date2*** but not ***date1***. If ***interval*** is Week ("ww"), however, the **DateDiff** function returns the number of calendar weeks between the two dates. It counts the number of Sundays between ***date1*** and ***date2***. **DateDiff** counts ***date2*** if it falls on a Sunday; but it doesn't count ***date1***, even if it does fall on a Sunday.

If ***date1*** refers to a later point in time than ***date2***, the **DateDiff** function returns a negative number.

The ***firstdayofweek*** argument affects calculations that use the "w" and "ww" interval symbols.

If *date1* or *date2* is a date literal, the specified year becomes a permanent part of that date. However, if *date1* or *date2* is enclosed in double quotation marks (" "), and you omit the year, the current year is inserted in your code each time the *date1* or *date2* expression is evaluated. This makes it possible to write code that can be used in different years.

When comparing December 31 to January 1 of the immediately succeeding year, **DateDiff** for Year ("yyyy") returns 1 even though only a day has elapsed.

DatePart Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctDatePartC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctDatePartX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctDatePartS"}

Returns a **Variant** (**Integer**) containing the specified part of a given date.

Syntax

DatePart(*interval, date*[,*firstdayofweek*[*, firstweekofyear*]])

The **DatePart** function syntax has these named arguments:

Part	Description
interval	Required. String expression that is the interval of time you want to return.
date	Required. **Variant** (**Date**) value that you want to evaluate.
firstdayofweek	Optional. A constant that specifies the first day of the week. If not specified, Sunday is assumed.
firstweekofyear	Optional. A constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs.

Settings

The ***interval*** argument has these settings:

Setting	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week
h	Hour
n	Minute
s	Second

The ***firstdayofweek*** argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use the NLS API setting.
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

The ***firstweekofyear*** argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use the NLS API setting.
vbFirstJan1	1	Start with week in which January 1 occurs (default).
vbFirstFourDays	2	Start with the first week that has at least four days in the new year.
vbFirstFullWeek	3	Start with first full week of the year.

Remarks

You can use the **DatePart** function to evaluate a date and return a specific interval of time. For example, you might use **DatePart** to calculate the day of the week or the current hour.

The ***firstdayofweek*** argument affects calculations that use the "w" and "ww" interval symbols.

If *date* is a date literal, the specified year becomes a permanent part of that date. However, if *date* is enclosed in double quotation marks (" "), and you omit the year, the current year is inserted in your code each time the *date* expression is evaluated. This makes it possible to write code that can be used in different years.

DateSerial Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctDateSerialC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctDateSerialX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctDateSerialS"}

Returns a **Variant** (**Date**) for a specified year, month, and day.

Syntax

DateSerial(*year*, *month*, *day*)

The **DateSerial** function syntax has these named arguments:

Part	Description
year	Required; **Integer**. Number between 100 and 9999, inclusive, or a numeric expression.
month	Required; **Integer**. Any numeric expression.
day	Required; **Integer**. Any numeric expression.

Remarks

To specify a date, such as December 31, 1991, the range of numbers for each **DateSerial** argument should be in the accepted range for the unit; that is, 1–31 for days and 1–12 for months. However, you can also specify relative dates for each argument using any numeric expression that represents some number of days, months, or years before or after a certain date.

The following example uses numeric expressions instead of absolute date numbers. Here the **DateSerial** function returns a date that is the day before the first day (`1 - 1`), two months before August (`8 - 2`), 10 years before 1990 (`1990 - 10`); in other words, May 31, 1980.

```
DateSerial(1990 - 10, 8 - 2, 1 - 1)
```

For the ***year*** argument, values between 0 and 99, inclusive, are interpreted as the years 1900–1999. For all other ***year*** arguments, use a four-digit year (for example, 1800).

When any argument exceeds the accepted range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 35 days, it is evaluated as one month and some number of days, depending on where in the year it is applied. If any single argument is outside the range -32,768 to 32,767, an error occurs. If the date specified by the three arguments falls outside the acceptable range of dates, an error occurs.

DateValue Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctDateValueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctDateValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctDateValueS"}

Returns a **Variant** (**Date**).

Syntax

DateValue(*date***)**

The required *date* argument is normally a string expression representing a date from January 1, 100 through December 31, 9999. However, *date* can also be any expression that can represent a date, a time, or both a date and time, in that range.

Remarks

If *date* is a string that includes only numbers separated by valid date separators, **DateValue** recognizes the order for month, day, and year according to the Short Date format you specified for your system. **DateValue** also recognizes unambiguous dates that contain month names, either in long or abbreviated form. For example, in addition to recognizing 12/30/1991 and 12/30/91, **DateValue** also recognizes December 30, 1991 and Dec 30, 1991.

If the year part of *date* is omitted, **DateValue** uses the current year from your computer's system date.

If the *date* argument includes time information, **DateValue** doesn't return it. However, if *date* includes invalid time information (such as "89:98"), an error occurs.

Day Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctDayC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctDayX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctDayS"}

Returns a **Variant** (**Integer**) specifying a whole number between 1 and 31, inclusive, representing the day of the month.

Syntax

Day(*date***)**

The required *date* argument is any **Variant**, numeric expression, string expression, or any combination, that can represent a date. If *date* contains **Null**, **Null** is returned.

Hour Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctHourC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctHourX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctHourS"}

Returns a **Variant** (**Integer**) specifying a whole number between 0 and 23, inclusive, representing the hour of the day.

Syntax

Hour(*time***)**

The required *time* argument is any **Variant**, numeric expression, string expression, or any combination, that can represent a time. If *time* contains **Null**, **Null** is returned.

Minute Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctMinuteC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctMinuteX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctMinuteS"}

Returns a **Variant** (**Integer**) specifying a whole number between 0 and 59, inclusive, representing the minute of the hour.

Syntax

Minute(*time***)**

The required *time* argument is any **Variant**, numeric expression, string expression, or any combination, that can represent a time. If *time* contains **Null**, **Null** is returned.

Month Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctMonthC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctMonthX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctMonthS"}

Returns a **Variant** (**Integer**) specifying a whole number between 1 and 12, inclusive, representing the month of the year.

Syntax

Month(*date***)**

The required *date* argument is any **Variant**, numeric expression, string expression, or any combination, that can represent a date. If *date* contains **Null**, **Null** is returned.

Now Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctNowC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctNowX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctNowS"}

Returns a **Variant** (**Date**) specifying the current date and time according your computer's system date and time.

Syntax

Now

Second Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctSecondC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctSecondX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctSecondS"}

Returns a **Variant** (**Integer**) specifying a whole number between 0 and 59, inclusive, representing the second of the minute.

Syntax

Second(*time***)**

The required *time* argument is any **Variant**, numeric expression, string expression, or any combination, that can represent a time. If *time* contains **Null**, **Null** is returned.

Time Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctTimeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctTimeX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctTimeS"}

Returns a **Variant** (**Date**) indicating the current system time.

Syntax

Time

Remarks

To set the system time, use the **Time** statement.

Time Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmTimeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmTimeX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmTimeS"}

Sets the system time.

Syntax

Time = *time*

The required *time* argument is any numeric expression, string expression, or any combination, that can represent a time.

Remarks

If *time* is a string, **Time** attempts to convert it to a time using the time separators you specified for your system. If it can't be converted to a valid time, an error occurs.

Timer Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctTimerC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctTimerX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctTimerS"}

Returns a **Single** representing the number of seconds elapsed since midnight.

Syntax

Timer

TimeSerial Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctTimeSerialC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctTimeSerialX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctTimeSerialS"}

Returns a **Variant** (**Date**) containing the time for a specific hour, minute, and second.

Syntax

TimeSerial(*hour, minute, second*)

The **TimeSerial** function syntax has these named arguments:

Part	Description
hour	Required; **Variant** (**Integer**). Number between 0 (12:00 A.M.) and 23 (11:00 P.M.), inclusive, or a numeric expression.
minute	Required; **Variant** (**Integer**). Any numeric expression.
second	Required; **Variant** (**Integer**). Any numeric expression.

Remarks

To specify a time, such as 11:59:59, the range of numbers for each **TimeSerial** argument should be in the normal range for the unit; that is, 0–23 for hours and 0–59 for minutes and seconds. However, you can also specify relative times for each argument using any numeric expression that represents some number of hours, minutes, or seconds before or after a certain time. The following example uses expressions instead of absolute time numbers. The **TimeSerial** function returns a time for 15 minutes before (`-15`) six hours before noon (`12 - 6`), or 5:45:00 A.M.

```
TimeSerial(12 - 6, -15, 0)
```

When any argument exceeds the normal range for that argument, it increments to the next larger unit as appropriate. For example, if you specify 75 minutes, it is evaluated as one hour and 15 minutes. If any single argument is outside the range -32,768 to 32,767, an error occurs. If the time specified by the three arguments causes the date to fall outside the acceptable range of dates, an error occurs.

TimeValue Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctTimeValueC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctTimeValueX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctTimeValueS"}

Returns a **Variant** (**Date**) containing the time.

Syntax

TimeValue(*time***)**

The required *time* argument is normally a string expression representing a time from 0:00:00 (12:00:00 A.M.) to 23:59:59 (11:59:59 P.M.), inclusive. However, *time* can also be any expression that represents a time in that range. If *time* contains **Null**, **Null** is returned.

Remarks

You can enter valid times using a 12-hour or 24-hour clock. For example, `"2:24PM"` and `"14:24"` are both valid *time* arguments.

If the *time* argument contains date information, **TimeValue** doesn't return it. However, if *time* includes invalid date information, an error occurs.

Weekday Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctWeekdayC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctWeekdayX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctWeekdayS"}

Returns a **Variant** (**Integer**) containing a whole number representing the day of the week.

Syntax

Weekday(*date***,** [***firstdayofweek***])**

The **Weekday** function syntax has these named arguments:

Part	Description
date	Required. **Variant**, numeric expression, string expression, or any combination, that can represent a date. If ***date*** contains **Null**, **Null** is returned.
firstdayofweek	Optional. A constant that specifies the first day of the week. If not specified, **vbSunday** is assumed.

Settings

The ***firstdayofweek*** argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use the NLS API setting.
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Return Values

The **Weekday** function can return any of these values:

Constant	Value	Description
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Year Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctYearC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctYearX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctYearS"}

Returns a **Variant** (**Integer**) containing a whole number representing the year.

Syntax

Year(*date***)**

The required *date* argument is any **Variant**, numeric expression, string expression, or any combination, that can represent a date. If *date* contains **Null**, **Null** is returned.

Array Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctArrayC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctArrayX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctArrayS"}

Returns a **Variant** containing an array.

Syntax

Array(*arglist***)**

The required *arglist* argument is a comma-delimited list of values that are assigned to the elements of the array contained within the **Variant**. If no arguments are specified, an array of zero length is created.

Remarks

The notation used to refer to an element of an array consists of the variable name followed by parentheses containing an index number indicating the desired element. In the following example, the first statement creates a variable named `A` as a **Variant**. The second statement assigns an array to variable `A`. The last statement assigns the value contained in the second array element to another variable.

```
Dim A As Variant
A = Array(10,20,30)
B = A(2)
```

The lower bound of an array created using the **Array** function is determined by the lower bound specified with the **Option Base** statement, unless **Array** is qualified with the name of the type library (for example **VBA.Array**). If qualified with the type-library name, **Array** is unaffected by **Option Base**.

Note A **Variant** that is not declared as an array can still contain an array. A **Variant** variable can contain an array of any type, except fixed-length strings and user-defined types. Although a **Variant** containing an array is conceptually different from an array whose elements are of type **Variant**, the array elements are accessed in the same way.

Const Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmConstC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmConstX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmConstS"}

Declares constants for use in place of literal values.

Syntax

[**Public** | **Private**] **Const** *constname* [**As** *type*] **=** *expression*

The **Const** statement syntax has these parts:

Part	Description
Public	Optional. Keyword used at module level to declare constants that are available to all procedures in all modules. Not allowed in procedures.
Private	Optional. Keyword used at module level to declare constants that are available only within the module where the declaration is made. Not allowed in procedures.
constname	Required. Name of the constant; follows standard variable naming conventions.
type	Optional. Data type of the constant; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String**, or **Variant**. Use a separate **As** *type* clause for each constant being declared.
expression	Required. Literal, other constant, or any combination that includes all arithmetic or logical operators except **Is**.

Remarks

Constants are private by default. Within procedures, constants are always private; their visibility can't be changed. In standard modules, the default visibility of module-level constants can be changed using the **Public** keyword. In class modules, however, constants can only be private and their visibility can't be changed using the **Public** keyword.

To combine several constant declarations on the same line, separate each constant assignment with a comma. When constant declarations are combined in this way, the **Public** or **Private** keyword, if used, applies to all of them.

You can't use variables, user-defined functions, or intrinsic Visual Basic functions (such as **Chr**) in expressions assigned to constants.

Note Constants can make your programs self-documenting and easy to modify. Unlike variables, constants can't be inadvertently changed while your program is running.

If you don't explicitly declare the constant type using **As** *type*, the constant has the data type that is most appropriate for *expression*.

Constants declared in a **Sub**, **Function**, or **Property** procedure are local to that procedure. A constant declared outside a procedure is defined throughout the module in which it is declared. You can use constants anywhere you can use an expression.

CreateObject Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctCreateObjectC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctCreateObjectX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctCreateObjectS"}

Creates and returns a reference to an ActiveX object.

Syntax

CreateObject(*class***)**

The *class* argument uses the syntax *appname***.***objecttype* and has these parts:

Part	Description
appname	Required; **Variant** (**String**). The name of the application providing the object.
objecttype	Required; **Variant** (**String**). The type or class of object to create.

Remarks

Every application that supports Automation provides at least one type of object. For example, a word processing application may provide an **Application** object, a **Document** object, and a **Toolbar** object.

To create an ActiveX object, assign the object returned by **CreateObject** to an object variable:

```
' Declare an object variable to hold the object
' reference. Dim as Object causes late binding.
Dim ExcelSheet As Object
Set ExcelSheet = CreateObject("Excel.Sheet")
```

This code starts the application creating the object, in this case, a Microsoft Excel spreadsheet. Once an object is created, you reference it in code using the object variable you defined. In the following example, you access properties and methods of the new object using the object variable, `ExcelSheet`, and other Microsoft Excel objects, including the `Application` object and the `Cells` collection.

```
' Make Excel visible through the Application object
ExcelSheet.Application.Visible = True
' Place some text in the first cell of the sheet
ExcelSheet.Cells(1, 1).Value = "This is column A, row 1"
' Save the sheet to C:\test.doc directory
ExcelSheet.SaveAs "C:\ TEST.DOC"
' Close Excel with the Quit method on the Application object
ExcelSheet.Application.Quit
' Release the object variable
Set ExcelSheet = Nothing
```

Declaring an object variable with the `As Object` clause creates a variable that can contain a reference to any type of object. However, access to the object through that variable is late bound; that is, the binding occurs when your program is run. To create an object variable that results in early binding; that is, binding when the program is compiled, declare the object variable with a specific class ID. For example, you can declare and create the following Microsoft Excel references:

```
Dim xlApp As Excel.Application
Dim xlBook As Excel.Workbook
Dim xlSheet As Excel.WorkSheet
Set xlApp = CreateObject("Excel.Application")
```

```
Set xlBook = xlApp.Workbooks.Add
Set xlSheet = xlBook.Worksheets(1)
```

The reference through an early-bound variable can give better performance, but can only contain a reference to the class specified in the declaration.

You can pass an object returned by the **CreateObject** function to a function expecting an object as an argument. For example, the following code creates and passes a reference to a Excel.Application object:

```
Call MySub (CreateObject("Excel.Application"))
```

Note Use **CreateObject** when there is no current instance of the object. If an instance of the object is already running, a new instance is started, and an object of the specified type is created. To use the current instance, or to start the application and have it load a file, use the **GetObject** function.

If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times **CreateObject** is executed.

Declare Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmDeclareC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmDeclareX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmDeclareS"}

Used at module level to declare references to external procedures in a dynamic-link library (DLL).

Syntax 1

[**Public** | **Private**] **Declare Sub** *name* **Lib "***libname***"** [**Alias "***aliasname***"**] [**(**[*arglist*]**)**]

Syntax 2

[**Public** | **Private**] **Declare Function** *name* **Lib "***libname***"** [**Alias "***aliasname***"**] [**(**[*arglist*]**)**] [**As** *type*]

The **Declare** statement syntax has these parts:

Part	Description
Public	Optional. Used to declare procedures that are available to all other procedures in all modules.
Private	Optional. Used to declare procedures that are available only within the module where the declaration is made.
Sub	Optional (either **Sub** or **Function** must appear). Indicates that the procedure doesn't return a value.
Function	Optional (either **Sub** or **Function** must appear). Indicates that the procedure returns a value that can be used in an expression.
name	Required. Any valid procedure name. Note that DLL entry points are case sensitive.
Lib	Required. Indicates that a DLL or code resource contains the procedure being declared. The **Lib** clause is required for all declarations.
libname	Required. Name of the DLL or code resource that contains the declared procedure.
Alias	Optional. Indicates that the procedure being called has another name in the DLL. This is useful when the external procedure name is the same as a keyword. You can also use **Alias** when a DLL procedure has the same name as a public variable, constant, or any other procedure in the same scope. **Alias** is also useful if any characters in the DLL procedure name aren't allowed by the DLL naming convention.
aliasname	Optional. Name of the procedure in the DLL or code resource. If the first character is not a number sign (**#**), *aliasname* is the name of the procedure's entry point in the DLL. If (**#**) is the first character, all characters that follow must indicate the ordinal number of the procedure's entry point.
arglist	Optional. List of variables representing arguments that are passed to the procedure when it is called.
type	Optional. Data type of the value returned by a **Function** procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String** (variable length only), or **Variant**, a user-defined type, or an object type.

The *arglist* argument has the following syntax and parts:

[**Optional**] [**ByVal** | **ByRef**] [**ParamArray**] *varname*[**( )**] [**As** *type*]

Part	Description
Optional	Optional. Indicates that an argument is not required. If used, all subsequent arguments in *arglist* must also be optional and declared using the **Optional** keyword. **Optional** can't be used for any argument if **ParamArray** is used.
ByVal	Optional. Indicates that the argument is passed by value.
ByRef	Indicates that the argument is passed by reference. **ByRef** is the default in Visual Basic.
ParamArray	Optional. Used only as the last argument in *arglist* to indicate that the final argument is an **Optional** array of **Variant** elements. The **ParamArray** keyword allows you to provide an arbitrary number of arguments. The **ParamArray** keyword can't be used with **ByVal**, **ByRef**, or **Optional**.
varname	Required. Name of the variable representing the argument being passed to the procedure; follows standard variable naming conventions.
( )	Required for array variables. Indicates that *varname* is an array.
type	Optional. Data type of the argument passed to the procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String** (variable length only), **Object**, **Variant**, a user-defined type, or an object type.

Remarks

For **Function** procedures, the data type of the procedure determines the data type it returns. You can use an **As** clause following *arglist* to specify the return type of the function. Within *arglist*, you can use an **As** clause to specify the data type of any of the arguments passed to the procedure. In addition to specifying any of the standard data types, you can specify **As Any** in *arglist* to inhibit type checking and allow any data type to be passed to the procedure.

Empty parentheses indicate that the **Sub** or **Function** procedure has no arguments and that Visual Basic should ensure that none are passed. In the following example, `First` takes no arguments. If you use arguments in a call to `First`, an error occurs:

```
Declare Sub First Lib "MyLib" ()
```

If you include an argument list, the number and type of arguments are checked each time the procedure is called. In the following example, `First` takes one **Long** argument:

```
Declare Sub First Lib "MyLib" (X As Long)
```

Note You can't have fixed-length strings in the argument list of a **Declare** statement; only variable-length strings can be passed to procedures. Fixed-length strings can appear as procedure arguments, but they are converted to variable-length strings before being passed.

Note The **vbNullString** constant is used when calling external procedures, where the external procedure requires a string whose value is zero. This is not the same thing as a zero-length string ("").

Deftype Statements

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vagrpDeftypeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vagrpDeftypeX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vagrpDefTypeS"}

Used at module level to set the default data type for variables, arguments passed to procedures, and the return type for **Function** and **Property Get** procedures whose names start with the specified characters.

Syntax

DefBool *letterrange*[**,** *letterrange*] **. . .**
DefByte *letterrange*[**,** *letterrange*] **. . .**
DefInt *letterrange*[**,** *letterrange*] **. . .**
DefLng *letterrange*[**,** *letterrange*] **. . .**
DefCur *letterrange*[**,** *letterrange*] **. . .**
DefSng *letterrange*[**,** *letterrange*] **. . .**
DefDbl *letterrange*[**,** *letterrange*] **. . .**
DefDec *letterrange*[**,** *letterrange*] **. . .**
DefDate *letterrange*[**,** *letterrange*] **. . .**
DefStr *letterrange*[**,** *letterrange*] **. . .**
DefObj *letterrange*[**,** *letterrange*] **. . .**
DefVar *letterrange*[**,** *letterrange*] **. . .**

The required *letterrange* argument has the following syntax:

letter1[**-***letter2*]

The *letter1* and *letter2* arguments specify the name range for which you can set a default data type. Each argument represents the first letter of the variable, argument, **Function** procedure, or **Property Get** procedure name and can be any letter of the alphabet. The case of letters in *letterrange* isn't significant.

Remarks

The statement name determines the data type:

Statement	Data Type
DefBool	**Boolean**
DefByte	**Byte**
DefInt	**Integer**
DefLng	**Long**
DefCur	**Currency**
DefSng	**Single**
DefDbl	**Double**
DefDec	**Decimal** (not currently supported)
DefDate	**Date**
DefStr	**String**
DefObj	**Object**
DefVar	**Variant**

For example, in the following program fragment, `Message` is a string variable:

```
DefStr A-Q
. . .
Message = "Out of stack space."
```

A **Def***type* statement affects only the module where it is used. For example, a **DefInt** statement in one module affects only the default data type of variables, arguments passed to procedures, and the return type for **Function** and **Property Get** procedures declared in that module; the default data type of variables, arguments, and return types in other modules is unaffected. If not explicitly declared with a **Def***type* statement, the default data type for all variables, all arguments, all **Function** procedures, and all **Property Get** procedures is **Variant**.

When you specify a letter range, it usually defines the data type for variables that begin with letters in the first 128 characters of the character set. However, when you specify the letter range A – Z, you set the default to the specified data type for all variables, including variables that begin with international characters from the extended part of the character set (128 – 255).

Once the range A – Z has been specified, you can't further redefine any subranges of variables using **Def***type* statements. Once a range has been specified, if you include a previously defined letter in another **Def***type* statement, an error occurs. However, you can explicitly specify the data type of any variable, defined or not, using a **Dim** statement with an **As** *type* clause. For example, you can use the following code at module level to define a variable as a **Double** even though the default data type is **Integer**:

```
DefInt A-Z
Dim TaxRate As Double
```

Def*type* statements don't affect elements of user-defined types because the elements must be explicitly declared.

Dim Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmDimC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmDimX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmDimS"}

Declares variables and allocates storage space.

Syntax

Dim [**WithEvents**] *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*] [**,** [**WithEvents**] *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*]] **. . .**

The **Dim** statement syntax has these parts:

Part	Description
WithEvents	Optional. Keyword that specifies that *varname* is an object variable used to respond to events triggered by an ActiveX object. **WithEvents** is valid only in class modules. You can declare as many individual variables as you like using **WithEvents**, but you can't create arrays with **WithEvents**. You can't use **New** with **WithEvents**.
varname	Required. Name of the variable; follows standard variable naming conventions.
subscripts	Optional. Dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax: [*lower* **To**] *upper* [**,** [*lower* **To**] *upper*] **. . .** When not explicitly stated in *lower*, the lower bound of an array is controlled by the **Option Base** statement. The lower bound is zero if no **Option Base** statement is present.
New	Optional. Keyword that enables implicit creation of an object. If you use **New** when declaring the object variable, a new instance of the object is created on first reference to it, so you don't have to use the **Set** statement to assign the object reference. The **New** keyword can't be used to declare variables of any intrinsic data type, can't be used to declare instances of dependent objects, and can’t be used with **WithEvents**.
type	Optional. Data type of the variable; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String** (for variable-length strings), **String** * *length* (for fixed-length strings), **Object**, **Variant**, a user-defined type, or an object type. Use a separate **As** *type* clause for each variable you declare.

Remarks

Variables declared with **Dim** at the module level are available to all procedures within the module. At the procedure level, variables are available only within the procedure.

Use the **Dim** statement at module or procedure level to declare the data type of a variable. For example, the following statement declares a variable as an **Integer**.

```
Dim NumberOfEmployees As Integer
```

Also use a **Dim** statement to declare the object type of a variable. The following declares a variable for a new instance of a worksheet.

```
Dim X As New Worksheet
```

If the **New** keyword is not used when declaring an object variable, the variable that refers to the object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**, which indicates that it doesn't refer to any particular instance of an object.

You can also use the **Dim** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Private**, **Public**, or **Dim** statement, an error occurs.

If you don't specify a data type or object type, and there is no **Def***type* statement in the module, the variable is **Variant** by default.

When variables are initialized, a numeric variable is initialized to 0, a variable-length string is initialized to a zero-length string (""), and a fixed-length string is filled with zeros. **Variant** variables are initialized to **Empty**. Each element of a user-defined type variable is initialized as if it were a separate variable.

Note When you use the **Dim** statement in a procedure, you generally put the **Dim** statement at the beginning of the procedure.

Enum Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmEnumC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmEnumX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmEnumS"}

Declares a type for an enumeration.

Syntax

[**Public** | **Private**] **Enum** *name*
  membername [= *constantexpression*]
  membername [= *constantexpression*]
    . . .
End Enum

The **Enum** statement has these parts:

Part	Description
Public	Optional. Specifies that the **Enum** type is visible throughout the project. **Enum** types are **Public** by default.
Private	Optional. Specifies that the **Enum** type is visible only within the module in which it appears.
name	Required. The name of the **Enum** type. The *name* must be a valid Visual Basic identifier and is specified as the type when declaring variables or parameters of the **Enum** type.
membername	Required. A valid Visual Basic identifier specifying the name by which a constituent element of the **Enum** type will be known.
constantexpression	Optional. Value of the element (evaluates to a **Long**). Can be another **Enum** type. If no *constantexpression* is specified, the value assigned is either zero (if it is the first *membername*), or 1 greater than the value of the immediately preceding *membername*.

Remarks

Enumeration variables are variables declared with an **Enum** type. Both variables and parameters can be declared with an **Enum** type. The elements of the **Enum** type are initialized to constant values within the **Enum** statement. The assigned values can't be modified at run time and can include both positive and negative numbers. For example:

```
Enum SecurityLevel
   IllegalEntry = -1
   SecurityLevel1 = 0
   SecurityLevel2 = 1
End Enum
```

An **Enum** statement can appear only at module level. Once the **Enum** type is defined, it can be used to declare variables, parameters, or procedures returning its type. You can't qualify an **Enum** type name with a module name. **Public Enum** types in a class module are not members of the class; however, they are written to the type library. **Enum** types defined in standard modules aren't written to type libraries. **Public Enum** types of the same name can't be defined in both standard modules and class modules, since they share the same name space. When two **Enum** types in different type libraries have the same name, but different elements, a reference to a variable of the type depends on which type library has higher priority in the **References**.

You can't use an **Enum** type as the target in a **With** block.

Erase Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmEraseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmEraseX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmEraseS"}

Reinitializes the elements of fixed-size arrays and releases dynamic-array storage space.

Syntax

Erase *arraylist*

The required *arraylist* argument is one or more comma-delimited array variables to be erased.

Remarks

Erase behaves differently depending on whether an array is fixed-size (ordinary) or dynamic. **Erase** recovers no memory for fixed-size arrays. **Erase** sets the elements of a fixed array as follows:

Type of Array	Effect of Erase on Fixed-Array Elements
Fixed numeric array	Sets each element to zero.
Fixed string array (variable length)	Sets each element to a zero-length string ("").
Fixed string array (fixed length)	Sets each element to zero.
Fixed **Variant** array	Sets each element to **Empty**.
Array of user-defined types	Sets each element as if it were a separate variable.
Array of objects	Sets each element to the special value **Nothing**.

Erase frees the memory used by dynamic arrays. Before your program can refer to the dynamic array again, it must redeclare the array variable's dimensions using a **ReDim** statement.

Event Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmEventC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmEventX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmEventS"}

Declares a user-defined event.

Syntax

[**Public**] **Event** *procedurename* [**(***arglist***)**]

The **Event** statement has these parts:

Part	Description
Public	Optional. Specifies that the **Event** visible throughout the project. **Events** types are **Public** by default. Note that events can only be raised in the module in which they are declared.
procedurename	Required. Name of the event; follows standard variable naming conventions.

The *arglist* argument has the following syntax and parts:

[**ByVal** | **ByRef**] *varname*[**( )**] [**As** *type*]

Part	Description
ByVal	Optional. Indicates that the argument is passed by value.
ByRef	Optional. Indicates that the argument is passed by reference. **ByRef** is the default in Visual Basic.
varname	Required. Name of the variable representing the argument being passed to the procedure; follows standard variable naming conventions.
type	Optional. Data type of the argument passed to the procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String** (variable length only), **Object**, **Variant**, a user-defined type, or an object type.

Remarks

Once the event has been declared, use the **RaiseEvent** statement to fire the event. A syntax error occurs if an **Event** declaration appears in a standard module. An event can't be declared to return a value. A typical event might be declared and raised as shown in the following fragments:

```
' Declare an event at module level of a class module

Event LogonCompleted (UserName as String)

Sub
   RaiseEvent LogonCompleted("AntoineJan")
End Sub
```

Note You can declare event arguments just as you do arguments of procedures, with the following exceptions: events cannot have named arguments, **Optional** arguments, or **ParamArray** arguments. Events do not have return values.

Function Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmFunctionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmFunctionX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmFunctionS"}

Declares the name, arguments, and code that form the body of a **Function** procedure.

Syntax

[**Public** | **Private** | **Friend**] [**Static**] **Function** *name* [**(***arglist***)**] [**As** *type*]
    [*statements*]
    [*name* **=** *expression*]
    [**Exit Function**]
    [*statements*]
    [*name* **=** *expression*]
End Function

The **Function** statement syntax has these parts:

Part	Description
Public	Optional. Indicates that the **Function** procedure is accessible to all other procedures in all modules. If used in a module that contains an **Option Private**, the procedure is not available outside the project.
Private	Optional. Indicates that the **Function** procedure is accessible only to other procedures in the module where it is declared.
Friend	Optional. Used only in a class module. Indicates that the **Function** procedure is visible throughout the project, but not visible to a controller of an instance of an object.
Static	Optional. Indicates that the **Function** procedure's local variables are preserved between calls. The **Static** attribute doesn't affect variables that are declared outside the **Function**, even if they are used in the procedure.
name	Required. Name of the **Function**; follows standard variable naming conventions.
arglist	Optional. List of variables representing arguments that are passed to the **Function** procedure when it is called. Multiple variables are separated by commas.
type	Optional. Data type of the value returned by the **Function** procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String**, or (except fixed length), **Object**, **Variant**, or any user-defined type. Arrays of any type can't be returned, but a **Variant** containing an array can.
statements	Optional. Any group of statements to be executed within the **Function** procedure.
expression	Optional. Return value of the **Function**.

The *arglist* argument has the following syntax and parts:

[**Optional**] [**ByVal** | **ByRef**] [**ParamArray**] *varname*[**( )**] [**As** *type*] [**=** *defaultvalue*]

Part	Description
Optional	Optional. Indicates that an argument is not required. If used, all subsequent arguments in *arglist* must also be optional

	and declared using the **Optional** keyword. **Optional** can't be used for any argument if **ParamArray** is used.
ByVal	Optional. Indicates that the argument is passed by value.
ByRef	Optional. Indicates that the argument is passed by reference. **ByRef** is the default in Visual Basic.
ParamArray	Optional. Used only as the last argument in *arglist* to indicate that the final argument is an **Optional** array of **Variant** elements. The **ParamArray** keyword allows you to provide an arbitrary number of arguments. It may not be used with **ByVal**, **ByRef**, or **Optional**.
varname	Required. Name of the variable representing the argument; follows standard variable naming conventions.
type	Optional. Data type of the argument passed to the procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported) **Date**, **String** (variable length only), **Object**, **Variant**. If the parameter is not **Optional**, a user-defined type or an object type may also be specified.
defaultvalue	Optional. Any constant or constant expression. Valid for **Optional** parameters only. If the type is an **Object**, an explicit default value can only be **Nothing**.

Remarks

If not explicitly specified using **Public**, **Private**, or **Friend**, **Function** procedures are public by default. If **Static** isn't used, the value of local variables is not preserved between calls. The **Friend** keyword can only be used in class modules. However, **Friend** procedures can be accessed by procedures in any module of a project. A **Friend** procedure does't appear in the type library of its parent class, nor can a **Friend** procedure be late bound.

Caution **Function** procedures can be recursive; that is, they can call themselves to perform a given task. However, recursion can lead to stack overflow. The **Static** keyword usually isn't used with recursive **Function** procedures.

All executable code must be in procedures. You can't define a **Function** procedure inside another **Function**, **Sub**, or **Property** procedure.

The **Exit Function** statement causes an immediate exit from a **Function** procedure. Program execution continues with the statement following the statement that called the **Function** procedure. Any number of **Exit Function** statements can appear anywhere in a **Function** procedure.

Like a **Sub** procedure, a **Function** procedure is a separate procedure that can take arguments, perform a series of statements, and change the values of its arguments. However, unlike a **Sub** procedure, you can use a **Function** procedure on the right side of an expression in the same way you use any intrinsic function, such as **Sqr**, **Cos**, or **Chr**, when you want to use the value returned by the function.

You call a **Function** procedure using the function name, followed by the argument list in parentheses, in an expression. See the **Call** statement for specific information on how to call **Function** procedures.

To return a value from a function, assign the value to the function name. Any number of such assignments can appear anywhere within the procedure. If no value is assigned to *name*, the procedure returns a default value: a numeric function returns 0, a string function returns a zero-length string (""), and a **Variant** function returns **Empty**. A function that returns an object reference returns **Nothing** if no object reference is assigned to *name* (using **Set**) within the **Function**.

The following example shows how to assign a return value to a function named `BinarySearch`. In this case, **False** is assigned to the name to indicate that some value was not found.

```
Function BinarySearch(. . .) As Boolean
. . .
   ' Value not found. Return a value of False.
   If lower > upper Then
      BinarySearch = False
      Exit Function
   End If
. . .
End Function
```

Variables used in **Function** procedures fall into two categories: those that are explicitly declared within the procedure and those that are not. Variables that are explicitly declared in a procedure (using **Dim** or the equivalent) are always local to the procedure. Variables that are used but not explicitly declared in a procedure are also local unless they are explicitly declared at some higher level outside the procedure.

Caution A procedure can use a variable that is not explicitly declared in the procedure, but a naming conflict can occur if anything you defined at the module level has the same name. If your procedure refers to an undeclared variable that has the same name as another procedure, constant, or variable, it is assumed that your procedure refers to that module-level name. Explicitly declare variables to avoid this kind of conflict. You can use an **Option Explicit** statement to force explicit declaration of variables.

Caution Visual Basic may rearrange arithmetic expressions to increase internal efficiency. Avoid using a **Function** procedure in an arithmetic expression when the function changes the value of variables in the same expression.

GetObject Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctGetObjectC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctGetObjectX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctGetObjectS"}

Returns a reference to an ActiveX object from a file.

Syntax

GetObject([*pathname*] [, *class*])

The **GetObject** function syntax has these named arguments:

Part	Description
pathname	Optional; **Variant** (**String**). The full path and name of the file containing the object to retrieve. If ***pathname*** is omitted, ***class*** is required.
class	Optional; **Variant** (**String**). A string representing the class of the object.

The ***class*** argument uses the syntax *appname*.*objecttype* and has these parts:

Part	Description
appname	Required; **Variant** (**String**). The name of the application providing the object.
objecttype	Required; **Variant** (**String**). The type or class of object to create.

Remarks

Use the **GetObject** function to access an ActiveX object from a file and assign the object to an object variable. Use the **Set** statement to assign the object returned by **GetObject** to the object variable. For example:

```
Dim CADObject As Object
Set CADObject = GetObject("C:\CAD\SCHEMA.CAD")
```

When this code is executed, the application associated with the specified ***pathname*** is started and the object in the specified file is activated.

If ***pathname*** is a zero-length string (""), **GetObject** returns a new object instance of the specified type. If the ***pathname*** argument is omitted, **GetObject** returns a currently active object of the specified type. If no object of the specified type exists, an error occurs.

Some applications allow you to activate part of a file. Add an exclamation point (**!**) to the end of the file name and follow it with a string that identifies the part of the file you want to activate. For information on how to create this string, see the documentation for the application that created the object.

For example, in a drawing application you might have multiple layers to a drawing stored in a file. You could use the following code to activate a layer within a drawing called `SCHEMA.CAD`:

```
Set LayerObject = GetObject("C:\CAD\SCHEMA.CAD!Layer3")
```

If you don't specify the object's ***class***, Automation determines the application to start and the object to activate, based on the file name you provide. Some files, however, may support more than one class of object. For example, a drawing might support three different types of objects: an **Application** object, a **Drawing** object, and a **Toolbar** object, all of which are part of the same file. To specify which object in a file you want to activate, use the optional ***class*** argument. For example:

```
Dim MyObject As Object
Set MyObject = GetObject("C:\DRAWINGS\SAMPLE.DRW", "FIGMENT.DRAWING")
```

In the above example, `FIGMENT` is the name of a drawing application and `DRAWING` is one of the object types it supports.

Once an object is activated, you reference it in code using the object variable you defined. In the preceding example, you access properties and methods of the new object using the object variable `MyObject`. For example:

```
MyObject.Line 9, 90
MyObject.InsertText 9, 100, "Hello, world."
MyObject.SaveAs "C:\DRAWINGS\SAMPLE.DRW"
```

Note Use the **GetObject** function when there is a current instance of the object or if you want to create the object with a file already loaded. If there is no current instance, and you don't want the object started with a file loaded, use the **CreateObject** function.

If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times **CreateObject** is executed. With a single-instance object, **GetObject** always returns the same instance when called with the zero-length string ("") syntax, and it causes an error if the ***pathname*** argument is omitted. You can't use **GetObject** to obtain a reference to a class created with Visual Basic.

Implements Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmImplementsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmImplementsX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmImplementsS"}

Specifies an interface or class that will be implemented in the class module in which it appears.

Syntax

Implements [*InterfaceName* | *Class*]

The required *InterfaceName* or *Class* is the name of an interface or class in a type library whose methods will be implemented by the corresponding methods in the Visual Basic class.

Remarks

An interface is a collection of prototypes representing the members (methods and properties) the interface encapsulates; that is, it contains only the declarations for the member procedures. A class provides an implementation of all of the methods and properties of one or more interfaces. Classes provide the code used when each function is called by a controller of the class. All classes implement at least one interface, which is considered the default interface of the class. In Visual Basic, any member that isn't explicitly a member of an implemented interface is implicitly a member of the default interface.

When a Visual Basic class implements an interface, the Visual Basic class provides its own versions of all the **Public** procedures specified in the type library of the Interface. In addition to providing a mapping between the interface prototypes and your procedures, the **Implements** statement causes the class to accept COM QueryInterface calls for the specified interface ID.

When you implement an interface or class, you must include all the **Public** procedures involved. A missing member in an implementation of an interface or class causes an error. If you don't place code in one of the procedures in a class you are implementing, you can raise the appropriate error (**Const** E_NOTIMPL = &H80004001) so a user of the implementation understands that a member is not implemented.

The **Implements** statement can't appear in a standard module.

LBound Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctLBoundC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctLBoundX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctLBoundS"}

Returns a **Long** containing the smallest available subscript for the indicated dimension of an array.

Syntax

LBound(*arrayname***[,** *dimension***])**

The **LBound** function syntax has these parts:

Part	Description
arrayname	Required. Name of the array variable; follows standard variable naming conventions.
dimension	Optional; **Variant** (**Long**). Whole number indicating which dimension's lower bound is returned. Use 1 for the first dimension, 2 for the second, and so on. If *dimension* is omitted, 1 is assumed.

Remarks

The **LBound** function is used with the **UBound** function to determine the size of an array. Use the **UBound** function to find the upper limit of an array dimension.

LBound returns the values in the following table for an array with the following dimensions:

```
Dim A(1 To 100, 0 To 3, -3 To 4)
```

Statement	Return Value
`LBound(A, 1)`	1
`LBound(A, 2)`	0
`LBound(A, 3)`	-3

The default lower bound for any dimension is either 0 or 1, depending on the setting of the **Option Base** statement. The base of an array created with the **Array** function is zero; it is unaffected by **Option Base**.

Arrays for which dimensions are set using the **To** clause in a **Dim**, **Private**, **Public**, **ReDim**, or **Static** statement can have any integer value as a lower bound.

Let Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmLetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmLetX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmLetS"}

Assigns the value of an expression to a variable or property.

Syntax

[**Let**] *varname* **=** *expression*

The **Let** statement syntax has these parts:

Part	Description
Let	Optional. Explicit use of the **Let** keyword is a matter of style, but it is usually omitted.
varname	Required. Name of the variable or property; follows standard variable naming conventions.
expression	Required. Value assigned to the variable or property.

Remarks

A value expression can be assigned to a variable or property only if it is of a data type that is compatible with the variable. You can't assign string expressions to numeric variables, and you can't assign numeric expressions to string variables. If you do, an error occurs at compile time.

Variant variables can be assigned either string or numeric expressions. However, the reverse is not always true. Any **Variant** except a **Null** can be assigned to a string variable, but only a **Variant** whose value can be interpreted as a number can be assigned to a numeric variable. Use the **IsNumeric** function to determine if the **Variant** can be converted to a number.

Caution Assigning an expression of one numeric type to a variable of a different numeric type coerces the value of the expression into the numeric type of the resulting variable.

Let statements can be used to assign one record variable to another only when both variables are of the same user-defined type. Use the **LSet** statement to assign record variables of different user-defined types. Use the **Set** statement to assign object references to variables.

Option Base Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmOptionBaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmOptionBaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmOptionBaseS"}

Used at module level to declare the default lower bound for array subscripts.

Syntax

Option Base {**0** | **1**}

Remarks

Because the default base is **0**, the **Option Base** statement is never required. If used, the statement must appear in a module before any procedures. **Option Base** can appear only once in a module and must precede array declarations that include dimensions.

Note The **To** clause in the **Dim**, **Private**, **Public**, **ReDim**, and **Static** statements provides a more flexible way to control the range of an array's subscripts. However, if you don't explicitly set the lower bound with a **To** clause, you can use **Option Base** to change the default lower bound to 1. The base of an array created with the the **ParamArray** keyword is zero; **Option Base** does not affect **ParamArray** (or the **Array** function, when qualified with the name of its type library, for example **VBA.Array**).

The **Option Base** statement only affects the lower bound of arrays in the module where the statement is located.

Option Compare Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmOptionCompareC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmOptionCompareX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmOptionCompareS"}

Used at module level to declare the default comparison method to use when string data is compared.

Syntax

Option Compare {**Binary** | **Text** | **Database**}

Remarks

If used, the **Option Compare** statement must appear in a module before any procedures.

The **Option Compare** statement specifies the string comparison method (**Binary**, **Text**, or **Database**) for a module. If a module doesn't include an **Option Compare** statement, the default text comparison method is **Binary**.

Option Compare Binary results in string comparisons based on a sort order derived from the internal binary representations of the characters. In Microsoft Windows, sort order is determined by the code page. A typical binary sort order is shown in the following example:

```
A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø
```

Option Compare Text results in string comparisons based on a case-insensitive text sort order determined by your system's locale. When the same characters are sorted using **Option Compare Text**, the following text sort order is produced:

```
(A=a) < ( À=à) < (B=b) < (E=e) < (Ê=ê) < (Z=z) < (Ø=ø)
```

Option Compare Database can only be used within Microsoft Access. This results in string comparisons based on the sort order determined by the locale ID of the database where the string comparisons occur.

Option Explicit Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmOptionExplicitC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmOptionExplicitX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmOptionExplicitS"}

Used at module level to force explicit declaration of all variables in that module.

Syntax

Option Explicit

Remarks

If used, the **Option Explicit** statement must appear in a module before any procedures.

When **Option Explicit** appears in a module, you must explicitly declare all variables using the **Dim**, **Private**, **Public**, **ReDim**, or **Static** statements. If you attempt to use an undeclared variable name, an error occurs at compile time.

If you don't use the **Option Explicit** statement, all undeclared variables are of **Variant** type unless the default type is otherwise specified with a **Def***type* statement.

Note Use **Option Explicit** to avoid incorrectly typing the name of an existing variable or to avoid confusion in code where the scope of the variable is not clear.

Option Private Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmOptionPrivateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmOptionPrivateX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmOptionPrivateS"}

When used in host applications that allow references across multiple projects, **Option Private Module** prevents a module's contents from being referenced outside its project. In host applications that don't permit such references, for example, standalone versions of Visual Basic, **Option Private** has no effect.

Syntax

Option Private Module

Remarks

If used, the **Option Private** statement must appear at module level, before any procedures.

When a module contains **Option Private Module**, the public parts, for example, variables, objects, and user-defined types declared at module level, are still available within the project containing the module, but they are not available to other applications or projects.

Note **Option Private** is only useful for host applications that support simultaneous loading of multiple projects and permit references between the loaded projects. For example, Microsoft Excel permits loading of multiple projects and **Option Private Module** can be used to restrict cross-project visibility. Although Visual Basic permits loading of multiple projects, references between projects are never permitted in Visual Basic.

Private Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmPrivateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmPrivateX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmPrivateS"}

Used at module level to declare private variables and allocate storage space.

Syntax

Private [**WithEvents**] *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*] [**,**[**WithEvents**] *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*]] **. . .**

The **Private** statement syntax has these parts:

Part	Description
WithEvents	Optional. Keyword that specifies that *varname* is an object variable used to respond to events triggered by an ActiveX object. **WithEvents** is valid only in class modules. You can declare as many individual variables as you like using **WithEvents**, but you can't create arrays with **WithEvents**. You can't use **New** with **WithEvents**.
varname	Required. Name of the variable; follows standard variable naming conventions.
subscripts	Optional. Dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax: [*lower* **To**] *upper* [**,**[*lower* **To**] *upper*] **. . .** When not explicitly stated in *lower*, the lower bound of an array is controlled by the **Option Base** statement. The lower bound is zero if no **Option Base** statement is present.
New	Optional. Keyword that enables implicit creation of an object. If you use **New** when declaring the object variable, a new instance of the object is created on first reference to it, so you don't have to use the **Set** statement to assign the object reference. The **New** keyword can't be used to declare variables of any intrinsic data type, can't be used to declare instances of dependent objects, and can’t be used with **WithEvents**.
type	Optional. Data type of the variable; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String** (for variable-length strings), **String** * *length* (for fixed-length strings), **Object**, **Variant**, a user-defined type, or an object type. Use a separate **As** *type* clause for each variable being defined.

Remarks

Private variables are available only to the module in which they are declared.

Use the **Private** statement to declare the data type of a variable. For example, the following statement declares a variable as an **Integer**:

```
Private NumberOfEmployees As Integer
```

You can also use a **Private** statement to declare the object type of a variable. The following statement declares a variable for a new instance of a worksheet.

```
Private X As New Worksheet
```

If the **New** keyword isn't used when declaring an object variable, the variable that refers to the object must be assigned an existing object using the **Set** statement before it can be used. Until it's assigned an object, the declared object variable has the special value **Nothing**, which indicates that it doesn't refer to any particular instance of an object.

If you don't specify a data type or object type, and there is no **Def***type* statement in the module, the variable is **Variant** by default.

You can also use the **Private** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Private**, **Public**, or **Dim** statement, an error occurs.

When variables are initialized, a numeric variable is initialized to 0, a variable-length string is initialized to a zero-length string (""), and a fixed-length string is filled with zeros. **Variant** variables are initialized to **Empty**. Each element of a user-defined type variable is initialized as if it were a separate variable.

Note When you use the **Private** statement in a procedure, you generally put the **Private** statement at the beginning of the procedure.

Property Get Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmPropertyGetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmPropertyGetX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmPropertyGetS"}

Declares the name, arguments, and code that form the body of a **Property** procedure, which gets the value of a property.

Syntax

[**Public** | **Private** | **Friend**] [**Static**] **Property Get** *name* [**(***arglist***)**] [**As** *type*]
    [*statements*]
    [*name* **=** *expression*]
    [**Exit Property**]
    [*statements*]
    [*name* **=** *expression*]
End Property

The **Property Get** statement syntax has these parts:

Part	Description
Public	Optional. Indicates that the **Property Get** procedure is accessible to all other procedures in all modules. If used in a module that contains an **Option Private** statement, the procedure is not available outside the project.
Private	Optional. Indicates that the **Property Get** procedure is accessible only to other procedures in the module where it is declared.
Friend	Optional. Used only in a class module. Indicates that the **Property Get** procedure is visible throughout the project, but not visible to a controller of an instance of an object.
Static	Optional. Indicates that the **Property Get** procedure's local variables are preserved between calls. The **Static** attribute doesn't affect variables that are declared outside the **Property Get** procedure, even if they are used in the procedure.
name	Required. Name of the **Property Get** procedure; follows standard variable naming conventions, except that the name can be the same as a **Property Let** or **Property Set** procedure in the same module.
arglist	Optional. List of variables representing arguments that are passed to the **Property Get** procedure when it is called. Multiple arguments are separated by commas. The name and data type of each argument in a **Property Get** procedure must be the same as the corresponding argument in a **Property Let** procedure (if one exists).
type	Optional. Data type of the value returned by the **Property Get** procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String** (except fixed length), **Object**, **Variant**, or user-defined type. Arrays of any type can't be returned, but a **Variant** containing an array can. The return *type* of a **Property Get** procedure must be the same data type as the last (or sometimes the only) argument in a corresponding **Property Let** procedure (if

	one exists) that defines the value assigned to the property on the right side of an expression.
statements	Optional. Any group of statements to be executed within the body of the **Property Get** procedure.
expression	Optional. Value of the property returned by the procedure defined by the **Property Get** statement.

The *arglist* argument has the following syntax and parts:

[**Optional**] [**ByVal** | **ByRef**] [**ParamArray**] *varname*[**( )**] [**As** *type*] [**=** *defaultvalue*]

Part	Description
Optional	Optional. Indicates that an argument is not required. If used, all subsequent arguments in *arglist* must also be optional and declared using the **Optional** keyword.
ByVal	Optional. Indicates that the argument is passed by value.
ByRef	Optional. Indicates that the argument is passed by reference. **ByRef** is the default in Visual Basic.
ParamArray	Optional. Used only as the last argument in *arglist* to indicate that the final argument is an **Optional** array of **Variant** elements. The **ParamArray** keyword allows you to provide an arbitrary number of arguments. It may not be used with **ByVal**, **ByRef**, or **Optional**.
varname	Required. Name of the variable representing the argument; follows standard variable naming conventions.
type	Optional. Data type of the argument passed to the procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String** (variable length only), **Object**, **Variant**. If the parameter is not **Optional**, a user-defined type or an object type may also be specified.
defaultvalue	Optional. Any constant or constant expression. Valid for **Optional** parameters only. If the type is an **Object**, an explicit default value can only be **Nothing**.

Remarks

If not explicitly specified using **Public**, **Private**, or **Friend**, **Property** procedures are public by default. If **Static** is not used, the value of local variables is not preserved between calls. The **Friend** keyword can only be used in class modules. However, **Friend** procedures can be accessed by procedures in any module of a project. A **Friend** procedure doesn't appear in the type library of its parent class, nor can a **Friend** procedure be late bound.

All executable code must be in procedures. You can't define a **Property Get** procedure inside another **Property**, **Sub**, or **Function** procedure.

The **Exit Property** statement causes an immediate exit from a **Property Get** procedure. Program execution continues with the statement following the statement that called the **Property Get** procedure. Any number of **Exit Property** statements can appear anywhere in a **Property Get** procedure.

Like a **Sub** and **Property Let** procedure, a **Property Get** procedure is a separate procedure that can take arguments, perform a series of statements, and change the values of its arguments. However, unlike a **Sub** or **Property Let** procedure, you can use a **Property Get** procedure on the right side of an expression in the same way you use a **Function** or a property name when you want to return the value of a property.

Property Let Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmPropertyLetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmPropertyLetX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmPropertyLetS"}

Declares the name, arguments, and code that form the body of a **Property Let** procedure, which assigns a value to a property.

Syntax

[**Public** | **Private** | **Friend**] [**Static**] **Property Let** *name* **(**[*arglist***,**] *value***)**
    [*statements*]
    [**Exit Property**]
    [*statements*]
End Property

The **Property Let** statement syntax has these parts:

Part	Description
Public	Optional. Indicates that the **Property Let** procedure is accessible to all other procedures in all modules. If used in a module that contains an **Option Private** statement, the procedure is not available outside the project.
Private	Optional. Indicates that the **Property Let** procedure is accessible only to other procedures in the module where it is declared.
Friend	Optional. Used only in a class module. Indicates that the **Property Let** procedure is visible throughout the project, but not visible to a controller of an instance of an object.
Static	Optional. Indicates that the **Property Let** procedure's local variables are preserved between calls. The **Static** attribute doesn't affect variables that are declared outside the **Property Let** procedure, even if they are used in the procedure.
name	Required. Name of the **Property Let** procedure; follows standard variable naming conventions, except that the name can be the same as a **Property Get** or **Property Set** procedure in the same module.
arglist	Required. List of variables representing arguments that are passed to the **Property Let** procedure when it is called. Multiple arguments are separated by commas. The name and data type of each argument in a **Property Let** procedure must be the same as the corresponding argument in a **Property Get** procedure.
value	Required. Variable to contain the value to be assigned to the property. When the procedure is called, this argument appears on the right side of the calling expression. The data type of *value* must be the same as the return type of the corresponding **Property Get** procedure.
statements	Optional. Any group of statements to be executed within the **Property Let** procedure.

The *arglist* argument has the following syntax and parts:

[**Optional**] [**ByVal** | **ByRef**] [**ParamArray**] *varname*[**( )**] [**As** *type*] [**=** *defaultvalue*]

Part	Description
Optional	Optional. Indicates that an argument is not required. If used, all subsequent arguments in *arglist* must also be optional and declared using the **Optional** keyword. Note that it is not possible for the right side of a **Property Let** expression to be **Optional**.
ByVal	Optional. Indicates that the argument is passed by value.
ByRef	Optional. Indicates that the argument is passed by reference. **ByRef** is the default in Visual Basic.
ParamArray	Optional. Used only as the last argument in *arglist* to indicate that the final argument is an **Optional** array of **Variant** elements. The **ParamArray** keyword allows you to provide an arbitrary number of arguments. It may not be used with **ByVal**, **ByRef**, or **Optional**.
varname	Required. Name of the variable representing the argument; follows standard variable naming conventions.
type	Optional. Data type of the argument passed to the procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String** (variable length only), **Object**, **Variant**. If the parameter is not **Optional**, a user-defined type, or an object type may also be specified.
defaultvalue	Optional. Any constant or constant expression. Valid for **Optional** parameters only. If the type is an **Object**, an explicit default value can only be **Nothing**.

Note Every **Property Let** statement must define at least one argument for the procedure it defines. That argument (or the last argument if there is more than one) contains the actual value to be assigned to the property when the procedure defined by the **Property Let** statement is invoked. That argument is referred to as *value* in the preceding syntax.

Remarks

If not explicitly specified using **Public**, **Private**, or **Friend**, **Property** procedures are public by default. If **Static** isn't used, the value of local variables is not preserved between calls. The **Friend** keyword can only be used in class modules. However, **Friend** procedures can be accessed by procedures in any module of a project. A **Friend** procedure doesn't appear in the type library of its parent class, nor can a **Friend** procedure be late bound.

All executable code must be in procedures. You can't define a **Property Let** procedure inside another **Property**, **Sub**, or **Function** procedure.

The **Exit Property** statement causes an immediate exit from a **Property Let** procedure. Program execution continues with the statement following the statement that called the **Property Let** procedure. Any number of **Exit Property** statements can appear anywhere in a **Property Let** procedure.

Like a **Function** and **Property Get** procedure, a **Property Let** procedure is a separate procedure that can take arguments, perform a series of statements, and change the value of its arguments. However, unlike a **Function** and **Property Get** procedure, both of which return a value, you can only use a **Property Let** procedure on the left side of a property assignment expression or **Let** statement.

Property Set Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmPropertySetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmPropertySetX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmPropertySetS"}

Declares the name, arguments, and code that form the body of a **Property** procedure, which sets a reference to an object.

Syntax

[**Public** | **Private** | **Friend**] [**Static**] **Property Set** *name* **(**[*arglist***,**] *reference***)**
    [*statements*]
    [**Exit Property**]
    [*statements*]
End Property

The **Property Set** statement syntax has these parts:

Part	Description
Optional	Optional. Indicates that the argument may or may not be supplied by the caller.
Public	Optional. Indicates that the **Property Set** procedure is accessible to all other procedures in all modules. If used in a module that contains an **Option Private** statement, the procedure is not available outside the project.
Private	Optional. Indicates that the **Property Set** procedure is accessible only to other procedures in the module where it is declared.
Friend	Optional. Used only in a class module. Indicates that the **Property Set** procedure is visible throughout the project, but not visible to a controller of an instance of an object.
Static	Optional. Indicates that the **Property Set** procedure's local variables are preserved between calls. The **Static** attribute doesn't affect variables that are declared outside the **Property Set** procedure, even if they are used in the procedure.
name	Required. Name of the **Property Set** procedure; follows standard variable naming conventions, except that the name can be the same as a **Property Get** or **Property Let** procedure in the same module.
arglist	Required. List of variables representing arguments that are passed to the **Property Set** procedure when it is called. Multiple arguments are separated by commas.
reference	Required. Variable containing the object reference used on the right side of the object reference assignment.
statements	Optional. Any group of statements to be executed within the body of the **Property** procedure.

The *arglist* argument has the following syntax and parts:

[**Optional**] [**ByVal** | **ByRef**] [**ParamArray**] *varname*[**( )**] [**As** *type*] [**=** *defaultvalue*]

Part	Description
Optional	Optional. Indicates that an argument is not required. If used, all subsequent arguments in *arglist* must also be optional and declared using the **Optional** keyword. Note that it is not

	possible for the right side of a **Property Set** expression to be **Optional**.
ByVal	Optional. Indicates that the argument is passed by value.
ByRef	Optional. Indicates that the argument is passed by reference. **ByRef** is the default in Visual Basic.
ParamArray	Optional. Used only as the last argument in *arglist* to indicate that the final argument is an **Optional** array of **Variant** elements. The **ParamArray** keyword allows you to provide an arbitrary number of arguments. It may not be used with **ByVal**, **ByRef**, or **Optional**.
varname	Required. Name of the variable representing the argument; follows standard variable naming conventions.
type	Optional. Data type of the argument passed to the procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String** (variable length only), **Object**, **Variant**. If the parameter is not **Optional**, a user-defined type, or an object type may also be specified.
defaultvalue	Optional. Any constant or constant expression. Valid for **Optional** parameters only. If the type is an **Object**, an explicit default value can only be **Nothing**.

Note Every **Property Set** statement must define at least one argument for the procedure it defines. That argument (or the last argument if there is more than one) contains the actual object reference for the property when the procedure defined by the **Property Set** statement is invoked. It is referred to as *reference* in the preceding syntax. It can't be **Optional**.

Remarks

If not explicitly specified using **Public**, **Private**, or **Friend**, **Property** procedures are public by default. If **Static** isn't used, the value of local variables is not preserved between calls. The **Friend** keyword can only be used in class modules. However, **Friend** procedures can be accessed by procedures in any module of a project. A **Friend** procedure doesn't appear in the type library of its parent class, nor can a **Friend** procedure be late bound.

All executable code must be in procedures. You can't define a **Property Set** procedure inside another **Property**, **Sub**, or **Function** procedure.

The **Exit Property** statement causes an immediate exit from a **Property Set** procedure. Program execution continues with the statement following the statement that called the **Property Set** procedure. Any number of **Exit Property** statements can appear anywhere in a **Property Set** procedure.

Like a **Function** and **Property Get** procedure, a **Property Set** procedure is a separate procedure that can take arguments, perform a series of statements, and change the value of its arguments. However, unlike a **Function** and **Property Get** procedure, both of which return a value, you can only use a **Property Set** procedure on the left side of an object reference assignment (**Set** statement).

Public Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmPublicC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmPublicX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmPublicS"}

Used at module level to declare public variables and allocate storage space.

Syntax

Public [**WithEvents**] *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*] [**,**[**WithEvents**] *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*]] **. . .**

The **Public** statement syntax has these parts:

Part	Description
WithEvents	Optional. Keyword specifying that *varname* is an object variable used to respond to events triggered by an ActiveX object. **WithEvents** is valid only in class modules. You can declare as many individual variables as you like using **WithEvents**, but you can't create arrays with **WithEvents**. You can't use **New** with **WithEvents**.
varname	Required. Name of the variable; follows standard variable naming conventions.
subscripts	Optional. Dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax: [*lower* **To**] *upper* [**,**[*lower* **To**] *upper*] **. . .** When not explicitly stated in *lower*, the lower bound of an array is controlled by the **Option Base** statement. The lower bound is zero if no **Option Base** statement is present.
New	Optional. Keyword that enables implicit creation of an object. If you use **New** when declaring the object variable, a new instance of the object is created on first reference to it, so you don't have to use the **Set** statement to assign the object reference. The **New** keyword can't be used to declare variables of any intrinsic data type, can't be used to declare instances of dependent objects, and can't be used with **WithEvents**.
type	Optional. Data type of the variable; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String**, (for variable-length strings), **String** * *length* (for fixed-length strings), **Object**, **Variant**, a user-defined type, or an object type. Use a separate **As** *type* clause for each variable being defined.

Remarks

Variables declared using the **Public** statement are available to all procedures in all modules in all applications unless **Option Private Module** is in effect; in which case, the variables are public only within the project in which they reside.

Caution The **Public** statement can't be used in a class module to declare a fixed-length string variable.

Use the **Public** statement to declare the data type of a variable. For example, the following statement declares a variable as an **Integer**:

```
Public NumberOfEmployees As Integer
```

Also use a **Public** statement to declare the object type of a variable. The following statement declares a variable for a new instance of a worksheet.

```
Public X As New Worksheet
```

If the **New** keyword is not used when declaring an object variable, the variable that refers to the object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**, which indicates that it doesn't refer to any particular instance of an object.

You can also use the **Public** statement with empty parentheses to declare a dynamic array. After declaring a dynamic array, use the **ReDim** statement within a procedure to define the number of dimensions and elements in the array. If you try to redeclare a dimension for an array variable whose size was explicitly specified in a **Private**, **Public**, or **Dim** statement, an error occurs.

If you don't specify a data type or object type and there is no **Def***type* statement in the module, the variable is **Variant** by default.

When variables are initialized, a numeric variable is initialized to 0, a variable-length string is initialized to a zero-length string (""), and a fixed-length string is filled with zeros. **Variant** variables are initialized to **Empty**. Each element of a user-defined type variable is initialized as if it were a separate variable.

ReDim Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmReDimC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmReDimX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmReDimS"}

Used at procedure level to reallocate storage space for dynamic array variables.

Syntax

ReDim [**Preserve**] *varname*(*subscripts*) [**As** *type*] [**,** *varname*(*subscripts*) [**As** *type*]] **. . .**

The **ReDim** statement syntax has these parts:

Part	Description
Preserve	Optional. Keyword used to preserve the data in an existing array when you change the size of the last dimension.
varname	Required. Name of the variable; follows standard variable naming conventions.
subscripts	Required. Dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax: [*lower* **To**] *upper* [**,**[*lower* **To**] *upper*] **. . .** When not explicitly stated in *lower*, the lower bound of an array is controlled by the **Option Base** statement. The lower bound is zero if no **Option Base** statement is present.
type	Optional. Data type of the variable; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String** (for variable-length strings), **String** * *length* (for fixed-length strings), **Object**, **Variant**, a user-defined type, or an object type. Use a separate **As** *type* clause for each variable being defined. For a **Variant** containing an array, *type* describes the type of each element of the array, but doesn't change the **Variant** to some other type.

Remarks

The **ReDim** statement is used to size or resize a dynamic array that has already been formally declared using a **Private**, **Public**, or **Dim** statement with empty parentheses (without dimension subscripts).

You can use the **ReDim** statement repeatedly to change the number of elements and dimensions in an array. However, you can't declare an array of one data type and later use **ReDim** to change the array to another data type, unless the array is contained in a **Variant**. If the array is contained in a **Variant**, the type of the elements can be changed using an **As** *type* clause, unless you're using the **Preserve** keyword, in which case, no changes of data type are permitted.

If you use the **Preserve** keyword, you can resize only the last array dimension and you can't change the number of dimensions at all. For example, if your array has only one dimension, you can resize that dimension because it is the last and only dimension. However, if your array has two or more dimensions, you can change the size of only the last dimension and still preserve the contents of the array. The following example shows how you can increase the size of the last dimension of a dynamic array without erasing any existing data contained in the array.

```
ReDim X(10, 10, 10)
. . .
ReDim Preserve X(10, 10, 15)
```

Similarly, when you use **Preserve**, you can change the size of the array only by changing the upper

bound; changing the lower bound causes an error.

If you make an array smaller than it was, data in the eliminated elements will be lost. If you pass an array to a procedure by reference, you can't redimension the array within the procedure.

When variables are initialized, a numeric variable is initialized to 0, a variable-length string is initialized to a zero-length string (""), and a fixed-length string is filled with zeros. **Variant** variables are initialized to **Empty**. Each element of a user-defined type variable is initialized as if it were a separate variable. A variable that refers to an object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**, which indicates that it doesn't refer to any particular instance of an object.

Caution The **ReDim** statement acts as a declarative statement if the variable it declares doesn't exist at module level or procedure level. If another variable with the same name is created later, even in a wider scope, **ReDim** will refer to the later variable and won't 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.

Note To resize an array contained in a **Variant**, you must explicitly declare the **Variant** variable before attempting to resize its array.

Rem Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmRemC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmRemX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmRemS"}

Used to include explanatory remarks in a program.

Syntax

Rem *comment*

You can also use the following syntax:

' *comment*

The optional *comment* argument is the text of any comment you want to include. A space is required between the **Rem** keyword and *comment*.

Remarks

If you use line numbers or line labels, you can branch from a **GoTo** or **GoSub** statement to a line containing a **Rem** statement. Execution continues with the first executable statement following the **Rem** statement. If the **Rem** keyword follows other statements on a line, it must be separated from the statements by a colon (**:**).

You can use an apostrophe (**'**) instead of the **Rem** keyword. When you use an apostrophe, the colon is not required after other statements.

Set Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmSetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmSetX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmSetS"}

Assigns an object reference to a variable or property.

Syntax

Set *objectvar* **=** {[**New**] *objectexpression* | **Nothing**}

The **Set** statement syntax has these parts:

Part	Description
objectvar	Required. Name of the variable or property; follows standard variable naming conventions.
New	Optional. **New** is usually used during declaration to enable implicit object creation. When **New** is used with **Set**, it creates a new instance of the class. If *objectvar* contained a reference to an object, that reference is released when the new one is assigned. The **New** keyword can't be used to create new instances of any intrinsic data type and can't be used to create dependent objects.
objectexpression	Required. Expression consisting of the name of an object, another declared variable of the same object type, or a function or method that returns an object of the same object type.
Nothing	Optional. Discontinues association of *objectvar* with any specific object. Assigning **Nothing** to *objectvar* releases all the system and memory resources associated with the previously referenced object when no other variable refers to it.

Remarks

To be valid, *objectvar* must be an object type consistent with the object being assigned to it.

The **Dim**, **Private**, **Public**, **ReDim**, and **Static** statements only declare a variable that refers to an object. No actual object is referred to until you use the **Set** statement to assign a specific object.

The following example illustrates how **Dim** is used to declare an array with the type `Form1`. No instance of `Form1` actually exists. **Set** then assigns references to new instances of `Form1` to the `myChildForms` variable. Such code might be used to create child forms in an MDI application.

```
Dim myChildForms(1 to 4) As Form1
Set myChildForms(1) = New Form1
Set myChildForms(2) = New Form1
Set myChildForms(3) = New Form1
Set myChildForms(4) = New Form1
```

Generally, when you use **Set** to assign an object reference to a variable, no copy of the object is created for that variable. Instead, a reference to the object is created. More than one object variable can refer to the same object. Because such variables are references to the object rather than copies of the object, any change in the object is reflected in all variables that refer to it. However, when you use the **New** keyword in the **Set** statement, you are actually creating an instance of the object.

Static Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmStaticC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmStaticX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmStaticS"}

Used at procedure level to declare variables and allocate storage space. Variables declared with the **Static** statement retain their values as long as the code is running.

Syntax

Static *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*] [**,** *varname*[**(**[*subscripts*]**)**] [**As** [**New**] *type*]] **. . .**

The **Static** statement syntax has these parts:

Part	Description
varname	Required. Name of the variable; follows standard variable naming conventions.
subscripts	Optional. Dimensions of an array variable; up to 60 multiple dimensions may be declared. The *subscripts* argument uses the following syntax: [*lower* **To**] *upper* [**,**[*lower* **To**] *upper*] **. . .** When not explicitly stated in *lower*, the lower bound of an array is controlled by the **Option Base** statement. The lower bound is zero if no **Option Base** statement is present.
New	Optional. Keyword that enables implicit creation of an object. If you use **New** when declaring the object variable, a new instance of the object is created on first reference to it, so you don't have to use the **Set** statement to assign the object reference. The **New** keyword can't be used to declare variables of any intrinsic data type and can't be used to declare instances of dependent objects.
type	Optional. Data type of the variable; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String**, (for variable-length strings), **String** * *length* (for fixed-length strings), **Object**, **Variant**, a user-defined type, or an object type. Use a separate **As** *type* clause for each variable being defined.

Remarks

Once module code is running, variables declared with the **Static** statement retain their value until the module is reset or restarted. Use the **Static** statement in nonstatic procedures to explicitly declare variables that are visible only within the procedure, but whose lifetime is the same as the module in which the procedure is defined.

Use a **Static** statement within a procedure to declare the data type of a variable that retains its value between procedure calls. For example, the following statement declares a fixed-size array of integers:

```
Static EmployeeNumber(200) As Integer
```

The following statement declares a variable for a new instance of a worksheet:

```
Static X As New Worksheet
```

If the **New** keyword isn't used when declaring an object variable, the variable that refers to the object must be assigned an existing object using the **Set** statement before it can be used. Until it is assigned an object, the declared object variable has the special value **Nothing**, which indicates that it doesn't refer to any particular instance of an object. When you use the **New** keyword in the declaration, an instance of the object is created on the first reference to the object.

If you don't specify a data type or object type, and there is no **Def***type* statement in the module, the variable is **Variant** by default.

Note The **Static** statement and the **Static** keyword are similar, but used for different effects. If you declare a procedure using the **Static** keyword (as in `Static Sub CountSales ()`), the storage space for all local variables within the procedure is allocated once, and the value of the variables is preserved for the entire time the program is running. For nonstatic procedures, storage space for variables is allocated each time the procedure is called and released when the procedure is exited. The **Static** statement is used to declare specific variables within nonstatic procedures to preserve their value for as long as the program is running.

When variables are initialized, a numeric variable is initialized to 0, a variable-length string is initialized to a zero-length string (""), and a fixed-length string is filled with zeros. **Variant** variables are initialized to **Empty**. Each element of a user-defined type variable is initialized as if it were a separate variable.

Note When you use **Static** statements within a procedure, put them at the beginning of the procedure with other declarative statements such as **Dim**.

Sub Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmSubC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmSubX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmSubS"}

Declares the name, arguments, and code that form the body of a **Sub** procedure.

Syntax

[**Private** | **Public** | **Friend**] [**Static**] **Sub** *name* [**(***arglist***)**]
    [*statements*]
    [**Exit Sub**]
    [*statements*]
End Sub

The **Sub** statement syntax has these parts:

Part	Description
Public	Optional. Indicates that the **Sub** procedure is accessible to all other procedures in all modules. If used in a module that contains an **Option Private** statement, the procedure is not available outside the project.
Private	Optional. Indicates that the **Sub** procedure is accessible only to other procedures in the module where it is declared.
Friend	Optional. Used only in a class module. Indicates that the **Sub** procedure is visible throughout the project, but not visible to a controller of an instance of an object.
Static	Optional. Indicates that the **Sub** procedure's local variables are preserved between calls. The **Static** attribute doesn't affect variables that are declared outside the **Sub**, even if they are used in the procedure.
name	Required. Name of the **Sub**; follows standard variable naming conventions.
arglist	Optional. List of variables representing arguments that are passed to the **Sub** procedure when it is called. Multiple variables are separated by commas.
statements	Optional. Any group of statements to be executed within the **Sub** procedure.

The *arglist* argument has the following syntax and parts:

[**Optional**] [**ByVal** | **ByRef**] [**ParamArray**] *varname*[**( )**] [**As** *type*] [**=** *defaultvalue*]

Part	Description
Optional	Optional. Keyword indicating that an argument is not required. If used, all subsequent arguments in *arglist* must also be optional and declared using the **Optional** keyword. **Optional** can't be used for any argument if **ParamArray** is used.
ByVal	Optional. Indicates that the argument is passed by value.
ByRef	Optional. Indicates that the argument is passed by reference. **ByRef** is the default in Visual Basic.
ParamArray	Optional. Used only as the last argument in *arglist* to indicate that the final argument is an **Optional** array of **Variant** elements. The **ParamArray** keyword allows you to provide an arbitrary number of arguments. **ParamArray**

	can't be used with **ByVal**, **ByRef**, or **Optional**.
varname	Required. Name of the variable representing the argument; follows standard variable naming conventions.
type	Optional. Data type of the argument passed to the procedure; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String** (variable-length only), **Object**, **Variant**. If the parameter is not **Optional**, a user-defined type, or an object type may also be specified.
defaultvalue	Optional. Any constant or constant expression. Valid for **Optional** parameters only. If the type is an **Object**, an explicit default value can only be **Nothing**.

Remarks

If not explicitly specified using **Public**, **Private**, or **Friend**, **Sub** procedures are public by default. If **Static** isn't used, the value of local variables is not preserved between calls. The **Friend** keyword can only be used in class modules. However, **Friend** procedures can be accessed by procedures in any module of a project. A **Friend** procedure doesn't appear in the type library of its parent class, nor can a **Friend** procedure be late bound.

Caution **Sub** procedures can be recursive; that is, they can call themselves to perform a given task. However, recursion can lead to stack overflow. The **Static** keyword usually is not used with recursive **Sub** procedures.

All executable code must be in procedures. You can't define a **Sub** procedure inside another **Sub**, **Function**, or **Property** procedure.

The **Exit Sub** keywords cause an immediate exit from a **Sub** procedure. Program execution continues with the statement following the statement that called the **Sub** procedure. Any number of **Exit Sub** statements can appear anywhere in a **Sub** procedure.

Like a **Function** procedure, a **Sub** procedure is a separate procedure that can take arguments, perform a series of statements, and change the value of its arguments. However, unlike a **Function** procedure, which returns a value, a **Sub** procedure can't be used in an expression.

You call a **Sub** procedure using the procedure name followed by the argument list. See the **Call** statement for specific information on how to call **Sub** procedures.

Variables used in **Sub** procedures fall into two categories: those that are explicitly declared within the procedure and those that are not. Variables that are explicitly declared in a procedure (using **Dim** or the equivalent) are always local to the procedure. Variables that are used but not explicitly declared in a procedure are also local unless they are explicitly declared at some higher level outside the procedure.

Caution A procedure can use a variable that is not explicitly declared in the procedure, but a naming conflict can occur if anything you defined at the module level has the same name. If your procedure refers to an undeclared variable that has the same name as another procedure, constant or variable, it is assumed that your procedure is referring to that module-level name. To avoid this kind of conflict, explicitly declare variables. You can use an **Option Explicit** statement to force explicit declaration of variables.

Note You can't use **GoSub**, **GoTo**, or **Return** to enter or exit a **Sub** procedure.

Type Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmTypeX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmTypeS"}

Used at module level to define a user-defined data type containing one or more elements.

Syntax

[**Private** | **Public**] **Type** *varname*
elementname [**(**[*subscripts*])] **As** *type*
[*elementname* [**(**[*subscripts*])] **As** *type*]
. . .
End Type

The **Type** statement syntax has these parts:

Part	Description
Public	Optional. Used to declare user-defined types that are available to all procedures in all modules in all projects.
Private	Optional. Used to declare user-defined types that are available only within the module where the declaration is made.
varname	Required. Name of the user-defined type; follows standard variable naming conventions.
elementname	Required. Name of an element of the user-defined type. Element names also follow standard variable naming conventions, except that keywords can be used.
subscripts	Optional. Dimensions of an array element. Use only parentheses when declaring an array whose size can change. The *subscripts* argument uses the following syntax: [*lower* **To**] *upper* [**,**[*lower* **To**] *upper*] **. . .** When not explicitly stated in *lower*, the lower bound of an array is controlled by the **Option Base** statement. The lower bound is zero if no **Option Base** statement is present.
type	Required. Data type of the element; may be **Byte**, **Boolean**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Decimal** (not currently supported), **Date**, **String** (for variable-length strings), **String** * *length* (for fixed-length strings), **Object**, **Variant**, another user-defined type, or an object type.

Remarks

The **Type** statement can be used only at module level. Once you have declared a user-defined type using the **Type** statement, you can declare a variable of that type anywhere within the scope of the declaration. Use **Dim**, **Private**, **Public**, **ReDim**, or **Static** to declare a variable of a user-defined type.

In standard modules, user-defined types are public by default. This visibility can be changed using the **Private** keyword. In class modules, however, user-defined types can only be private and the visibility can't be changed using the **Public** keyword.

Line numbers and line labels aren't allowed in **Type...End Type** blocks.

User-defined types are often used with data records, which frequently consist of a number of related elements of different data types.

The following example shows the use of fixed-size arrays in a user-defined type:

```
Type StateData
   CityCode (1 To 100) As Integer  ' Declare a static array.
   County As String * 30
End Type

Dim Washington(1 To 100) As StateData
```

In the preceding example, `StateData` includes the `CityCode` static array, and the record `Washington` has the same structure as `StateData`.

When you declare a fixed-size array within a user-defined type, its dimensions must be declared with numeric literals or constants rather than variables.

The setting of the **Option Base** statement determines the lower bound for arrays within user-defined types.

UBound Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctUBoundC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctUBoundX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctUBoundS"}

Returns a **Long** containing the largest available subscript for the indicated dimension of an array.

Syntax

UBound(*arrayname***[,** *dimension***])**

The **UBound** function syntax has these parts:

Part	Description
arrayname	Required. Name of the array variable; follows standard variable naming conventions.
dimension	Optional; **Variant** (**Long**). Whole number indicating which dimension's upper bound is returned. Use 1 for the first dimension, 2 for the second, and so on. If *dimension* is omitted, 1 is assumed.

Remarks

The **UBound** function is used with the **LBound** function to determine the size of an array. Use the **LBound** function to find the lower limit of an array dimension.

UBound returns the following values for an array with these dimensions:

```
Dim A(1 To 100, 0 To 3, -3 To 4)
```

Statement	Return Value
`UBound(A, 1)`	100
`UBound(A, 2)`	3
`UBound(A, 3)`	4

Array Function Example

This example uses the **Array** function to return a **Variant** containing an array.

```
Dim MyWeek, MyDay
MyWeek = Array("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")
' Return values assume lower bound set to 1 (using Option Base
' statement).
MyDay = MyWeek(2)  ' MyDay contains "Tue".
MyDay = MyWeek(4)  ' MyDay contains "Thu".
```

Const Statement Example

This example uses the **Const** statement to declare constants for use in place of literal values. **Public** constants are declared in the General section of a standard module, rather than a class module. **Private** constants are declared in the General section of any type of module.

```
' Constants are Private by default.
Const MyVar = 459

' Declare Public constant.
Public Const MyString = "HELP"

' Declare Private Integer constant.
Private Const MyInt As Integer = 5

' Declare multiple constants on same line.
Const MyStr = "Hello", MyDouble As Double = 3.4567
```

CreateObject Function Example

This example uses the **CreateObject** function to set a reference (`xlApp`) to Microsoft Excel. It uses the reference to access the **Visible** property of Microsoft Excel, and then uses the Microsoft Excel **Quit** method to close it. Finally, the reference itself is released.

```
Dim xlApp As Object  ' Declare variable to hold the reference.

Set xlApp = CreateObject("excel.application")
   ' You may have to set Visible property to True
   ' if you want to see the application.
xlApp.Visible = True
   ' Use xlApp to access Microsoft Excel's
   ' other objects.
xlApp.Quit  ' When you finish, use the Quit method to close
Set xlApp = Nothing  ' the application, then release the reference.
```

Declare Statement Example

This example shows how the **Declare** statement is used at the module level of a standard module to declare a reference to an external procedure in a dynamic-link library (DLL). You can place the **Declare** statements in class modules if the **Declare** statements are **Private**.

```
' In Microsoft Windows (16-bit):
Declare Sub MessageBeep Lib "User" (ByVal N As Integer)
' Assume SomeBeep is an alias for the procedure name.
Declare Sub MessageBeep Lib "User" Alias "SomeBeep"(ByVal N As Integer)
' Use an ordinal in the Alias clause to call GetWinFlags.
Declare Function GetWinFlags Lib "Kernel" Alias "#132"() As Long

' In 32-bit Microsoft Windows systems, specify the library USER32.DLL,
' rather than USER.DLL. You can use conditional compilation to write
' code that can run on either Win32 or Win16.
#If Win32 Then
   Declare Sub MessageBeep Lib "User32" (ByVal N As Long)
#Else
   Declare Sub MessageBeep Lib "User" (ByVal N As Integer)
#End If
```

Deftype Statements Example

This example shows various uses of the **Def*type*** statements to set default data types of variables and function procedures whose names start with specified characters. The default data type can be overridden only by explicit assignment using the **Dim** statement. **Def*type*** statements can only be used at the module level (that is, not within procedures).

```
' Variable names beginning with A through K default to Integer.
DefInt A-K
' Variable names beginning with L through Z default to String.
DefStr L-Z
CalcVar = 4 ' Initialize Integer.
StringVar = "Hello there"  ' Initialize String.
AnyVar = "Hello"  ' Causes "Type mismatch" error.
Dim Calc As Double' Explicitly set the type to Double.
Calc = 2.3455  ' Assign a Double.

' Deftype statements also apply to function procedures.
CalcNum = ATestFunction(4) ' Call user-defined function.
' ATestFunction function procedure definition.
Function ATestFunction(INumber)
   ATestFunction = INumber * 2     ' Return value is an integer.
End Function
```

Dim Statement Example

This example shows the **Dim** statement used to declare variables. It also shows the **Dim** statement used to declare arrays. The default lower bound for array subscripts is 0 and can be overridden at the module level using the **Option Base** statement.

```
' AnyValue and MyValue are declared as Variant by default with values
' set to Empty.
Dim AnyValue, MyValue

' Explicitly declare a variable of type Integer.
Dim Number As Integer

' Multiple declarations on a single line. AnotherVar is of type Variant
' because its type is omitted.
Dim AnotherVar, Choice As Boolean, BirthDate As Date

' DayArray is an array of Variants with 51 elements indexed, from
' 0 thru 50, assuming Option Base is set to 0 (default) for
' the current module.
Dim DayArray(50)

' Matrix is a two-dimensional array of integers.
Dim Matrix(3, 4) As Integer

' MyMatrix is a three-dimensional array of doubles with explicit
' bounds.
Dim MyMatrix(1 To 5,  4 To 9,  3 To 5) As Double

' BirthDay is an array of dates with indexes from 1 to 10.
Dim BirthDay(1 To 10) As Date

' MyArray is a dynamic array of variants.
Dim MyArray()
```

Enum Statement Example

The following example shows the **Enum** statement used to define a collection of named constants. In this case, the constants are colors you might choose to design data entry forms for a database.

```
Public Enum InterfaceColors
   icMistyRose = &HE1E4FF&
   icSlateGray = &H908070&
   icDodgerBlue = &HFF901E&
   icDeepSkyBlue = &HFFBF00&
   icSpringGreen = &H7FFF00&
   icForestGreen = &H228B22&
   icGoldenrod = &H20A5DA&
   icFirebrick = &H2222B2&
End Enum
```

Erase Statement Example

This example uses the **Erase** statement to reinitialize the elements of fixed-size arrays and deallocate dynamic-array storage space.

```
' Declare array variables.
Dim NumArray(10) As Integer' Integer array.
Dim StrVarArray(10) As String' Variable-string array.
Dim StrFixArray(10) As String * 10 ' Fixed-string array.
Dim VarArray(10) As Variant' Variant array.
Dim DynamicArray() As Integer' Dynamic array.
ReDim DynamicArray(10)  ' Allocate storage space.
Erase NumArray ' Each element set to 0.
Erase StrVarArray ' Each element set to zero-length
   ' string ("").
Erase StrFixArray ' Each element set to 0.
Erase VarArray ' Each element set to Empty.
Erase DynamicArray' Free memory used by array.
```

Event Statement Example

The following example uses events to count off seconds during a demonstration of the fastest 100 meter race. The code illustrates all of the event-related methods, properties, and statements, including the **Event** statement.

The class that raises an event is the event source, and the classes that implement the event are the sinks. An event source can have multiple sinks for the events it generates. When the class raises the event, that event is fired on every class that has elected to sink events for that instance of the object.

The example also uses a form (`Form1`) with a button (`Command1`), a label (`Label1`), and two text boxes (`Text1` and `Text2`). When you click the button, the first text box displays "From Now" and the second starts to count seconds. When the full time (9.84 seconds) has elapsed, the first text box displays "Until Now" and the second displays "9.84"

The code for `Form1` specifies the initial and terminal states of the form. It also contains the code executed when events are raised.

```
Option Explicit

Private WithEvents mText As TimerState

Private Sub Command1_Click()
Text1.Text = "From Now"
    Text1.Refresh
    Text2.Text = "0"
    Text2.Refresh
Call mText.TimerTask(9.84)
End Sub

Private Sub Form_Load()
    Command1.Caption = "Click to Start Timer"
    Text1.Text = ""
    Text2.Text = ""
    Label1.Caption = "The fastest 100 meter run took this long:"
    Set mText = New TimerState
    End Sub

Private Sub mText_ChangeText()
    Text1.Text = "Until Now"
    Text2.Text = "9.84"
End Sub

Private Sub mText_UpdateTime(ByVal dblJump As Double)
    Text2.Text = Str(Format(dblJump, "0"))
    DoEvents
End Sub
```

The remaining code is in a class module named TimerState. The **Event** statements declare the procedures initiated when events are raised.

```
Option Explicit
Public Event UpdateTime(ByVal dblJump As Double)
Public Event ChangeText()

Public Sub TimerTask(ByVal Duration As Double)
    Dim dblStart As Double
    Dim dblSecond As Double
```

```
    Dim dblSoFar As Double
    dblStart = Timer
    dblSoFar = dblStart

    Do While Timer < dblStart + Duration
        If Timer - dblSoFar >= 1 Then
            dblSoFar = dblSoFar + 1
            RaiseEvent UpdateTime(Timer - dblStart)
        End If
    Loop

    RaiseEvent ChangeText

End Sub
```

Function Statement Example

This example uses the **Function** statement to declare the name, arguments, and code that form the body of a **Function** procedure. The last example uses hard-typed, initialized **Optional** arguments.

```
' The following user-defined function returns the square root of the
' argument passed to it.
Function CalculateSquareRoot(NumberArg As Double) As Double
   If NumberArg < 0 Then   ' Evaluate argument.
      Exit Function  ' Exit to calling procedure.
   Else
      CalculateSquareRoot = Sqr(NumberArg)     ' Return square root.
   End If
End Function
```

Using the **ParamArray** keyword enables a function to accept a variable number of arguments. In the following definition, `FirstArg` is passed by value.

```
Function CalcSum(ByVal FirstArg As Integer, ParamArray OtherArgs())
Dim ReturnValue
' If the function is invoked as follows:
ReturnValue = CalcSum(4, 3 ,2 ,1)
' Local variables are assigned the following values: FirstArg = 4,
' OtherArgs(1) = 3, OtherArgs(2) = 2, and so on, assuming default
' lower bound for arrays = 1.
```

Optional arguments can have default values and types other than **Variant**.

```
' If a function's arguments are defined as follows:
Function MyFunc(MyStr As String, Optional MyArg1 As _ Integer = 5, Optional
MyArg2 = "Dolly")
Dim RetVal
' The function can be invoked as follows:
RetVal = MyFunc("Hello", 2, "World")     ' All 3 arguments supplied.
RetVal = MyFunc("Test", , 5) ' Second argument omitted.
' Arguments one and three using named-arguments.
RetVal = MyFunc(MyStr:="Hello ", MyArg1:=7)
```

GetObject Function Example

This example uses the **GetObject** function to get a reference to a specific Microsoft Excel worksheet (`MyXL`). It uses the worksheet's **Application** property to make Microsoft Excel visible, to close it, and so on. Using two API calls, the DetectExcel **Sub** procedure looks for Microsoft Excel, and if it is running, enters it in the Running Object Table. The first call to **GetObject** causes an error if Microsoft Excel isn't already running. In the example, the error causes the ExcelWasNotRunning flag to be set to True. The second call to **GetObject** specifies a file to open. If Microsoft Excel isn't already running, the second call starts it and returns a reference to the worksheet represented by the specified file, mytest.xls. The file must exist in the specified location; otherwise, the Visual Basic error Automation error is generated. Next the example code makes both Microsoft Excel and the window containing the specified worksheet visible. Finally, if there was no previous version of Microsoft Excel running, the code uses the **Application** object's **Quit** method to close Microsoft Excel. If the application was already running, no attempt is made to close it. The reference itself is released by setting it to **Nothing**.

```
' Declare necessary API routines:
Declare Function FindWindow Lib "user32" Alias _
"FindWindowA" (ByVal lpClassName as String, _
               ByVal lpWindowName As Long) As Long

Declare Function SendMessage Lib "user32" Alias _
"SendMessageA" (ByVal hWnd as Long,ByVal wMsg as Long _
               ByVal wParam as Long _
               ByVal lParam As Long) As Long

Sub GetExcel()
   Dim MyXL As Object   ' Variable to hold reference
   ' to Microsoft Excel.
   Dim ExcelWasNotRunning As Boolean     ' Flag for final release.

' Test to see if there is a copy of Microsoft Excel already running.
   On Error Resume Next ' Defer error trapping.
' Getobject function called without the first argument returns a
' reference to an instance of the application. If the application isn't
' running, an error occurs.
   Set MyXL = Getobject(, "Excel.Application")
   If Err.Number <> 0 Then ExcelWasNotRunning = True
   Err.Clear   ' Clear Err object in case error occurred.

' Check for Microsoft Excel. If Microsoft Excel is running,
' enter it into the Running Object table.
   DetectExcel

Set the object variable to reference the file you want to see.
   Set MyXL = Getobject("c:\vb4\MYTEST.XLS")

' Show Microsoft Excel through its Application property. Then
' show the actual window containing the file using the Windows
' collection of the MyXL object reference.
   MyXL.Application.Visible = True
   MyXL.Parent.Windows(1).Visible = True
   ' Do manipulations of your
   ' file here.
   ' ...
' If this copy of Microsoft Excel was not running when you
```

```
' started, close it using the Application property's Quit method.
' Note that when you try to quit Microsoft Excel, the
' title bar blinks and a message is displayed asking if you
' want to save any loaded files.
   If ExcelWasNotRunning = True Then
      MyXL.Application.Quit
   End IF

   Set MyXL = Nothing   ' Release reference to the
   ' application and spreadsheet.
End Sub

Sub DetectExcel()
' Procedure dectects a running Excel and registers it.
   Const WM_USER = 1024
   Dim hWnd As Long
' If Excel is running this API call returns its handle.
   hWnd = FindWindow("XLMAIN", 0)
   If hWnd = 0 Then  ' 0 means Excel not running.
      Exit Sub
   Else
   ' Excel is running so use the SendMessage API
   ' function to enter it in the Running Object Table.
      SendMessage hWnd, WM_USER + 18, 0, 0
   End If
End Sub
```

Implements Statement Example

The following example shows how to use the **Implements** statement to make a set of declarations available to multiple classes. By sharing the declarations through the **Implements** statement, neither class has to make any declarations itself.

Assume there are two forms. The Selector form has two buttons, Customer Data and Supplier Data. To enter name and address information for a customer or a supplier, the user clicks the Customer button or the Supplier button on the Selector form, and then enters the name and address using the Data Entry form. The Data Entry form has two text fields, Name and Address.

The following code for the shared declarations is in a class called PersonalData:

```
Public Name As String
Public Address As String
```

The code supporting the customer data is in a class module called Customer:

```
Implements PersonalData
Private Property Get PersonalData_Address() As String
PersonalData_Address = "CustomerAddress"
End Property

Private Property Let PersonalData_Address(ByVal RHS As String)
'
End Property

Private Property Let PersonalData_Name(ByVal RHS As String)
'
End Property

Private Property Get PersonalData_Name() As String
PersonalData_Name = "CustomerName"
End Property
```

The code supporting the supplier data is in a class module called Supplier:

```
Implements PersonalData

Private Property Get PersonalData_Address() As String
PersonalData_Address = "SupplierAddress"
End Property

Private Property Let PersonalData_Address(ByVal RHS As String)
'
End Property

Private Property Let PersonalData_Name(ByVal RHS As String)
'
End Property

Private Property Get PersonalData_Name() As String
PersonalData_Name = "SupplierName"
End Property
```

The following code supports the Selector form:

```
Private cust As New Customer
Private sup As New Supplier
```

```
Private Sub Command1_Click()
Dim frm2 As New Form2
    Set frm2.PD = cust
    frm2.Show 1
End Sub

Private Sub Command2_Click()
Dim frm2 As New Form2
    Set frm2.PD = sup
    frm2.Show 1
End Sub
```

The following code supports the Data Entry form:

```
Private m_pd As PersonalData
Private Sub Form_Load()
    With m_pd
        Text1 = .Name
        Text2 = .Address
    End With
End Sub
Public Property Set PD(Data As PersonalData)
    Set m_pd = Data
End Property
```

LBound Function Example

This example uses the **LBound** function to determine the smallest available subscript for the indicated dimension of an array. Use the **Option Base** statement to override the default base array subscript value of 0.

```
Dim Lower
Dim MyArray(1 To 10, 5 To 15, 10 To 20)  ' Declare array variables.
Dim AnyArray(10)
Lower = Lbound(MyArray, 1) ' Returns 1.
Lower = Lbound(MyArray, 3) ' Returns 10.
Lower = Lbound(AnyArray)' Returns 0 or 1, depending on
   ' setting of Option Base.
```

Let Statement Example

This example assigns the values of expressions to variables using the explicit **Let** statement.

```
Dim MyStr, MyInt
' The following variable assignments use the Let statement.
Let MyStr = "Hello World"
Let MyInt = 5
```

The following are the same assignments without the **Let** statement.

```
Dim MyStr, MyInt
MyStr = "Hello World"
MyInt = 5
```

Option Base Statement Example

This example uses the **Option Base** statement to override the default base array subscript value of 0. The **LBound** function returns the smallest available subscript for the indicated dimension of an array. The **Option Base** statement is used at the module level only.

```
Option base 1   ' Set default array subscripts to 1.

Dim Lower
Dim MyArray(20), TwoDArray(3, 4)    ' Declare array variables.
Dim ZeroArray(0 To 5)' Override default base subscript.
' Use LBound function to test lower bounds of arrays.
Lower = LBound(MyArray) ' Returns 1.
Lower = LBound(TwoDArray, 2) ' Returns 1.
Lower = LBound(ZeroArray)   ' Returns 0.
```

Option Compare Statement Example

This example uses the **Option Compare** statement to set the default string comparison method. The **Option Compare** statement is used at the module level only.

```
' Set the string comparison method to Binary.
Option compare Binary   ' That is, "AAA" is less than "aaa".
' Set the string comparison method to Text.
Option compare Text  ' That is, "AAA" is equal to "aaa".
```

Option Explicit Statement Example

This example uses the **Option Explicit** statement to force explicit declaration of all variables. Attempting to use an undeclared variable causes an error at compile time. The **Option Explicit** statement is used at the module level only.

```
Option explicit' Force explicit variable declaration.
Dim MyVar' Declare variable.
MyInt = 10  ' Undeclared variable generates error.
MyVar = 10  ' Declared variable does not generate error.
```

Option Private Statement Example

This example demonstrates the **Option Private** statement, which is used at module level to indicate that the entire module is private. With **Option Private Module**, module-level parts not declared **Private** are available to other modules in the project, but not to other projects or applications.

```
Option private Module' Indicates that module is private.
```

Private Statement Example

This example shows the **Private** statement being used at the module level to declare variables as private; that is, they are available only to the module in which they are declared.

```
Private Number As Integer  ' Private Integer variable.
Private NameArray(1 To 5) As String' Private array variable.
' Multiple declarations, two Variants and one Integer, all Private.
Private MyVar, YourVar, ThisVar As Integer
```

Property Get Statement Example

This example uses the **Property Get** statement to define a property procedure that gets the value of a property. The property identifies the current color of a pen as a string.

```
Dim CurrentColor As Integer
Const BLACK = 0, RED = 1, GREEN = 2, BLUE = 3

' Returns the current color of the pen as a string.
Property Get PenColor() As String
   Select Case CurrentColor
      Case RED
         PenColor = "Red"
      Case GREEN
         PenColor = "Green"
      Case BLUE
         PenColor = "Blue"
   End Select
End Property

' The following code gets the color of the pen
' calling the Property Get procedure.
ColorName = PenColor
```

Property Let Statement Example

This example uses the **Property Let** statement to define a procedure that assigns a value to a property. The property identifies the pen color for a drawing package.

```
Dim CurrentColor As Integer
Const BLACK = 0, RED = 1, GREEN = 2, BLUE = 3

' Set the pen color property for a Drawing package.
' The module-level variable CurrentColor is set to
' a numeric value that identifies the color used for drawing.
Property Let PenColor(ColorName As String)
   Select Case ColorName   ' Check color name string.
      Case "Red"
         CurrentColor = RED  ' Assign value for Red.
      Case "Green"
         CurrentColor = GREEN      ' Assign value for Green.
      Case "Blue"
         CurrentColor = BLUE ' Assign value for Blue.
      Case Else
         CurrentColor = BLACK      ' Assign default value.
   End Select
End Property

' The following code sets the PenColor property for a drawing package
' by calling the Property let procedure.

PenColor = "Red"
```

Property Set Statement Example

This example uses the **Property Set** statement to define a property procedure that sets a reference to an object.

```
' The Pen property may be set to different Pen implementations.
Property Set Pen(P As Object)
   Set CurrentPen = P    ' Assign Pen to object.
End Property
```

Public Statement Example

This example uses the **Public** statement at the module level (General section) of a standard module to explicitly declare variables as public; that is, they are available to all procedures in all modules in all applications unless **Option Private Module** is in effect.

```
Public Number As Integer' Public Integer variable.
Public NameArray(1 To 5) As String ' Public array variable.
' Multiple declarations, two Variants and one Integer, all Public.
Public MyVar, YourVar, ThisVar As Integer
```

ReDim Statement Example

This example uses the **ReDim** statement to allocate and reallocate storage space for dynamic-array variables. It assumes the **Option Base** is **1**.

```
Dim MyArray() As Integer' Declare dynamic array.
Redim MyArray(5)   ' Allocate 5 elements.
For I = 1 To 5 ' Loop 5 times.
   MyArray(I) = I ' Initialize array.
Next I
```

The next statement resizes the array and erases the elements.

```
Redim MyArray(10) ' Resize to 10 elements.
For I = 1 To 10' Loop 10 times.
   MyArray(I) = I ' Initialize array.
Next I
```

The following statement resizes the array but does not erase elements.

```
Redim Preserve MyArray(15) ' Resize to 15 elements.
```

Rem Statement Example

This example illustrates the various forms of the **Rem** statement, which is used to include explanatory remarks in a program.

```
Dim MyStr1, MyStr2
MyStr1 = "Hello": Rem Comment after a statement separated by a colon.
MyStr2 = "Goodbye"   ' This is also a comment; no colon is needed.
```

Set Statement Example

This example uses the **Set** statement to assign object references to variables. `YourObject` is assumed to be a valid object with a Text property.

```
Dim YourObject, MyObject, MyStr
Set MyObject = YourObject  ' Assign object reference.
' MyObject and YourObject refer to the same object.
YourObject.Text = "Hello World"    ' Initialize property.
MyStr = MyObject.Text' Returns "Hello World".

' Discontinue association. MyObject no longer refers to YourObject.
Set MyObject = Nothing  ' Release the object.
```

Static Statement Example

This example uses the **Static** statement to retain the value of a variable for as long as module code is running.

```
' Function definition.
Function KeepTotal(Number)
    ' Only the variable Accumulate preserves its value between calls.
    Static Accumulate
    Accumulate = Accumulate + Number
    KeepTotal = Accumulate
End Function

' Static function definition.
Static Function MyFunction(Arg1, Arg2, Arg3)
    ' All local variables preserve value between function calls.
    Accumulate = Arg1 + Arg2 + Arg3
    Half = Accumulate / 2
    MyFunction = Half
End Function
```

Sub Statement Example

This example uses the **Sub** statement to define the name, arguments, and code that form the body of a **Sub** procedure.

```
' Sub procedure definition.
' Sub procedure with two arguments.
Sub SubComputeArea(Length, TheWidth)
   Dim Area As Double   ' Declare local variable.
   If Length = 0 Or TheWidth = 0 Then
   ' If either argument = 0.
      Exit Sub ' Exit Sub immediately.
   End If
   Area = Length * TheWidth  ' Calculate area of rectangle.
   Debug.Print Area  ' Print Area to Debug window.
End Sub
```

Type Statement Example

This example uses the **Type** statement to define a user-defined data type. The **Type** statement is used at the module level only. If it appears in a class module, a **Type** statement must be preceded by the keyword **Private**.

```
Type EmployeeRecord  ' Create user-defined type.
   ID As Integer  ' Define elements of data type.
   Name As String * 20
   Address As String * 30
   Phone As Long
   HireDate As Date
End Type
Sub CreateRecord()
   Dim MyRecord As EmployeeRecord  ' Declare variable.

   ' Assignment to EmployeeRecord variable must occur in a procedure.
   MyRecord.ID = 12003  ' Assign a value to an element.
End Sub
```

UBound Function Example

This example uses the **UBound** function to determine the largest available subscript for the indicated dimension of an array.

```
Dim Upper
Dim MyArray(1 To 10, 5 To 15, 10 To 20)  ' Declare array variables.
Dim AnyArray(10)
Upper = UBound(MyArray, 1) ' Returns 10.
Upper = UBound(MyArray, 3) ' Returns 20.
Upper = UBound(AnyArray)' Returns 10.
```

Ambiguous selection

Either you have not selected a keyword or you have requested Help on a component of the integrated development environment (IDE). If you were trying to select a keyword, try reselecting a single keyword. If you were trying to get information on the IDE, click one of the following:

Code or Module Window

Immediate Window or Pane

Locals Windows or pane

Object Browser

Watch Window or Pane

As

The **As** keyword is used in these contexts:

Const Statement

Declare Statement

Dim Statement

Function Statement

Name Statement

Open Statement

Private Statement

Property Get Statement

Property Let Statement

Property Set Statement

Public Statement

ReDim Statement

Static Statement

Sub Statement

Type Statement

Binary

The **Binary** keyword is used in these contexts:

Open Statement

Option Compare Statement

ByRef

The **ByRef** keyword is used in these contexts:

Call Statement

Declare Statement

Function Statement

Property Get Statement

Property Let Statement

Property Set Statement

Sub Statement

ByVal

The **ByVal** keyword is used in these contexts:

Call Statement

Declare Statement

Function Statement

Property Get Statement

Property Let Statement

Property Set Statement

Sub Statement

Date

The **Date** keyword is used in these contexts:

Date Data Type

Date Function

Date Statement

Else

The **Else** keyword is used in these contexts:

If...Then...Else Statement

Select Case Statement

Error

The **Error** keyword is used in these contexts:

Error Function

Error Statement

On Error Statement

For

The **For** keyword is used in these contexts:

For...Next Statement

For Each...Next Statement

Open Statement

Get

The **Get** keyword is used in these contexts:

Get Statement

Property Get Statement

Input

The **Input** keyword is used in these contexts:

Input Function

Input # Statement

Line Input # Statement

Open Statement

Is

The **Is** keyword is used in these contexts:

If...Then...Else Statement

Is Operator

Select Case Statement

Len

The **Len** keyword is used in these contexts:

Len Function

Open Statement

Let

The **Let** keyword is used in these contexts:

Let Statement

Property Let Statement

Lock

The **Lock** keyword is used in these contexts:

Lock, Unlock Statements

Open Statement

Mid

The **Mid** keyword is used in these contexts:

Mid Function

Mid Statement

New

The **New** keyword is used in these contexts:

Dim Statement

Private Statement

Public Statement

Set Statement

Static Statement

Next

The **Next** keyword is used in these contexts:

For...Next Statement

For Each...Next Statement

On Error Statement

Resume Statement

On

The **On** keyword is used in these contexts:

On Error Statement

On...GoSub Statement

On...GoTo Statement

Option

The **Option** keyword is used in these contexts:

Option Base Statement

Option Compare Statement

Option Explicit Statement

Option Private Statement

Optional

The **Optional** keyword is used in these contexts:

Declare Statement

Function Statement

Property Get Statement

Property Let Statement

Property Set Statement

Sub Statement

ParamArray

The **ParamArray** keyword is used in these contexts:

Declare Statement

Function Statement

Property Get Statement

Property Let Statement

Property Set Statement

Sub Statement

Print

The **Print** keyword is used in these contexts:

Print Method

Print # Statement

Private

The **Private** keyword is used in these contexts:

Const Statement

Declare Statement

Function Statement

Option Private Statement

Private Statement

Property Get Statement

Property Let Statement

Property Set Statement

Sub Statement

Type Statement

Property

The **Property** keyword is used in these contexts:

Property Get Statement

Property Let Statement

Property Set Statement

Public

The **Public** keyword is used in these contexts:

Const Statement

Declare Statement

Function Statement

Property Get Statement

Property Let Statement

Property Set Statement

Public Statement

Sub Statement

Type Statement

Resume

The **Resume** keyword is used in these contexts:

On Error Statement

Resume Statement

Seek

The **Seek** keyword is used in these contexts:

Seek Function

Seek Statement

Set

The **Set** keyword is used in these contexts:

Set Statement

Property Set Statement

Static

The **Static** keyword is used in these contexts:

Function Statement

Property Get Statement

Property Let Statement

Property Set Statement

Static Statement

Sub Statement

Step

The **Step** keyword is used in these contexts:

For...Next Statement

For Each...Next Statement

String

The **String** keyword is used in these contexts:

String Data Type

String Function

Then

The **Then** keyword is used in these contexts:

#If...Then...#Else Directive

If...Then...Else Statement

Time

The **Time** keyword is used in these contexts:

Time Function

Time Statement

To

The **To** keyword is used in these contexts:

Dim Statement

For...Next Statement

Lock, Unlock Statements

Private Statement

Public Statement

ReDim Statement

Select Case Statement

Static Statement

Type Statement

WithEvents

The **WithEvents** keyword is used in these contexts:

Dim Statement

Private Statement

Public Statement

Document Conventions

Visual Basic documentation uses the following typographic conventions.

Convention	Description
Sub, **If**, **ChDir**, **Print**, **True**, **Debug**	Words in bold with initial letter capitalized indicate language-specific keywords.
setup	Words you are instructed to type appear in bold.
object, *varname*, *arglist*	Italic, lowercase letters indicate placeholders for information you supply.
pathname, ***filenumber***	Bold, italic, and lowercase letters indicate placeholders for arguments where you can use either positional or named-argument syntax.
[*expressionlist*]	In syntax, items inside brackets are optional.
{**While** | **Until**}	In syntax, braces and a vertical bar indicate a mandatory choice between two or more items. You must choose one of the items unless all of the items are also enclosed in brackets. For example: [{**This** | **OrThat**}]
ESC, ENTER	Words in small capital letters indicate key names and key sequences.
ALT+F1, CTRL+R	A plus sign (+) between key names indicates a combination of keys. For example, ALT+F1 means hold down the ALT key while pressing the F1 key.

Code Conventions

The following code conventions are used:

Sample Code	Description
`MyString = "Hello, world!"`	This font is used for code, variables, and error message text.
`' This is a comment.`	An apostrophe (') introduces code comments.

`MyVar = "This is an " _`
`& "example" _`
`& " of how to continue code."`
	A space and an underscore (_) continue a line of code.

Error Function Example

This example uses the **Error** function to print error messages that correspond to the specified error numbers.

```
Dim ErrorNumber
For ErrorNumber = 61 To 64 ' Loop through values 61 - 64.
   Debug.Print Error(ErrorNumber)  ' Print error to Debug window.
Next ErrorNumber
```

Error Statement Example

This example uses the **Error** statement to simulate error number 11.

```
On Error Resume Next ' Defer error handling.
Error 11 ' Simulate the "Division by zero" error.
```

On Error Statement Example

This example first uses the **On Error GoTo** statement to specify the location of an error-handling routine within a procedure. In the example, an attempt to delete an open file generates error number 55. The error is handled in the error-handling routine, and control is then returned to the statement that caused the error. The **On Error GoTo 0** statement turns off error trapping. Then the **On Error Resume Next** statement is used to defer error trapping so that the context for the error generated by the next statement can be known for certain. Note that **Err.Clear** is used to clear the **Err** object's properties after the error is handled.

```
Sub OnErrorStatementDemo()
   On Error GoTo ErrorHandler      ' Enable error-handling routine.
   Open "TESTFILE" For Output As #1      ' Open file for output.
   Kill "TESTFILE"   ' Attempt to delete open
            ' file.
   On Error Goto 0   ' Turn off error trapping.
   On Error Resume Next ' Defer error trapping.
   ObjectRef = GetObject("MyWord.Basic") ' Try to start nonexistent
            ' object, then test for
'Check for likely Automation errors.
   If Err.Number = 440 Or Err.Number = 432 Then
      ' Tell user what happened. Then clear the Err object.
      Msg = "There was an error attempting to open the Automation object!"
      MsgBox Msg, , "Deferred Error Test"
      Err.Clear   ' Clear Err object fields
   End If
Exit Sub    ' Exit to avoid handler.
ErrorHandler:  ' Error-handling routine.
   Select Case Err.Number  ' Evaluate error number.
      Case 55  ' "File already open" error.
         Close #1 ' Close open file.
      Case Else
         ' Handle other situations here...
   End Select
   Resume   ' Resume execution at same line
            ' that caused the error.
End Sub
```

Resume Statement Example

This example uses the **Resume** statement to end error handling in a procedure, and then resume execution with the statement that caused the error. Error number 55 is generated to illustrate using the **Resume** statement.

```
Sub ResumeStatementDemo()
   On Error GoTo ErrorHandler      ' Enable error-handling routine.
   Open "TESTFILE" For Output As #1      ' Open file for output.
   Kill "TESTFILE"   ' Attempt to delete open file.
   Exit Sub ' Exit Sub to avoid error handler.
ErrorHandler:  ' Error-handling routine.
   Select Case Err.Number  ' Evaluate error number.
      Case 55  ' "File already open" error.
         Close #1 ' Close open file.
      Case Else
         ' Handle other situations here....
   End Select
   Resume   ' Resume execution at same line
   ' that caused the error.
End Sub
```

Error Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctErrorX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctErrorS"}

Returns the error message that corresponds to a given error number.

Syntax

Error[**(***errornumber***)**]

The optional *errornumber* argument can be any valid error number. If *errornumber* is a valid error number, but is not defined, **Error** returns the string "Application-defined or object-defined error." If *errornumber* is not valid, an error occurs. If *errornumber* is omitted, the message corresponding to the most recent run-time error is returned. If no run-time error has occurred, or *errornumber* is 0, **Error** returns a zero-length string ("").

Remarks

Examine the property settings of the **Err** object to identify the most recent run-time error. The return value of the **Error** function corresponds to the **Description** property of the **Err** object.

Error Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmErrorX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmErrorS"}

Simulates the occurrence of an error.

Syntax

Error *errornumber*

The required *errornumber* can be any valid error number.

Remarks

The **Error** statement is supported for backward compatibility. In new code, especially when creating objects, use the **Err** object's **Raise** method to generate run-time errors.

If *errornumber* is defined, the **Error** statement calls the error handler after the properties of **Err** object are assigned the following default values:

Property	Value
Number	Value specified as argument to **Error** statement. Can be any valid error number.
Source	Name of the current Visual Basic project.
Description	String expression corresponding to the return value of the **Error** function for the specified **Number**, if this string exists. If the string doesn't exist, **Description** contains a zero-length string ("").
HelpFile	The fully qualified drive, path, and file name of the appropriate Visual Basic Help file.
HelpContext	The appropriate Visual Basic Help file context ID for the error corresponding to the **Number** property.
LastDLLError	Zero.

If no error handler exists or if none is enabled, an error message is created and displayed from the **Err** object properties.

Note Not all Visual Basic host applications can create objects. See your host application's documentation to determine whether it can create classes and objects.

On Error Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmOnErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmOnErrorX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmOnErrorS"}

Enables an error-handling routine and specifies the location of the routine within a procedure; can also be used to disable an error-handling routine.

Syntax

On Error GoTo *line*
On Error Resume Next
On Error GoTo 0

The **On Error** statement syntax can have any of the following forms:

Statement	Description
On Error GoTo *line*	Enables the error-handling routine that starts at *line* specified in the required *line* argument. The *line* argument is any line label or line number. If a run-time error occurs, control branches to *line*, making the error handler active. The specified *line* must be in the same procedure as the **On Error** statement; otherwise, a compile-time error occurs.
On Error Resume Next	Specifies that when a run-time error occurs, control goes to the statement immediately following the statement where the error occurred where execution continues. Use this form rather than **On Error GoTo** when accessing objects.
On Error GoTo 0	Disables any enabled error handler in the current procedure.

Remarks

If you don't use an **On Error** statement, any run-time error that occurs is fatal; that is, an error message is displayed and execution stops.

An "enabled" error handler is one that is turned on by an **On Error** statement; an "active" error handler is an enabled handler that is in the process of handling an error. If an error occurs while an error handler is active (between the occurrence of the error and a **Resume**, **Exit Sub**, **Exit Function**, or **Exit Property** statement), the current procedure's error handler can't handle the error. Control returns to the calling procedure. If the calling procedure has an enabled error handler, it is activated to handle the error. If the calling procedure's error handler is also active, control passes back through previous calling procedures until an enabled, but inactive, error handler is found. If no inactive, enabled error handler is found, the error is fatal at the point at which it actually occurred. Each time the error handler passes control back to a calling procedure, that procedure becomes the current procedure. Once an error is handled by an error handler in any procedure, execution resumes in the current procedure at the point designated by the **Resume** statement.

Note An error-handling routine is not a **Sub** procedure or **Function** procedure. It is a section of code marked by a line label or line number.

Error-handling routines rely on the value in the **Number** property of the **Err** object to determine the cause of the error. The error-handling routine should test or save relevant property values in the **Err** object before any other error can occur or before a procedure that might cause an error is called. The property values in the **Err** object reflect only the most recent error. The error message associated with **Err.Number** is contained in **Err.Description**.

On Error Resume Next causes execution to continue with the statement immediately following the statement that caused the run-time error, or with the statement immediately following the most recent call out of the procedure containing the **On Error Resume Next** statement. This statement allows execution to continue despite a run-time error. You can place the error-handling routine where the error would occur, rather than transferring control to another location within the procedure. An **On Error Resume Next** statement becomes inactive when another procedure is called, so you should execute an **On Error Resume Next** statement in each called routine if you want inline error handling within that routine.

Note The **On Error Resume Next** construct may be preferable to **On Error GoTo** when handling errors generated during access to other objects. Checking **Err** after each interaction with an object removes ambiguity about which object was accessed by the code. You can be sure which object placed the error code in **Err.Number**, as well as which object originally generated the error (the object specified in **Err.Source**).

On Error GoTo 0 disables error handling in the current procedure. It doesn't specify line 0 as the start of the error-handling code, even if the procedure contains a line numbered 0. Without an **On Error GoTo 0** statement, an error handler is automatically disabled when a procedure is exited.

To prevent error-handling code from running when no error has occurred, place an **Exit Sub**, **Exit Function**, or **Exit Property** statement immediately before the error-handling routine, as in the following fragment:

```
Sub InitializeMatrix(Var1, Var2, Var3, Var4)
   On Error GoTo ErrorHandler
   . . .
   Exit Sub
ErrorHandler:
   . . .
   Resume Next
End Sub
```

Here, the error-handling code follows the **Exit Sub** statement and precedes the **End Sub** statement to separate it from the procedure flow. Error-handling code can be placed anywhere in a procedure.

Untrapped errors in objects are returned to the controlling application when the object is running as an executable file. Within the development environment, untrapped errors are only returned to the controlling application if the proper options are set. See your host application's documentation for a description of which options should be set during debugging, how to set them, and whether the host can create classes.

If you create an object that accesses other objects, you should try to handle errors passed back from them unhandled. If you cannot handle such errors, map the error code in **Err.Number** to one of your own errors, and then pass them back to the caller of your object. You should specify your error by adding your error code to the **vbObjectError** constant. For example, if your error code is 1052, assign it as follows:

```
Err.Number = vbObjectError + 1052
```

Note System errors during calls to dynamic-link libraries (DLL) do not raise exceptions and cannot be trapped with Visual Basic error trapping. When calling DLL functions, you should check each return value for success or failure (according to the API specifications), and in the event of a failure, check the value in the **Err** object's **LastDLLError** property.

Resume Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmResumeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmResumeX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmResumeS"}

Resumes execution after an error-handling routine is finished.

Syntax

Resume [**0**]
Resume Next
Resume *line*

The **Resume** statement syntax can have any of the following forms:

Statement	Description
Resume	If the error occurred in the same procedure as the error handler, execution resumes with the statement that caused the error. If the error occurred in a called procedure, execution resumes at the statement that last called out of the procedure containing the error-handling routine.
Resume Next	If the error occurred in the same procedure as the error handler, execution resumes with the statement immediately following the statement that caused the error. If the error occurred in a called procedure, execution resumes with the statement immediately following the statement that last called out of the procedure containing the error-handling routine (or **On Error Resume Next** statement).
Resume *line*	Execution resumes at *line* specified in the required *line* argument. The *line* argument is a line label or line number and must be in the same procedure as the error handler.

Remarks

If you use a **Resume** statement anywhere except in an error-handling routine, an error occurs.

#Else clause must be preceded by a matching #If

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgLbElseNoMatchingIfC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLbElseNoMatchingIfS"}

#Else is a conditional compilation directive. This error has the following cause and solution:

- An **#Else** clause was detected that isn't preceded by a matching **#If** or **#ElseIf**.

  Check to see if a preceding **#If** has been separated from this **#Else** by an **#End If**. Note that only one **#Else** is permitted in each **#If** block, so two successive **#Else** clauses cause this error.

For additional information, select the item in question and press F1.

#Else If, #Else, or #End If must be preceded by a matching #If

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgLbNoMatchingIfC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLbNoMatchingIfS"}

#Else If, **#Else**, and **#End If** are conditional compilation directives. This error has the following cause and solution:

- An **#Else If**, **#Else**, or **#End If** was detected that isn't preceded by a matching **#If** clause.

  Check to see if the intended **#If** has been separated from the clause in question by an intervening block or if the intended **#If** is preceded by a number sign (**#**) sign. If everything else is in order, place an **#If** clause in the appropriate position.

For additional information, select the item in question and press F1.

#ElseIf must be preceded by a matching #If or #ElseIf and followed by an #ElseIf, #Else, or #End If

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgLbBadElseIfC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLbBadElseIfS"}

#ElseIf is a conditional compilation directive. This error has the following causes and solutions:

- An **#ElseIf** has been detected that isn't preceded by an **#If** or **#ElseIf**.
  Place an **#If** statement before the **#ElseIf** or remove an incorrectly placed preceding **#End If**.
- An **#ElseIf** has been detected that is preceded by an **#Else** or **#End If**.
  Appropriately terminate the preceding **#If** block, or change the preceding **#Else** to an **#ElseIf**.

For additional information, select the item in question and press F1.

A compatible ActiveX component must be a Visual Basic executable or a DLL

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidCompatibleServerS"}

A compatible ActiveX component is one that you specify as a compatible ActiveX component. This error has the following cause and solution:

- Visual Basic tried to access an object you specified as a compatible ActiveX component, but the file specified wasn't an executable file or dynamic-link library (DLL) created by Visual Basic.

  Only .exe files and DLLs created by Visual Basic are valid entries in the Compatible ActiveX Component field of the **Project Properties** dialog box accessed through the **Project** menu. If possible, load the project into Visual Basic and choose the **Make Project.exe File** command from the **File** menu to create a Visual Basic executable file. If the file is already an executable file that wasn't created by Visual Basic, or if the file can't be loaded into Visual Basic, consult the documentation of the file to find out if it can be converted to a Visual Basic executable file or if the vendor can supply an executable file created by Visual Basic.

For additional information, select the item in question and press F1.

Add-in can't reference project

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantRefProjectS"}

Projects are dynamic, but add-ins are static. This error has the following cause and solution:

- You tried to create an add-in from a project that references another project.

  The semantics of projects can change in a way that affects referencing projects and add-ins. Therefore, you can't create an add-in from a project that references projects.

For additional information, select the item in question and press F1.

A module is not a valid type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgModuleAsTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgModuleAsTypeS"}

A standard module doesn't represent a class and can't be instantiated in the form of a variable. This error has the following cause and solution:

- You used the name of a standard module in a **Dim** or **Set** declaration.

  Check the spelling of the module name and make sure it corresponds to a form, MDI form, or class module.

For additional information, select the item in question and press F1.

A procedure with a ParamArray argument can't be called with named arguments

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgParamArrayNamedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgParamArrayNamedS"}

All arguments in a call to a procedure defined with a **ParamArray** must be positional. This error has the following cause and solution:

- Named-argument syntax appears in a procedure call.

  The named-argument calling syntax can't be used to call a procedure that includes a **ParamArray** parameter. To supply only some elements of the **ParamArray**, use commas as placeholders for those elements you want to omit. For example, in the following call, if the **ParamArray** arguments begin after `Arg2`, values are being passed only for the first, third, and sixth values in the **ParamArray**:

  ```
  MySub Arg1, Arg2, 7,, 44,,3
  ```

Note The **ParamArray** always represents the last items in the argument list.

For additional information, select the item in question and press F1.

Ambiguous name detected

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgAmbiguousNameC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgAmbiguousNameS"}

The identifier conflicts with another identifier or requires qualification. This error has the following causes and solutions:

- More than one object in the same scope may have elements with the same name.

  Qualify the element name by including the object name and a period. For example:

  object.property

  Module-level identifiers and project-level identifiers (module names and referenced project names) may be reused in a procedure, although it makes programs harder to maintain and debug. However, if you want to refer to both items in the same procedure, the item having wider scope must be qualified. For example, if `MyID` is declared at the module level of `MyModule`, and then a procedure-level variable is declared with the same name in the module, references to the module-level variable must be appropriately qualified:

  ```
  Dim MyID As String
  Sub MySub
     MyModule.MyID = "This is module-level variable"
     Dim MyID As String
     MyID = "This is the procedure-level variable"
     Debug.Print MyID
     Debug.Print MyModule.MyID
  End Sub
  ```

- An identifier declared at module-level conflicts with a procedure name. For example, this error occurs if the variable `MyID` is declared at module level, and then a procedure is defined with the same name:

  ```
  Public MyID
  Sub MyID
     . . .
  End Sub
  ```

  In this case, you must change one of the names because qualification with a common module name would not resolve the ambiguity. Procedure names are **Public** by default, but variable names are **Private** unless specified as **Public**.

For additional information, select the item in question and press F1.

AppleScript error

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgAppleScriptErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgAppleScriptErrorS"}

An error occurred during processing of your script. This error has the following causes and solutions:

- The error could be related to the target application.
  Check whether identifiers in the script match those in the application.
- The error could be related to AppleScript.
  Make sure the syntax used in your script is valid AppleScript syntax.

For additional information, select the item in question and press F1.

Application-defined or object-defined error

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsguserdefinedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgUserDefinedS"}

This message is displayed when an error generated with the **Raise** method or **Error** statement doesn't correspond to an error defined by Visual Basic for Applications. It is also returned by the **Error** function for arguments that don't correspond to errors defined by Visual Basic for Applications. Thus it may be an error you defined, or one that is defined by an object, including host applications like Microsoft Excel, Visual Basic, and so on. For example, Visual Basic forms generate form-related errors that can't be generated from code simply by specifying a number as an argument to the **Raise** method or **Error** statement. This message has the following causes and solutions:

- Your application executed an **Err.Raise** *n* or **Error** *n* statement, but the number *n* isn't defined by Visual Basic for Applications.

  If this was what was intended, you must use **Err.Raise** and specify additional arguments so that an end user can understand the nature of the error. For example, you can include a description string, source, and help information. To regenerate an error that you trapped, this approach will work if you don't execute **Err.Clear** before regenerating the error. If you execute **Err.Clear** first, you must fill in the additional arguments to the **Raise** method. Look at the context in which the error occurred, and make sure you are regenerating the same error.
- It may be that in accessing objects from other applications, an error was propagated back to your program that can't be mapped to a Visual Basic error.

  Check the documentation for any objects you have accessed. The **Err** object's **Source** property should contain the programmatic ID of the application or object that generated the error. To understand the context of an error returned by an object, you may want to use the **On Error Resume Next** construct in code that accesses objects, rather than the **On Error GoTo** *line* syntax.

Note In the past, programmers often used a loop to print out a list of all trappable error message strings. Typically this was done with code such as the following:

```
For index = 1 to 500
   Debug.Print Error$(index)
Next index
```

Such code still lists all the Visual Basic for Applications error messages, but displays "Application-defined or object-defined error" for host-defined errors, for example those in Visual Basic that relate to forms, controls, and so on. Many of these are trappable run-time errors. You can use the Help **Search** dialog box to find the list of trappable errors specific to your host application. Click **Search**, type **Trappable** in the first text box, and then click **Show Topics**. Select **Trappable Errors** in the lower list box and click **Go To**.

For additional information, select the item in question and press F1.

Argument required for Property Let or Property Set

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgPropByValC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgPropByvalS"}

The purpose of **Property Let** and **Property Set** procedures is to give a new value to a property. This error has the following causes and solutions:

- In setting the property, the value doesn't appear in the right place.

  Place the value to which you want to set the property on the right side of the expression setting the property value.
- In the procedure definition, the parameter defined to receive the value passed on the right side of the expression is missing or misplaced.

  Specify a parameter for the value argument list in the procedure definition. If the procedure takes more than one argument, the property-value parameter must appear last in the list.

For additional information, select the item in question and press F1.

Array already dimensioned

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgArrayAlreadyDimC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgArrayAlreadyDimS"}

A static array can only be dimensioned once. This error has the following causes and solutions:

- You attempted to change the dimensions of a static array with a **ReDim** statement; only dynamic arrays can be redimensioned.

  Either remove the redimensioning or use a dynamic array. To define a dynamic array, use a **Dim**, **Public**, **Private**, or **Static** statement with empty parentheses. For example:

  ```
  Dim MyArray()
  ```

  In a procedure, you can define a dynamic array with the **ReDim** or **Static** statement using a variable for the number of elements:

  ```
  ReDim MyArray(n)
  ```

- An **Option Base** statement occurs after array dimensions are set.

  Make sure any **Option Base** statement precedes all array declarations.

For additional information, select the item in question and press F1.

Array argument must be ByRef

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgArrayMustBeByRefC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgArrayMustBeByrefS"}

Arrays declared with **Dim**, **ReDim**, or **Static** can't be passed **ByVal**. This error has the following cause and solution:

- You tried to pass a whole array **ByVal**.

  An individual element of an array can be passed **ByVal** (by value), but a whole array must be passed **ByRef** (by reference). Note that **ByRef** is the default. If you must pass an array **ByVal** to prevent changes to the array's elements from being propagated back to the caller, you can pass the array argument in its own set of parentheses, or you can place it into a **Variant**, and then pass the **Variant** to the **ByVal** parameter, as follows:

  ```
  Dim MyVar As Variant
  MyVar = OldArray()
  ```

For additional information, select the item in question and press F1.

Beginning of search scope has been reached; do you want to continue from the end?

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgSearchLowerBoundS"}

Your search was unsuccessful. This condition has the following cause and solution:

- Your upward search has reached the beginning of the specified scope.
  You can continue searching from the end of the search scope, or cancel and change the scope of the search.

Block If without End If

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedEndifC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedEndIfS"}

An error occurred due to an incomplete statement. This error has the following cause and solution:

- An **If** statement is used without a corresponding **End If** statement.

  A multiline **If** statement must terminate with a matching **End If** statement. For nested **If...End If** statements, make sure there is a correctly matched **If...End If** structure inside each enclosing **If...End If** structure.

For additional information, select the item in question and press F1.

Breakpoint not allowed on this line

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgNOBpCaseC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoBpCaseS"}

Breakpoints can only be placed on certain parts of statements. This error has the following causes:

- You tried to place a breakpoint on a line that can't accept a breakpoint, for example:
  - A line that contains only comments.
  - A line that contains only line labels.
  - A line that contains only declarations (**Const**, **Dim**, **Static**, **Type**, and so on).
  - Any line in a hidden module.
  - Any line in the **Immediate** window.

For additional information, select the item in question and press F1.

ByRef argument type mismatch

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgArgTypeMismatchC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgArgTypeMismatchS"}

An argument passed **ByRef** (by reference), the default, must have the precise data type expected in the procedure. This error has the following cause and solution:

- You passed an argument of one type that could not be coerced to the type expected.

  For example, this error occurs if you try to pass an **Integer** variable when a **Long** is expected. If you want coercion to occur, even if it causes information to be lost, you can pass the argument in its own set of parentheses. For example, to pass the **Variant** argument `MyVar` to a procedure that expects an **Integer** argument, you can write the call as follows:

  ```
  Dim MyVar
  MyVar = 3.1415
  Call SomeSub((MyVar))

  Sub SomeSub (MyNum As Integer)
     MyNum = MyNum + MyNum
  End Sub
  ```

  Placing the argument in its own set of parentheses forces evaluation of it as an expression. During this evaluation, the fractional portion of the number is rounded (not truncated) to make it conform to the expected argument type. The result of the evaluation is placed in a temporary location, and a reference to the temporary location is received by the procedure. Thus, the original `MyVar` retains its value.

Note If you don't specify a type for a variable, the variable receives the default type, **Variant**. This isn't always obvious. For example, the following code declares two variables, the first, `MyVar`, is a **Variant**; the second, `AnotherVar`, is an **Integer**.

```
Dim MyVar, AnotherVar As Integer
```

For additional information, select the item in question and press F1.

Calling convention not supported by Visual Basic

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidVtableCcS"}

Visual Basic doesn't support all procedure calling conventions, which specify the order in which arguments must be passed and the way that argument types must be specified. This error has the following cause and solution:

- The procedure was called using a calling convention that Visual Basic doesn't support. For example, Visual Basic doesn't support the Pascal calling convention in a 16-bit version of the Microsoft Windows environment.

  If the calling convention isn't supported by Visual Basic, the procedure can't be called from Visual Basic. Check the object's documentation to see if an alternative is provided.

For additional information, select the item in question and press F1.

Can't add a reference to the specified file

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCannotAddReferenceS"}

Not all object libraries or type libraries can be accessed by Visual Basic. This error has the following cause and solution:

- You tried to use the **Add References** dialog box to add a reference to a type library or object library that can't be used by Visual Basic.

  Check the documentation for the object represented by the library to see if it's available in some other form that Visual Basic can use.

For additional information, select the item in question and press F1.

Can't assign or coerce array of fixed-length string or user-defined type to Variant

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgAssignArrayVariantC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgAssignArrayVariantS"}

A **Variant** can only accept assignment of data having a valid **VarType**. This error has the following causes and solutions:

- You tried to pass an array of fixed-length strings. When a single fixed-length string is assigned to a **Variant**, it's coerced to a variable-length string, but this can't be done for an array of fixed-length strings.

  If you must pass the array, use a loop to assign the individual elements of the array to the elements of a temporary array of variable-length strings. You can then assign the array to a variant and use **Erase** to deallocate the temporary array. However, you can't deallocate a fixed-size array with **Erase**.
- You tried to pass a fixed-length string or user-defined type to the **VarType** function or **TypeName** function.

  An argument to the **VarType** or **TypeName** function must be a valid **Variant** type.
- You tried to assign a user-defined type to a **Variant** variable.

  Although you can't directly assign a whole variable of user-defined type to a **Variant**, you can use the **Array** function to assign the individual elements of a variable of user-defined type to a **Variant**. This produces a **Variant** containing an array of variants. The **VarType** of each element in this array of variants corresponds to the original type of each element of the user-defined type.
- You tried to pass an array of fixed-length strings or user-defined types as an argument in a procedure call that requires a **Variant** argument. Note that any time a procedure is late bound, that is, when the call must be constructed at run time, all arguments must be passed as **Variant** types. For example, the following code causes this error:

  ```
  Dim MyForm As Object ' Because MyForm is Object, binding is late.
  Set MyForm = New Form1
  Dim StringArray(10) As String * 12
  ' The next line generates the error.
  MyForm.MyProc StringArray
  ```

  For the string array, use a loop to assign each individual member of the array to a temporary array of variable-length strings. You can then assign that array to a **Variant** to pass to the procedure. For an array of user-defined types, you can use the **Array** function to assign the individual elements of a variable of user-defined type to a **Variant**. This produces a **Variant** containing an array of variants. The **VarType** of each element in this array of variants corresponds to the original type of each element of the user-defined type.

For additional information, select the item in question and press F1.

Can't assign to an array

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgAssignArrayC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgAssignArrayS"}

Each element of an array must have its value assigned individually. This error has the following causes and solutions:

- You inadvertently tried to assign a single value to an array variable without specifying the element to which the value should be assigned.

  To assign a single value to an array element, you must specify the element in a subscript. For example, if `MyArray` is an integer array, the expression `MyArray = 5` is invalid, but the following expression is valid:

  ```
  MyArray(UBound(MyArray)) = 5
  ```

- You tried to assign a whole array to another array. For example, if `Arr1` is an array and `Arr2` is another array, the following two assignments are both invalid:

  ```
  Arr1 = Arr2 ' Invalid assignment.
  Arr1() = Arr2()' Invalid assignment.
  ```

  To assign one array to another, you must individually assign the elements. For example:

  ```
  For count = LBound(Arr2) to UBound(Arr2)
     Arr1(count) = Arr2(count)
  Next count
  ```

  Note that you can place a whole array in a **Variant**, resulting in a single variant variable containing the whole array:

  ```
  Dim MyArr As Variant
  MyVar = Arr2()
  ```

  You then reference the elements of the array in the variant with the same subscript notation as for a normal array, for example:

  ```
  MyVar(3) = MyVar(1) + MyVar(5)
  ```

For additional information, select the item in question and press F1.

Can't change data types of array elements

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgRedimTypeMismatchC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgRedimTypeMismatchS"}

ReDim can only be used to change the number of elements in an array. This error has the following cause and solution:

- You tried to redeclare the data type of an array using **ReDim**.

  Declare a new array of the type you want, and then use the conversion functions to assign each element of the old array to the corresponding element of the new array.

  You can also place the array in a **Variant** variable. This can be done with a simple assignment:

  ```
  Dim MyVar As Variant
  MyVar = MyIntegerArray()
  ```

  This creates a **Variant** containing an array tagged as the type of the original array. You can then assign variables of any valid **VarType** to the elements of the array within a variant.

For additional information, select the item in question and press F1.

Can't define a Public user-defined type within an object module

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgRecordDefInClassC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgRecordDefInClassS"}

A user-defined type that appears within an object module can't be **Public**. This error has the following cause and solution:

- You tried to define a **Public** user-defined type in an object module.

  Move the user-defined type definition to a standard module, and then declare variables of the type in the object module or other modules, as appropriate. If you only want the type to be available in the module in which it appears, you can place its **Type...End Type** definition in the object module and precede its definition with the **Private** keyword.

For additional information, select the item in question and press F1.

Can't display hidden procedure

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgProcInHiddenModuleS"}

Everything in a hidden module is hidden. This error has the following cause and solution:

- You tried to view a procedure in a hidden module.

  Unhide the module to view the procedure. To view a line that generated an error in a hidden procedure, run the code after unhiding the module to view the error.

For additional information, select the item in question and press F1.

Can't edit module

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCannotEditS"}

Modules marked as protected can be viewed, but not edited. This error has the following cause and solution:

- You tried to edit a protected module.
  Remove the protection from the module, and then try editing again.

For additional information, select the item in question and press F1.

Can't enter break mode at this time

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgCantDebugProjChangedAtRunC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantDebugProjChangedAtRunS"}

Break mode is the state in which a program is still running, but its activity is suspended. This error has the following cause and solution:

- You tried to enter break mode, for example, by pressing CTRL+BREAK, pressing the **Break** button on the **Standard** toolbar or the **Debug** toolbar, or by executing a breakpoint in the running code.

  A change was made programmatically to the project using the extensibility (add-in) object model. This prevents the program from having execution suspended. You can continue running, or end execution, but can't suspend execution.

For additional information, select the item in question and press F1.

Can't execute immediate statements in design mode

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantExecImmedInWaitModeS"}

Design time is any time that your code isn't being executed, or is suspended and waiting to be continued. This error has the following cause and solution:

- You tried to execute a statement in the **Immediate** window, but your code isn't in a running mode. Start your code running, and then suspend program execution. You can then execute code in the **Immediate** window.

For additional information, place the cursor in the **Immediate** window and press F1.

Can't find project or library

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBrokenLibRefS"}

You can't run your code until all missing references are resolved. This error has the following causes and solutions:

- A referenced project could not be found, or a referenced object library corresponding to the language of the project could not be found.

  Unresolved references are prefixed with MISSING in the **References** dialog box. Select the missing reference to display the path and language of the missing project or library. Follow these steps to resolve the reference or references:

  To resolve the references

  1. Display the **References** dialog box.
  2. Select the missing reference.
  3. Start the **Object Browser**.
  4. Use the **Browse** dialog box to find the missing reference.
  5. Click **OK**.
  6. Repeat the preceding steps until all missing references are resolved.

  Once you find a missing item, the MISSING prefix is removed to indicate that the link is reestablished. If the file name of a referenced project has changed, a new reference is added, and the old reference must be removed.

  To remove a reference that is no longer required, simply clear the check box next to the unnecessary reference. Note that the references to the Visual Basic object library and host-application object library can't be removed.

  Applications may support different language versions of their object libraries. To find out which language version is required, click the reference and check the language indicated at the bottom of the dialog box.

  Object libraries may be standalone files with the extension .OLB or they can be integrated into a dynamic-link library (DLL) They can exist in different versions for each platform. Therefore, when projects are moved across platforms, for example, from Macintosh to Microsoft Windows, the correct language version of the referenced library for that platform must be available in the location specified in your host application documentation.

Object library file names are generally constructed as follows:

- Windows (version 3.1 and earlier): Application Code + Language Code + [Version].OLB. For example:

  The object library for French Visual Basic for Applications, Version 2 was vafr2.olb.

  The French Microsoft Excel 5.0 object library was xlfr50.olb.
- Macintosh: Application Name Language Code [Version] OLB. For example:

  The object library for French Visual Basic for Applications, Version 2 was VA FR 2 OLB.

  The French Microsoft Excel 5.0 object library was MS Excel FR 50 OLB.

If you can't find a missing project or library on your system, contact the referencing project's author. If the missing library is a Microsoft application object library, you can obtain it as follows:

- If you have access to Microsoft electronic technical support services, refer to the technical support section of this Help file. Under electronic services, you will find instructions on how to use the appropriate service option.
- If you don't have access to Microsoft electronic technical support services, Microsoft object libraries are available upon request as an application note from Microsoft. Information on how to contact your local Microsoft product support organization is also available in the technical support section of this Help file.

For additional information, select the item in question and press F1.

Can't Get or Put an object reference variable or a variable of user-defined type containing an object reference

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgGetPutObjRecordc"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgGetPutObjRecordS"}

An object reference is temporary and can easily become invalid between closing and opening a file. This error has the following cause and solution:

- The variable in your **Get** or **Put** statement contains, or is declared to contain, a reference to an object.

  If the variable is an object reference you can't use it with **Get** and **Put** statements. To place the value of some or all of the object's properties in the file, each property must be individually specified.
- The user-defined type variable in your **Get** or **Put** statement contains an element that is an object reference.

  If the variable's **Type** statement contains an element representing an object (for example, it is defined in a class module, has **Object** data type, is a form or a control, and so on), remove it from the definition, or define a new type for use with the **Get** and **Put** statements that has no **Object** type element in its definition.

  If you have elements in the user-defined type with **Variant** type, make sure no object reference is assigned to that element. A **Variant** can accept such an assignment, but will cause this error if its user-defined type is used in a **Get** or **Put**.

  Note that you can use **Input #**, **Line Input #**, **Print #**, or **Write #** to write the default property of an object to disk.

For additional information, select the item in question and press F1.

Can't load module; invalid format

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantLoadModuleS"}

A module must be in text format. This error has the following cause and solution:

- You attempted to load a binary format file as a module.

  Some versions of Visual Basic permit you to save code in both binary and text formats. If possible, reload the file in the application in which it was last saved and save it as text.

For additional information, select the item in question and press F1.

Can't make an assignment to a read-only property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgReadOnlyPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgReadOnlyPropertyS"}

Some properties can't accept assignments. This error has the following cause and solution:

- You tried to assign a value to a property that can't accept assignment.

  In some cases, a property can accept assignment only at specific times. For example, a property might accept assignments at design time, but not at run time. Check the Help for the specific property to see when it can accept assignments, if ever.

For additional information, select the item in question and press F1.

Can't perform requested operation since the module is hidden

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgCantOperateOnHiddenModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantOperateOnHiddenModuleS"}

This error pertains to the Visual Basic extensibility (add-in) object model. This error has the following cause and solution:

- The module is hidden. You can't modify a hidden module or obtain information from a hidden module.

  Unhide the module and try the operation again.

For additional information, select the item in question and press F1.

Can't place conditional breakpoint on an array

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgConditionalWatchOnArrayC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgConditionalWatchOnArrayS"}

An array has no single value associated with it. This error has the following cause and solution:

- You tried to set a conditional breakpoint (sometimes called a watchpoint) that would suspend program execution when the value of a whole array changed.

  Set the conditional breakpoint on a specific element of the array.

For additional information, select the item in question and press F1.

Can't record into running module

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgRecordModCantChangeS"}

You must end execution to begin recording. This error has the following cause and solution:

- You tried to record a macro while executing module code.
  Stop execution or press CTRL+BREAK, and then turn on recording.

For additional information, select the item in question and press F1.

Can't remove default reference

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgCantRemoveReferenceC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantRemoveReferenceS"}

A default reference always exists. This error has the following cause and solution:

- You tried to remove a reference that must always be available, for example, a type-library or object-library reference.

  Don’t attempt to remove the reference.

For additional information, select the item in question and press F1.

Case Else outside Select Case

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgCaseElseNoSelectC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCaseElseNoSelectS"}

A **Case Else** statement can only occur between matching **Select Case** and **End Select** statements. This error has the following cause and solution:

- You placed a **Case Else** statement outside the **Select Case...End Select** block.
  Move the **Case Else** statement within a **Select Case...End Select** block.

For additional information, select the item in question and press F1.

Case without Select Case

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgCaseNoSelectC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCaseNoSelectS"}

A **Case** statement must occur within a **Select Case...End Select Block**. This error has the following cause and solution:

- A **Case** statement can't be matched with a preceding **Select Case** statement.

  Check other control structures within the **Select Case...Case** structure and verify that they are correctly matched. For example, an **If** without a matching **End If** inside the **Select Case...End Select** structure generates this error.

For additional information, select the item in question and press F1.

Circular dependencies between modules

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgcircultypeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCircularTypeS"}

Circular references between modules, constants, and user-defined types aren't allowed. This error has the following cause and solution:

- A user-defined type or constant in one module references a user-defined type or constant in a second module, which in turn references another user-defined type or constant in the first module.

  Remove the dependent references.

For additional information, select the item in question and press F1.

Object module must implement all procedures in interface

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInterfaceIncompleteC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInterfaceIncompleteS"}

An interface is a collection of unimplemented procedure prototypes. This error has the following cause and solution:

- You specified an interface in an **Implements** statement, but you didn't add code for all the procedures in the interface.

  You must write code for each of the procedures specified in the interface. An empty procedure is adequate; the procedure should implement the required behavior.

For additional information, select the item in question and press F1.

Code execution has been interrupted

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBreakModeInterruptS"}

Code execution can be suspended when necessary. This condition has the following cause and solution:

- A CTRL+BREAK (Microsoft Windows), ESC (Microsoft Excel) or COMMAND+PERIOD (Macintosh) key combination has been encountered.

  In the error dialog box, click **Debug** to enter break mode, **Continue** to resume, or **End** to stop execution.

For additional information, select the item in question and press F1.

Compile error in hidden module: <module name>

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCompErrInHiddenModuleS"}

A protected module can't be displayed. This error has the following cause and solution:

- There is a compilation error in the code of the specified module, but it can't be displayed because the project is protected.

  Unprotect the project, and then run the code again to view the error.

For additional information, select the item in question and press F1.

Constant expression required

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgRequiredConstExprC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgRequiredConstExprS"}

A constant must be initialized. This error has the following causes and solutions:

- You tried to initialize a constant with a variable, an instance of a user-defined type, an object, or the return value of a function call.

  Initialize constants with literals, previously declared constants, or literals and constants joined by operators (except the **Is** logical operator).
- You tried to declare an array using a variable to specify the number of elements.

  To declare a dynamic array within a procedure, declare the array with **ReDim** and specify the number of elements with a variable.

For additional information, select the item in question and press F1.

Constants, fixed-length strings, arrays, and Declare statements not allowed as Public members of an object module

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgBadClassMemberC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadClassMemberS"}

Not all variables in an object module can be declared as **Public**. However, procedures are **Public** by default, and **Property** procedures can be used to simulate variables syntactically. This error has the following causes and solutions:

- You declared a **Public** constant in an object module.

  Although you can't declare a **Public** constant in an object module, you can create a **Property Get** procedure with the same name. If you don't create a **Property Let** or **Property Set** procedure with that name, you are in effect creating a read-only property that can be used the same way you would use a constant.
- You declared a **Public** fixed-length string in an object module.

  You can simulate fixed-length strings with a set of **Property** procedures that either truncate the string data when it exceeds the permitted length, or notify the user that the length has been exceeded.
- You declared a **Public** array in an object module.

  Although a procedure can't return an array, it can return a **Variant** that contains an array. To simulate a **Public** array in a class module, use a set of **Property** procedures that accept and return a **Variant** containing an array.
- You placed a **Declare** statement in an object module.

  Declare statements are implicitly public. Precede the **Declare** statement with the **Private** keyword.

For additional information, select the item in question and press F1.

Current module doesn't support Print method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgNotIOBPrintC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNotIOBPrintS"}

Not all methods and properties are appropriate in all modules. This error has the following causes and solutions:

- You tried to use the **Print** method on an object that can't display anything. For example, you can't use the **Print** method without qualification in a standard module.

  Remove the reference to the **Print** method, or qualify it with an appropriate object. For example, qualify it with the **Debug** object to display its arguments in the **Immediate** window during debugging.
- You tried to use the **Line**, **Circle**, **PSet**, or **Scale** method on an object that can't accept them. For example, they can't appear unqualified in a standard module or an Automation class module.

For additional information, select the item in question and press F1.

Cyclic reference of projects not allowed

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCycleDetectedS"}

Circular references to projects aren't allowed. For example, if `MyProj` references `YourProj`, then `YourProj` (or a project that references `YourProj`) can't reference `MyProj`. This error has the following cause and solution:

- You tried to add a reference to a project that is already part of the project.
  Remove the circular reference or references.

For additional information, select the item in question and press F1.

Definitions of property procedures for the same property are inconsistent or contain optional parameters or a ParamArray

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInconsistentPropFuncsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInconsistentPropFuncsS"}

The parameters for **Property Get**, **Property Let**, and **Property Set** procedures for the same property must match exactly, except that the **Property Let** has one extra parameter, whose type must match the return type of the corresponding **Property Get**, and the **Property Set** has one more parameter than the corresponding **Property Get**, whose type is either **Variant**, **Object**, a class name, or an object library type specified in an object library. This error has the following causes and solutions:

- The number of parameters for the **Property Get** procedure isn't one less than the number of parameters for the matching **Property Let** or **Property Set** procedure.

  Add a parameter to **Property Let** or **Property Set** or remove a parameter from **Property Get**, as appropriate.
- The parameter types of **Property Get** must exactly match the corresponding parameters of **Property Let** or **Property Set**, except for the extra **Property Set** parameter.

  Modify the parameter declarations in the corresponding procedure definitions so they are appropriately matched.
- The parameter type of the extra parameter of the **Property Let** must match the return type of the corresponding **Property Get** procedure.

  Modify either the extra parameter declaration in the **Property Let** or the return type of the corresponding **Property Get** so they are appropriately matched.
- The parameter type of the extra parameter of the **Property Set** can differ from the return type of the corresponding **Property Get**, but it must be either a **Variant**, **Object**, class name, or a valid object library type.

  Make sure the extra parameter of the **Property Set** procedure is either a **Variant**, **Object**, class name, or object library type.
- You defined a **Property** procedure with an **Optional** or a **ParamArray** parameter.

  ParamArray and **Optional** parameters aren't permitted in **Property** procedures. Redefine the procedures without using these keywords.

For additional information, select the item in question and press F1.

Def*type* statements must precede declarations

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgDefTypeInvC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDefTypeInvS"}

Def*type* statements include **DefInt**, **DefDbl**, **DefCur**, and so on. This error has the following causes and solutions:

- A variable declaration precedes a **Def*type*** statement at module level.
  Move the **Def*type*** statement to precede all variable declarations.
- A **Def*type*** statement appears in a procedure.
  Move the **Def*type*** statement to module level, preceding all variable declarations.

For additional information, select the item in question and press F1.

Destination label too far away; loop, Select Case, or block If too large

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgBranchToOBigC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBranchTooBigS"}

Procedures can be as large as 64K from beginning to end, but because branching can occur either forward or backward within a procedure, such branching is limited to 32,767 bytes in either direction. This error has the following causes and solutions:

- You have a branching statement (**GoTo**, **GoSub**) whose destination label is farther away than 32,767 bytes from the source branching statement.

  Move the label closer, or make the procedure smaller.
- You have a very large loop structure that occupies more than 32K of memory from beginning to end.

  Make the loop smaller.
- You have a very large block **If** structure that contains a **Then** or **Else** clause that occupies more than 32K of memory from beginning to end.

  Reduce the size of the offending portion of the structure.

For additional information, select the item in question and press F1.

Do without Loop

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedLoopC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedLoopS"}

Each **Do** must be matched with a terminating **Loop**. This error has the following cause and solution:

- A **Do** statement was used without a terminating **Loop** statement.

  Check that other control structures within the **Do...Loop** structure are correctly matched. For example, a block **If** without a matching **End If** inside the **Do...Loop** structure may generate this error.

For additional information, select the item in question and press F1.

Duplicate declaration in current scope

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgMultiplyDefinedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgMultiplyDefinedS"}

The specified name is already used at this level of scope. For example, two variables can have the same name if they are defined in different procedures, but not if they are defined within the same procedure. This error has the following causes and solutions:

- A new variable or procedure has the same name as an existing variable or procedure. For example:

  ```
  Sub MySub()
     Dim A As Integer
     Dim A As Variant
     . . .    ' Other declarations or procedure code here.
  End Sub
  ```

  Check the current procedure, module, or project and remove any duplicate declarations.
- A **Const** statement uses the same name as an existing variable or procedure.

  Remove or rename the constant in question.
- You declared a fixed array more than once.

  Remove or rename one of the arrays.

Search for the duplicate name. When specifying the name to search for, omit any type-declaration character because a conflict occurs if the names are the same and the type-declaration characters are different.

Note that a module-level variable can have the same name as a variable declared in a procedure, but when you want to refer to the module-level variable within the procedure, you must qualify it with the module name. Module names and the names of referenced projects can be reused as variable names within procedures and can also be qualified.

For additional information, select the item in question and press F1.

Duplicate definition

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDuplicateDefnS"}

You can only define a conditional compiler constant to have one value. This error has the following cause and solution:

- You specified two different values for the same conditional compiler constant, for example:

  ```
  #Const Mac = 0
  #Const Mac = 1
  ```

  Remove one of the definitions.

For additional information, select the item in question and press F1.

Duplicate Deftype statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgDupDefTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDupDefTypeS"}

Def*type* statements in the same scope must specify unique letter ranges. This error has the following cause and solution:

- Part or all of the letter range for this **Def***type* statement is already included in another **Def***type* statement.

  Rewrite the statement so there is no overlap in the letter ranges.

For additional information, select the item in question and press F1.

Duplicate Option statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgDupOptionC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDupOptionS"}

Only one **Option** statement of each kind may occur in each module. This error has the following cause and solution:

- You defined more than one **Option Base**, **Option Compare**, **Option Explicit**, or **Option Private** statement in this module.

  Remove any duplicates in the module.

For additional information, select the item in question and press F1.

Edit can't be undone — proceed anyway?

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgAskCantUndoS"}

You won't be able to choose the **Undo** command to restore the current state after you perform this edit. This error has the following cause and solution:

- This typically occurs when an edit is simply too large to be saved.
  Try performing the edit in smaller increments.

For additional information, select the item in question and press F1.

Else without If

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgElseNoMatchingIfC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgElseNoMatchingIfS"}

An **Else** must be preceded by an **If**. This error has the following cause and solution:

- An **Else** statement was used without a corresponding **If** statement.

  Check other control structures within the **If...End If** structure and verify that they are correctly matched. Also check that the block **If** is correctly formatted.

For additional information, select the item in question and press F1.

Empty Enum type not allowed

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgEmptyEnumC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEmptyEnumS"}

An **Enum** type specifies the name and members for an enumeration. This error has the following cause and solution:

- You named an enumeration in an **Enum…End Enum** block, but failed to specify any members. Add at least one member to the **Enum…End Enum** block.

For additional information, select the item in question and press F1.

Empty watch expression

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEmptyWatchExprS"}

The **Add Watch** dialog box requires entry of an expression. This error has the following cause and solution:

- You didn't supply a watch expression in the **Add Watch** dialog box.
  Add an expression to the **Add Watch** dialog box or press the ESC key.

For additional information, select the item in question and press F1.

End If without block If

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgEndifnomatchingifC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEndIfNoMatchingIfS"}

An **End If** statement must have a corresponding **If** statement. This error has the following cause and solution:

- The **If** clause was omitted or is separated from the **End If**.

  Check other control structures within the **If...End If** structure and verify that they are correctly matched. Also check that the block **If** is correctly formatted.

For additional information, select the item in question and press F1.

End of search scope has been reached; do you want to continue from the beginning?

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgSearchUpperBoundS"}

Your search was unsuccessful. This error has the following cause and solution:

- Your downward search has reached the end of the specified scope.
  You can continue searching from the beginning of the search scope, or cancel and change the scope of the search.

End Select without Select Case

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgEndSelectNoSelectC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEndSelectNoSelectS"}

End Select must be matched with a preceding **Select Case**. This error has the following cause and solution:

- You used an **End Select** statement without a corresponding **Select Case** statement.

  This is usually due to an extra **End Select** below a **Select Case** block, or leaving behind the **End Select** statement when copying a **Select Case** block from one procedure to another. Check each **End Select** statement to make sure it terminates a **Select Case** structure.

For additional information, select the item in question and press F1.

End With without With

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgEndWithWithoutWithC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEndWithWithoutWithS"}

End With must be matched with a preceding **With**. This error has the following cause and solution:

- You used an **End With** statement without a corresponding **With** statement.

  Check other control structures within the **With...End With** structure and verify that they are correctly matched. For example, an **If** without a matching **End If** inside the **With...End With** structure can cause this error.

For additional information, select the item in question and press F1.

Error accessing the system registry

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgRegistryAccessS"}

Some operations must access your system's registration database. This error has the following cause and solution:

- The registration database for your system has been corrupted.
  Run the **Setup** program for the host application again.

For additional information, select the item in question and press F1.

Event handler is invalid

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgCantInsertEventHandlerC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantInsertEventHandlerS"}

The parameter list of an event-handling procedure must precisely match the declaration of the event. This error has the following cause and solution:

- Your event-handling procedure has the wrong number of parameters.
  Eliminate extra parameters or add the missing ones.
- One or more of your event-handling procedure parameters has the wrong data type.
  Make the parameter types match those of the event declaration.
- Your event-handling procedure is a **Function** rather than a **Sub**.
  Make your procedure a **Sub**. An event handler can't return a value.
- Another type library uses the event name for a type of its own.
  Qualify the name with the name of the proper type library to avoid the ambiguity.

For additional information, select the item in question and press F1.

Procedure declaration doesn't match description of event having same name

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEventProcSigMismatchS"}

Your class module has a procedure name that conflicts with the name of an event. This error has the following cause and solution:

- A procedure has the same name as an event, but does not have the same signature (that is, the number and types of the parameters). This can occur if you do something such as add a new parameter to an event procedure. For example, if you modify the definition of a form's Form_Load event procedure as follows, this error will occur:

  ```
  Sub Form_Load (MyParam As Integer)
  . . .
  End Sub
  ```

  If the procedure isn't the event procedure corresponding to the event, change its name. If the procedure corresponds to the event, make the parameter list agree with that required by the event (if any).

For additional information, select the item in question and press F1.

Exit Do not within Do...Loop

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExitDoNotWithinDoC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExitDoNotWithinDoS"}

Exit Do is only valid within a **Do...Loop** statement. This error has the following cause and solution:

- You used an **Exit Do** statement outside a **Do...Loop** statement.
  Make sure a valid **Do** statement precedes the **Exit Do**.

For additional information, select the item in question and press F1.

Exit For not within For...Next

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExitForNotWithinForC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExitForNotWithinForS"}

Exit For is only valid within a **For...Next** loop. This error has the following cause and solution:

- You used an **Exit For** statement outside a **For...Next** statement.
  Make sure a valid **For** statement precedes the **Exit For**.

For additional information, select the item in question and press F1.

Exit Function not allowed in Sub or Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExitFuncofSubC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExitFuncOfSubS"}

An **Exit** statement must match the procedure in which it occurs. This error has the following cause and solution:

- You used **Exit Function** in a **Sub** or **Property** procedure.
  Use **Exit Sub** or **Exit Property** for these types of procedures.

For additional information, select the item in question and press F1.

Exit Property not allowed in Function or Sub

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExitPropNotC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExitPropNotS"}

An **Exit** statement must match the procedure in which it occurs. This error has the following cause and solution:

- You used **Exit Property** in a **Sub** or **Function** procedure.
  Use the proper **Exit** statement for this type of procedure.

For additional information, select the item in question and press F1.

Exit Sub not allowed in Function or Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExitSubofFuncC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExitSubOfFuncS"}

An **Exit** statement must match the procedure in which it occurs. This error has the following cause and solution:

- You used **Exit Sub** in a **Function** or **Property** procedure.
  Use the proper **Exit** statement for this type of procedure.

For additional information, select the item in question and press F1.

Expected array

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedArrayC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedArrayS"}

A variable name with a subscript indicates the variable is an array. This error has the following cause and solution:

- The syntax you specified is appropriate for an array, but no array with this name is in scope.

  Check to make sure the name of the variable is spelled correctly. Unless the module contains **Option Explicit**, a variable is created on first use. If you misspell the name of an array variable, the variable may be created, but not as an array.

For additional information, select the item in question and press F1.

Expected End Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgEndFuncNotEndSubC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEndFuncNotEndSubS"}

An **End** *procedure* statement must match the procedure in which it occurs. This error has the following cause and solution:

- You used **End Property** or **End Sub** to end a **Function** procedure.
  Use **End Function** for this type of procedure.

For additional information, select the item in question and press F1.

Expected End Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgEndPropC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEndPropS"}

An **End** *procedure* statement must match the procedure in which it occurs. This error has the following cause and solution:

- You used **End Function** or **End Sub** to end a **Property** procedure.
  Use **End Property** for this type of procedure.

For additional information, select the item in question and press F1.

Expected End Sub

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgEndSubNotEndFuncC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEndSubNotEndFuncS"}

An **End** *procedure* statement must match the procedure in which it occurs. This error has the following cause and solution:

- You used **End Function** or **End Property** to end a **Sub** procedure.
  Use **End Sub** for this type of procedure.

For additional information, select the item in question and press F1.

Expected End With

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedEndWithC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedEndWithS"}

A **With** block must be terminated. This error has the following cause and solution:

- You did not properly terminate a **With** block.
  Place an **End With** statement at the end of the block.

For additional information, select the item in question and press F1.

Expected Function or variable

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedFuncC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedFuncS"}

The syntax of your statement indicates a variable or function call. This error has the following cause and solution:

- The name isn't that of a known variable or **Function** procedure.

  Check the spelling of the name. Make sure that any variable or function with that name is visible in the portion of the program from which you are referencing it. For example, if a function is defined as **Private** or a variable isn't defined as **Public**, it's only visible within its own module.
- You are trying to inappropriately assign a value to a procedure name. For example if `MySub` is a **Sub** procedure, the following code generates this error:

  ```
  MySub = 237 ' Causes Expected Function or variable error
  ```

  Although you can use assignment syntax with a **Property Let** procedure or with a **Function** that returns an object or a **Variant** containing an object, you can't use assignment syntax with a **Sub**, **Property Get**, or **Property Set** procedure.

For additional information, select the item in question and press `F1`.

Expected procedure, not module

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedFuncNotModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedFuncNotModuleS"}

There is no procedure by this name in the current scope, but there is a module by this name. You can call a procedure, but not a module. This error has the following cause and solution:

- The name of a module is used as a procedure call.

  Check the spelling of the procedure name, and make sure the procedure you are trying to call isn't private to another module.

For additional information, select the item in question and press F1.

Expected procedure, not project or library

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedFuncnotprojectC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedFuncNotProjectS"}

There is no procedure by this name in the current scope, but there is a project by this name. You can call a procedure, but not a project. This error has the following cause and solution:

- The name of a project is used as a procedure call.

  Check the spelling of the procedure name, and make sure the procedure you are trying to call isn't private to another module.

For additional information, select the item in question and press F1.

Expected procedure, not user-defined type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedFuncNotRecordC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedFuncNotRecordS"}

There is no procedure by this name in the current scope, but there is a user-defined type by this name. You can call a procedure, but not a user-defined type. This error has the following cause and solution:

- The name of a user-defined type is used as a procedure call.

  Check the spelling of the procedure name, and make sure the procedure you are trying to call isn't private to another module.

For additional information, select the item in question and press F1.

Expected procedure, not variable

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedFuncNotVarC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedFuncNotVarS"}

There is no procedure by this name in the current scope, but there is a variable by this name. You can call a procedure, but not a variable. This error has the following cause and solution:

- The name of a variable is used as a procedure call.

  The error may also be caused by misspelling the name of a valid procedure, because that can be misconstrued as an implicitly defined variable. Check the spelling of the procedure name, and make sure the procedure you are trying to call isn't private to another module.

For additional information, select the item in question and press F1.

Expected Sub, Function, or Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedSubC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedSubS"}

The syntax of your statement indicates a **Sub**, **Function,** or **Property** procedure invocation. This error has the following cause and solution:

- The specified name isn't that of a **Sub**, **Function**, or **Property** procedure in scope in this part of your program.

  Check the spelling of the name. Note that if the procedure is defined as **Private**, it can only be called from within its module.

For additional information, select the item in question and press F1.

Expected user-defined type, not project

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedTypeNotProjC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedTypeNotProjS"}

There is no user-defined type by this name in the current scope, but there is a project by this name. You can define a variable as having user-defined type, but not project type. This error has the following cause and solution:

- The name of a project is used as a user-defined type.

  Check the spelling of the name of the user-defined type, and make sure the user-defined type isn't private to another module.

For additional information, select the item in question and press F1.

Expected variable or procedure, not Enum type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgEnumExpectedVarC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEnumExpectedVarS"}

The name of an **Enum** type only appears in a statement declaring an enumeration of the type or as a qualifier. This error has the following cause and solution:

- An **Enum** type name is used instead of the name of an enumeration variable of the type.

  Declare a variable of the **Enum** type or find a previous declaration in the current scope and use that variable.
- An **Enum** type name is used instead of a variable or procedure name.

  Check the spelling of the identifier that caused the error. Use the name of a variable or procedure where you specified an **Enum** type.

For additional information, select the item in question and press F1.

Expected variable or procedure, not module

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgModuleExpectedVarC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgModuleExpectedVarS"}

There is no variable or procedure by this name in the current scope, but there is a module by this name. This error has the following cause and solution:

- The name of a module is used as a variable or procedure.

  Check the spelling of the variable or procedure name, and make sure the name you want to refer to isn't private to another module. A module name can be a qualifier, but can't stand alone.

For additional information, select the item in question and press F1.

Expected variable or procedure, not project

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgProjectExpectedVarC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgProjectExpectedVarS"}

There is no variable or procedure by this name in the current scope, but there is a project by this name. This error has the following cause and solution:

- The name of a project is used as a variable or procedure.

  Check the spelling of the variable or procedure name, and make sure the name you want to refer to isn't private to another module. A project name can be a qualifier, but can't stand alone.

For additional information, select the item in question and press F1.

Expected: <various>

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedTokensS"}

An expected part of the syntax was not found. The error is usually located to the left of the selected item, but isn't always obvious. For example, you can invoke a **Sub** procedure with or without the **Call** keyword. However, if you use the **Call** keyword, you must enclose the argument list in parentheses. This error has the following causes and solutions:

- **Expected: End of Statement**. Improper use of parentheses in a procedure invocation:

  ```
  X = Workbook.Add F:= 5  ' Error due to no parentheses.
  Call MySub 5          ' Error due to no parentheses.
  ```

  Use parentheses in a function call that specifies arguments or with a **Sub** procedure invocation that uses the **Call** keyword.

- **Expected:)**. Incorrect syntax for a procedure call. For example, a function call can't stand by itself, and **Sub** procedure calls sometimes require the **Call** keyword, depending on how you specify their arguments.

  ```
  Workbook.Add (X:=5, Y:=7)  ' Function call without expression.
  YourSub(5, 7)            ' Sub invocation without Call.
  ```

  Always use function calls in expressions. If you have multiple arguments enclosed in parentheses in a **Sub** procedure call, you must use the **Call** keyword.

- **Expected: Expression**. For example, when pasting code from the **Object Browser**, you may have forgotten to specify a value for a named argument.

  ```
  Workbook.Add (X:= )   ' Error because no value assigned to
                        ' named argument.
  ```

  Either add a value for the argument, or delete the argument if it's optional.

- **Expected: Variable**. For example, you may have used restricted keywords for variable names. In the following example, the **Input #** statement expects a variable as the second argument. Since **Type** is a restricted keyword, it can't be used as a variable name.

  ```
  Input # 1, Type' Type keyword invalidly used as
                  ' variable name.
  ```

  Rename the variable so it doesn't conflict with restricted keywords.

For additional information, select the item in question and press F1.

External name not defined

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExtNameUndefS"}

You can indicate external names by enclosing them in brackets (**[ ]**). This error has the following cause and solution:

- The identifier specified within brackets has no meaning to the host application.
  Check the spelling of the name to be sure it's consistent with that expected by the host.

For additional information, select the item in question and press F1.

File format no longer supported

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgUnsupFormatS"}

Not all file formats are readable under all circumstances. This error has the following cause and solution:

- You are trying to load a binary file created with an earlier version of Visual Basic.

  Load the file into the version of Visual Basic with which the file was created, save it in text format, then try to load it again.

For additional information, select the item in question and press F1.

Fixed or static data can't be larger than 64K

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgSizeTooBigS"}

Fixed and static data include nonautomatic variables, fixed-length strings, and fixed arrays. This error has the following causes and solutions:

- You attempted to allocate more than 64K of module-level data.

  Reduce the amount of declared data. Note that although the size limit for module-level data is 64K, module-level variable-length strings and arrays can exceed this limit.
- You attempted to allocate more than 64K of static procedure-level data in the module.

  Reduce the amount of this type of data declared. Static data from all procedures in a module is limited to a total of 64K (not 64K per procedure). Note that static variable-length strings and arrays can exceed this limit.
- The size of a user-defined type exceeds 64K.

  Reduce the size of the user-defined type. Generally the size of a user-defined type equals the sum of the sizes specified for its elements. On some platforms there may be padding between the elements to keep them aligned on word boundaries. If you nest one user-defined type in another, the size of the nested type must be included in the size of the new type.
- In a procedure, you tried to declare a variable of user-defined type that requires more than 32K.

  Although the size limit of a variable of user-defined type is 64K at module level, variables of user-defined type in procedures can't exceed 32K. Reduce the size required for the user-defined type, or use a module-level variable.
- The size of a fixed-length string declared within a procedure exceeds 65,464.

  Reduce the length of the fixed-length string. Note that variable-length strings can exceed this limit.

For additional information, select the item in question and press F1.

For control variable already in use

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgForIndexInUseC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgForIndexInUseS"}

When you nest **For...Next** loops, you must use different control variables in each one. This error has the following cause and solution:

- An inner **For** loop uses the same counter as an enclosing **For** loop.

  Check nested loops for repetition. For example, if the outer loop uses `For Count = 1 To 25`, the inner loops can't use `Count` as the control variables.

For additional information, select the item in question and press F1.

For Each control variable must be Variant or Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgForEachVariantC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgForEachVariantS"}

A control variable is the *element* part of the **For Each...Next** statement syntax. This error has the following causes and solutions:

- A collection has a control variable that isn't a **Variant** or **Object** type.
  Make sure the *element* part of the **For Each...Next** is a **Variant** or **Object**.
- An array has a control variable that isn't a **Variant**.
  Make sure the *element* part of the **For Each...Next** is a **Variant**.

For additional information, select the item in question and press F1.

For Each control variable on arrays must be Variant

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgForEachAryVariantC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgForEachAryVariantS"}

A control variable is the *element* part of the **For Each...Next** statement syntax. This error has the following cause and solution:

- An array has a control variable that isn't a **Variant**.
  Make sure the *group* part of the **For Each...Next** is a **Variant** type variable.
- For additional information, select the item in question and press F1.

For Each may not be used on array of user-defined type or fixed-length strings

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgForEachAryTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgForEachAryTypeS"}

For Each constructs are only valid for collections and arrays of intrinsic types, including arrays of objects. Also, arrays of fixed-length strings can't be iterated using **For Each**. This error has the following causes and solutions:

- The elements of the array in your **For Each** construct have a user-defined type.
  Use an ordinary **For...Next** loop to iterate the elements of the array.
- The elements of the array in your **For Each** construct have a fixed-length string type.
  Use an ordinary **For...Next** loop to iterate the elements of the array.

For additional information, select the item in question and press F1.

For Each can only iterate over a collection object or an array

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgForEachCollAryC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgForEachCollAryS"}

The **For Each** construct can only be used with collections and arrays. This error has the following cause and solution:

- You specified an object that isn't a collection or array as the *group* part of the **For Each** syntax.

  Check the spelling of the item over which you want to iterate to make sure it corresponds to a collection or array in scope in this part of your code.

For additional information, select the item in question and press F1.

For without Next

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedNextC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedNextS"}

Every **For** statement must have a matching **Next** statement. This error has the following cause and solution:

- A **For** statement is used without a corresponding **Next** statement.
  Check for an incorrectly matched **For...Next** structure inside the outer **For...Next** structure.

For additional information, select the item in question and press F1.

Forward reference to user-defined type

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgForwardTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgForwardTypeS"}

A user-defined type must be defined before it can be referenced. This error has the following causes and solutions:

- You declared a variable with a user-defined type before the definition of the user-defined type appears. In the following example, the variable `OtherVar` is declared before its type (`OtherType`) is known:

  ```
  Type MyType
     OtherVar As OtherType
  End Type

  Type OtherType
     WholeVar As Integer
     RealVar As Double
  End Type
  ```

  Reposition the type definitions so that the forward reference doesn't occur.

- You nested a user-defined type within itself.

  ```
  Type MyType
     MyVar As Integer
     OtherVar As MyType
  End Type
  ```

  Remove the self-referencing nested type. This may occur indirectly if you nest a type within another type in which the first is already declared. Check the definition of each nested type to eliminate duplication.

For additional information, select the item in question and press F1.

Function call on left side of assignment must return Variant or Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgFunctionLHSC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgFunctionLHSS"}

A function call can appear on the left side of an assignment, but only if the return value of the function is an **Object** or **Variant**. This error has the following cause and solution:

- The return type of the function on the left side of the assignment isn't a **Variant** or **Object**.

  Change the return type. Note that if the return value is an object or a **Variant** that contains an object, the assignment is to the default property of the object. If the **Variant** returned isn't an object, the assignment has no effect.
- Everything in the call is correct, however, it can't be completed. For example, you may be trying to set a property that can only be set at design time.

  Enter design mode and set the property in the **Property** window. Remove the code that tried to set the property programmatically.

For additional information, select the item in question and press F1.

Function marked as restricted or uses a type not supported in Visual Basic

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidTypeLibFunctionS"}

Not every procedure that appears in a type library or object library can be accessed by every programming language. The creator of a type or object library can designate some functions as restricted to prevent their use by macro languages. This error has the following causes and solutions:

- You tried to use a function with a restricted specification.

  You can't use the function in your program. If you have documentation for the object represented by the library, check to see if a method is provided that gives equivalent functionality.
- You tried to use a function that requires a parameter type or has a return type that isn't available in Visual Basic.

  Sometimes you can simulate return types with Visual Basic equivalents. Check the subtypes of the **Variant** data type . This may also work for non-Basic parameter types that are expected as references. However, you can't pass a **Variant** data type by value in an effort to simulate a non-Basic type.

For additional information, select the item in question and press F1.

Identifier too long

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIdTooLongS"}

Identifiers can't be more than 255 characters long. This error has the following cause and solution:

- The identifier exceeds 255 characters.
  Reduce the length of the identifier.

For additional information, select the item in question and press F1.

Identifier under cursor isn't a procedure name

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgViewProcNotProcNameS"}

You tried to view a procedure, but the identifier at the insertion point was not a procedure name. This error has the following cause and solution:

- Identifier isn't a procedure name.
  Check the spelling of the identifier.

For additional information, select the item in question and press F1.

Incorrect DLL version

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIncorrectVersionS"}

Each version of Visual Basic works only with its corresponding dynamic-link library (DLL). This error has the following cause and solution:

- Your version of the Visual Basic dynamic-link library doesn't match the version expected by this host application. The program is attempting to call routines in a DLL, but the version of the library is inconsistent with either Visual Basic or the host application

  Obtain the correct version of the library, and make sure earlier versions don't precede the proper one on your path.

For additional information, select the item in question and press F1.

Incorrect OLE version

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIncorrectOleVersionS"}

Your versions of the OLE dynamic-link libraries (DLL) don't match those expected by the host application. In Microsoft Windows, the application searches for the DLLs first in the current directory, then along your path settings, and then in the WINDOWS\SYSTEM directory. This error has the following cause and solution:

- Earlier OLE DLLs were encountered in the search before the DLLs expected by the host application.

  You should not try to use both versions of the DLLs.

Note that on the Macintosh, OLE files are normally only found in the Extensions folder so it is unlikely that this error will occur.

For additional information, select the item in question and press F1.

Insufficient Immediate window memory to create variable

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgVarInNestedPaneS"}

Memory in the **Immediate** window is limited. This error has the following cause and solution:

- You specified a variable in the **Immediate** window that must be instantiated, since it wasn't created in the program's code context.

  Delete the reference to the variable in the **Immediate** window, or declare the variable in the program's code context so that it doesn't have to be created in the **Immediate** window.

For additional information, select the item in question and press F1.

Insufficient memory to save Undo information

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantSaveUndoInfoS"}

You won't be able to choose the **Undo** command to restore the current state after you perform this edit. This error has the following cause and solution:

- The edit is too large to save for **Undo**.
  Try performing the edit in smaller increments.

For additional information, select the item in question and press F1.

Insufficient project information to load project on platform or with version now being used

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgCantLoadFromCanonicalC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantLoadFromCanonicalS"}

When you save a project, extra information is saved to permit loading the project with a later version or on a different platform. This error has the following cause and solution:

- When you saved this project, you received an error indicating that insufficient memory was available to save all the information that might be necessary to load the project with a different version or to load it on a different platform.

  Load the file using the Visual Basic version in which it was previously saved, or on the platform in which it was previously saved, and then save it again. If you don't receive any error messages during that save, you should be able to load it in the current version or on the current platform.

For additional information, select the item in question and press F1.

Interface not valid for Implements

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvalInterfForImplementsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalInterfForImplementsS"}

Not all interfaces can be implemented in Visual Basic. This error has the following cause and solution:

- The interface contains some element that can't be supported by Visual Basic. For example, Visual Basic has no equivalent to the unsigned long integer type, Visual Basic can't designate a procedure parameter as "out-only." Although Visual Basic supports the use of the underscore character (_) in Visual Basic identifiers, it can't implement an interface that uses underscore characters in the names of its members.

  You can't implement the interface in Visual Basic.

For additional information, select the item in question and press F1.

Invalid Access mode

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgWrongAccessC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgWrongAccessS"}

In your **Open** statement, you specified a type of access that was invalid for the specified file type. This error has the following causes and solutions:

- You attempted to open a file for **Input**, but specified an illegal access mode.

  You can omit the access mode specification when opening a file for input, but if you specify it, the access mode must be **Read**. Both **Write** and **Read Write** are invalid access modes on a file opened for **Input**.
- You attempted to open a file for **Append**, but specified an invalid access mode.

  You can omit the access mode specification when opening a file for append, but if you specify it, the access mode must be **Write**. Both **Read** and **Read Write** are invalid access modes on a file opened for **Append**.
- You attempted to open a file for **Output**, but specified an invalid access mode.

  You can omit the access mode specification when opening a file for output, but if you specify it, the access mode must be **Write**. Both **Read** and **Read Write** are invalid access modes on a file opened for **Output**.

For additional information, select the item in question and press F1.

Invalid attribute in Sub, Function, or Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvalidAttrInSubC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidAttrInSubS"}

Some attributes are invalid within procedures. This error has the following cause and solution:

- A **Public** or **Private** attribute appears within the body of a procedure definition.

  Remove the attribute from the procedure. To give the variable wider scope, move the declaration to module level. Variables declared within procedures are always **Private**.

For additional information, select the item in question and press F1.

Invalid character

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIllegalCharS"}

The character can't be used in the current context. This error has the following cause and solution:

- You probably used an invalid character, such as a bracket or hyphen, as part of a variable name. Compare the spelling of the name with its declaration.

For additional information, select the item in question and press F1.

Invalid data format

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvDataReadS"}

The data read from a file wasn't in the expected format. This error has the following causes and solutions:

- A project file or object library file is either corrupted or in a format that can't be understood.

  Get a new version of the project file or object library file.
- You may have attempted to load an .exe file into a module.

  Load the source code instead.
- You may have used the **References** dialog box and **Object Browser** to add a reference to a file that isn't a valid object library or contains a Basic project in a format not supported by the host application. For example, Microsoft Excel can't understand .bas or .frm files, or Microsoft Project files containing Basic code.

  Load the questionable file into the application in which it was created, and then save it in a compatible format. For example, object library source code can be processed through MkTypLib; and QuickBasic, and Visual Basic code can be saved in text format, and so on.

For additional information, select the item in question and press F1.

Invalid data type for constant

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgConstTypeErrorS"}

Not all types of data can be assigned to constants. This error has the following cause and solution:

- You tried to declare the type of a constant to be a user-defined type, an object, or an array. Remove the declaration or redeclare as a variable.

For additional information, select the item in question and press F1.

Invalid in Immediate window

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgILLImmedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIIIImmedS"}

Not all statements are permitted in the **Immediate** window. This error has the following causes and solutions:

- A declarative statement was used. For example, **Const**, **Declare**, **Def***type*, **Dim**, **Function**, **Option Base**, **Option Explicit**, **Option Compare**, **Option Private**, **Private**, **Public**, property procedure declaration statements (**Property Let**, **Property Set**, and **Property Get**), **ReDim**, **Static**, **Sub**, and **Type** are not allowed in the **Immediate** window.

  Remove the declarative statements from the **Immediate** window.
- A control flow statement was used, for example, **Sub**, **Function**, **Property**, **GoSub**, **GoTo**, **Return**, and **Resume**.

  Remove these statements from the **Immediate** window.
- There is no logical connection made between separated physical lines in the **Immediate** window, so statements formatted as multiple physical lines, such as a block **If** statement, can't be properly executed.

  Such blocks can be typed on a single physical line, with each statement separated from the next by a colon (**:**). Conversely, you can extend a single statement across physical lines in the **Immediate** window by using the line-continuation character, which is a space followed by an underscore (**_**).
- You tried to execute some code in the **Immediate** window that invalidates the current state of your program and requires you to reinitialize the program.

  Remove the code in question from the **Immediate** window.

For additional information, select the item in question and press F1.

Invalid inside procedure

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvInsideProcC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvInsideProcS"}

The statement can't occur in a **Sub** or **Function** procedure. This error has the following cause and solution:

- One of the following statements appears in a procedure: **Declare**, **Def***type*, **Private**, **Public**, **Option Base**, **Option Compare**, **Option Explicit**, **Option Private**, **Enum** and **Type**.

  Remove the statement from the procedure. The statements can be placed at module level.

For additional information, select the item in question and press F1.

Invalid length for fixed-length string

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgIllegalFixedStringLenC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIllegalFixedStringLenS"}

This error has the following causes and solutions:

- A fixed-length string is declared with zero length.
  A fixed-length string must have at least one character.
- A fixed-length string exceeds the limit of 65,526.
  Reduce the length specified for the fixed-length string.

Note Variable-length strings can exceed the upper limit and can have zero length.

For additional information, select the item in question and press F1.

Invalid Next control variable reference

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgNextForMatchC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNextForMatchS"}

The numeric variable in the **Next** part of a **For...Next** loop must match the variable in the **For** part. This error has the following cause and solution:

- The variable in the **Next** part of a **For...Next** loop differs from the variable in the **For** part. For example:

  ```
  For Counter = 1 To 10
     MyVar = Counter
  Next Count
  ```

  Check the spelling of the variable in the **Next** part to be sure it matches the **For** part. Also, be sure you haven't inadvertently deleted parts of the enclosing loop that used the variable.

For additional information, select the item in question and press F1.

Invalid or unqualified reference

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgBadWithRefC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadWithRefS"}

An identifier beginning with a period is valid only within a **With** block. This error has the following cause and solution:

- The identifier begins with a period.
  Complete the qualification of the identifier or remove the period.

For additional information, select the item in question and press F1.

Invalid outside procedure

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvOutsideProcC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvOutsideProcS"}

The statement must occur within a **Sub** or **Function**, or a property procedure (**Property Get**, **Property Let**, **Property Set**). This error has the following cause and solution:

- An executable statement, **Static** or **ReDim**, appears at module level.

  Static is unnecessary at module level, since all module-level variables are static. Use **Dim** instead of **ReDim** at module level. To create a dynamic array at module level, declare it with **Dim** using empty parentheses.

Note At module level, you can use only comments and declarative statements, such as **Const**, **Declare**, **Def***type*, **Dim**, **Option Base**, **Option Compare**, **Option Explicit**, **Option Private**, **Private**, **Public**, and **Type**. The **Sub**, **Function**, and **Property** statements occur outside the body of their procedures, but within the procedure declaration.

For additional information, select the item in question and press F1.

Invalid ParamArray use

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgillegalparamarrayuseC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIllegalParamArrayUseS"}

The parameter defined as **ParamArray** is used incorrectly in the procedure. This error has the following causes and solutions:

- You attempted to pass **ParamArray** as an argument to another procedure that expects an array or a **ByRef Variant**.

  Assign the **ParamArray** parameter to a **Variant**, and then pass the variant.
- You attempted to use an **Erase** or **ReDim** statement with a **ParamArray** parameter within its procedure.

  Remove the **Erase** or **ReDim**. These operations can't be performed on the **ParamArray** parameter.

For additional information, select the item in question and press F1.

Invalid procedure name

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidProcNameS"}

There are restrictions in procedure-naming beyond the rules for naming identifiers. This error has the following cause and solution:

- You attempted to define a procedure, but the name used for the procedure is invalid because the host already uses that identifier for another purpose. For example, if the host application is Microsoft Excel, you can't define a procedure with the name `R1C1` because that identifier is already used by Microsoft Excel.

  Choose another name for the procedure.
- Your procedure name is that of a restricted keyword, exceeds 255 characters, or doesn’t begin with a letter.

  Choose a different name for the procedure.

For additional information, select the item in question and press `F1`.

Invalid qualifier

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgQualNotObjectRecordC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgQualNotObjectRecordS"}

Qualifiers are used for disambiguation. This error has the following cause and solution:

- The qualifier does not identify a project, module, object, or a variable of user-defined type within the current scope.

  Check the spelling of the qualifier. Make sure that the qualifying identifier is within the current scope. For example, a variable of user-defined type in a **Private** module is visible only within that module.

For additional information, select the item in question and press F1.

Invalid ReDim

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvalidReDimC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidRedimS"}

Not every array can be redimensioned. This error has the following causes and solutions:

- A variable was implicitly declared a **Variant**, and you attempted to use **ReDim** to change it to an array.

  A **Variant** can contain an array, but if it isn't explicitly declared, you can't use **ReDim** to make it into an array. Declare the **Variant** before using **ReDim** to specify the number of elements it can contain. For example, in the following code, `ReDim AVar(10)` causes an invalid **ReDim** error, but `ReDim BVar(10)` does not:

  ```
  AVar = 1 ' Implicit declaration of AVar.
  ReDim AVar(10) ' Causes invalid ReDim error.
  .
  .
  .
  Dim BVar ' Explicit declaration of BVar.
  ReDim BVar(10) ' No error.
  ```

- You tried to use **ReDim** to change more than one dimension of an array contained within a **Variant**.

  You can only use **ReDim** to change the size of the last dimension of an array in a **Variant**. To create an array with multiple dimensions that can be redimensioned, the array can't be contained within a **Variant**, and you have to declare it the normal way.

- You can use **ReDim** only to change the number of elements in a normal array, not the type of those elements.

  If you want an array in which you can change the types of the elements, use an array contained within a **Variant**. If you declare the array first, changing the types and the number of its elements can be accomplished as follows:

  ```
  Dim MyVar As Variant ' Declare the variable.
  ReDim MyVar(10) As String  ' ReDim it as array of String subtypes.
  ReDim MyVar(20) As Integer ' ReDim it as array of Integer subtypes.
  ReDim MyVar(5) As Variant  ' ReDim it as array of Variant subtypes.
  ```

- You attempted to use **ReDim** with an array that is a member of an Automation object.

  Remove the **ReDim**.

Note If you don't specify a type for a variable, the variable receives the default type, **Variant**. This isn't always obvious. For example, the following code declares two variables, the first, `MyVar`, is a **Variant**; the second, `AnotherVar`, is an **Integer**.

```
Dim MyVar, AnotherVar As Integer
```

For additional information, select the item in question and press `F1`.

Invalid syntax for conditional compiler constant declarations

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidConstValSyntaxS"}

Entering conditional compiler constants in an **Options** dialog box differs from declaring constants in code. This error has the following cause and solution:

- You used improper syntax when entering a constant declaration in the in an **Options** dialog box.

  The only valid syntax is a simple assignment of an integer value to the identifier. Make sure the syntax for the entry is as follows, with each constant separated by a colon (**:**):

  constantname **=** [{+ | − }]*integervalue* **:** [{+ | − }]*constantname* **=** *integervalue* [...]

For additional information, select the item in question and press F1.

Invalid type-declaration character

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgIncorrectTypeCharC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIncorrectTypeCharS"}

Type-declaration characters are valid, but don't exist for all data types; they aren't permitted in some situations. This error has the following causes and solutions:

- A type-declaration character is appended to a variable declared in a **Private**, **Public**, or **Static** statement with an **As** clause.

  Remove the type-declaration character.
- A type-declaration character is appended to an inconsistent literal. For example, since the ampersand (**&**) is the type-declaration character for a **Long** integer, appending it to a literal of a different type causes this error:

  ```
  10.253&
  ```

  Remove the type-declaration character or replace it with the correct one.

For additional information, select the item in question and press F1.

Invalid use of AddressOf operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvalidAddressOfUseC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidAddressOfUseS"}

The **AddressOf** operator modifies an argument to pass the address of a function rather than passing the result of the function call. This error has the following cause and solution:

- You tried to use **AddressOf** with the name of a class method.

  Only the names of Visual Basic procedures in a .bas module can be modified with **AddressOf**. You can't specify a class method.
- The procedure name modified by **AddressOf** is defined in a module in a different project.
- You tried to modify the name a DLL function or a function defined in a type library with **AddressOf**.
- DLL and type library functions can't be modified with **AddressOf**.

  The procedure definition must be in a module in the current project. Move the definition to a module in this project or include its current module in the project.

For additional information, select the item in question and press F1.

Invalid use of Me keyword

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvalidMeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidMeS"}

The **Me** keyword can appear in class modules. This error has the following causes and solutions:

- The **Me** keyword appeared in a standard module.

  The **Me** keyword can't appear in a standard module because a standard module doesn't represent an object. If you copied the code in question from a class module, you have to replace the **Me** keyword with the specific object or form name to preserve the original reference.
- The **Me** keyword appeared on the left side of a **Set** assignment, for example:

  ```
  Set Me = MyObject ' Causes "Invalid use of Me keyword" message.
  ```

  Remove the **Set** assignment.

Note The **Me** keyword can appear on the left side of a **Let** assignment, in which case the default property of the object represented by **Me** is set. For example:

```
Let Me = MyObject ' Valid assignment with explicit Let.
Me = MyObject  ' Valid assignment with implicit Let.
```

For additional information, select the item in question and press F1.

Invalid use of New keyword

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvalidNewC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidNewS"}

The **New** keyword can only be applied to a creatable object (an instance of a class or Automation object). This error has the following causes and solutions:

- You tried to instantiate something that can have only one instance. For example, you tried to create a new instance of a module by specifying `Module1` in a statement like the following:

  ```
  Dim MyMod As New Module1
  ```

  You can't create the new instance, since a module can have only one instance.

- You tried to instantiate an Automation object, but it was not a creatable object. For example, you tried to create a new instance of a list box by specifying ListBox in a statement like the following:

  ```
  ' Valid syntax to create the variable.
  Dim MyListBox As ListBox
  Dim MyFormInst As Form
  ' Invalid syntax to instantiate the object.
  Set MyFormInst = New Form
  Set MyListBox = New ListBox
  ```

  ListBox and Form are class names, not specific object names. You can use them to specify that a variable will be a reference to a certain object type, as with the valid **Dim** statements above. But you can't use them to instantiate the objects themselves in a **Set** statement. You must specify a specific object, rather than the generic class name, in the **Set** statement:

  ```
  ' Valid syntax to create new instance of a form or list box.
  Set MyFormInst = New Form1
  Set MyListBox = New List1
  ```

For additional information, select the item in question and press F1.

Invalid use of object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgConstQualC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgConstQualS"}

You tried to use an object in an incorrect way. This error has the following causes and solutions:

- You tried to discontinue an object reference by assigning **Nothing** to it but omitted the **Set** keyword:

  ```
  MyObject = Nothing
  ```

  Use the **Set** statement to set an object to **Nothing**. Assuming `MyObject` is an object, you must set it to **Nothing** with the **Set** statement:

  ```
  Set MyObject = Nothing
  ```

  Omitting the **Set** keyword is an implicit use of **Let**, which causes an attempt to perform a value assignment, rather than a reference assignment. **Nothing** can't be used in a value assignment.
- You attempted to use **Nothing** in an expression.

  Rewrite the expression without the **Nothing**.

For additional information, select the item in question and press `F1`.

Invalid use of Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgUnsuitablefuncpropmatchC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgUnsuitableFuncPropMatchS"}

You are using one kind of **Property** procedure where a different kind is expected. This error has the following causes and solutions:

- You are trying to write to a property that is read-only.

  If the only property procedure defined for the property is a **Property Get**, you can't assign a value to the property. Either write an appropriate **Property Let** procedure, or don't attempt to write to the property.
- You are trying to read a property that is write-only.

  If the only property procedure defined for the property is a **Property Let**, you can't read the value of the property. Either write an appropriate **Property Get** procedure, or don't attempt to write to the property.
- You are trying to set a reference but the property has only **Property Get** or **Property Let** procedures.

  Either write a **Property Set** procedure for the property, or don't try to set a reference to it.

For additional information, select the item in question and press F1.

Invalid watch expression

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadWatchExprS"}

The specified watch expression isn't a valid expression. This error has the following cause and solution:

- The watch expression is syntactically incorrect.

  Check the syntax of all components in the expression. Note that the syntax for a watch expression corresponds to the locale of the project in which the expression being watched is defined.

For additional information, select the item in question and press F1.

Label not defined

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgLabelNotDefinedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLabelNotDefinedS"}

This error has the following cause and solution:

- A line label or line number is referred to (for example in a **GoTo** statement), but doesn't occur within the scope of the reference.

  The label must be within the procedure that contains the reference. Line labels are visible only in their own procedures.

For additional information, select the item in question and press F1.

Language/country setting has changed

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDefaultLCIDChangeS"}

This error has the following cause and solution:

- You changed the default language/country setting for the Visual Basic environment. Be aware that this may cause problems, although by itself it is legal.

For additional information, select the item in question and press F1.

Line isn't an executable statement

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoExecStmtS"}

Declarations and comments are not executable statements. This error has the following cause and solution:

- You chose the **Step To Cursor** command, but the cursor was not on a line containing an executable statement.

  Place the cursor on an executable statement near the current line.

For additional information, choose the **Step To Cursor** command on the **Run** menu and press F1.

Line too long

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLineTooLongS"}

A physical line of Visual Basic code can contain up to 1023 characters. This error has the following cause and solution:

- A line contains too many characters.

  You can create a longer logical line by joining physical lines with a line-continuation character, a space followed by an underscore (_). Up to 10 physical lines can be joined with line-continuation characters to form a single logical line. Thus, a logical line could potentially contain a total of 10,230 characters. Beyond that, you must break the line into individual statements or assign some expressions to intermediate variables.

For additional information, select the item in question and press F1.

Loop without Do

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgLoopNoMatchingDoC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLoopNoMatchingDoS"}

A **Do** loop must begin with a **Do** statement. This error has the following cause and solution:

- You have an unterminated loop block nested within another loop.
  Check to verify that the correct **Do...Loop** syntax is used.

For additional information, select the item in question and press F1.

LSet allowed only on strings and user-defined types

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgLsetTypeErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLsetTypeErrorS"}

LSet is used to left align data within strings and variables of user-defined type. This error has the following causes and solutions:

- The specified variable isn't a string or user-defined type.

  If you are trying to block assign one array to another, **LSet** does not work. You must use a loop to assign each element individually.
- You tried to use **LSet** with an object.

  LSet can also be used to assign the elements of a user-defined type variable to the elements of a different, but compatible, user-defined type. Although objects are similar to user-defined types, you can't use **LSet** on them. Similarly, you can't use **LSet** on variables of user-defined types that contain strings, objects, or variants.

For additional information, select the item in question and press F1.

LSet not allowed

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgLsetOwnerRecordC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLsetOwnerRecordS"}

LSet is used to left-align data within strings and variables of user-defined type. This error has the following cause and solution:

- You tried to use **LSet** on a user-defined type containing strings, objects, or variants.
  You must assign the elements individually from this variable of user-defined type to another.

For additional information, select the item in question and press F1.

Maximum number of watch expressions added

There is a limit to the number of watch expressions the **Watch** window can contain. This error has the following cause and solution:

- You added the maximum number of watch expressions to the **Watch** window.

  Remove one watch expression for each new one you want to add, or display the value you want to know about in the **Immediate** window.

Method or data member not found

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgIdentNotMemberC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIdentNotMemberS"}

The collection, object, or user-defined type doesn't contain the referenced member. This error has the following causes and solutions:

- You misspelled the object or member name.

  Check the spelling of the names and check the **Type** statement or the object documentation to determine what the members are and the proper spelling of the object or member names.
- You specified a collection index that's out of range.

  Check the **Count** property to determine whether a collection member exists. Note that collection indexes begin at 1 rather than zero, so the **Count** property returns the highest possible index number.

For additional information, select the item in question and press F1.

Member identifier already exists in object module from which this object module derives

Identifiers used for object module members can't conflict with names already used in an object module from which they derive. This error has the following cause and solution:

- A procedure or data member identifier in your object module uses an identifier already used in the object module from which it derives. For example, a form has a **BackColor** property, so the following code would cause this error:

  ```
  ' Form already has a BackColor property.
  Dim BackColor As Integer' Generates the error.

  Function BackColor() ' Generates the error.
  End Function
  ```

  Change the identifier that conflicts with the member identifier in your object module.

Note The following names cannot be used as property or method names because they belong to the underlying IUnknown and IDispatch interfaces: QueryInterface, AddRef, Release, GetTypeInfoCount, GetTypeInfo, GetIDsOfNames, Invoke. Using these names causes a compilation error.

For additional information, select the item in question and press F1.

Method not valid without suitable object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgIllegalMethodC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIllegalMethodS"}

Not all methods can be performed by all objects. This error has the following cause and solution:

- You called a method without specifying an object, and the method isn't valid for the implicit object. For example, you can't use the **Line** method in a standard module without a valid object qualifier because a standard module can't display the output of the **Line** method.

  Explicitly qualify the method call with an object that can accept the method. For example, you can specify a form or picture box with the **Line** method.

Note Other methods that need an explicit object qualifier when used in a standard module include **Circle**, **Print**, and **PSet**.

For additional information, select the item in question and press F1.

Missing end bracket

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgMissingEndBrackS"}

Brackets in a statement must occur in matching pairs. This error has the following cause and solution:

- An opening bracket isn't matched by a closing bracket.
  Place a closing bracket at the end of the material to be bracketed.

For additional information, select the item in question and press F1.

Module not found

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgModuleNotFoundS"}

Modules aren't loaded from a code reference — they must be part of the project. This error has the following cause and solution:

- The requested module doesn't exist in the specified project. For example, the statement `MyModule.SomeVar = 5` generates this error when `MyModule` isn't visible in the project `MyProject`.

  See your host application documentation for information on including the module in the project.

Module too large

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgModTooLargeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgModTooLargeS"}

A module contains code within the project. This error has the following cause and solution:

- There is too much code in the module.

  Create a new module and move some of the procedures from this module to the new one. If the current module contains module-level declarations of data that must be visible to the procedures in the new module, declare that data as **Public**.

Note Comments aren't counted as lines of code. Therefore, deleting comments doesn't prevent this error.

For additional information, select the item in question and press F1.

Must be first statement on the line

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgMustbe1stStatementC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgMustBe1stStatementS"}

Not all keywords can appear at the beginning of a line of code. This error has the following causes and solutions:

- You preceded a **Sub**, **Function**, or **Property** statement with another statement on the same line.

  A **Sub**, **Function**, or **Property** statement must always be the first statement on any line in which it appears (unless preceded by the keyword **Public**, **Private**, or **Static**).
- You preceded an **End If**, **Else**, or **ElseIf** statement with another statement on the same line.

  An **End If**, **Else**, or **ElseIf** (only when used in a block **If** structure) statement must always be the first statement on any line in which it appears.

For additional information, select the item in question and press F1.

Name conflicts with existing module, project, or object library

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgModNameConflictS"}

Modules, object libraries, and referenced projects must be uniquely named within a project. This error has the following causes and solutions:

- There is already a module, project, or object library with this name referenced in this project. A file name extension isn't considered part of the name, so different extensions can't be used to distinguish one file from another.

  Use a different name for one of the duplicate module, project, or object library references.
- You attempted to add a reference to a project or object library whose file name (without an extension) is the same as the name of one of the current project's modules.

  Change either the module name or the name of the file that could not be added.

For additional information, select the item in question and press F1.

Named argument already specified

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNamedAlreadySpecifiedS"}

You can use a named argument only once in the argument list of each procedure invocation. This error has the following causes and solutions:

- You specified the same named argument more than once in a single call. For example, if the procedure `MySub` expects the named arguments `Arg1` and `Arg2`, the following call would generate this error:

  ```
  Call MySub(Arg1 := 3, Arg1 := 5)
  ```

  Remove one of the duplicate specifications.
- You specified the same argument both by position and with a named argument, for example:

  ```
  Call MySub(1, Arg1 := 3)
  ```

  Remove one of the duplicate specifications.

For additional information, select the item in question and press F1.

Named arguments not allowed

Named arguments aren't permitted in all situations. This error has the following causes and solutions:

- You tried to specify a named argument as an array index, for example:

  ```
  MyVar = MyArray(MyNamedArg := 1)
  ```

  Use an ordinary variable or constant expression as an array index.
- You tried to specify a named argument with an object, for example:

  ```
  MyVar = MyObject(MyNamedArg := 1)
  ```

  Use a variable or constant expression if the object requires an argument. For example, if the default for an object is a method, the object's name represents the default method. If it needs arguments, specify them positionally.
- You tried to specify a named argument with an external name:

  ```
  MyVar = [MyName](MyNamedArg := 1)
  ```

  Use an ordinary variable or constant expression if the external name needs an argument.
- You tried to specify a named argument with a data member of an object, for example:

  ```
  MyVar = [MyObject].MyProperty(MyNamedArg := 1)
  ```

  Use an ordinary variable or constant expression if the data member needs an argument.

For additional information, select the item in question and press F1.

Next without For

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgNextnomatchingforC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNextNoMatchingForS"}

A **Next** statement must have a preceding **For** statement that matches. This error has the following cause and solution:

- A **Next** statement is used without a corresponding **For** statement.

  Check other control structures within the **For...Next** structure and verify that they are correctly matched. For example, an **If** without a matching **End If** inside the **For...Next** structure generates this error.

For additional information, select the item in question and press F1.

No text selected

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoSelectionS"}

You must select some text to perform this operation. This error has the following cause and solution:

- You didn't select any text prior to initiating a Find or Replace operation.

  Some host applications permit you to specify selection scope when searching for and replacing text, even though no text is actually selected. To select text, place the pointer at the beginning of the desired selection, then hold down the SHIFT key and use either the arrow keys or click the end of the text to complete the selection. If you don't want to search selected text, undo the **Selection** option.

For additional information, select the item in question and press F1.

Not enough memory to completely save project

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgOOmemForCanonicalSaveC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOOmemForCanonicalSaveS"}

When a project is saved, extra information is usually saved to make it possible to load the project in a later version or on a different platform. This error has the following cause and solution:

- Memory available is insufficient to save information that will be necessary if you attempt to load the project in a later version of Visual Basic or on a different platform.

  If you plan to open the project in a later version of Visual Basic or on another platform, close some applications and try to save again.

For additional information, select the item in question and press F1.

No watch expression selected

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEmptyInstWatchExprS"}

You must have an expression selected when you choose the **Instant Watch** command. This error has the following cause and solution:

- You didn't select an expression to watch before choosing **Instant Watch**.
  Highlight the watch expression in the **Code** window before choosing **Instant Watch**.

For additional information, select the item in question and press F1.

Object library feature not supported

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgUnsupportedTypeLibFeatureS"}

It's possible to have features in an object library that have no equivalent in Visual Basic. This error has the following cause and solution:

- An attempt was made to access an object library data type or function that can't be supported by Visual Basic.

  Contact the creator of the object library for more information on when it is appropriate to use the object library, and the tools with which it should be used.

For additional information, select the item in question and press F1.

Object library for Visual Basic for Applications not found

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgMissingVBATypelibS"}

The Visual Basic for Applications object library is no longer a standalone file; it is integrated into the dynamic-link library (DLL).

Under unusual circumstances a previous version of the object library (vaxxx.olb or vaxxxx.olb) corresponding to the language of the project might be needed, but not found. This error has the following causes and solutions:

- The object library is missing completely, isn't in the expected directory, or is an incorrect version.

  Search your disk to make sure the object library is in the correct directory, as specified in the host-application documentation.

If the missing library is a language version that is installed by the host application, it may be easiest to simply rerun the setup program. If a project requires a different language object library than the one that accompanies your host application (for example, if someone sends you a project written on a machine set up for a different language), make sure the correct language version of the Visual Basic object library is included with the project and it is installed in the expected location.

Applications may support different language versions of their object libraries. To find out which language version is required, display the **References** dialog box, and see which language is indicated at the bottom of the dialog box.

Object libraries exist in different versions for each platform. Therefore, when projects are moved across platforms, for example, from Macintosh to Microsoft Windows, the correct language version of the referenced library for that platform must be available in the location specified in your host application documentation. Note that some language codes are two characters while others are three characters.

The Visual Basic object library file name is constructed as follows:

- Windows: Application Code + Language Code + [Version].OLB. For example:

  The French Visual Basic for Applications object library for version 2 was vafr2.olb.
- Macintosh: Application Name Language Code [Version] OLB. For example:

  The French Visual Basic for Applications object library for version 2 was VA FR 2 OLB.

If you can't find a missing project or object library on your system, contact the referencing project's author. If the missing library is a Microsoft application object library, you can obtain it as follows:

- If you have access to Microsoft electronic technical support services, refer to the technical support section of this Help file. Under electronic services, you will find instructions on how to use the appropriate service option.
- If you don't have access to Microsoft electronic technical support services, Microsoft object libraries are available upon request as an application note from Microsoft. Information on how to contact your local Microsoft product support organization is also available in the technical support section of this Help file.

For additional information, select the item in question and press F1.

Object library not registered

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLibNotRegisteredS"}

The Visual Basic for Applications object library is no longer a standalone file; it is integrated into the dynamic-link library (DLL).

In earlier versions, when you started an application that uses Visual Basic for Applications, certain object libraries were loaded. This error has the following cause and solution:

- An attempt was made to load a previous version of the Visual Basic for Applications object library (vaxxx.olb) or host-application object libraries. However, the correct language version of these object libraries could not be found in the system registry.

  Reregister your application. On the Macintosh, delete the vba.ini file from the Macintosh Preferences folder, and restart your application.

For additional information, select the item in question and press F1.

Object library's language setting incompatible with current project

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLangMismatchS"}

The reference couldn't be added. This error has the following cause and solution:

- You attempted to add a reference to an object library whose locale isn't compatible with the locale of the current project. The reference was not added. To use that object library, a project whose locale is compatible with it must be created.

  Try registering both Visual Basic for Applications and the host application for the given language. The object library then becomes available in the **References** dialog box.

  Note When Visual Basic is the host application, it isn't possible to change a project's language setting. Any object libraries used must be compatible with the English/U.S. setting.

Type not supported in Visual Basic

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvalidTypeInfoKindC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidTypeInfoKindS"}

Not all types are supported in Visual Basic. This error has the following cause and solution:

- You tried to use a type in your program that has no equivalent in Visual Basic for Applications. For example, Visual Basic has no pointer or unsigned integer type, so if you try to create a variable of one of those types from an object library, this error occurs. In the following example that follows, even though `Rainbow` may be a valid structure, Visual Basic can't create a variable of that type if it contains a type Visual Basic doesn't recognize:

  ```
  Dim MyVar As Rainbow ' Causes error.
  ```

  If the type is a valid parameter type for a function in an object library, this error means only that you can't create a variable of that type in your own code. Although you can't always declare variables with a data type specified in an object's documentation, there is often a Visual Basic equivalent. For example, although Visual Basic has no pointer type, you can pass a pointer to a function to an API function by using the **AddressOf** operator. Also, check the **Variant** type's subtypes. You can often use them as equivalents of types not offered directly in Visual Basic. In some cases, however, Visual Basic simply has no equivalent. For example, data pointers aren't available.

For additional information, select the item in question and press F1.

Only comments may appear after End Sub, End Function, or End Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgStmtBetweenProcsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgStmtBetweenProcsS"}

Only comments, directives, and declarations are permitted outside procedures. This error has the following cause and solution:

- You placed executable code outside a procedure.

  Any nondeclarative lines outside a procedure must begin with a comment delimiter (**'**). Declarative statements must appear before the first procedure declaration. Comments are ignored when the code executes.

For additional information, select the item in question and press F1.

Optional argument must be Variant

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgOptParamMustBeVariantC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOptParamMustBeVariantS"}

Optional arguments can have any intrinsic data type, but it must be a type with a single default value. This error has the following cause and solution:

- You tried to specify **Optional** with a parameter that has no default value, for example, an array. Make sure any argument specified as **Optional** has a default value.

For additional information, select the item in question and press F1.

Option Private Module not permitted in object module

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgOptPrivModInClassC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOptPrivModInClassS"}

Option Private Module makes the contents of a module unavailable to other projects, while preserving their availability to your project. This error has the following cause and solution:

- The statement **Option Private Module** appears in an object module.

  Remove the **Option Private Module** statement from the module. Object modules have the characteristic of **Option Private Module** by default. Changing the default can't be done from code. See your host application's documentation for information on giving object module members wider visibility.

For additional information, select the item in question and press F1.

Out of memory; some watches might have been deleted

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOOMemWatchesDeletedS"}

When your system runs out of memory, watch expressions are deleted to make enough memory available to permit recovery from the condition. This error has the following cause and solution:

- You have too many applications, projects, or modules loaded.

  To continue working, remove any dispensable elements from memory, including unnecessary applications, projects, or modules that may be loaded.

For additional information, select the item in question and press F1.

Out of resources

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOutOfHandlesS"}

Certain types of resources called "handles" are used internally for many operations. This error has the following cause and solution:

- Your code has reached a resource limit.

  Try saving, closing, and reloading your code. If you still receive this error message, reduce the number of objects referenced in the code.

For additional information, select the item in question and press F1.

ParamArray must be declared as an array of Variant

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgparamarraynotarrayC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgParamArrayNotArrayS"}

Each argument to a **ParamArray** parameter can be of a different data type. Therefore, the parameter itself must be declared as an array of **Variant** type. You can also supply any number of arguments to a **ParamArray**. When the call is made, each argument supplied in the call becomes a corresponding element of the **Variant** array. For example:

```
Sub MySub(ParamArray VarArg())
   . . .
End Sub
Call MySub ("First arg", 2, 3.54)
```

This error has the following causes and solutions:

- In the definition of the procedure, the **ParamArray** parameter is defined as an array of a type other than **Variant**.

  Redeclare the parameter type as an array of **Variant** elements.
- No data type was specified for the **ParamArray** parameter, but the procedure definition is within the scope of a **Def***type* statement, so the parameter is implicitly declared as having a type other than **Variant**.

  Use an explicit **As Variant** clause in the specification of the **ParamArray** parameter.

For additional information, select the item in question and press F1.

Please see the Readme file for more information on this error

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgSeeReadmeS"}

Some topics were added to this product after Help was completed. Help explanations for these can be found in the host application's Readme file.

For additional information, select the item in question and press F1.

Private Enum types cannot be used as parameter or return types for public procedures, or as public data members

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgPrivEnumInPubFuncC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgPrivEnumInPubFuncS"}

A **Public** procedure is visible to all modules in a project, while a **Private Enum** type is not visible outside its own module. This error has the following cause and solution:

- Your **Public** procedure is in a **Public** class, but it returns a value or has a parameter that is defined in a standard module or in a **Private** class.

  Declare the **Enum Public**. It must be in a class module.

For additional information, select the item in question and press F1.

Procedure too large

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgProcedureTooLargeS"}

When compiled, the code for a procedure can't exceed 64K. This error has the following cause and solution:

- Code for this procedure exceeds 64K when compiled.
  Break this, and any other large procedures, into two or more smaller procedures.

For additional information, select the item in question and press F1.

Project contains too many procedure, variable, and constant names

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgTooManySymbolsS"}

A project's procedure, variable, constant, and parameter names are stored in a name table. This error has the following cause and solution:

- The number of names in the project's name table exceeds 32,768.

  The name table may contain some temporary duplicates. You can compact the name table by saving the project to a disk, and then closing it. If the problem persists after you reopen the project, reduce the number of names by reusing local variable names in multiple procedures, and then recompact the table by saving the project, closing it, and reopening it.

For additional information, select the item in question and press F1.

Project not found

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgProjectNotFoundS"}

The project was not found. This error has the following cause and solution:

- You specified a project name that can't be found.
  Check the way you specified the project for any errors.

For additional information, select the item in question and press F1.

Qualified name disallowed in module scope

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgQualifiedNameDisallowedS"}

Under some circumstances, some host applications don't permit procedure calls that include qualified names. This error has the following cause and solution:

- You specified a module name in a procedure call using dot notation (*qualifier.item*).

  If you are receiving this error it is probably because the host application already knows the specified qualifier and doesn't need that information in the procedure call. In such a case, you can simply omit the qualifier altogether and the host application will make the procedure call correctly. Check the host application's documentation to find the reason for any other restrictions on qualified names.

For additional information, select the item in question and press F1.

Qualifier must be collection

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgQualNotObjectS"}

The use of an exclamation point between two identifiers is specific to collections. This error has the following cause and solution:

- You used a name on the left side of the exclamation point (**!**) that isn't the name of a collection.

  If the name is supposed to represent a collection, check to make sure the name is spelled correctly. Note that the exclamation point is also the type-declaration character for the **Single** data type. If the name in question isn't supposed to be a collection, perhaps the **!** type-declaration character appended to a variable name has been concatenated with another name.

For additional information, select the item in question and press F1.

Range has no values

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIllegalArrayBoundS"}

There are limitations on the way you can specify the number of elements in an array. This error has the following cause and solution:

- You specified your array boundaries incorrectly. For example, the following ranges are invalid:

  ```
  Dim MyArray(10 To -5)' Descending order not permitted.
  Dim MyArray(0 To 0)     ' No elements in the array.
  ```

  Check to be sure your syntax is correct. For example, the following range is valid:

  ```
  Dim MyArray(-5 To 10)
  ```

For additional information, select the item in question and press F1.

RSet allowed only on strings

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgRsetTypeErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgRsetTypeErrorS"}

RSet is used to right align string data within fixed-length or variable-length strings. This error has the following cause and solution:

- You tried to use the **RSet** statement on a variable that isn't a string.

  If appropriate, try converting the variable to a string. Otherwise, don't use **RSet**.

Note Although the **LSet** statement can be used to assign the elements of one user-defined type variable to the elements of a different, but compatible, user-defined type, such assignments are discouraged because they can't be guaranteed to be portable.

For additional information, select the item in question and press F1.

Run-time error <number>:

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgRunTimeErrorS"}

This error is user-defined, and has the following cause and solution:

- An error was generated that doesn't correspond to a Visual Basic error.

  Handle the error as indicated in the documentation for the object or object library that generated the error.

For additional information, select the item in question and press F1.

Search string too long or complex

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgSearchTooLongC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgSearchTooLongS"}

Visual Basic pattern-matching capabilities aren't unlimited. This error has the following cause and solution:

- The search string specified is too long, or the combination of wildcard character specifications can't be understood.

  Reduce the length or complexity of the search string and retry the search.

For additional information, select the item in question and press F1.

Search string must be specified

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgEmptySearchStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEmptySearchStringS"}

You have to specify a search string. This error has the following cause and solution:

- You tried to perform a search operation without a string for comparison.

  You must specify the search string. While you can specify a zero-length string ("") as a replacement string, a zero-length string has no meaning as a search string.

For additional information, select the item in question and press F1.

Search text isn't found

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgSrchNotFoundC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgSrchNotFoundS"}

No search text was found. This error has the following causes and solutions:

- There is no matching text between the start point and end point of the specified search scope. You can widen the scope of the search if you want to continue searching.
- You failed to provide a left bracket when using a pattern-matching expression. Provide the left bracket.

For additional information, select the item in question and press F1.

Seek failed: can't read/write from the disk

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgseekerrC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgSeekErrS"}

Seek statements are carried out directly to disk. This error has the following causes and solutions:

- You attempted to read from a disk or file that is write-protected, read-only, or locked.

  Remove the write-protected attribute or change the read-only attribute or lock. Note that if the file is locked by another process, you can't remove the lock.
- The file has become unavailable, for example, if a removable disk has been physically changed.

  If the file has been moved to another disk, access it from there. Otherwise, you can't access the file.
- You attempted to read from a project file, an object library, or a type library, but the file has been corrupted.

  Obtain a new copy of the library or project file.

For additional information, select the item in question and press F1.

Select Case without End Select

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedEndSelectC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedEndSelectS"}

Each **Select Case** construct must be terminated with an **End Select** statement. This error has the following cause and solution:

- You used a **Select Case** statement without a corresponding **End Select** statement.

  Check if there is an incorrectly matched **Select Case...End Select** structure inside an outer **Select Case...End Select** structure. If you have nested **Select Case** statements, each must have a matching **End Select**.

For additional information, select the item in question and press F1.

Selected watch expression invalid

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgBadInstWatchExprC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadInstWatchExprS"}

It isn't always possible to select a valid watch expression. This error has the following causes and solutions:

- You chose the **Instant Watch** command, but the selected expression isn't a valid expression. For example, you can't watch a comment or a **Sub** procedure call.

  Select the expression in such a way that it is valid, or choose **Add Watch** and type in a valid expression.
- The watch expression must have code syntax corresponding to the locale of the project that defines the expression being watched.

  Rewrite the expression in a way that is valid for the locale.

For additional information, select the item in question and press F1.

Set Next Statement can only apply to executable lines within current procedure

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalLineForNextStmtS"}

You can choose the **Set Next Statement** command to indicate where a suspended program should begin when execution is continued. This error has the following causes and solutions:

- When you chose the command, the cursor was on a line that didn't contain an executable statement.

  Place the cursor on a line with an executable statement and try again. Declarations, line labels, and comments aren't executable, so lines with only declarations, labels, and comments can't be the targets of the **Set Next Statement** command.
- When you chose the command, the cursor was on a line outside the currently executing procedure.

  Place the cursor on a line within the currently executing procedure.

SHARE.EXE required

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgShareRequiredS"}

The program share.exe must be running when you start Visual Basic. This error has the following cause and solution:

- You didn't start share.exe before starting Visual Basic.

  Close your application, start share.exe, then restart your application. To avoid this problem in the future, place an invocation of share.exe in your autoexec.bat file so share.exe automatically starts when you turn on your machine.

For additional information, select the item in question and press F1.

Specified library or project already referenced

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLibAlreadyAvailS"}

The **Add References** dialog box displays referenced libraries and projects. This error has the following cause and solution:

- You chose the **Browse** button in the **Add References** dialog box, then selected a type library or project that was already listed in the **Available References** section of the **Add References** dialog box.

  To make a project or type library part of your project, display the **References** dialog box and make sure the type library or project name is checked.

For additional information, select the item in question and press F1.

Statement invalid inside Type block

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvInsideTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvInsideTypeS"}

Only element names, their **As** *type* clauses, and comments are allowed within a **Type...End Type** statement block. This error has the following cause and solution:

- You placed an invalid statement in a user-defined type definition.
  Remove anything that isn't a comment, an element name, or an **As** *type* clause.

For additional information, select the item in question and press F1.

Statement invalid outside Type block

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsginvoutsidetypeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvOutsideTypeS"}

The syntax for declaring variables outside a **Type...End Type** statement block is different from the syntax for declaring the elements of the user-defined type. This error has the following causes and solutions:

- You tried to declare a variable outside a **Type...End Type** block or outside a statement.

  When declaring a variable with an **As** clause outside a **Type...End Type** block, use one of the declaration statements, **Dim**, **ReDim**, **Static**, **Public**, or **Private**. For example, the first declaration of `MyVar` in the following code generates this error; the second and third declarations of `MyVar` are valid:

  ```
  MyVar As Double' Invalid declaration syntax.

  Dim MyVar As Double

  Type AType
     MyVar As Double   ' This is valid declaration syntax
  End Type ' because it's inside a Type block.
  ```

- You used an **End Type** statement without a corresponding **Type** statement.

  Check for an unmatched **End Type**, and either precede its block with a **Type** statement, or delete the **End Type** statement if it isn't needed.

For additional information, select the item in question and press F1.

Statement too complex

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgstatementtoocomplexC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgStatementTooComplexS"}

Visual Basic is unable to analyze this statement. This error has the following causes and solutions:

- Your statement can't be parsed due to its complexity.

  Try breaking the statement into several smaller components or replace complex conditional clauses with a combination of logical operators and **If...Then...Else** statements.
- Your statement or function uses too many nested function calls.

  Make function calls earlier and assign the results to specific variables; then use the variables in the statement that is causing the complexity error.

For additional information, select the item in question and press F1.

Statements or labels invalid between Select Case and first Case

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgExpectedCaseC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedCaseS"}

You can place nothing but a comment between the **Select Case** statement and the first **Case** clause. This error has the following cause and solution:

- You placed a statement between **Select Case** and its first **Case** clause. For example:

```
Select Case SomeVar
   ' This is a comment and is valid.
   Stop  ' Even a Stop statement is invalid here.
   Case SomeValue
   . . .
End Select
```

  The **Select Case** statement must be immediately followed by its first **Case** statement. If the intervening expression is a comment, precede it with a comment delimiter (**'**). Otherwise, place the expression where it belongs or delete it.

For additional information, select the item in question and press F1.

Syntax error

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgsyntaxC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgSyntaxS"}

Visual Basic can't determine what action to take. This error has the following causes and solutions:

- A keyword or argument is misspelled.

  Keywords and the names of named arguments must exactly match those specified in their syntax specifications. Check online Help, and then correct the spelling.
- Punctuation is incorrect.

  For example, when you omit optional arguments positionally, you must substitute a comma (**,**) as a placeholder for the omitted argument.
- A procedure isn't defined.

  Check the spelling of the procedure name.
- You tried to specify both **Optional** and **ParamArray** in the same procedure declaration.

  A **ParamArray** argument can't be **Optional**. Choose one and delete the other.
- You tried to define an event procedure with an **Optional** or **ParamArray** parameter.

  Remove the **Optional** or **ParamArray** keyword from the parameter specification.
- You tried to use a named argument in a **RaiseEvent** statement.

  Events do not support named arguments.

For additional information, select the item in question and press F1.

The edit may make the object module incompatible with the previously specified compatible ActiveX component

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIncompatibleSrvrS"}

If a Compatible ActiveX component already exists as a previously distributed executable file or dynamic-link library (DLL), you must be careful not to change its interface. This warning has the following cause and solution:

- You are trying to edit the code of a object module that already is represented by an executable file.

  If you make changes that affect the interface to the object, the class will not be upward compatible with the previous version and so it will not be possible to use the new version in place of the old version for compiled code.

  In Visual Basic, the name of the Compatible ActiveX component appears in the dialog box displayed when you choose **Project Options** from the **Tools** menu.

  Important To accept the edit, click **OK** in the error message dialog box. If you want to undo the edit, click the **Cancel** button.

For additional information, select the item in question and press F1.

The specified region has been searched

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoMoreSrchTextFoundS"}

When you click the **Find Next** and **Replace** buttons in the **Replace** dialog box, the total number of replacements isn't specified. This condition has the following cause and effect:

- When doing a search and replace operation, you can specify the region to be covered — selection, procedure, module, or project.

  If a region is selected, it is the default search region. If no region is selected, the current module is the default search region. To change the scope of a search, select a different option in the **Replace** dialog box.

For additional information, select the item in question and press F1.

The specified region has been searched and 1 replacement was made

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgReplacementDoneS"}

When you click the **Replace All** button in the **Replace** dialog box, this message specifies the total number of replacements. This condition has the following cause and effect:

- During the search and replace operation, only one instance of the specified text was found, and the replacement was made. You can specify the region to be covered — selection, procedure, module, or project.

  If a region is selected, it is the default search region. If no region is selected, the current module is the default search region. To change the scope of a search, select a different option in the **Replace** dialog box.

For additional information, select the item in question and press F1.

The specified region has been searched and the replacements were made

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgReplaceDoneS"}

When you click the **Replace All** button in the **Replace** dialog box, this message specifies the total number of replacements. This condition has the following cause and effect:

- When doing a search and replace operation, you can specify the region to be covered — selection, procedure, module, or project.

  If a region is selected, it is the default search region. If no region is selected, the current module is the default search region. To change the scope of a search, select a different option in the **Replace** dialog box.

For additional information, select the item in question and press F1.

The .vbp file for this project contains an invalid or corrupted library reference ID

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgBadLibIDC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadLibIDS"}

When you save a project for which a reference has been selected from the **References** dialog box, an entry is made in the project's .vbp file (called the .mak file in earlier versions of Visual Basic). For example, the entry for a data access object is:

```
Reference=*\G{00025E01-0000-0000-C000-000000000046}#0.0#0#C:
\WINDOWS\SYSTEM\DAO2516.DLL#Microsoft
DAO 2.5 Object Library
```

This error occurs when such a reference has been edited or corrupted. This error has the following cause and solution:

- A reference in the .vbp file has become invalid.

  Delete the incorrect line from the .vbp file and check the appropriate object library in the **References** dialog box from the **Tools** menu. Then save the project, and the correct information will be entered in the .vbp file.

For additional information, select the item in question and press F1.

This component doesn't raise any events

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgObjectDoesNotFireEventsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgObjectDoesNotFireEventsS"}

An event procedure must correspond to an event that can be raised by an object. This error has the following cause and solution:

- You wrote an event procedure for an object that doesn’t raise events.
  You can't write an event procedure that doesn’t correspond to an event.
- You tried to use **WithEvents** on a class that doesn’t raise events.
  You can't use **WithEvents** on a class that doesn’t raise events.

For additional information, select the item in question and press F1.

Too many arguments

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgTooManyArgsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgTooManyArgsS"}

A procedure can have only 60 arguments. This error has the following cause and solution:

- You specified more than 60 arguments.

  If you must specify more arguments, define a user-defined type to collect multiple arguments of different types, or use a **ParamArray** as the final argument and pass multiple values to it. You can also pass multiple arguments by placing them in an array.

For additional information, select the item in question and press F1.

Too many dimensions

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgTooManyDimsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgTooManyDimsS"}

Arrays can have no more than 60 dimensions. This error has the following causes and solutions:

- You tried to declare an array with more than 60 dimensions.

  Reduce the number of dimensions.
- Your array declaration is within the specified limits, but there isn't enough memory to actually create the array.

  Either make more memory available or reduce the number of dimensions.If your array is an array of **Variant** type or an array contained within a **Variant**, you may be able to create the array with the same number of dimensions by redeclaring it with the data type of its elements. For example, if it contains only integers, declaring it as an array of **Integer** type uses less memory than if each element is a **Variant**.

For additional information, select the item in question and press F1.

Too many line continuations

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgTooManyLineContsS"}

There is a limit to the number of lines you can join with line-continuation characters. This error has the following cause and solution:

- Your code has more than 10 consecutive lines joined with line-continuation characters.

  Make some of the constituent lines physically longer to reduce the number of line-continuation characters needed, or break the construct into more than one statement.

For additional information, select the item in question and press F1.

Too many local, nonstatic variables

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInsufficientLocalSpaceC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInsufficientLocalSpaceS"}

Local, nonstatic variables are variables that are defined within a procedure and reinitialized each time the procedure is called. This error has the following cause and solution:

- The sum of the memory requirements for this procedure's local, nonstatic variables and compiler-generated temporary variables exceeds 32K.

  Declare some of your variables with the **Static** statement where appropriate. **Static** variables retain their value between procedure invocations because they are allocated from different memory resources than nonstatic variables.

For additional information, select the item in question and press F1.

Too many module-level variables

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInsufficientModSpaceC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInsufficientModSpaceS"}

Module-level variables are those declared in the Declarations section of a module, before the module's procedures. This error has the following cause and solution:

- The sum of the memory requirements for all module-level variables in this module exceeds 64K.

  This is the storage limit for this module. If appropriate, you can declare some of your variables as **Public** in another module, or if some module-level variables are used only in one procedure, you can declare them within that procedure. If you declared variables at module level because you want them to retain their value between procedure invocations, you can instead declare them as **Static** within the procedure in which they are referenced.

Note Available space can vary among operating systems.

For additional information, select the item in question and press F1.

Type-declaration character doesn't match declared data type

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgTypecharNoMatchS"}

The data type of a variable can't be changed by appending the type-declaration character for another type. This error has the following cause and solution:

- You declared a variable of a specific type, referenced a variable of the same name in the same scope, and then appended an inconsistent type-declaration character.

  If you want to be able to change the type of data assigned to a variable, declare the variable as a **Variant**. If you simply appended an incorrect type-declaration character, delete or change it.

For additional information, select the item in question and press F1.

Type-declaration character not allowed

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgtypecharnotallowedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgTypecharNotallowedS"}

While using type-declaration characters is valid in Visual Basic, some data types (including **Byte**, **Boolean**, **Date**, **Object**, and **Variant**) have no associated type-declaration characters. This error has the following causes and solutions:

- You tried to use a type-declaration character in the declaration of a variable that uses the **As** clause, for example, with **Dim**, **Static**, **Public**, and so on.

  Either remove the type-declaration character or remove the **As** clause.
- You tried to use a type-declaration character in reference to a variable that was implicitly declared without a type-declaration character:

  ```
  MyVar = 20  ' Implicit declaration.
  MyVar% = 25 ' Generates an error.
  ```

  Either remove the type-declaration character or redeclare the original variable.

Note If an explicit variable declaration contains a type-declaration character, inclusion of the character is optional in later references. For example:

```
Dim MyStr$
MyStr = "Because it was explicitly declared, the $ is optional."
```

For additional information, select the item in question and press F1.

Type-declaration character required

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgTypecharRequiredS"}

The necessity of using type-declaration characters depends on the form of the identifier's declaration. This error has the following cause and solution:

- A variable that was originally implicitly declared with a type-declaration characters was referenced without a type-declaration character. For example:

  ```
  MyStr$ = "Implicit declaration"
  MyStr = "Trying to refer to MyStr$, but error results" _
   & "from calling it MyStr."
  ```

  Either make the declaration explicit, or add the type-declaration character to later references.

Note If an explicit variable declaration contains a type-declaration character, inclusion of the character is optional in later references. For example:

```
Dim MyStr$
MyStr = "Because it was explicitly declared, the $ is optional."
```

For additional information, select the item in question and press F1.

Type mismatch: array or user-defined type expected

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgaggregateparamtypemismatchC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgaggregateParamTypeMismatchS"}

The type of an argument or parameter includes whether or not it is an array or a user-defined type. This error has the following cause and solution:

- Your argument specified a single element of an array or user-defined type, or a simple variable, literal, or constant. However, it is being passed to a parameter that expects a whole array or user-defined type.

  Either change the argument or change the definition of the parameter.
- Your argument specified an array or user-defined type, but it was not of the same type as the parameter.

  Either pass an array of the expected type or change the definition of the parameter declaration.

For additional information, select the item in question and press F1.

Unable to read from the disk

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgReadFaultS"}

There is a general problem reading from a disk. This error has the following causes and solutions:

- The disk drive door is improperly closed.
  If the disk is removable, check the drive door. Close it properly if necessary.
- The disk may be inserted upside down.
  Reinsert the disk right-side up.
- The disk may not be properly formatted.
  Reformat the disk if you are willing to lose the data it currently contains.
- You are experiencing network problems.
  Try restarting your network.
- You placed a disk of a certain density in a drive that can't read it.
  Use a different drive or copy the material you want to a disk that can be read on your drive.

For additional information, select the item in question and press F1.

Unable to write to the disk

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgWriteFaultS"}

There is a general problem writing to a disk. This error has the following causes and solutions:

- The disk drive door is improperly closed.
  If the disk is removable, check the drive door. Close it properly if necessary.
- The disk may be inserted upside down.
  Reinsert the disk right-side up.
- The disk may not be properly formatted.
  Reformat the disk if you are willing to lose the data it currently contains.
- You are experiencing network problems.
  Try restarting your network.
- You placed a disk of a certain density in a drive that can't write to it.
  Use a different drive or copy the material you want to a disk that can be read on your drive.

For additional information, select the item in question and press F1.

Unexpected error; please contact Microsoft Technical Support

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgUnexpectedErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgUnexpectedErrorS"}

This condition may or may not be serious. This error has the following cause and solution:

- An unanticipated error occurred in the host application or in Visual Basic.

  You don't have to assume that this error is serious. However, you should note the number of the error and report it to Microsoft Technical Support.

For additional information, select the item in question and press F1.

Unmatched brackets in search string

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadSearchStringS"}

An opening bracket must be matched by a closing bracket. This error has the following cause and solution:

- You opened a bracket pair in a search string, but did not close it with a matching closing bracket. Place a closing bracket in the appropriate place.

For additional information, select the item in question and press F1.

Unrecognized project language

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgUnknownLcidS"}

The specified code locale for the project to be loaded isn't currently supported by this application. This error has the following causes and solutions:

- The project was created on a system that supports the code locale, but was then moved to a system where that code locale isn't recognized. For example, the ole2nls.dll on the current machine may be a version that doesn't recognize the code locale.

  Install the proper dynamic-link library (DLL) on the current system.
- The correct object library for the project was not found.

  Make sure the correct object libraries are available, for example, make sure your path includes their directories.

For additional information, select the item in question and press F1.

User-defined type can't be passed ByVal

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgrecordmustbebyrefC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgRecordMustBeByrefS"}

User-defined types can only be passed by reference (the default), not by value. The error may not be reported until the call is made. This error has the following cause and solution:

- You placed a **ByVal** keyword in the definition of a parameter that represented a user-defined type.

  Remove the **ByVal** keyword. To keep changes from being propagated back to the caller, **Dim** a temporary variable of the type and pass the temporary variable into the procedure.

For additional information, select the item in question and press F1.

User-defined types and fixed-length strings not allowed as the type of a Public member of an object module; Private object modules not allowed as the type of a public member of a public object modules

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgRecordUsedInClassC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgRecordUsedInClassS"}

You can't create **Public** members of **Private** user-defined types. This error has the following causes and solutions:

- You declared a **Public** variable in an object module with the name of a **Private** user-defined type.

  Declare the variable **Private**, move the **Type...End Type** statement defining the user-defined type to a standard module, or remove the declaration altogether.
- You tried to define a **Public** procedure in an object module with a return type or parameter of user-defined type.

  Declare the procedure **Private**.

For additional information, select the item in question and press F1.

User-defined type not defined

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgundefinedtypeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgUndefinedTypeS"}

You can create your own data types in Visual Basic, but they must be defined first in a **Type...End Type** statement or in a properly registered object library or type library. This error has the following causes and solutions:

- You tried to declare a variable or argument with an undefined data type or you specified an unknown class or object name.

  Use the **Type** statement in a module to define a new data type. If you are trying to create a reference to a class, the class must be visible to the project. If you are referring to a class in your program, you must have a class module of the specified name in your project. Check the spelling of the type name or name of the object.
- The type you want to declare is in another module but has been declared **Private**.

  Move the definition of the type to a standard module where it can be **Public**.
- The type is a valid type, but the object library or type library in which it is defined isn't registered in Visual Basic.

  Display the **References** dialog box, and then select the appropriate object library or type library. For example, if you don't check the **Data Access Object** in the **References** dialog box, types like Database, Recordset, and TableDef aren't recognized and references to them in code cause this error.

For additional information, select the item in question and press F1.

User-defined type without members not allowed

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgzerolenrecordsdisallowedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgZeroLenRecordsDisallowedS"}

User-defined types must have at least one element. This error has the following cause and solution:

- You specified an empty user-defined type in a **Type...End Type** definition.
  Check the **Type** statement for unintended comment delimiters.

For additional information, select the item in question and press F1.

Variable not defined

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgimplicitvarnotallowedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgImplicitVarNotAllowedS"}

You use the **Option Explicit** statement to protect your modules from having undeclared variables and to eliminate the possibility of inadvertently creating new variables when typographical errors occur. This error has the following cause and solution:

- You used an **Option Explicit** statement to require the explicit declaration of variables, but you used a variable without declaring it.

  Explicitly declare the variable, or change the spelling of the variable to match that of the intended variable.

For additional information, select the item in question and press F1.

Variable not yet created in this context

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgVariableNotInstantiatedS"}

A variable has to be created before it can be displayed in the **Watch** window or the **Immediate** window, and before it can have values assigned to it in the **Immediate** window. This error has the following causes and solutions:

- You tried to display the value of a local variable that you just entered in your code before executing at least a **Single Step** command in break mode.

  Step into the code to force compilation of the new statement.
- You tried to display the value of a local variable that you just added in a procedure farther down the call chain by moving to the procedure using the **Calls** dialog box.

  You have to actually return to the procedure before you can display the variable in its context.

For additional information, select the item in question and press F1.

Variable required — can't assign to this expression

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsglvaluerequiredC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLvalueRequiredS"}

This error typically occurs when you attempt to assign a value to something that can't accept the assignment. This error has the following causes and solutions:

- You attempted to use a numeric expression as an argument to the **Len** function.

  The **Len** function doesn't accept a numeric expression, a numeric literal, or a binary numeric expression, but it does accept either a string or numeric variable, a string expression, or a variable of user-defined type.
- You used a function call or an expression as an argument to **Input #**, **Let**, **Get**, or **Put**. For example, you may have used an argument that appears to be a valid reference to an array variable, but instead is a call to a function of the same name.

  Input #, **Let**, **Get**, and **Put** don't accept function calls as arguments.
- You attempted to assign a value to an identifier previously declared as a constant.

  Choose another name for the identifier.
- You tried to use a nonvariable as a loop counter in a **For...Next** construction.

  Use a variable as the counter.
- You tried to assign a value to a read-only property or to an expression that consists of more than one variable (such as X + Y). An assignment places a value at a memory location. The specified expression must represent a single, writable location.

  Rewrite the assignment to a single variable name that can accept the data.
- You tried to use an undeclared variable that is defined as a constant in a type library.

  Either use a different name for the variable, or declare it explicitly.

For additional information, select the item in question and press F1.

Warning: custom language settings not portable

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCustomLCIDChangeS"}

Not all language settings are portable. This warning has the following cause and solution:

- You used a custom language setting in your code.

  When you choose a custom language/country setting for your code, the language conventions used in your code match those set in the **Control Panel** of your system. You can use custom code locale settings, but your code may not work in other locales or on other systems with different settings. The host application parses some strings based on the **Control Panel** settings of the machine on which it is running.

  If the **Control Panel** settings on the target machine aren't the same as those on the machine on which the code was written, strings parsed by a host application don't work, for example, code that depends on a locale-specific decimal separator. Therefore, you should not use a custom language setting unless you don't intend to send your code to other users. If you plan to send your code to other users, select a predefined locale.

For additional information, select the item in question and press F1.

Wend without While

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgwendwithoutwhileC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgWendWithoutWhileS"}

Every **Wend** statement must be preceded by a matching **While** statement. This error has the following cause and solution:

- You used a **Wend** statement without a preceding **While** statement.
  Check for an unterminated loop nested within the **While...Wend** loop that's causing the error.

For additional information, select the item in question and press F1.

While without Wend

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgexpectedwendC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgExpectedWendS"}

A **While** statement is used without a corresponding **Wend** statement. This error has the following cause and solution:

- You opened a **While...Wend** construct, but did not close it.
  Check for an incorrectly matched **While...Wend** structure inside the outer **While...Wend** structure.

For additional information, select the item in question and press F1.

With object must be user-defined type, Object, or Variant

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgbadwithoperandC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadWithOperandS"}

The **With...End With** block can't be used with all variable types. This error has the following cause and solution:

- You tried to use a variable that was not of **Object** type, user-defined type, or **Variant** type containing an object within a **With** block.

  Check to see if you misspelled the name of the object, user-defined type, or **Variant** variable.

For additional information, select the item in question and press F1.

Wrong number of dimensions

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgarrayaritymismatchC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgArrayArityMismatchS"}

You must reference an array with indexes corresponding to the same number of dimensions as appear in the array's declaration. This error has the following cause and solution:

- You referred to an array with a different number of dimensions than it actually contains. For example, referring to an array as `X(2,4)` (an array with two dimensions) when it has been defined as `Dim X(5)` (an array with one), generates this error.

  Check the declaration of the array and, in the reference, include one index for each dimension in the declaration.

For additional information, select the item in question and press `F1`.

This action will reset your project, proceed anyway?

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCannotEditInBreakModeS"}

You can edit code in break mode, but some edits prevent continuing execution. This error has the following cause and solution:

- You attempted an edit that prevents continued execution, for example, you declared a static variable.

  If you choose **Yes**, execution will terminate and you can edit your code. If you choose **No**, you can continue running the code from the point at which it was suspended.

For additional information, select the item in question and press F1.

You must terminate the #If block with an #End If

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgLbExpectedEndIfC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLbExpectedEndIfS"}

#If is a conditional compilation directive. This error has the following cause and solution:

- An **#If** block was detected that isn't terminated by an **#End If**.
  Add an **#End If** in the appropriate position.

For additional information, select the item in question and press F1.

Argument not optional (Error 449)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgParameterNotOptionalC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgParameterNotOptionalS"}

The number and types of arguments must match those expected. This error has the following causes and solutions:

- Incorrect number of arguments.

  Supply all necessary arguments. For example, the **Left** function requires two arguments; the first representing the character string being operated on, and the second representing the number of characters to return from the left side of the string. Because neither argument is optional, both must be supplied.
- Omitted argument isn't optional.

  An argument can only be omitted from a call to a user-defined procedure if it was declared **Optional** in the procedure declaration. Either supply the argument in the call or declare the parameter **Optional** in the definition.

For additional information, select the item in question and press F1.

Bad DLL calling convention (Error 49)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgDLLBadCallingConvC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDLLBadCallingConvS"}

Arguments passed to a dynamic-link library (DLL) routine must exactly match those expected by the routine. Calling conventions deal with number, type, and order of arguments. This error has the following causes and solutions:

- Your program is calling a routine in a DLL that's being passed the wrong type of arguments.

  Make sure all argument types agree with those specified in the declaration of the routine you are calling.
- Your program is calling a routine in a DLL that's being passed the wrong number of arguments.

  Make sure you are passing the same number of arguments indicated in the declaration of the routine you are calling.
- Your program is calling a routine in a DLL, but isn't using the StdCall calling convention.

  If the DLL routine expects arguments by value, then make sure **ByVal** is specified for those arguments in the declaration for the routine.
- Your **Declare** statement includes **CDecl**.

  The **CDecl** keyword applies only to the Macintosh.

For additional information, select the item in question and press F1.

Bad file mode (Error 54)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgBadFileModeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadFileModeS"}

Statements used in manipulating file contents must be appropriate to the mode in which the file was opened. This error has the following causes and solutions:

- A **Put** or **Get** statement is specifying a sequential file.
  Put and **Get** can only refer to files opened for **Random** or **Binary** access.
- A **Print #** statement specifies a file opened for an access mode other than **Output** or **Append**.
  Use a different statement to place data in the file or reopen the file in an appropriate mode.
- An **Input #** statement specifies a file opened for an access mode other than **Input**.
  Use a different statement to place data in the file or reopen the file in **Input** mode.
- You attempted to write to a read-only file.
  Change the read/write status of the file or don't try to write to it.

For additional information, select the item in question and press F1.

Bad file name or number (Error 52)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgBadFileNameorNumberC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadFileNameOrNumberS"}

An error occurred trying to access the specified file. This error has the following causes and solutions:

- A statement refers to a file with a file number or file name that is:
  - Not specified in the **Open** statement or was specified in an **Open** statement, but has since been closed.

    Specify the file name in an **Open** statement. Note that if you invoked the **Close** statement without arguments, you may have inadvertently closed all currently open files, invalidating all file numbers.
  - Out of the range of file numbers (1 – 511).

    If your code is generating file numbers algorithmically, make sure the numbers are valid.
- There is an invalid name or number.

  File names must conform to operating system conventions as well as Basic file-naming conventions. In Microsoft Windows, use the following conventions for naming files and directories:
  - The name of a file or directory can have two parts: a name and an optional extension. The two parts are separated by a period, for example, myfile.new.
  - The name can contain up to 255 characters.
  - The name must start with either a letter or number. It can contain any uppercase or lowercase characters (file names aren't case-sensitive) except the following characters: quotation mark (**"**), apostrophe (**'**), slash (**/**), backslash (\), colon (**:**), and vertical bar (**|**).
  - The name can contain spaces.
  - The following names are reserved and can't be used for files or directories: CON, AUX, COM1, COM2, COM3, COM4, LPT1, LPT2, LPT3, PRN, and NUL. For example, if you try to name a file PRN in an **Open** statement, the default printer will simply become the destination for **Print #** and **Write #** statements directed to the file number specified in the **Open** statement.
  - The following are examples of valid Microsoft Windows file names:

    ```
    LETTER.DOC
    My Memo.Txt
    BUDGET.92
    12345678.901
    Second Try.Rpt
    ```
  - On the Macintosh, a file name can include any character except the colon (**:**), and can contain spaces. Null characters (**Chr(0)**) aren't allowed in any file names.

For additional information, select the item in question and press F1.

Bad record length (Error 59)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgBadRecordLengthC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadRecordLenS"}

The length of a record variable in a **Get** or **Put** statement must be the length specified in its corresponding **Open** statement. This error has the following causes and solutions:

- The record variable's length differs from the length specified in the corresponding **Open** statement.

  Make sure the sum of the sizes of fixed-length variables in the user-defined type defining the record variable's type is the same as the value stated in the **Open** statement's **Len** clause. In the following example, assume `RecVar` is a variable of the appropriate type. You can use the **Len** function to specify the length, as follows:

  ```
  Open MyFile As #1 Len = Len(RecVar)
  ```

- The variable in a **Put** statement is (or includes) a variable-length string.

  Because a 2-byte descriptor is always added to a variable-length string placed in a random access file with **Put**, the variable-length string must be at least 2 characters shorter than the record length specified in the **Len** clause of the **Open** statement.

- The variable in a **Put** statement is (or includes) a **Variant**.

  Like variable-length strings, **Variant** data types also require a 2-byte descriptor. Variants containing variable-length strings require a 4-byte descriptor. Therefore, for variable-length strings in a **Variant**, the string must be at least 4 bytes shorter than the record length specified in the **Len** clause.

For additional information, select the item in question and press `F1`.

Bad record number (Error 63)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgBadRecordNumberC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadRecordNumS"}

An error occurred during the attempted file access. This error has the following cause and solution:

- The record number in a **Put** or **Get** statement is less than or equal to zero.

  Check the calculations used in generating the record number. Make sure that the variables containing the record number or used in calculating record numbers are spelled correctly. A misspelled variable name is implicitly declared and initialized to zero, unless you have properly placed **Option Explicit** in the module.

For additional information, select the item in question and press F1.

Can't create necessary temporary file (Error 322)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantCreateTempFileS"}

Creating an executable file requires creation of temporary files. This error has the following cause and solution:

- The drive that contains the directory specified by the TEMP environment variable is full.

  Delete files from the full drive or specify a different drive in the TEMP environment variable.
- The TEMP environment variable specifies an invalid or read-only drive or directory.

  Specify a valid drive for the TEMP environment variable or remove the read-only restriction from the currently specified drive or directory.

For additional information, select the item in question and press F1.

Can't perform requested operation (Error 17)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantContinueS"}

An operation can't be carried out if it would invalidate the current state of the project. This error has the following cause and solution:

- The requested operation would invalidate the current state of the project. For example, the error occurs if you use the **References** dialog box to add a reference to a new project or object library while a program is in break mode.

  Stop execution of the current code, and then retry the operation.
- An attempt was made to programmatically modify currently running code. For example, your code may have tried to read code from a disk file into a currently running module.

  Although you can modify modules in the project while they aren't actually running, you can't make modifications to a running module. To make such changes, you must stop the module from running, make the additions or changes, and then restart execution.

For additional information, select the item in question and press F1.

Can't rename with different drive (Error 74)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgDifferentDriveC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDifferentDriveS"}

The **Name** statement must rename the file to the current drive. This error has the following cause and solution:

- You tried to move a file to a different drive using the **Name** statement.
  Use **FileCopy** to write the file to another drive, and then delete the old file with a **Kill** statement.

For additional information, select the item in question and press F1.

Can't save file to TEMP directory (Error 735)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgCantSaveFileToTEMPC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantSaveFileToTEMPS"}

Components often need to save temporary information to disk. This error has the following cause and solution:

- Component can't find a directory named TEMP.

  Create a directory named TEMP and set the TEMP environment variable equal to its path.
- The drive or partition containing the TEMP directory lacks sufficient space to save information.

  Make some space on the drive by erasing unnecessary files, or create a TEMP directory on another partition and set the TEMP environment variable equal to its path.

For additional information, select the item in question and press F1.

Class doesn't support Automation (Error 430)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgOLENotSupportedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOLENotSupportedS"}

Not all objects expose an Automation interface. This error has the following cause and solution:

- The class you specified in the **GetObject** or **CreateObject** function call was found, but has not exposed a programmability interface.

  You can't write code to control an object's behavior unless it has been exposed for Automation. Check the documentation of the application that created the object for limitations on the use of Automation with this class of object.

For additional information, select the item in question and press F1.

Code resource lock error (Error 455)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCodeResourceLockErrorS"}

When you access a code resource, you must lock it. This error has the following cause and solution:

- A call was made to a procedure in a code resource. The code resource was found, but an error occurred when an attempt was made to lock the resource.

  Check for an error returned by Hlock, for example, "`Illegal on empty handle`" or "`Illegal on free block`". This error can only occur on the Macintosh.

For additional information, select the item in question and press F1.

Code resource not found (Error 454)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCodeResourceNotFoundS"}

This error can only occur on the Macintosh. This error has the following cause and solution:

- A call was made to a procedure in a code resource, but the code resource could not be found. Check to be sure the resource is available and properly referenced.

For additional information, select the item in question and press F1.

Connection to type library or object library for remote process has been lost (Error 442)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgLostTLBC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLostTLBS"}

During remote access (that is, when accessing an object that is part of another process or is running on another machine), the connection to the library containing object information was broken. This error has the following cause and solution:

- You lost your connection to the remote process's object library or type library.

  To reconnect, follow these steps:

  1. Restart the **Application** object.
  2. In the error dialog box through which you entered this Help topic, click **OK** to display the **References** dialog box.
  3. The lost reference appears with the word MISSING to its left.
  4. Remove the lost reference.
  5. In the **References** dialog box, click the check box for the object you started in step 1.

For additional information, select the item in question and press F1.

Device I/O error (Error 57)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIOErrorS"}

External devices are sometimes subject to unanticipated errors. This error has the following cause and solution:

- An input or output error occurred while your program was using a device such as a printer or disk drive.

  Make sure the device is operating properly, and then retry the operation.

For additional information, select the item in question and press F1.

Device unavailable (Error 68)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDevUnavailableS"}

This error has the following causes and solutions:

- The device you are trying to access either is not online or doesn't exist.

  Check power to the device and any cables connecting your computer to the device. If you are trying to access a printer over a network, make sure there is a logical connection between your computer and the printer, for example, a connection associating LPT1 with the network printer ID.
- Your network connection may have been broken.

  Reconnect to the network and try the operation again.

For additional information, select the item in question and press F1.

Disk full (Error 61)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgDiskFullC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDiskFullS"}

This error has the following causes and solutions:

- There isn't enough room on the disk for the completion of a **Print #**, **Write #**, or **Close** operation.
  Move some files to another disk or delete some files.
- There isn't enough room on the disk to create required files.
  Move some files to another disk or delete some files.

For additional information, select the item in question and press F1.

Disk not ready (Error 71)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDiskNotReadyS"}

This error has the following causes and solutions:

- There is no disk in the specified drive.
  Put a disk in the drive and retry the operation.
- The drive door of the specified drive is open.
  Close the drive door and retry the operation.

For additional information, select the item in question and press F1.

Division by zero (Error 11)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDivByZeroS"}

Division by zero isn't possible. This error has the following cause and solution:

- The value of an expression being used as a divisor is zero.

  Check the spelling of variables in the expression. A misspelled variable name can implicitly create a numeric variable that is initialized to zero. Check previous operations on variables in the expression, especially those passed into the procedure as arguments from other procedures.

For additional information, select the item in question and press F1.

Error in loading DLL (Error 48)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgDLLLoadErrC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDLLLoadErrS"}

A dynamic link library (DLL) is a library specified in the **Lib** clause of a **Declare** statement. This error has the following causes and solutions:

- The file isn't DLL-executable.
  If the file is a source-text file, it must be compiled and linked to DLL executable form.
- The file isn't a Microsoft Windows DLL.
  Obtain the Microsoft Windows DLL equivalent of the file.
- The file is an early Microsoft Windows DLL that is incompatible with Microsoft Windows protect mode.
  Obtain an updated version of the DLL.
- The DLL references another DLL that isn't present.
  Obtain the referenced DLL and make it available to the other DLL.
- The DLL or one of the referenced DLLs isn't in a directory specified by your path.
  Move the DLL to a referenced directory or place its current directory on the path.

For additional information, select the item in question and press F1.

File already exists (Error 58)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgFileAlreadyExistsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgFileAlreadyExistsS"}

This error has the following causes and solutions:

- This error occurs at run time when the new file name, for example, one specified in a **Name** statement, is identical to a file name that already exists.

  Specify a new file name in the **Name** statement or delete the old file before specifying it in a **Name** statement.
- You used the **Save As** command to save a currently loaded project, but the project name already exists.

  Use a different project name if you don't want to replace the other project.

For additional information, select the item in question and press F1.

File already open (Error 55)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgFileAlreadyOpenC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgFileAlreadyOpenS"}

Sometimes a file must be closed before another **Open** or other operation can occur. This error has the following causes and solutions:

- A sequential-output mode **Open** statement was executed for a file that is already open.

  You must close a file opened for one type of sequential access before opening it for another. For example, you must close a file opened for **Input** before opening it for **Output**.
- A statement, for example, **Kill**, **SetAttr**, or **Name**, refers to an open file.

  Close the file before executing the statement.

For additional information, select the item in question and press F1.

File not found (Error 53)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgFileNotFoundS"}

The file was not found where specified. This error has the following causes and solutions:

- A statement, for example, **Kill**, **Name**, or **Open**, refers to a file that doesn't exist.

  Check the spelling of the file name and the path specification.
- An attempt has been made to call a procedure in a dynamic-link library (DLL), but the library file name specified in the **Lib** clause of the **Declare** statement can't be found.

  Check the spelling of the file name and the path specification.
- In the development environment, this error occurs if you attempt to open a project or load a text file that doesn't exist.

  Check the spelling of the project name or file name and the path specification.

For additional information, select the item in question and press F1.

For loop not initialized (Error 92)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgIllegalForC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIllegalForS"}

For loop counters must be initialized. This error has the following cause and solution:

- You jumped into the middle of a **For...Next** loop.
  Remove the jump into the loop. Placing labels inside a **For...Next** loop isn't recommended.

For additional information, select the item in question and press F1.

Input past end of file (Error 62)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgEndofFileC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEndOfFileS"}

You can't read past the end-of-file position. This error has the following cause and solution:

- An **Input #** or **Line Input #** statement is reading from a file in which all data has been read or from an empty file.

  Use the **EOF** function immediately before the **Input #** statement to detect the end of file.
- You used the **EOF** function with a file opened for **Binary** access.

  EOF only works with files opened for sequential **Input** access. Use **Seek** and **Loc** with files opened for **Binary** access.

For additional information, select the item in question and press F1.

Internal error (Error 51)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInternalErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInternalErrorS"}

Make sure this error wasn't generated by the **Error** statement or **Raise** method. This error has the following cause and solution:

- An internal malfunction has occurred in Visual Basic.

  Unless this call was generated by the **Error** statement or **Raise** method, contact Microsoft Product Support Services to report the conditions under which the message appeared.

For additional information, select the item in question and press F1.

Invalid Clipboard format (Error 460)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidClipboardFormatS"}

The Clipboard only accepts data in certain specified formats. This error has the following cause and solution:

- You tried to place data from a component on the Clipboard, but the data is in a format incompatible with the Clipboard.

  Consult the documentation for the component to determine whether you can use the Clipboard for transferring data from the component.

For additional information, select the item in question and press F1.

Invalid file format (Error 321)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvalidFileFormatErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidFileFormatErrorS"}

Disk files often have data stored in proprietary formats. This error has the following cause and solution:

- You tried to load a file into a component, but the format of the data in the file was incompatible with the component.

  Consult the documentation for the component to determine the proper format for disk file data and whether the component provides support for converting from one format to another.
- You tried to save component data to a file, but the format of the data was incompatible with the format of the file.

  Consult the documentation for the component to determine whether it provides support for converting from one format to another.

For additional information, select the item in question and press F1.

Invalid ordinal (Error 452)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidOrdinalS"}

Your call to a dynamic-link library (DLL) indicated to use a number instead of a procedure name, using the **#***num* syntax. This error has the following causes and solutions:

- An attempt to convert the *num* expression to an ordinal failed.
  Make sure the expression represents a valid number or call the procedure by name.
- The *num* specified doesn't specify any function in the DLL.
  Make sure *num* identifies a valid function in the DLL.
- A type library has an invalid declaration resulting in internal use of an invalid ordinal number.
  Comment out code to isolate the procedure call causing the problem. Write a **Declare** statement for the procedure and report the problem to the type library vendor.

For additional information, select the item in question and press F1.

Invalid pattern string (Error 93)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgBadPatStrC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadPatStrS"}

The pattern string specified in the **Like** operation of a search is invalid. This error has the following cause and solution:

- A common example of an invalid character list expression is `[a-b`, where the right bracket is missing.

  Review the valid characters for list expressions.

For additional information, select the item in question and press F1.

Invalid procedure call or argument (Error 5)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgillegalfunccallC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIllegalFuncCallS"}

Some part of the call can't be completed. This error has the following causes and solutions:

- An argument probably exceeds the range of permitted values. For example, the **Sin** function can only accept values within a certain range. Positive arguments less than 2,147,483,648 are accepted, while 2,147,483,648 generates this error.

  Check the ranges permitted for arguments.
- This error can also occur if an attempt is made to call a procedure that isn't valid on the current platform. For example, some procedures may only be valid for Microsoft Windows, or for the Macintosh, and so on.

  Check platform-specific information about the procedure.

For additional information, select the item in question and press F1.

Invalid property-array index (Error 381)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvalidPropertyArrayIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidPropertyArrayIndexS"}

A property value may consist of an array of values. This error has the following cause and solution:

- A component's property array could have a lower bound of zero and an upper bound equal to the number of elements in the array minus 1. Alternatively, the lower bound could be 1 and the upper bound could equal the number of elements in the array.

  Check the component's documentation to make sure your index is within the valid range for the specified property.

For additional information, select the item in question and press F1.

Invalid use of Null (Error 94)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgCantUseNullC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantUseNullS"}

Null is a **Variant** subtype used to indicate that a data item contains no valid data. This error has the following cause and solution:

- You are trying to obtain the value of a **Variant** variable or an expression that is **Null**. For example:

  ```
  MyVar = Null
  For Count = 1 To MyVar
      . . .
  Next Count
  ```

  Make sure the variable contains a valid value.

For additional information, select the item in question and press F1.

Named argument not found (Error 448)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNamedParamNotFoundS"}

A named argument may not be used in a procedure invocation unless it appears in the procedure definition. This error has the following cause and solution:

- You specified a named argument, but the procedure was not defined to accept an argument by that name.

  Check the spelling of the argument name.

For additional information, select the item in question and press F1.

Need property-array index (Error 385)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgNeedPropertyarrayIndexC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNeedPropertyarrayIndexS"}

This property value consists of an array rather than a single value. This error has the following cause and solution:

- You didn't specify the index for the property array you tried to access.

  Check the component's documentation to find the range for the indexes appropriate to the array. Specify an appropriate index in your property access statement.

For additional information, select the item in question and press F1.

Object doesn't support current locale setting (Error 447)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLocaleSettingNotSupportedS"}

Not all objects support all locale settings. This error has the following causes and solutions:

- You attempted to access an object that doesn't support the locale setting for the current project. For example, if your current project has the locale setting Canadian French, the object you are trying to access must support that locale setting.

  Check which locale settings the object supports.
- The object relies on national language support in a dynamic-link library (DLL), for example, OLE2NLS.DLL, that may be out of date.

  Obtain a more recent version of the DLL, one that supports the current project locale.

For additional information, select the item in question and press F1.

Object doesn't support named arguments (Error 446)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNamedArgsNotSupportedS"}

Not all objects support named arguments. This error has the following cause and solution:

- You tried to access an object whose methods don't support named arguments.

  Specify arguments positionally when performing methods on this object. See the object's documentation for more information on argument positions and types.

For additional information, select the item in question and press F1.

Object doesn't support this action (Error 445)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgActionNotSupportedS"}

Not all objects support all actions. This error has the following cause and solution:

- You referenced a method or property that isn't supported by this object.

  See the object's documentation for more information on the object and check the spellings of properties and methods.

For additional information, select the item in question and press F1.

Object not a collection (Error 451)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgNotEnumC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNotEnumS"}

Certain properties, methods, and operations can only apply to **Collection** objects. This error has the following cause and solution:

- You specified an operation or property that is exclusive to collections, but the object isn't a collection.

  Check the spelling of the object or property name, or verify that the object is a **Collection** object. Also look at the **Add** method used to add the object to the collection to be sure the syntax is correct and that any identifiers were spelled correctly.

For additional information, select the item in question and press F1.

Object required (Error 424)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgNotObjectC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNotObjectS"}

References to properties and methods often require an explicit object qualifier. This error has the following causes and solutions:

- You referred to an object property or method, but didn't provide a valid object qualifier.

  Specify an object qualifier if you didn't provide one. For example, although you can omit an object qualifier when referencing a form property from within the form's own module, you must explicitly specify the qualifier when referencing the property from a standard module.
- You supplied an object qualifier, but it isn't recognized as an object.

  Check the spelling of the object qualifier and make sure the object is visible in the part of the program in which you are referencing it. In the case of **Collection** objects, check any occurrences of the **Add** method to be sure the syntax and spelling of all the elements are correct.
- You supplied a valid object qualifier, but some other portion of the call contained an error.

  An incorrect path as an argument to a host application's **File Open** command could cause the error. Check arguments.
- You didn't use the **Set** statement in assigning an object reference.

  If you assign the return value of a **CreateObject** call to a **Variant** variable, an error doesn't necessarily occur if the **Set** statement is omitted. In the following code example, an implicit instance of Microsoft Excel is created, and its default property (the string "Microsoft Excel") is returned and assigned to the **Variant** `RetVal`. A subsequent attempt to use `RetVal` as an object reference causes this error:

  ```
  Dim RetVal                              ' Implicitly a Variant.
  ' Default property is assigned to Type 8 Variant RetVal.
  RetVal = CreateObject("Excel.Application")
  RetVal.Visible = True            ' Error occurs here.
  ```

  Use the **Set** statement when assigning an object reference.
- In rare cases, this error occurs when you have a valid object but are attempting to perform an invalid action on the object. For example, you may receive this error if you try to assign a value to a read-only property.

  Check the object's documentation and make sure the action you are trying to perform is valid.

For additional information, select the item in question and press `F1`.

Object variable or With block variable not set (Error 91)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgObjNotSetC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgObjNotSetS"}

There are two steps to creating an object variable. First you must declare the object variable. Then you must assign a valid reference to the object variable using the **Set** statement. Similarly, a **With...End With** block must be initialized by executing the **With** statement entry point. This error has the following causes and solutions:

- You attempted to use an object variable that isn't yet referencing a valid object.

  Specify or respecify a reference for the object variable. For example, if the **Set** statement is omitted in the following code, an error would be generated on the reference to `MyObject`:

  ```
  Dim MyObject As Object  ' Create object variable.
  Set MyObject = Sheets(1)' Create valid object reference.
  MyCount = MyObject.Count' Assign Count value to MyCount.
  ```

- You attempted to use an object variable that has been set to **Nothing**.

  ```
  Set MyObject = Nothing  ' Release the object.
  MyCount = MyObject.Count' Make a reference to a released object.
  ```

  Respecify a reference for the object variable. For example, use a new **Set** statement to set a new reference to the object.

- The object is a valid object, but it wasn't set because the object library in which it is described hasn't been selected in the **References** dialog box.

  Select the object library in the **Add References** dialog box.

- The target of a **GoTo** statement is inside a **With** block.

  Don't jump into a **With** block. Make sure the block is initialized by executing the **With** statement entry point.

- You specified a line inside a **With** block when you chose the **Set Next Statement** command.

  The **With** block must be initialized by executing the **With** statement.

For additional information, select the item in question and press F1.

Automation error (Error 440)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgOLEAutomationErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOLEAutomationErrorS"}

When you access Automation objects, specific types of errors can occur. This error has the following cause and solution:

- An error occurred while executing a method or getting or setting a property of an object variable. The error was reported by the application that created the object.

  Check the properties of the **Err** object to determine the source and nature of the error. Also try using the **On Error Resume Next** statement immediately before the accessing statement, and then check for errors immediately following the accessing statement.

For additional information, select the item in question and press F1.

Object doesn't support this property or method (Error 438)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOLENoPropOrMethodS"}

Not all objects support all properties and methods. This error has the following cause and solution:

- You specified a method or property that doesn't exist for this Automation object.

  See the object's documentation for more information on the object and check the spellings of properties and methods.
- You specified a **Friend** procedure to be called late bound.

  The name of a **Friend** procedure must be known at compile time. It can't appear in a late-bound call.

For additional information, select the item in question and press F1.

Out of memory (Error 7)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOutOfMemoryS"}

More memory was required than is available, or a 64K segment boundary was encountered. This error has the following causes and solutions:

- You have too many applications, documents, or source files open.

  Close any unnecessary applications, documents, or source files that are open.
- You have a module or procedure that's too large.

  Break large modules or procedures into smaller ones. This doesn't save memory, but it can prevent hitting 64K segment boundaries.
- You are running Microsoft Windows in standard mode.

  Restart Microsoft Windows in enhanced mode.
- You are running Microsoft Windows in enhanced mode, but have run out of virtual memory.

  Increase virtual memory by freeing some disk space, or at least ensure that some space is available.
- You have terminate-and-stay-resident programs running.

  Eliminate terminate-and-stay-resident programs.
- You have many device drivers loaded.

  Eliminate unnecessary device drivers.
- You have run out of space for **Public** variables.

  Reduce the number of **Public** variables.

For additional information, select the item in question and press F1.

Out of stack space (Error 28)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgOutofStackC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOutOfStackS"}

The stack is a working area of memory that grows and shrinks dynamically with the demands of your executing program. This error has the following causes and solutions:

- You have too many active **Function**, **Sub**, or **Property** procedure calls.

  Check that procedures aren't nested too deeply. This is especially true with recursive procedures, that is, procedures that call themselves. Make sure recursive procedures terminate properly. Use the **Calls** dialog box to view which procedures are active (on the stack).
- Your local variables require more local variable space than is available.

  Try declaring some variables at the module level instead. You can also declare all variables in the procedure static by preceding the **Property**, **Sub**, or **Function** keyword with **Static**. Or you can use the **Static** statement to declare individual **Static** variables within procedures.
- You have too many fixed-length strings.

  Fixed-length strings in a procedure are more quickly accessed, but use more stack space than variable-length strings, because the string data itself is placed on the stack. Try redefining some of your fixed-length strings as variable-length strings. When you declare variable-length strings in a procedure, only the string descriptor (not the data itself) is placed on the stack. You can also define the string at module level where it requires no stack space. Variables declared at module level are **Public** by default, so the string is visible to all procedures in the module.
- You have too many nested **DoEvents** function calls.

  Use the **Calls** dialog box to view which procedures are active on the stack.
- Your code triggered an event cascade.

  An event cascade is caused by triggering an event that calls an event procedure that's already on the stack. An event cascade is similar to an unterminated recursive procedure call, but it's less obvious, since the call is made by Visual Basic rather than by an explicit call in your code. Use the **Calls** dialog box to view which procedures are active (on the stack).

To display the **Calls** dialog box, select the **Calls** button to the right of the **Procedure** box in the **Debug** window or choose the **Calls** command. For additional information, select the item in question and press F1.

Out of string space (Error 14)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOutOfStrSpaceS"}

Visual Basic permits you to use very large strings. However, the requirements of other programs and the way you manipulate your strings may cause this error. This error has the following causes and solutions:

- Expressions requiring that temporary strings be created for evaluation may cause this error. For example, the following code causes an `Out of string space` error on some operating systems:

  ```
  MyString = "Hello"
  For Count = 1 To 100
     MyString = MyString & MyString
  Next Count
  ```

  Assign the string to a variable of another name.
- Your system may have run out of memory, which prevented a string from being allocated.

  Remove any unnecessary applications from memory to create more space.

For additional information, select the item in question and press F1.

Overflow (Error 6)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOverflowS"}

An overflow results when you try to make an assignment that exceeds the limitations of the target of the assignment. This error has the following causes and solutions:

- The result of an assignment, calculation, or data type conversion is too large to be represented within the range of values allowed for that type of variable.

  Assign the value to a variable of a type that can hold a larger range of values.
- An assignment to a property exceeds the maximum value the property can accept.

  Make sure your assignment fits the range for the property to which it is made.

For additional information, select the item in question and press F1.

Path not found (Error 76)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgPathNotFoundC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgPathNotFoundS"}

The path to a file includes the drive specification plus the directories and subdirectories that must be traversed to locate the file. A path can be relative or absolute. This error has the following cause and solution:

- During a file-access or disk-access operation, for example, **Open**, **MkDir**, **ChDir**, or **RmDir**, the operating system was unable to find the specified path.

  Respecify the path.

For additional information, select the item in question and press F1.

Path/File access error (Error 75)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgPathFileAccessC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgPathFileAccessS"}

During a file-access or disk-access operation, for example, **Open**, **MkDir**, **ChDir**, or **RmDir**, the operating system couldn't make a connection between the path and the file name. This error has the following causes and solutions:

- The file specification isn't correctly formatted.

  A file name can contain a fully qualified (absolute) or relative path. A fully qualified path starts with the drive name (if the path is on another drive) and lists the explicit path from the root to the file. Any path that isn't fully qualified is relative to the current drive and directory.
- You attempted to save a file that would replace an existing read-only file.

  Change the read-only attribute of the target file or save the file with a different file name.
- You attempted to open a read-only file in sequential **Output** or **Append** mode.

  Open the file in **Input** mode or change the read-only attribute of the file.

For additional information, select the item in question and press F1.

Permission denied (Error 70)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgpermissiondeniedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgPermissionDeniedS"}

An attempt was made to write to a write-protected disk or to access a locked file. This error has the following causes and solutions:

- You tried to open a write-protected file for sequential **Output** or **Append**.

  Open the file for **Input** or change the write-protection attribute of the file.
- You tried to open a file on a disk that is write-protected for sequential **Output** or **Append**.

  Remove the write-protection device from the disk or open the file for **Input**.
- You tried to write to a file that another process locked.

  Wait to open the file until the other process releases it.
- You attempted to access the registry, but your user permissions don't include this type of registry access.

  On 32-bit Microsoft Windows systems, a user must have the correct permissions for access to the system registry. Change your permissions or have them changed by the system administrator.

For additional information, select the item in question and press F1.

Printer error (Error 482)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgPrinterErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgPrinterErrorS"}

This error has the following cause and solution:

- An error occurred at the printer, but no other information was returned to the computer that sent the file.

  You must physically examine the printer. Make sure all connections between computer and printer are solid. Most printers provide a display for error information such as "Out of paper," "Offline," and so on.

For additional information, select the item in question and press F1.

Property Get can't be executed at run time (Error 393)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgGetNotSupportedAtRuntimeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgGetNotSupportedAtRuntimeS"}

Not all properties return run-time information. This error has the following cause and solution:

- The property you are trying to access is read-only at run time.
  Terminate execution and view the property in design time.

For additional information, select the item in question and press F1.

Property Get can't be executed on write-only property (Error 394)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgGetNotSupportedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgGetNotSupportedS"}

Not all properties return information. This error has the following cause and solution:

- The property you are trying to access doesn't return information.
  You can't determine the state of the property.

For additional information, select the item in question and press F1.

Property or method not found (Error 423)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoSuchControlOrPropertyS"}

The spelling of an object name must exactly match the definition in its object library. This error has the following cause and solution:

- You referred to an *object*.*method* or *object*.*property*, but *method* or *property* isn't defined.

  You may have misspelled the name of the *object*. To see what properties and methods are defined for an *object*, display the **Object Browser**. Select the appropriate object library to view a list of available properties and methods.

For additional information, select the item in question and press F1.

Property not found (Error 422)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgPropertyNotFoundC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgPropertyNotFoundS"}

Not all objects support the same set of properties. This error has the following cause and solution:

- This object doesn't support the specified property.

  Check the spelling of the property name. Also, you may be trying to access something like a "text" property when the object actually supports a "caption" or some similarly named property. Check the object's documentation.

For additional information, select the object in question and press F1.

Property Set can't be executed at run time (Error 382)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgSetNotSupportedAtRunTimeC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgSetNotSupportedAtRunTimeS"}

It may not be possible to obtain a reference to a property at run time.This error has the following cause and solution:

- You tried to get a reference to a property that's either read-only or write-only at run time.

  Since you can use a reference for both reading and writing, the property must provide run-time support for both operations for a reference to be obtained at run time.

For additional information, select the item in question and press F1.

Property Set can't be used with a read-only property (Error 383)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgSetNotSupportedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgSetNotSupportedS"}

It may not be possible to obtain a reference to a property at run time.This error has the following cause and solution:

- You tried to get a reference to a property that's read-only at run time.

  Since you can use a reference for both reading and writing, the property must provide run-time support for both operations for a reference to be obtained at run time. You can only use a **Property Get** with this property.

For additional information, select the item in question and press F1.

Property Set not permitted (Error 387)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgSetNotPermittedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgSetNotPermittedS"}

Not all properties support returning a reference. This error has the following cause and solution:

- You tried to get a reference to a property that doesn't return a reference.
  Restrict your access to this property to **Property Let** and **Property Get**.

For additional information, select the item in question and press F1.

Replacements too long (Error 746)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgReplacementsTooLongC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgReplacementsTooLongS"}

A replacement may not exceed a specified maximum length. This error has the following cause and solution:

- You specified a replacement that exceeded the length permitted.
  Consult the component documentation for maximum length restriction.

For additional information, select the item in question and press F1.

Resume without error (Error 20)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgResumewoerrC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgResumeWOErrS"}

A **Resume** statement can only appear within an error handler and can only be executed in an active error handler. This error has the following causes and solutions:

- You placed a **Resume** statement outside error-handling code.

  Move the statement into an error handler, or delete it.
- Your code jumped into an error handler even though there was no error.

  Perhaps you misspelled a line label. Jumps to labels can't occur across procedures, so search the procedure for the label that identifies the error handler. If you find a duplicate label specified as the target of a **GoTo** statement that isn't an **On Error GoTo** statement, change the line label to agree with its intended target.

For additional information, select the item in question and press F1.

Return without GoSub (Error 3)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgReturnwoGoSubC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgReturnWOGoSubS"}

A **Return** statement must have a corresponding **GoSub** statement. This error has the following cause and solution:

- You have a **Return** statement that can't be matched with a **GoSub** statement.
  Make sure your **GoSub** statement wasn't inadvertently deleted.

Unlike **For...Next**, **While...Wend**, and **Sub...End Sub**, which are matched at compile time, **GoSub** and **Return** are matched at run time.

For additional information, select the item in question and press F1.

Specified DLL function not found (Error 453)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidDllFunctionNameS"}

The dynamic-link library (DLL) in a user library reference was found, but the DLL function specified wasn't found within the DLL. This error has the following causes and solutions:

- You specified an invalid ordinal in the function declaration.

  Check for the proper ordinal or call the function by name.
- You gave the right DLL name, but it isn't the version that contains the specified function.

  You may have the correct version on your machine, but if the directory containing the wrong version precedes the directory containing the correct one in your path, the wrong DLL is accessed. Check your machine for different versions. If you have an early version, contact the supplier for a later version.
- If you are working on a 32-bit Microsoft Windows platform, both the DLL name and alias (if used) must be correct.

  Make sure the DLL name and alias are correct.
- Some 32-bit DLLs contain functions with slightly different versions to accommodate both Unicode and ANSI strings. An "A" at the end of the function name specifies the ANSI version. A "W" at the end of the function name specifies the Unicode version.

  If the function takes string-type arguments, try appending an "A" to the function name.

For additional information, select the item in question and press F1.

Expression too complex (Error 16)

{ewc HLP95EN.DLL,DYNALINK,"See Also":""}

The number of subexpressions allowed in a floating-point expression varies among platforms. For example, on 32-bit Microsoft Windows operating systems, the limit is 8 levels of nested floating-point expressions. This error has the following cause and solution:

- A floating-point expression contains too many nested subexpressions.

  Break the expression into as many separate expressions as necessary to prevent the error from occurring.

Note In earlier versions of Visual Basic, Error 16 was "String expression too complex." That error condition can no longer occur. However, if you have early code that traps and handles that error, you should remove it to prevent confusion with this new error.

For additional information, select the item in question and press F1.

Search text not found (Error 744)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgSearchTextNotFoundC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgSearchTextNotFoundS"}

This error has the following cause and solution:

- The text you specified wasn't found.
  The text doesn't exist.

For additional information, select the item in question and press F1.

Sub, Function, or Property not defined (Error 35)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgundefinedprocC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgUndefinedProcS"}

A **Sub**, **Function**, or **Property** procedure must be defined to be called. This error has the following causes and solutions:

- You misspelled the name of your procedure.

  Check the spelling and correct it.
- You tried to call a procedure from another project without explicitly adding a reference to that project in the **References** dialog box.

To add a reference

1. Display the **References** dialog box.
2. Find the name of the project containing the procedure you want to call. If the project name doesn't appear in the **References** dialog box, click the **Browse** button to search for it.
3. Click the check box to the left of the project name.
4. Click **OK**.

- The specified procedure isn't visible to the calling procedure.

  Procedures declared **Private** in one module can't be called from procedures outside the module. If **Option Private Module** is in effect, procedures in the module aren't available to other projects. Search to locate the procedure.
- You declared a dynamic-link library (DLL) routine, but the routine isn't in the specified library.
- Check the ordinal (if you used one) or the name of the routine. Make sure your version of the DLL is the correct one. The routine may only exist in later versions of the DLL. If the directory containing the wrong version precedes the directory containing the correct one in your path, the wrong DLL is accessed. You gave the right DLL name, but it isn't the version that contains the specified function.

For additional information, select the item in question and press F1.

Subscript out of range (Error 9)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgOutOfBoundsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOutOfBoundsS"}

Elements of arrays and members of collections can only be accessed within their defined ranges. This error has the following causes and solutions:

- You referenced a nonexistent array element.

  The subscript may be larger or smaller than the range of possible subscripts, or the array may not have dimensions assigned at this point in the application. Check the declaration of the array to verify its upper and lower bounds. Use the **UBound** and **LBound** functions to condition array accesses if you're working with arrays that are redimensioned. If the index is specified as a variable, check the spelling of the variable name.
- You declared an array but didn't specify the number of elements. For example, the following code causes this error:

  ```
  Dim MyArray() As Integer
  MyArray(8) = 234  ' Causes Error 9.
  ```

  Visual Basic doesn't implicitly dimension unspecified array ranges as 0 – 10. Instead, you must use **Dim** or **ReDim** to specify explicitly the number of elements in an array.
- You referenced a nonexistent collection member.

  Try using the **For Each...Next** construct instead of specifying index elements.
- You used a shorthand form of subscript that implicitly specified an invalid element.

  For example, when you use the **!** operator with a collection, the **!** implicitly specifies a key. For example, *object***!***keyname***.**value is equivalent to *object***.**item**(***keyname***).**value. In this case, an error is generated if *keyname* represents an invalid key in the collection. To fix the error, use a valid key name or index for the collection.

For additional information, select the item in question and press F1.

This component doesn't support the set of events (Error 459)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgObjDoesNotSupportEventsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgObjDoesNotSupportEventsS"}

Not every component supports client sinking of events. This error has the following cause and solution:

- You tried to use a **WithEvents** variable with a component that can't work as an event source for the specified set of events. For example, you may be sinking events of an object, then create another object that **Implements** the first object. Although you might think you could sink the events from the implemented object, that isn't automatically the case. **Implements** only implements an interface for methods and properties.

  You can't sink events for a component that doesn't source events.

For additional information, select the item in question and press F1.

Too many DLL application clients (Error 47)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgTooManyClientsS"}

The dynamic-link library (DLL) for Visual Basic can only accommodate access by 50 host applications at a time. This error has the following cause and solution:

- Your application and other applications that are Visual Basic hosts (some of which may be accessed by your application) are all attempting to access the Visual Basic DLL at the same time.

  Reduce the number of open applications accessing Visual Basic.

For additional information, select the item in question and press F1.

Too many files (Error 67)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgTooManyFilesS"}

There is a limit to the number of disk files that can be open at one time. This error has the following causes and solutions:

- MS-DOS operating system: More files have been created in the root directory than the operating system permits.

  The MS-DOS operating system limits the number of files that can be in the root directory, usually 512. If your program is opening, closing, or saving files in the root directory, change your program so that it uses a subdirectory.
- MS-DOS operating system: More files have been opened than the number specified in the **files=** setting in your CONFIG.SYS file.

  Increase the number specified in the **files=** setting in your CONFIG.SYS file and restart your computer.
- Macintosh: Your program tried to open more than 40 files.

  On the Macintosh, the standard limit is 40 files. This limit can be changed using a utility to modify the **MaxFiles** parameter of the boot block.

For additional information, select the item in question and press F1.

Type mismatch (Error 13)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgTypeMismatchC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgTypeMismatchS"}

Visual Basic is able to convert and coerce many values to accomplish data type assignments that weren't possible in earlier versions. However, this error can still occur and has the following causes and solutions:

- The variable or property isn't of the correct type. For example, a variable that requires an integer value can't accept a string value unless the whole string can be recognized as an integer.

  Try to make assignments only between compatible data types. For example, an **Integer** can always be assigned to a **Long**, a **Single** can always be assigned to a **Double**, and any type (except a user-defined type) can be assigned to a **Variant**.
- An object was passed to a procedure that is expecting a single property or value.

  Pass the appropriate single property or call a method appropriate to the object.
- A module or project name was used where an expression was expected, for example:

  ```
  Debug.Print MyModule
  ```

  Specify an expression that can be displayed.
- You attempted to mix traditional Basic error handling with **Variant** values having the **Error** subtype (10, **vbError**), for example:

  ```
  Error CVErr(n)
  ```

  To regenerate an error, you must map it to an intrinsic Visual Basic or a user-defined error, and then generate that error.
- A **CVErr** value can't be converted to **Date**. For example:

  ```
  MyVar = CDate(CVErr(9))
  ```

  Use a **Select Case** statement or some similar construct to map the return of **CVErr** to such a value.
- At run time, this error typically indicates that a **Variant** used in an expression has an incorrect subtype, or a **Variant** containing an array appears in a **Print #** statement.

  To print arrays, create a loop that displays each element individually.

For additional information, select the item in question and press F1.

User interrupt occurred (Error 18)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgUserInterruptS"}

The process of breaking execution is useful, but if left unhandled, it results in termination of the application. This error has the following cause and solution:

- A user pressed CTRL+BREAK or another interrupt key.

  In the development environment, continue execution. To provide for the occurrence of this condition at run time, handle the error in an appropriate way.

For additional information, select the item in question and press F1.

Variable uses a type not supported in Visual Basic (Error 458)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidTypeLibVariableS"}

Not every variable that appears in a type library or object library can be used by every programming language. This error has the following cause and solution:

- You tried to use a variable defined in a type library or object library that has a data type that isn't supported by Visual Basic.

  You can't use a variable of a type not recognized by Visual Basic in a Visual Basic program.

For additional information, select the item in question and press F1.

Wrong number of arguments or invalid property assignment (Error 450)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgfuncaritymismatchC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgFuncArityMismatchS"}

The number of arguments to a procedure must match the number of parameters in the procedure's definition. This error has the following causes and solutions:

- The number of arguments in the call to the procedure wasn't the same as the number of required arguments expected by the procedure.

  Check the argument list in the call against the procedure declaration or definition.
- You specified an index for a control that isn't part of a control array.

  The index specification is interpreted as an argument but neither an index nor an argument is expected, so the error occurs. Remove the index specification, or follow the procedure for creating a control array. Set the **Index** property to a nonzero value in the control's property sheet or property window at design time.
- You tried to assign a value to a read-only property, or you tried to assign a value to a property for which no **Property Let** procedure exists.

  Assigning a value to a property is the same as passing the value as an argument to the object's **Property Let** procedure. Properly define the **Property Let** procedure; it must have one more argument than the corresponding **Property Get** procedure. If the property is meant to be read-only, you can't assign a value to it.

For additional information, select the item in question and press F1.

ActiveX component can't create object or return reference to this object (Error 429)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgCantCreateObjectC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantCreateObjectS"}

Creating objects requires that the object's class be registered in the system registry and that any associated dynamic-link libraries (DLL) be available. This error has the following causes and solutions:

- The class isn't registered. For example, the system registry has no mention of the class, or the class is mentioned, but specifies either a file of the wrong type or a file that can't be found.

  If possible, try to start the object's application. If the registry information is out of date or wrong, the application should check the registry and correct the information. If starting the application doesn't fix the problem, rerun the application's setup program.
- A DLL required by the object can't be used, either because it can't be found, or it was found but was corrupted.

  Make sure all associated DLLs are available. For example, the Data Access Object (DAO) requires supporting DLLs that vary among platforms. You may have to rerun the setup program for such an object if that is what is causing this error.
- The object is available on the machine, but it is a licensed Automation object, and can't verify the availability of the license necessary to instantiate it.

  Some objects can be instantiated only after the component finds a license key, which verifies that the object is registered for instantiation on the current machine. When a reference is made to an object through a properly installed type library or object library, the correct key is supplied automatically.

  If the attempt to instantiate is the result of a **CreateObject** or **GetObject** call, the object must find the key. In this case, it may search the system registry or look for a special file that it creates when it is installed, for example, one with the extension .lic. If the key can't be found, the object can't be instantiated. If an end user has improperly set up the object's application, inadvertently deleted a necessary file, or changed the system registry, the object may not be able to find its key. If the key can't be found, the object can't be instantiated. In this case, the instantiation may work on the developer's system, but not on the user's system. It may be necessary for the user to reinstall the licensed object.
- You are trying to use the **GetObject** function to retrieve a reference to class created with Visual Basic.

  GetObject can't be used to obtain a reference to a class created with Visual Basic.
- Access to the object has explicitly been denied.

  For example, you may be trying to access a data object that's currently being used and is locked to prevent deadlock situations. If that's the case, you may be able to access the object at another time.

For additional information, select the item in question and press F1.

Invalid property value (Error 380)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidPropertyValueS"}

Most properties only accept values of a certain type, within a certain range. This error has the following cause and solution:

- An inappropriate value has been assigned to a property.

  See the property's Help topic to determine what types and range of values are appropriate for the property.

For additional information, select the property in question and press F1.

Invalid picture (Error 481)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvalidPictureC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidPictureS"}

Only a picture can be assigned to the **Picture** property or **Picture** object. This error has the following cause and solution:

- You attempted to assign something to a **Picture** property or a **Picture** object that isn't recognized as an icon, bitmap, or Microsoft Windows metafile.

  See the Help topics for the **Picture** object or **Picture** property of the object you're using to determine the acceptable picture types.

For additional information, select the item in question and press F1.

Invalid format in resource file (Error 325)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidResourceFormatS"}

When accessing a resource file, the resource you are trying to retrieve must have a valid format. This error has the following cause and solution:

- The resource file you're trying to retrieve has an invalid format, or a specific resource in the file has an invalid format.

  If the file contents have been damaged, reinstall the file from its original disk. If the error continues, contact the provider of the resource for a new file or a version of the resource with the expected format.

For additional information, select the item in question and press F1.

This key is already associated with an element of this collection (Error 457)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgDuplicateKeyC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDuplicateKeyS"}

A key is a string specified in the **Add** method that uniquely identifies a specific member of a collection. This error has the following cause and solution:

- You specified a key for a collection member that already identifies another member of the collection.

  Choose a different key for this member.

For additional information, select the item in question and press F1.

This array is fixed or temporarily locked (Error 10)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgArrayLockedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgArrayLockedS"}

Not all arrays can be redimensioned. Even arrays specifically declared to be dynamic and arrays within **Variant** variables are sometimes locked temporarily. This error has the following causes and solutions:

- You tried to use **ReDim** to change the number of elements of a fixed-size array . For example, in the following code, the fixed array `FixedArr` is received by `SomeArr` in the `NextOne` procedure, and then an attempt is made to resize `SomeArr`:

```
Sub FirstOne
   Dim FixedArr(25) As Integer   ' Create a fixed-size array and
   NextOne FixedArr()   ' pass it to another procedure.
End Sub

Sub NextOne(SomeArr() As Integer)
   ReDim SomeArr(35)    ' Error 10 occurs here.
   . . .
End Sub
```

  Make the original array dynamic rather than fixed by declaring it with **ReDim** (if the array is declared within a procedure), or by declaring it without specifying the number of elements (if the array is declared at module level).

- You tried to redimension a module-level dynamic array, in which one element has been passed as an argument to a procedure. For example, in the following code, `ModArray` is a dynamic, module-level array whose forty-fifth element is being passed by reference to the `Test` procedure:

```
Dim ModArray () As Integer ' Create a module-level dynamic array.
   . . .

Sub AliasError()
   ReDim ModArray (1 To 73) As Integer
   Test ModArray (45)   ' Pass an element of the module-level
                        ' array to the Test procedure.
End Sub

Sub Test(SomeInt As Integer)
   ReDim ModArray (1 To 40) As Integer  ' Error occurs here.
End Sub
```

  There is no need to pass an element of the module-level array in this case, since it's visible within all procedures in the module. However, if an element is passed, the array is locked to prevent a deallocation of memory for the reference parameter within the procedure, causing unpredictable behavior when the procedure returns.

- You attempted to assign a value to a **Variant** variable containing an array, but the **Variant** is currently locked. For example, if your code uses a **For Each...Next** loop to iterate over a variant containing an array, the array is locked on entry into the loop, and then released at the termination of the loop:

```
SomeArray = Array(9,8,7,6,5,4,3,2,1)

For Each X In SomeArray
   SomeArray = 301   ' Causes error since array is locked.
Next X
```

  Use a **For...Next** rather than a **For Each...Next** loop to iterate. When an array is the object of a **For Each...Next** loop, you can read the array, but not write to it.

For additional information, select the item in question and press F1.

File name or class name not found during Automation operation (Error 432)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgOLEFileNotFoundC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOLEFileNotFoundS"}

The **GetObject** function requires either a valid file name with a path specification, or the name of a class that is registered with the system. This error has the following cause and solution:

- The name specified for file name or class in a call to the **GetObject** function could not be found. Check the names and try again. Make sure the name used for the *class* parameter matches that registered with the system.

For additional information, select the item in question and press F1.

Automation object doesn't have a default value (Error 443)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgOLENoDefaultC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOLENoDefaultS"}

When you specify an object name without a property or method, Visual Basic assumes you are referencing the object's default member (property or method). However, not all objects expose a default member. This error has the following cause and solution:

- Visual Basic can't determine the default member for the specified object.
  Check the object's documentation and give an explicit specification for the property or method.

For additional information, select the item in question and press F1.

Unexpected error; quitting

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantBootS"}

An unexpected error occurred**,** and Visual Basic was unable to continue. This may be a hardware problem or an effect of other software in your system.

Valid values are whole numbers from 1 to 32

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgPREFTabstopsErrorS"}

You attempted to set the **TabStop Width** beyond the permitted range. A number between 1 and 32 (inclusive) is acceptable for this field.

Valid values are whole numbers from 2 to 60

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgPrefGridErrorS"}

You attempted to set the **Grid Width** or **Grid Height** beyond the permitted range. A number between 2 and 60 (inclusive) is acceptable for this field.

Specified item is a binary form and can't be loaded into Visual Basic

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCANTLOADVB1FRMS"}

Binary forms created with earlier versions of Visual Basic can't be loaded into this version of Visual Basic. Only Visual Basic versions 2.0 and 3.0 support converting binary forms to ASCII. If you have a copy of Visual Basic 2.0 or 3.0, you can load the form created with version 1.0 and those versions and save it as text. Then you can import the form in Visual Basic.

Project file is read-only

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbmsgProjectFileIsReadonlyC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbmsgProjectFileIsReadonlyS"}

The project file is not writable. To change it, you must save it under a different name or remove the read-only attribute.

File is read-only

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbmsgFileIsReadonlyvb5C"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbmsgFileIsReadonlyvb5S"}

The file is not writable. To change it, you must save it under a different name or remove the read-only attribute.

A procedure of that name already exists

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNEWPROCEXISTSS"}

Procedure names within the same scope must be unique. Change the name of the procedure.

The project file specified contains an invalid key value. Valid range is 0 to 'item3'

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgINVALIDRANGEYVALUES"}

The range of key values is restricted. Keep the key value within the range specified.

The project name is too long. Name has been truncated

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbmsgTheProjectNameIsTooLongNameHasBeenTruncatedC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbmsgTheProjectNameIsTooLongNameHasBeenTruncatedS"}

If a project name exceeds permitted length, it is truncated to the maximum permitted length.

The project file specified contains an invalid key. The project can't be loaded

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgVBPCORRUPTKEYINVALIDS"}

You can't load the project while the project file contains the invalid key.

Specified item is an invalid key. The specified file can't be loaded

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgINVALIDFILEKEYS"}

You can't load the file while it contains the invalid key.

The project file specified contains invalid key value

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgINVALIDPROJKEYVALUES"}

The specified key value is invalid. Please change it.

Not enough memory to run; quitting

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOOMBootS"}

Visual Basic couldn't obtain enough memory to run. Close other applications and try again.

Wrong version of operating system; requires specified build of Windows NT 3.51 or specified build of Windows 95

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgWrongBADOSVERS"}

To run this version of Visual Basic, you must be running the specified build of Windows 95 or Windows NT version 3.51.

The specified system error occurred

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgHRESULTS"}

Visual Basic encountered an error that was generated by the system or an external component.

The file specified is marked as a version not supported by the current version of Visual Basic and won't be loaded

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgFORMVERNOTRECOGNIZEDS"}

You tried to load a form from Visual Basic version 1.0 or a form from a version later than this version of Visual Basic. This is not supported.

The project file specified is corrupt and can't be loaded

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgVBPCORRUPTS"}

Visual Basic can't read the project file. This can occur if the file has been modified by an editor outside of Visual Basic. To fix the problem, undo any changes made to the file.

Unable to write specified Designer cache file; will just use regular files on Load

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNODESIGNERCACHES"}

Your ActiveX designer was unable to write information to a cache file, so the designer will load using uncached information.

To improve performance the next time you start Visual Basic, an ActiveX designer writes a cache file. The designer couldn't write to the cache file due to low disk space or invalid permissions on the drive. Make sure you have enough disk space available and that you have write permissions to the drive. If the problem persists, contact the ActiveX designer's vendor.

Not enough memory to load file

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoMemS"}

Visual Basic couldn't obtain enough memory to load the file. Close other applications and try again.

Duplicate procedure name

{ewc HLP95EN.DLL,DYNALINK,"Applies To":"vamsgDupProcA"}

You defined (or are attempting to define) more than one version of a procedure with the same name. Rename one of the procedures. Ensure that duplicate **Declare** statements for the same procedure have the same **Alias** clause.

Can't find Windows Help .exe file

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgHelpLoadFailedS"}

The Windows Help application isn't available. If winhelp.exe is on your computer, make sure it is on your path. If it isn't on your computer, run Microsoft Windows 95 or Microsoft Windows NT setup to install it.

Can't display system information

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantDisplaySysInfoS"}

Visual Basic can't display system information when you choose **System Info** from the **About Microsoft Visual Basic** dialog box. Your system may not have enough memory, or a required file may be corrupted or missing.

TThe application description can't be more than 2000 characters long

You can't have an application description longer than 2000 characters long. Delete some characters from the description.

Must specify which item(s) to print

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgMustSpecifyPrintS"}

You haven't specified what to print. Check at least one of the **Form** or **Code** check boxes in the **Print** dialog box after choosing **Print** from the **File** menu.

Specified name is not a legal object name

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIllegalNameS"}

Form and control names must start with a letter and can be a maximum of 40 characters — including letters, numbers, and underscores (_).

Note that the **Name** property of a form or control is different from the **Label** properties — **Caption**, **Text**, and **Value** — that label or display the contents of a control at run time. These properties can be restricted keywords, can begin with a number, and can contain nonalphanumeric characters.

Name specified conflicts with existing module, project, or object library

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCLASSNAMES"}

This error has the following causes:

- You loaded a form, User Control , User Document, or Property Page that contains objects with conflicting names. This can occur if the file was edited outside of Visual Basic.
- This error can occur when you rename a form to the same name as the project, another form, or a module. It can also occur when you rename the project to the same name as a form or module.

Can't set the project name at this time

The project name can't be set at this time.

Invalid object use (Error 425)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOBJILLEGALUSES"}

Most objects can only be used in specific ways, usually when trying to use an object outside the scope for which it is relevant. For example, an object reference passed into a drag-and-drop event can probably only be used in the context of that event.

Can't print form image to this type of printer (Error 486)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgGFXFORMPRINTERRORS"}

You must have a graphics-capable printer to print a form image.

The specified object can't be used as an owner form for Show() (Error 371)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgINVALIDOWNERFORMS"}

You must use an appropriate object with the **Show** method.

Specified ActiveX control was not found (Error 363)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLDNOSUCHMODELS"}

The form being loaded contains an ActiveX control that isn't part of the current project. This error has the following causes and solutions:

- You may have manually edited the project's .vbp file to add a form containing an ActiveX control that isn't already part of the project.

  After the project loads, use the **Add File** command on the **File** menu to add the ActiveX control to the project.
- You may have manually edited the project's .vbp file to add a form containing an earlier version of an ActiveX control than the ActiveX control that is already part of the project.

  After the project loads, delete the earlier version from the form and put the later version of the control on the form.

ActiveX component not correctly registered (Error 336)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgREGBADREGISTRATIONS"}

The ActiveX component has not been properly registered in the system registry.

Data value named not found (Error 327)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgPBAGELEMENTNOTFOUNDS"}

The data value could not be found.

Invalid picture (Error 481)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsggfxInvalidPictureS"}

An invalid graphics format was assigned to the **Picture** property. This error has the following cause and solution:

- You tried to assign a graphics format other than a bitmap, icon, or Windows metafile to the **Picture** property of a form or control.

  Ensure that the file you are trying to load into the **Picture** property is a valid graphics file supported by Visual Basic.

Printer error (Error 482)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsggfxPrinterErrorS"}

There is some problem that prevents printing. This error has the following causes and solutions:

- You don't have a printer installed from the Windows **Control Panel**.
  Open the **Control Panel**, double-click the **Printers** icon, and click **Add Printer** to install a printer.
- Your printer isn't online.
  Physically switch the printer online.
- Your printer is jammed or out of paper.
  Physically correct the problem.
- You tried to print a form to a printer that can accept only text.
  Switch to an installed printer that can print graphics.

Printer driver does not support specified property (Error 483)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsggfxDriverNotSupportPropS"}

The printer driver for the printer in use doesn't support this property of the **Printer** object. This error has the following cause and solution:

- The effect of the properties of the **Printer** object depends on the driver supplied by the printer manufacturer. Some property settings may have no effect, or several different property settings may all have the same effect. Settings outside the accepted range may or may not produce an error.

  For more information, see the manufacturer's documentation for the specific driver.

Problem getting printer information from the system; make sure the printer is set up correctly. (Error 484)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsggfxDriverOrWinINIErrorS"}

There is some problem that prevents getting printer information from the system. This error has the following causes and solutions:

- You don't have a printer installed from the Windows **Control Panel**.
  Open the **Control Panel**, double-click the **Printers** icon, and click **Add Printer** to install a printer.
- Your printer isn't online.
  Physically switch the printer online.
- Your printer is jammed or out of paper.
  Physically correct the problem.

Invalid picture type (Error 485)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsggfxInvalidPictureTypeS"}

The resource file picture format you tried to load doesn't match the specified property of the object. This error has the following causes and solutions:

- You tried to use the **LoadResPicture** method to load a bitmap resource as the **Icon** property of a form.

  Change the property to the **Picture** property or change the *format* argument of **LoadResPicture** to **vbResIcon**.
- You tried to use the **LoadResPicture** method to load a cursor resource as some property of an object or control other than the **MousePointer** property.

  Change the property reference to **MousePointer**.

Can't empty Clipboard (Error 520)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgclip2CantEmptyS"}

The Clipboard was opened but could not be emptied. This error has the following cause and solution:

- Another application is using the Clipboard and won't release it to your application.
  Set an error trap for this situation in your code and provide a message box with **Retry** and **Cancel** buttons to allow the user to try again after a short pause.

Can't open Clipboard (Error 521)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgclip2CantOpenS"}

The Clipboard has already been opened by another application. This error has the following cause and solution:

- Another application is using the Clipboard and won't release it to your application.

  Set an error trap for this situation in your code and provide a message box with **Retry** and **Cancel** buttons to allow the user to try again after a short pause.

Specified format doesn't match format of data (Error 461)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgclipWrongFormatS"}

The specified Clipboard format is incompatible with the method being executed. This error has the following causes and solutions:

- You tried to use the **GetText** method or **SetText** method with a Clipboard format other than **vbCFText** or **vbCFLink**.

  Before using these methods, use the **GetFormat** method to test whether the current contents of the Clipboard matches the specified format.
- You tried to use the **GetData** method or **SetData** method with a Clipboard format other than **vbCFBitmap**, **vbCFDIB**, or **vbCFMetafile**.

  Before using these methods, use the **GetFormat** method to test whether the current contents of the Clipboard matches the specified graphics format.

Invalid Clipboard format (Error 460)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgclipInvalidFormatS"}

The specified Clipboard format is incompatible with the method being executed. This error has the following causes and solutions:

- You tried to use the **GetText** method or **SetText** method with a Clipboard format other than **vbCFText** or **vbCFLink**.

  Remove the invalid format and specify one of the two valid formats.
- You tried to use the **GetData** or **SetData** method with a Clipboard format other than **vbCFBitmap**, **vbCFDIB**, or **vbCFMetafile**.

  Remove the invalid format and specify one of the three valid graphics formats.

License information for this component not found. You don't have an appropriate license to use this functionality in the design environment (Error 429)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgobjNoLicenseS"}

You are not a licensed user of the ActiveX control. This error has the following cause and solution:

- You tried to place an ActiveX control on a form at design time or tried to add a form with an ActiveX control on it to a project, but the associated information in the registry could not be found.

  The information in the registry may have been deleted or become corrupted. Reinstall the ActiveX control or contact the control vendor.

Form already displayed; can't show modally (Error 400)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgmodShowingFormS"}

You can't use the **Show** method to display a visible form as modal. This error has the following cause and solution:

- You tried to use **Show**, with the *style* argument set to 1 – **vbModal**, on an already visible form.

  Use either the **Unload** statement or the **Hide** method on the form before trying to show it as a modal form.

Must close or hide topmost modal form first (Error 402)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgmodNotTopModalS"}

The modal form you are trying to close or hide isn't on top of the z-order. This error has the following cause and solution:

- Another modal form is higher in the z-order than the modal form you tried to close or hide.

  First use either the **Unload** statement or the **Hide** method on any modal form higher in the z-order. A modal form is a form displayed by the **Show** method, with the *style* argument set to 1 – **vbModal**.

Can't load or unload this object (Error 361)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgldNotLoadableS"}

A **Load** statement or **Unload** statement has referenced an invalid object or control. This error has the following causes and solutions:

- You tried to load or unload an object that isn't a loadable object, such as Screen, Printer, or Clipboard.

  Delete the erroneous statement from your code.
- You tried to load or unload an existing control that isn't part of a control array. For example, assuming that a **TextBox** with the **Name** property Text1 exists, `Load Text1` will cause this error.

  Delete the erroneous statement from your code or change the reference to a control in a control array.
- You tried to unload a **Menu** control in the Click event of its parent menu.

  Unload the **Menu** control with some other procedure.
- You tried to unload the last visible menu item of a **Menu** control.

  Check the setting of the **Visible** property for the other menu items in the control array before trying to unload a menu item, or delete the erroneous statement from your code.

Object already loaded (Error 360)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIdAlreadyLoadedS"}

The control in the control array has already been loaded. This error has the following cause and solution:

- You tried to add a control to a control array at run time with the **Load** statement but the index value you referred to already exists.

  Change the index reference to a new value or check whether your code is executing the same **Load** statement with the same index value reference more than once.

Form not found (Error 424)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOBJNOSUCHFORMS"}

The form was not found. This error has the following cause and solution:

- You tried to add a form to the **UserForms** collection using the **Add** method, but there is no form class of that name. For example, `UserForms.Add "MyForm"`, where `MyForm` doesn't exist.

  Make sure that the class name is available to your project.

Unable to unload within this context (Error 365)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIdCantUnloadHereS"}

In some situations you are not allowed to unload a form or a control on a form. This error has the following causes and solutions:

- There is an **Unload** statement in the Paint event for the form or for a control on the form that has the Paint event.

  Remove the **Unload** statement from the Paint event.
- There is an **Unload** statement in the Change, Click, or DropDown events of a **ComboBox**.

  Remove the **Unload** statement from the event.
- There is an **Unload** statement in the Scroll event of an **HScrollBar** or **VScrollBar** control.

  Remove the **Unload** statement from the event.
- There is an **Unload** statement in the Resize event of a **Data**, **Form**, **MDIForm**, or **PictureBox** control.

  Remove the **Unload** statement from the event.
- There is an **Unload** statement in the Resize event of an **MDIForm** that is trying to unload an MDI child form.

  Remove the **Unload** statement from the event.
- There is an **Unload** statement in the RePosition event or Validate event of a **Data** control.

  Remove the **Unload** statement from the event.
- There is an **Unload** statement in the ObjectMove event of an **OLE Container** control.

  Remove the **Unload** statement from the event.

Specified system DLL could not be loaded (Error 298)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgddeDllNotLoadedS"}

A .dll file provided by the operating system; for example, DDEML.DLL, VERSION.DLL, or WINSPOOL.DRV couldn't be found. This error has the following causes and solutions:

- The file isn't on the proper path.
  Ensure that the DLL is on the Windows System path.
- The DLL is corrupted or was deleted.
  Reload the DLL.
- There isn't enough memory or swap space.
  Try to free up some memory by closing other applications.

Can't use character device names in file names (Error 320)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgfileCharDevInvalidS"}

From within Visual Basic, you can't give a file the same name as a character device driver. This error has the following cause and solution:

- You tried to use a file name such as AUX, CON, COM1, COM2, LPT1, LPT2, LPT3, LPT4, or NUL. Give the file another name.

Invalid file format (Error 321)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgfileCorruptS"}

A Visual Basic form file is damaged. This error has the following causes and solutions:

- The form has a damaged ActiveX control.

  Try replacing the ActiveX control on the form.
- The number of properties in the current version of the ActiveX control doesn't match the expected number of properties.

  Try replacing the ActiveX control with an earlier version or later version.

Illegal parameter. Can't write arrays (Error 328)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCANTWRITEARRAYSS"}

An illegal parameter was passed to the method. This error has the following cause and solution:

- In the WriteProperties event of your User Control, you tried to do a **PropBag.WriteProperty** X, where X is an array. This isn't supported.

  You must write out each element of the array individually.

Could not access system registry (Error 335)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgregBadAccessS"}

An attempt to read from or write to the system registry failed.

ActiveX Component not found (Error 337)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgregServerNotFoundS"}

The required .exe or .dll file can't be found.

ActiveX Component did not run correctly (Error 338)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgregBadServerS"}

The ActiveX component's .exe file failed to run correctly. There may be a problem with the information in the registry.

Object was unloaded (Error 364)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgIdUnloadedS"}

A form was unloaded from its own **_Load** procedure. This error has the following cause and solution:

- A form with an **Unload** statement in its **_Load** procedure was implicitly loaded. For example, the following will implicitly load `YourForm` if it isn't already loaded:

  ```
  MyForm.BackColor = YourForm.BackColor.
  ```

  Remove the **Unload** statement from the **Form_Load** procedure.

The file specified is out of date. This program requires a later version (Error 368)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgldBadVBXVersionS"}

An ActiveX control was found to be out of date. This error has the following cause and solution:

- The number of properties in the current version of the ActiveX control don't match the expected number of properties.

  Replace the ActiveX control with a later version.

Invalid property value (Error 380)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgpropIllegalValueS"}

An inappropriate value is assigned to a property. This error has the following cause and solution:

- You tried to set one of the properties of an object or control to a value outside its permissible range. Change the property value to a valid setting. For example, the **MousePointer** property must be set to an integer from 0 – 15 or 99.

Specified property can't be set at run time (Error 382)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgpropNoWriteRunS"}

The property is read-only at run time. This error has the following cause and solution:

- You tried to set or change a property whose value can only be set at design time.
  Remove the reference to the property from your code or change the reference to only return the value of the property at run time.

Specified property is read-only (Error 383)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgpropNoWriteS"}

The property is read-only at both design time and run time. This error has the following cause and solution:

- You tried to set or change a property whose value can only be read.

  Remove the reference to the property from your code or change the reference to only return the value of the property at run time.

Specified property can't be read at run time (Error 393)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgpropNoReadRunS"}

The property is only available at design time. This error has the following cause and solution:

- You tried to read a property at run time that is only accessible at design time.
  Change your code and remove the reference to the property.

Specified property is write-only (Error 394)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgpropNoReadS"}

The property can't be read. This error has the following cause and solution.

- A property can't be read; for example, `ctl.property = 3` might be legal, but "`Print ctl.property`" would generate this error.

  Change your code and remove the reference to the property.

Permission to use object denied (Error 419)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgobjAccessDeniedS"}

You don't have the necessary permissions for the specified object. This error has the following cause and solution:

- You don't currently have the authority to access this object.
  To change your permission assignments, see your system administrator or the object's creator.

Can't create AutoRedraw image (Error 480)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsggfxCantMakeBitmapS"}

Visual Basic can't create a persistent bitmap for automatic redraw of the form or picture. This error has the following cause and solution:

- There isn't enough available memory for the **AutoRedraw** property to be set to **True**.

  Set the **AutoRedraw** property to **False** and perform your own redraw in the Paint event procedure, or make the **PictureBox** control or **Form** object smaller and try the operation again with **AutoRedraw** set to **True**.

Out of memory (Error 31001)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOutOfMemS"}

Your system could not allocate or access enough memory or disk space for the specified operation.

No object (Error 31004)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoObjectS"}

You tried to perform an action on an object that doesn't exist.

Class is not set (Error 31018)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoClassS"}

If you don't specify a source document (**SourceDoc** property) when setting **Action** = 0 (**CreateEmbed** method), the **Class** property must be set to the name of a class available on your system.

To display a list of the available class names at design time, right-click the **OLE** container control and choose the **Insert Object** command.

Unable to activate object (Error 31027)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantActivateS"}

The object's source document can't be loaded, or the application that created the object isn't available.

This error occurs when you try to activate a linked object (set **Action** = 7) and the file specified in the **SourceDoc** property has been deleted or no longer exists.

This error also occurs when you activate an object (set **Action** = 7), and the action specified by the **Verb** property isn't valid. Some applications that provide objects may support different verbs, depending on the state of the application. All the verbs supported by an application are listed in the **ObjectVerbs** property list. However, some verbs may not be valid for the application's current state.

Unable to create embedded object (Error 31032)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantEmbedS"}

The application that is creating the object can't create the object as specified in the **SourceDoc** property.

For example, this error occurs if you try to embed a spreadsheet object and **SourceDoc** specifies a spreadsheet that is too large to be loaded by the spreadsheet application.

Error saving to file (Error 31036)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgErrorSavingFileS"}

Visual Basic can't write the object to the specified file (set **Action** = 11 or 18). Possible causes:

- The **FileNumber** property is invalid.
- The specified file wasn't opened in Binary mode.
- There isn't enough disk space.

Error loading from file (Error 31037)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbmsgErrorLoadingFileC;vbproBooksOnlineJumpTopic"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vamsgErrorLoadingFileA"}

An error occurred while attempting to read the specified file (set **Action** = 12). Possible causes:

- The **FileNumber** property is invalid.
- The file wasn't opened in Binary mode.
- The file wasn't saved properly (set **Action** = 11).
- The file is corrupted.
- The file position isn't located at the beginning of a valid OLE object.

At line specified: The Form name specified is not valid; can't load this form

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidFormNameS"}

The ASCII file contains a form name that isn't a valid string in Visual Basic. The form isn't loaded.

At line specified: Can't create embedded object specified

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbmsgLineitem1CannotCreateEmbeddedObjectInitem2C"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbmsgLineitem1CannotCreateEmbeddedObjectInitem2S"}

An embedded object could not be created while loading a form, User Control, User Document, or Property Page from a text file. For example, you would get this error if you previously inserted a Microsoft Word document onto the form, and then removed Microsoft Word from your system. This message is written to the error log file.

At line specified: Can't create embedded object specified; license not found

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOBJNOTLICENSEDS"}

An embedded object could not be created during the load of a form, User Control, User Document, or Property Page from a text file, due to the license file not being found. You must have a license to use this object. Check with the object's vendor for more information. This message is written to the error log file.

At line specified: Maximum nesting level for controls exceeded specified item.

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgMaxNestLevelS"}

The ASCII file contains controls nested more than 7 levels deep.

At line specified: All controls must precede menus; can't load specified control

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgMenusOnlyS"}

A control appeared in an incorrect location in the ASCII form file. All controls must be loaded before menus.

At line specified: Parent menu specified can't be loaded as a separator

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgParentIsSeparatorS"}

The ASCII file contains a **Menu** control whose parent or top-level menu is defined as a menu separator. Top-level menus can't be menu separators. The separator won't be set.

At line specified: Can't set checked property in menu specified. Parent menu can't be checked

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoCheckTopLevelS"}

A top-level **Menu** control appeared in the ASCII form file with its **Checked** property set to **True**. Top-level menus can't be checked. The **Menu** control will be loaded, but its **Checked** property won't be set.

At line specified: Can't set Shortcut property in menu specified. Parent menu cannot have a shortcut key

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoShortCutKeyS"}

A top-level **Menu** control appeared in the ASCII form file with a shortcut key defined. Top-level menus can't have a shortcut key. The **Menu** control will be loaded, but its **Shortcut** property won't be set.

At line specified: The form name specified is already in use; can't load this form

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgFormNameInUseS"}

The ASCII file contains a form with a name that is already being used in the application. The form isn't loaded.

Conflicting attributes were found in specified item. The defaults will be used

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCONFLICTFLAGSS"}

Some of the attribute statements in a form, User Control, Property Page, User Document, or Class Module conflict with the required settings for their type. For example, forms must always have the **VB_PredeclaredId** attribute equal to **True**. This error can occur if the file was modified by an editor other than Visual Basic.

Specified item is a read-only file

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgFileIsReadOnlyS"}

You asked to save to a file that is read-only. Read-only files are shown in the **Project Explorer** as  . You can't save to read-only files. Use **Save As** instead.

At line specified: Could not create specified reference

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgINVALIDREFERENCES"}

The reference indicated at the specified line could not be created.

At line specified: The file specified could not be loaded

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgFileNotLoadedS"}

Syntax errors are preventing Visual Basic from parsing and loading a file, or form name conflicts prevent loading of an ASCII form. The form won't be loaded and the form name won't be displayed in the **Project Explorer**.

Make sure that the file causing this error is a valid ASCII form in the correct format and that no conflicts exist among the different forms in the project.

Errors occurred during load

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgERRORLOGOTHERS"}

Check the error log for more descriptive information.

Specified item could not be loaded. Remove it from the list of available add-ins?

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgAddInCantLoadS"}

Visual Basic couldn't load the add-in that you tried to select from the **Available Add-Ins** list in the **Add-In Manager** dialog box. Click **Yes** to remove it from the list or click **No** to leave it on the list. In either case, you can't load it.

Visual Basic can't load the specified item because it is not in the system registry. Please ensure that all add-ins have been installed correctly.

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgAddInBadRegS"}

Visual Basic couldn't load the add-in that you tried to select from the **Available Add-Ins** list in the **Add-In Manager** dialog box, because it was not registered properly or is no longer registered in the system registry.

Display more load errors?

{ewc HLP95EN.DLL,DYNALINK,"Applies To":"vamsgDISPLAYMORELOADERRA"}

Click **OK** to see more errors, otherwise click **Cancel**.

Specified item could not be loaded

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoFileLoadS"}

Syntax errors are preventing Visual Basic from parsing and loading a file, or form name conflicts prevent loading of an ASCII form. The form won't be loaded and the form name won't be displayed in the **Project Explorer**.

Make sure that the file causing this error is a valid ASCII form in the correct format and that no conflicts exist among the different forms in the project.

Version number missing or invalid

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgVersionMissingS"}

The version signature was not found, or the specified version isn't recognized. Make sure that the first line that is not blank or a comment in the ASCII form specifies the correct version.

Specified item is referenced by specified project and cannot be updated

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCONTROLREFERENCEDS"}

The specified control is referenced by more than one project. Updating it for one project might invalidate it for another.

Can't remove control or reference; in use

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCCDLLInUseS"}

The ActiveX control or reference that you tried to remove, is being used by one of the forms in the project. First delete the control or referenced object from the form, and then cancel the selction in the list.

Can't quit at this time

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantExitDlgS"}

You can't quit Windows while Visual Basic is in run mode or break mode. You also can't quit Windows while a dialog box or message box is displayed.

Error loading the specified item. An error was encountered loading a property. Continue?

{ewc HLP95EN.DLL,DYNALINK,"Applies To":"vamsgBadPropContLoadQSA"}

You may have tried to load a form with controls whose names conflict with forms already in the project. For example, loading Form2 that contains a Form1 control triggers this error.

Error loading the specified item. A control could not be loaded due to load error. Continue?

{ewc HLP95EN.DLL,DYNALINK,"Applies To":"vamsgBadCtlContLoadQSA"}

This error message appears after another error has occurred. After you've taken the appropriate action for that error, you will see this error message. To load the control anyway, click **Yes**; to cancel the loading, click **No**.

Specified item has an old file format. When saved, it will be saved in a newer format.

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOldRevS"}

This file was created with an earlier version of Visual Basic. When you save it, it will be saved in the file format of the current version.

Specified item already exists in project

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgProjFileExistsS"}

The file you specified is already part of the project. You can't add the same file to a project more than once. You can't save a file with the same name as another file in the project.

Errors during load. Refer to specified item for details

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgErrorLogS"}

Something unexpected appeared in the ASCII form file. Visual Basic created a log file to provide more detail about the errors. You should examine the log file to determine the severity of the problem. Sometimes you can safely ignore the errors (for example, `Version number missing or invalid...`). Other times, however, the errors could cause the form not to run as expected (for example, `Class MyClass in control MyControl was not a loaded control class`).

String value too long to process; form load aborted

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgValueTooLongS"}

A string embedded in the form being loaded was too long to process.

One or more files in the project have changed. Do you want to save the changes now?

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDirtyFilesS"}

One of the property settings or a code statement has changed for one or more of the files in the project, and the changes have not yet been saved. This is your last chance to save the changes before loading another project.

Specified item could not be registered

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoFileRegisterS"}

The specified file couldn't be registered in the system registry. The error is related to type libraries used by components of the Visual Basic development environment and indicates that the specified file's entry in the system registry is corrupted, or that the DLL itself is missing or corrupted.

Other applications are currently accessing an object in your program. Ending the program now could cause errors in those programs. End program at this time?

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgServerRunningS"}

An Automation object in the Visual Basic program you are currently running is being accessed by at least one other application. Terminating your program could cause an error in the other application. Click **Yes** to quit your program or click **No** to continue it.

One or more of the properties in the specified item was bad. Some or all of the properties might not be set correctly

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgUnhappyMakPropsS"}

The specified project file has one or more invalid property settings. This error occurs only in regard to a project file.

Conflicting names were found in the specified item. The specified name will be used

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCONFLICTNAMESS"}

The name of a form occurs twice in the source file for a form, User Control, Property Page, or User Document. This error can occur if the file was modified by an editor other than Visual Basic.

At line specified: The control name specified is invalid

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidCtlNameS"}

The ASCII file contains a control name that isn't a valid string in Visual Basic. The control isn't loaded.

At line specified: The property name s[ecified is invalid

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidPropNameS"}

The ASCII file contains a property name that isn't a valid property for that control.

The item at the specified line has a quoted string where the property name should be.

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgQuotedStringS"}

A quoted string appeared in the ASCII form file where the property name was expected. Property names are not placed inside quotation marks (" ").

At line specified: Property specified had an invalid property index

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidIndexS"}

The ASCII file contains a property name with a property index greater than 255.

At line specified: Property specified could not be loaded

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCannotLoadPropS"}

The ASCII file contains an unknown property. The property will be skipped when loading the form.

At line specified: Property specified must be a quoted string.

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgMustBeQuotedS"}

The ASCII file contains a property that should appear inside quotation marks (" "), but the quotation marks are missing. This line in the form description is ignored.

At line specified: Property specified had an invalid value

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidPropValueS"}

The ASCII file contains a property with a value that isn't correct for this control. The property is set with its default value.

At line specified: Property specified had an invalid file reference

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidFileRefS"}

The ASCII file contains a reference to a file that Visual Basic couldn't find in the specified directory.

At line specified: Property specified could not be set

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCouldNotSetS"}

Visual Basic can't set the property of the specified control as indicated by the form description in the ASCII file.

At line specified: Class specified of specified control was not a loaded control class

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadClassNameS"}

The ASCII file contains a control class that Visual Basic doesn't recognize. Add the custom control with this class to your project.

At line specified: Did not find an index property, and control specified already exists

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoIndexFoundS"}

The control can't be loaded because there is no index, and it has the same name as a previously loaded control.

At line specified: Control name too long; truncated as specified

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCtlNameTooLongS"}

The ASCII file contains a control name longer than 40 characters. The control will be loaded with the name truncated to 40 characters.

At line specified: Class name too long; truncated as specified

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgClassNameTooLongS"}

The ASCII file contains a class name longer than 40 characters. The class will be loaded with the name truncated to 40 characters.

At line specified: Syntax error: property specified was missing an equal sign (=).

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgMissingEqualS"}

The ASCII file contains a property name and value without an equal sign (**=**) between them. The property isn't loaded.

At line specified: Can't load control specified; containing control not a valid container

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalContainerS"}

You attempted to load a control into a control which isn't a valid container.

At line specified: Can't load control specified; name already in use

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCtlClsMismatchS"}

The control named in the ASCII text file couldn't be loaded because its name is already in use in the application.

At line specified: Missing or invalid control class in file specified

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgMissingControlClassS"}

The ASCII file contains an unknown control class in the form description, or the class name isn't a valid string in Visual Basic.

At line specified: Missing or invalid control name in file specified

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgMissingControlNameS"}

The ASCII file contains an unknown control name in the form description, or the control name isn't a valid string in Visual Basic.

At line specified: Can't load control specified; license not found

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCTLNOTLICENSEDS"}

An ActiveX control could not be created while loading a form, User Control, User Document, or Property Page from a text file. The control may be missing or corrupted. Reinstall the control and try again. This message is written to the error log file.

At line specified: The CLSID specified is invalid.

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgINVALIDCLSIDS"}

An object could not be loaded while loading a Form, User Control, User Document, or Property Page from a text file. The CLSID specified in the file is not valid. Applies only to objects that are properties, such as the **Font** object. This message is written to the error log file.

Could not create reference to specified item

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBADREFERENCESTRS"}

There was an error establishing a reference while loading the file, so the reference was not added.

Specified item will not be loaded. Name is already in use

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDUPLICATENAMES"}

You can't have two items of the specified kind loaded at the same time.

ChDir Statement Example

This example uses the **ChDir** statement to change the current directory or folder.

```
' Change current directory or folder to "MYDIR".
ChDir "MYDIR"

' Assume "C:" is the current drive. The following statement changes
' the default directory on drive "D:". "C:" remains the current drive.
ChDir "D:\WINDOWS\SYSTEM"
```

ChDrive Statement Example

This example uses the **ChDrive** statement to change the current drive.

```
ChDrive "D" ' Make "D" the current drive.
```

CurDir Function Example

This example uses the **CurDir** function to return the current path.

```
' Assume current path on C drive is "C:\WINDOWS\SYSTEM".
' Assume current path on D drive is "D:\EXCEL".
' Assume C is the current drive.
Dim MyPath
MyPath = CurDir' Returns "C:\WINDOWS\SYSTEM".
MyPath = CurDir("C") ' Returns "C:\WINDOWS\SYSTEM".
MyPath = CurDir("D") ' Returns "D:\EXCEL".
```

Dir Function Example

This example uses the **Dir** function to check if certain files and directories exist.

```
Dim MyFile, MyPath, MyName
' Returns "WIN.INI" if it exists.
MyFile = Dir("C:\WINDOWS\WIN.INI")

' Returns filename with specified extension. If more than one *.ini
' file exists, the first file found is returned.
MyFile = Dir("C:\WINDOWS\*.INI")

' Call Dir again without arguments to return the next *.INI file in the
' same directory.
MyFile = Dir

' Return first *.TXT file with a set hidden attribute.
MyFile = Dir("*.TXT", vbHidden)

' Display the names in C:\ that represent directories.
MyPath = "c:\" ' Set the path.
MyName = Dir(MyPath, vbDirectory)   ' Retrieve the first entry.
Do While MyName <> ""' Start the loop.
   ' Ignore the current directory and the encompassing directory.
   If MyName <> "." And MyName <> ".." Then
      ' Use bitwise comparison to make sure MyName is a directory.
      If (GetAttr(MyPath & MyName) And vbDirectory) = vbDirectory Then
         Debug.Print MyName  ' Display entry only if it
      End If    ' it represents a directory.
   End If
   MyName = Dir    ' Get next entry.
Loop
```

Environ Function Example

This example uses the **Environ** function to supply the entry number and length of the `PATH` statement from the environment-string table.

```
Dim EnvString, Indx, Msg, PathLen  ' Declare variables.
Indx = 1 ' Initialize index to 1.
Do
   EnvString = Environ(Indx) ' Get environment
             ' variable.
   If Left(EnvString, 5) = "PATH=" Then  ' Check PATH entry.
      PathLen = Len(Environ("PATH"))     ' Get length.
      Msg = "PATH entry = " & Indx & " and length = " & PathLen
      Exit Do
   Else
      Indx = Indx + 1   ' Not PATH entry,
   End If   ' so increment.
Loop Until EnvString = ""
If PathLen > 0 Then
   MsgBox Msg  ' Display message.
Else
   MsgBox "No PATH environment variable exists."
End If
```

FileCopy Statement Example

This example uses the **FileCopy** statement to copy one file to another. For purposes of this example, assume that `SRCFILE` is a file containing some data.

```
Dim SourceFile, DestinationFile
SourceFile = "SRCFILE"  ' Define source file name.
DestinationFile = "DESTFILE" ' Define target file name.
FileCopy SourceFile, DestinationFile     ' Copy source to target.
```

FileDateTime Function Example

This example uses the **FileDateTime** function to determine the date and time a file was created or last modified. The format of the date and time displayed is based on the locale settings of your system.

```
Dim MyStamp
' Assume TESTFILE was last modified on February 12, 1993 at 4:35:47 PM.
' Assume English/U.S. locale settings.
MyStamp = FileDateTime("TESTFILE") ' Returns "2/12/93 4:35:47 PM".
```

FileLen Function Example

This example uses the **FileLen** function to return the length of a file in bytes. For purposes of this example, assume that `TESTFILE` is a file containing some data.

```
Dim MySize
MySize = FileLen("TESTFILE") ' Returns file length (bytes).
```

GetAttr Function Example

This example uses the **GetAttr** function to determine the attributes of a file and directory or folder.

```
Dim MyAttr
' Assume file TESTFILE has hidden attribute set.
MyAttr = GetAttr("TESTFILE") ' Returns 2.

' Returns nonzero if hidden attribute is set on TESTFILE.
Debug.Print MyAttr And vbHidden

' Assume file TESTFILE has hidden and read-only attributes set.
MyAttr = GetAttr("TESTFILE") ' Returns 3.

' Returns nonzero if hidden attribute is set on TESTFILE.
Debug.Print MyAttr And (vbHidden + vbReadOnly)

' Assume MYDIR is a directory or folder.
MyAttr = GetAttr("MYDIR")  ' Returns 16.
```

Kill Statement Example

This example uses the **Kill** statement to delete a file from a disk.

```
' Assume TESTFILE is a file containing some data.
Kill "TestFile"' Delete file.

' Delete all *.TXT files in current directory.
Kill "*.TXT"
```

MkDir Statement Example

This example uses the **MkDir** statement to create a directory or folder. If the drive is not specified, the new directory or folder is created on the current drive.

```
MkDir "MYDIR"  ' Make new directory or folder.
```

Name Statement Example

This example uses the **Name** statement to rename a file. For purposes of this example, assume that the directories or folders that are specified already exist.

```
Dim OldName, NewName
OldName = "OLDFILE": NewName = "NEWFILE" ' Define filenames.
Name OldName As NewName ' Rename file.

OldName = "C:\MYDIR\OLDFILE": NewName = "C:\YOURDIR\NEWFILE"
Name OldName As NewName ' Move and rename file.
```

QBColor Function Example

This example uses the **QBColor** function to change the **BackColor** property of the form passed in as `MyForm` to the color indicated by `ColorCode`. **QBColor** accepts integer values between 0 and 15.

```
Sub ChangeBackColor (ColorCode As Integer, MyForm As Form)
   MyForm.BackColor = QBColor(ColorCode)
End Sub
```

RGB Function Example

This example shows how the **RGB** function is used to return a whole number representing an **RGB** color value. It is used for those application methods and properties that accept a color specification. The object `MyObject` and its property are used for illustration purposes only. If `MyObject` does not exist, or if it does not have a Color property, an error occurs.

```
Dim RED, I, RGBValue, MyObject
Red = RGB(255, 0, 0) ' Return the value for Red.
I = 75' Initialize offset.
RGBValue = RGB(I, 64 + I, 128 + I) ' Same as RGB(75, 139, 203).
MyObject.Color = RGB(255, 0, 0)    ' Set the Color property of
   ' MyObject to Red.
```

RmDir Statement Example

This example uses the **RmDir** statement to remove an existing directory or folder.

```
' Assume that MYDIR is an empty directory or folder.
RmDir "MYDIR"  ' Remove MYDIR.
```

SetAttr Statement Example

This example uses the **SetAttr** statement to set attributes for a file.

```
SetAttr "TESTFILE", vbHidden ' Set hidden attribute.
SetAttr "TESTFILE", vbHidden + vbReadOnly' Set hidden and read-only
   ' attributes.
```

ChDir Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmChDirC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmChDirX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmChDirS"}

Changes the current directory or folder.

Syntax

ChDir *path*

The required *path* argument is a string expression that identifies which directory or folder becomes the new default directory or folder. The *path* may include the drive. If no drive is specified, **ChDir** changes the default directory or folder on the current drive.

Remarks

The **ChDir** statement changes the default directory but not the default drive. For example, if the default drive is C, the following statement changes the default directory on drive D, but C remains the default drive:

```
ChDir "D:\TMP"
```

ChDrive Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmChDriveC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmChDriveX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmChDriveS"}

Changes the current drive.

Syntax

ChDrive *drive*

The required *drive* argument is a string expression that specifies an existing drive. If you supply a zero-length string (""), the current drive doesn't change. If the *drive* argument is a multiple-character string, **ChDrive** uses only the first letter.

CurDir Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctCurDirC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctCurDirX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctCurDirS"}

Returns a **Variant** (**String**) representing the current path.

Syntax

CurDir[(*drive*)]

The optional *drive* argument is a string expression that specifies an existing drive. If no drive is specified or if *drive* is a zero-length string (""), **CurDir** returns the path for the current drive.

Dir Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctDirC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctDirX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctDirS"}

Returns a **String** representing the name of a file, directory, or folder that matches a specified pattern or file attribute, or the volume label of a drive.

Syntax

Dir[**(***pathname*[**,** *attributes*]**)**]

The **Dir** function syntax has these parts:

Part	Description
pathname	Optional. String expression that specifies a file name—may include directory or folder, and drive. A zero-length string ("") is returned if *pathname* is not found.
attributes	Optional. Constant or numeric expression, whose sum specifies file attributes. If omitted, all files are returned that match *pathname*.

Settings

The *attributes* argument settings are:

Constant	Value	Description
vbNormal	0	Normal
vbHidden	2	Hidden
vbSystem	4	System file
vbVolume	8	Volume label; if specified, all other attributes are ignored
vbDirectory	16	Directory or folder

Note These constants are specified by Visual Basic for Applications and can be used anywhere in your code in place of the actual values.

Remarks

Dir supports the use of multiple-character (*****) and single-character (**?**) wildcards to specify multiple files.

You must specify *pathname* the first time you call the **Dir** function, or an error occurs. If you also specify file attributes, *pathname* must be included.

Dir returns the first file name that matches *pathname*. To get any additional file names that match *pathname*, call **Dir** again with no arguments. When no more file names match, **Dir** returns a zero-length string (""). Once a zero-length string is returned, you must specify *pathname* in subsequent calls or an error occurs. You can change to a new *pathname* without retrieving all of the file names that match the current *pathname*. However, you can't call the **Dir** function recursively. Calling **Dir** with the **vbDirectory** attribute does not continually return subdirectories.

Tip Because file names are retrieved in no particular order, you may want to store returned file names in an array, and then sort the array.

Environ Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctEnvironC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctEnvironX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctEnvironS"}

Returns the **String** associated with an operating system environment variable.

Syntax

Environ({*envstring* | *number*})

The **Environ** function syntax has these named arguments:

Part	Description
envstring	Optional. String expression containing the name of an environment variable.
number	Optional. Numeric expression corresponding to the numeric order of the environment string in the environment-string table. The ***number*** argument can be any numeric expression, but is rounded to a whole number before it is evaluated.

Remarks

If ***envstring*** can't be found in the environment-string table, a zero-length string ("") is returned. Otherwise, **Environ** returns the text assigned to the specified ***envstring***; that is, the text following the equal sign (=) in the environment-string table for that environment variable.

If you specify ***number***, the string occupying that numeric position in the environment-string table is returned. In this case, **Environ** returns all of the text, including ***envstring***. If there is no environment string in the specified position, **Environ** returns a zero-length string.

FileCopy Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmFileCopyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmFileCopyX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmFileCopyS"}

Copies a file.

Syntax

FileCopy *source*, *destination*

The **FileCopy** statement syntax has these named arguments:

Part	Description
source	Required. String expression that specifies the name of the file to be copied. The ***source*** may include directory or folder, and drive.
destination	Required. String expression that specifies the target file name. The ***destination*** may include directory or folder, and drive.

Remarks

If you try to use the **FileCopy** statement on a currently open file, an error occurs.

FileDateTime Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctFileDateTimeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctFileDateTimeX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctFileDateTimeS"}

Returns a **Variant** (**Date**) that indicates the date and time when a file was created or last modified.

Syntax

FileDateTime(*pathname***)**

The required *pathname* argument is a string expression that specifies a file name. The *pathname* may include the directory or folder, and the drive.

FileLen Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctFileLenC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctFileLenX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctFileLenS"}

Returns a **Long** specifying the length of a file in bytes.

Syntax

FileLen(*pathname***)**

The required *pathname* argument is a string expression that specifies a file. The *pathname* may include the directory or folder, and the drive.

Remarks

If the specified file is open when the **FileLen** function is called, the value returned represents the size of the file immediately before it was opened.

Note To obtain the length of an open file, use the **LOF** function.

GetAttr Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctGetAttrC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctGetAttrX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctGetAttrS"}

Returns an **Integer** representing the attributes of a file, directory, or folder.

Syntax

GetAttr(*pathname***)**

The required *pathname* argument is a string expression that specifies a file name. The *pathname* may include the directory or folder, and the drive.

Return Values

The value returned by **GetAttr** is the sum of the following attribute values:

Constant	Value	Description
vbNormal	0	Normal
vbReadOnly	1	Read-only
vbHidden	2	Hidden
vbSystem	4	System
vbDirectory	16	Directory or folder
vbArchive	32	File has changed since last backup

Note These constants are specified by Visual Basic for Applications. The names can be used anywhere in your code in place of the actual values.

Remarks

To determine which attributes are set, use the **And** operator to perform a bitwise comparison of the value returned by the **GetAttr** function and the value of the individual file attribute you want. If the result is not zero, that attribute is set for the named file. For example, the return value of the following **And** expression is zero if the Archive attribute is not set:

```
Result = GetAttr(FName) And vbArchive
```

A nonzero value is returned if the Archive attribute is set.

Kill Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmKillC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmKillX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmKillS"}

Deletes files from a disk.

Syntax

Kill *pathname*

The required *pathname* argument is a string expression that specifies one or more file names to be deleted. The *pathname* may include the directory or folder, and the drive.

Remarks

Kill supports the use of multiple-character (*****) and single-character (**?**) wildcards to specify multiple files.

An error occurs if you try to use **Kill** to delete an open file.

Note To delete directories, use the **RmDir** statement.

MkDir Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmMkDirC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmMkDirX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmMkDirS"}

Creates a new directory or folder.

Syntax

MkDir *path*

The required *path* argument is a string expression that identifies the directory or folder to be created. The *path* may include the drive. If no drive is specified, **MkDir** creates the new directory or folder on the current drive.

Name Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmNameC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmNameX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmNameS"}

Renames a disk file, directory, or folder.

Syntax

Name *oldpathname* **As** *newpathname*

The **Name** statement syntax has these parts:

Part	Description
oldpathname	Required. String expression that specifies the existing file name and location—may include directory or folder, and drive.
newpathname	Required. String expression that specifies the new file name and location—may include directory or folder, and drive. The file name specified by *newpathname* can't already exist.

Remarks

Both *newpathname* and *oldpathname* must be on the same drive. If the path in *newpathname* exists and is different from the path in *oldpathname*, the **Name** statement moves the file to the new directory or folder and renames the file, if necessary. If *newpathname* and *oldpathname* have different paths and the same file name, **Name** moves the file to the new location and leaves the file name unchanged. Using **Name**, you can move a file from one directory or folder to another, but you can't move a directory or folder.

Using **Name** on an open file produces an error. You must close an open file before renaming it. **Name** arguments cannot include multiple-character (*****) and single-character (**?**) wildcards.

QBColor Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctQBColorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctQBColorX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctQBColorS"}

Returns a **Long** representing the RGB color code corresponding to the specified color number.

Syntax

QBColor(*color***)**

The required *color* argument is a whole number in the range 0–15.

Settings

The *color* argument has these settings:

Number	Color	Number	Color
0	Black	8	Gray
1	Blue	9	Light Blue
2	Green	10	Light Green
3	Cyan	11	Light Cyan
4	Red	12	Light Red
5	Magenta	13	Light Magenta
6	Yellow	14	Light Yellow
7	White	15	Bright White

Remarks

The *color* argument represents color values used by earlier versions of Basic (such as Microsoft Visual Basic for MS-DOS and the Basic Compiler). Starting with the least-significant byte, the returned value specifies the red, green, and blue values used to set the appropriate color in the RGB system used by Visual Basic for Applications.

RGB Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctRGBC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctRGBX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctRGBS"}

Returns a **Long** whole number representing an RGB color value.

Syntax

RGB(*red*, *green*, *blue*)

The **RGB** function syntax has these named arguments:

Part	Description
red	Required; **Variant** (**Integer**). Number in the range 0–255, inclusive, that represents the red component of the color.
green	Required; **Variant** (**Integer**). Number in the range 0–255, inclusive, that represents the green component of the color.
blue	Required; **Variant** (**Integer**). Number in the range 0–255, inclusive, that represents the blue component of the color.

Remarks

Application methods and properties that accept a color specification expect that specification to be a number representing an RGB color value. An RGB color value specifies the relative intensity of red, green, and blue to cause a specific color to be displayed.

The value for any argument to **RGB** that exceeds 255 is assumed to be 255.

The following table lists some standard colors and the red, green, and blue values they include:

Color	Red Value	Green Value	Blue Value
Black	0	0	0
Blue	0	0	255
Green	0	255	0
Cyan	0	255	255
Red	255	0	0
Magenta	255	0	255
Yellow	255	255	0
White	255	255	255

RmDir Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmRmDirC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmRmDirX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmRmDirS"}

Removes an existing directory or folder.

Syntax

RmDir *path*

The required *path* argument is a string expression that identifies the directory or folder to be removed. The *path* may include the drive. If no drive is specified, **RmDir** removes the directory or folder on the current drive.

Remarks

An error occurs if you try to use **RmDir** on a directory or folder containing files. Use the **Kill** statement to delete all files before attempting to remove a directory or folder.

SetAttr Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmSetAttrC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmSetAttrX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmSetAttrS"}

Sets attribute information for a file.

Syntax

SetAttr *pathname*, *attributes*

The **SetAttr** statement syntax has these named arguments:

Part	Description
pathname	Required. String expression that specifies a file name—may include directory or folder, and drive.
attributes	Required. Constant or numeric expression, whose sum specifies file attributes.

Settings

The ***attributes*** argument settings are:

Constant	Value	Description
vbNormal	0	Normal (default)
vbReadOnly	1	Read-only
vbHidden	2	Hidden
vbSystem	4	System file
vbArchive	32	File has changed since last backup

Note These constants are specified by Visual Basic for Applications. The names can be used anywhere in your code in place of the actual values.

Remarks

A run-time error occurs if you try to set the attributes of an open file.

Close Statement Example

This example uses the **Close** statement to close all three files opened for **Output**.

```
Dim I, FileName
For I = 1 To 3 ' Loop 3 times.
   FileName = "TEST" & I   ' Create file name.
   Open FileName For Output As #I  ' Open file.
   Print #I, "This is a test."     ' Write string to file.
Next I
Close ' Close all 3 open files.
```

EOF Function Example

This example uses the **EOF** function to detect the end of a file. This example assumes that `MYFILE` is a text file with a few lines of text.

```
Dim InputData
Open "MYFILE" For Input As #1' Open file for input.
Do While Not EOF(1)   ' Check for end of file.
   Line Input #1, InputData  ' Read line of data.
   Debug.Print InputData   ' Print to Debug window.
Loop
Close #1 ' Close file.
```

FileAttr Function Example

This example uses the **FileAttr** function to return the file mode and file handle of an open file.

```
Dim FileNum, Mode, Handle
FileNum = 1 ' Assign file number.
Open "TESTFILE" For Append As FileNum    ' Open file.
Mode = FileAttr(FileNum, 1)' Returns 8 (Append file mode).
Handle = FileAttr(FileNum, 2)' Returns file handle.
Close FileNum  ' Close file.
```

FreeFile Function Example

This example uses the **FreeFile** function to return the next available file number. Five files are opened for output within the loop, and some sample data is written to each.

```
Dim MyIndex, FileNumber
For MyIndex = 1 To 5 ' Loop 5 times.
   FileNumber = FreeFile   ' Get unused file
      ' number.
   Open "TEST" & MyIndex For Output As #FileNumber   ' Create filename.
   Write #FileNumber, "This is a sample."      ' Output text.
   Close #FileNumber ' Close file.
Next MyIndex
```

Get Statement Example

This example uses the **Get** statement to read data from a file into a variable. This example assumes that `TESTFILE` is a file containing five records of the user-defined type `Record`.

```
Type Record ' Define user-defined type.
   ID As Integer
   Name As String * 20
End Type

Dim MyRecord As Record, Position   ' Declare variables.
' Open sample file for random access.
Open "TESTFILE" For Random As #1 Len = Len(MyRecord)
' Read the sample file using the Get statement.
Position = 3' Define record number.
Get #1, Position, MyRecord ' Read third record.
Close #1 ' Close file.
```

Input # Statement Example

This example uses the **Input #** statement to read data from a file into two variables. This example assumes that `TESTFILE` is a file with a few lines of data written to it using the **Write #** statement; that is, each line contains a string in quotations and a number separated by a comma, for example, ("Hello", 234).

```
Dim MyString, MyNumber
Open "TESTFILE" For Input As #1    ' Open file for input.
Do While Not EOF(1)   ' Loop until end of file.
   Input #1, MyString, MyNumber    ' Read data into two variables.
   Debug.Print MyString, MyNumber  ' Print data to Debug window.
Loop
Close #1 ' Close file.
```

Input Function Example

This example uses the **Input** function to read one character at a time from a file and print it to the **Debug** window. This example assumes that `TESTFILE` is a text file with a few lines of sample data.

```
Dim MyChar
Open "TESTFILE" For Input As #1    ' Open file.
Do While Not EOF(1)  ' Loop until end of file.
   MyChar = Input(1, #1)   ' Get one character.
   Debug.Print MyChar   ' Print to Debug window.
Loop
Close #1 ' Close file.
```

Line Input # Statement Example

This example uses the **Line Input #** statement to read a line from a sequential file and assign it to a variable. This example assumes that `TESTFILE` is a text file with a few lines of sample data.

```
Dim TextLine
Open "TESTFILE" For Input As #1    ' Open file.
Do While Not EOF(1)   ' Loop until end of file.
   Line Input #1, TextLine ' Read line into variable.
   Debug.Print TextLine ' Print to Debug window.
Loop
Close #1 ' Close file.
```

Loc Function Example

This example uses the **Loc** function to return the current read/write position within an open file. This example assumes that `TESTFILE` is a text file with a few lines of sample data.

```
Dim MyLocation, MyLine
Open "TESTFILE" For Binary As #1    ' Open file just created.
Do While MyLocation < LOF(1) ' Loop until end of file.
   MyLine = MyLine & Input(1, #1)  ' Read character into variable.
   MyLocation = Loc(1)  ' Get current position within file.
' Print to Debug window.
   Debug.Print MyLine; Tab; MyLocation
Loop
Close #1 ' Close file.
```

Lock, Unlock Statements Example

This example illustrates the use of the **Lock** and **Unlock** statements. While a record is being modified, access by other processes to the record is denied. This example assumes that `TESTFILE` is a file containing five records of the user-defined type `Record`.

```
Type Record ' Define user-defined type.
   ID As Integer
   Name As String * 20
End Type
Dim MyRecord As Record, RecordNumber     ' Declare variables.
' Open sample file for random access.
Open "TESTFILE" For Random Shared As #1 Len = Len(MyRecord)
RecordNumber = 4   ' Define record number.
Lock #1, RecordNumber' Lock record.
Get #1, RecordNumber, MyRecord     ' Read record.
MyRecord.ID = 234 ' Modify record.
MyRecord.Name = "John Smith"
Put #1, RecordNumber, MyRecord     ' Write modified record.
Unlock #1, RecordNumber ' Unlock current record.
Close #1 ' Close file.
```

LOF Function Example

This example uses the **LOF** function to determine the size of an open file. This example assumes that `TESTFILE` is a text file containing sample data.

```
Dim FileLength
Open "TESTFILE" For Input As #1    ' Open file.
FileLength = LOF(1)  ' Get length of file.
Close #1 ' Close file.
```

Open Statement Example

This example illustrates various uses of the **Open** statement to enable input and output to a file.

The following code opens the file `TESTFILE` in sequential-input mode.

```
Open "TESTFILE" For Input As #1
' Close before reopening in another mode.
Close #1
```

This example opens the file in Binary mode for writing operations only.

```
Open "TESTFILE" For Binary Access Write As #1
' Close before reopening in another mode.
Close #1
```

The following example opens the file in Random mode. The file contains records of the user-defined type `Record`.

```
Type Record ' Define user-defined type.
   ID As Integer
   Name As String * 20
End Type

Dim MyRecord As Record  ' Declare variable.
Open "TESTFILE" For Random As #1 Len = Len(MyRecord)
' Close before reopening in another mode.
Close #1
```

This code example opens the file for sequential output; any process can read or write to file.

```
Open "TESTFILE" For Output Shared As #1
' Close before reopening in another mode.
Close #1
```

This code example opens the file in Binary mode for reading; other processes can't read file.

```
Open "TESTFILE" For Binary Access Read Lock Read As #1
```

Print # Statement Example

This example uses the **Print #** statement to write data to a file.

```
Open "TESTFILE" For Output As #1    ' Open file for output.
Print #1, "This is a test" ' Print text to file.
Print #1,' Print blank line to file.
Print #1, "Zone 1"; Tab ; "Zone 2" ' Print in two print zones.
Print #1, "Hello" ; " " ; "World"  ' Separate strings with space.
Print #1, Spc(5) ; "5 leading spaces "   ' Print five leading spaces.
Print #1, Tab(10) ; "Hello"' Print word at column 10.

' Assign Boolean, Date, Null and Error values.
Dim MyBool, MyDate, MyNull, MyError
MyBool = False : MyDate = #February 12, 1969# : MyNull = Null
MyError = CVErr(32767)
' True, False, Null, and Error are translated using locale settings of
' your system. Date literals are written using standard short date
' format.
Print #1, MyBool ; " is a Boolean value"
Print #1, MyDate ; " is a date"
Print #1, MyNull ; " is a null value"
Print #1, MyError ; " is an error value"
Close #1 ' Close file.
```

Put Statement Example

This example uses the **Put** statement to write data to a file. Five records of the user-defined type `Record` are written to the file.

```
Type Record ' Define user-defined type.
   ID As Integer
   Name As String * 20
End Type
Dim MyRecord As Record, RecordNumber     ' Declare variables.
' Open file for random access.
Open "TESTFILE" For Random As #1 Len = Len(MyRecord)
For RecordNumber = 1 To 5  ' Loop 5 times.
   MyRecord.ID = RecordNumber      ' Define ID.
   MyRecord.Name = "My Name" & RecordNumber    ' Create a string.
   Put #1, RecordNumber, MyRecord  ' Write record to file.
Next RecordNumber
Close #1 ' Close file.
```

Reset Statement Example

This example uses the **Reset** statement to close all open files and write the contents of all file buffers to disk. Note the use of the **Variant** variable `FileNumber` as both a string and a number.

```
Dim FileNumber
For FileNumber = 1 To 5 ' Loop 5 times.
   ' Open file for output. FileNumber is concatenated into the string
   ' TEST for the filename, but is a number following a #.
   Open "TEST" & FileNumber For Output As #FileNumber
   Write #FileNumber, "Hello World"      ' Write data to file.
Next FileNumber
Reset ' Close files and write contents
      ' to disk.
```

Seek Function Example

This example uses the **Seek** function to return the current file position. The example assumes `TESTFILE` is a file containing records of the user-defined type `Record`.

```
Type Record ' Define user-defined type.
   ID As Integer
   Name As String * 20
End Type
```

For files opened in Random mode, **Seek** returns number of next record.

```
Dim MyRecord As Record  ' Declare variable.
Open "TESTFILE" For Random As #1 Len = Len(MyRecord)
Do While Not EOF(1)  ' Loop until end of file.
   Get #1, , MyRecord   ' Read next record.
   Debug.Print Seek(1)  ' Print record number to Debug
         ' window.
Loop
Close #1 ' Close file.
```

For files opened in modes other than Random mode, **Seek** returns the byte position at which the next operation takes place. Assume `TESTFILE` is a file containing a few lines of text.

```
Dim MyChar
Open "TESTFILE" For Input As #1    ' Open file for reading.
Do While Not EOF(1)  ' Loop until end of file.
   MyChar = Input(1, #1)   ' Read next character of data.
   Debug.Print Seek(1)  ' Print byte position to Debug
         ' window.
Loop
Close #1 ' Close file.
```

Seek Statement Example

This example uses the **Seek** statement to set the position for the next read or write within a file. This example assumes `TESTFILE` is a file containing records of the user-defined type `Record`.

```
Type Record ' Define user-defined type.
   ID As Integer
   Name As String * 20
End Type
```

For files opened in Random mode, **Seek** sets the next record.

```
Dim MyRecord As Record, MaxSize, RecordNumber  ' Declare variables.
' Open file in random-file mode.
Open "TESTFILE" For Random As #1 Len = Len(MyRecord)
MaxSize = LOF(1) \ Len(MyRecord)   ' Get number of records in file.
' The loop reads all records starting from the last.
For RecordNumber = MaxSize To 1 Step - 1
   Seek #1, RecordNumber   ' Set position.
   Get #1, , MyRecord   ' Read record.
Next RecordNumber
Close #1 ' Close file.
```

For files opened in modes other than Random mode, **Seek** sets the byte position at which the next operation takes place. Assume `TESTFILE` is a file containing a few lines of text.

```
Dim MaxSize, NextChar, MyChar
Open "TESTFILE" For Input As #1    ' Open file for input.
MaxSize = LOF(1)  ' Get size of file in bytes.
' The loop reads all characters starting from the last.
For NextChar = MaxSize To 1 Step -1
   Seek #1, NextChar ' Set position.
   MyChar = Input(1, #1)   ' Read character.
Next NextChar
Close #1 ' Close file.
```

Spc Function Example

This example uses the **Spc** function to position output in a file and in the **Debug** window.

```
' The Spc function can be used with the Print # statement.
Open "TESTFILE" For Output As #1    ' Open file for output.
Print #1, "10 spaces between here"; Spc(10); "and here."
Close #1 ' Close file.
```

The following statement causes the text to be printed in the **Debug** window (using the **Print** method), preceded by 30 spaces.

```
Debug.Print Spc(30); "Thirty spaces later..."
```

Tab Function Example

This example uses the **Tab** function to position output in a file and in the **Debug** window.

```
' The Tab function can be used with the Print # statement.
Open "TESTFILE" For Output As #1   ' Open file for output.
' The second word prints at column 20.
Print #1, "Hello"; Tab(20); "World."
' If the argument is omitted, cursor is moved to the next print zone.
Print #1, "Hello"; Tab; "World"
Close #1 ' Close file.
```

The **Tab** function can also be used with the **Print** method. The following statement prints text starting at column 10.

```
Debug.Print Tab(10); "10 columns from start."
```

Width # Statement Example

This example uses the **Width #** statement to set the output line width for a file.

```
Dim I
Open "TESTFILE" For Output As #1    ' Open file for output.
Width #1, 5 ' Set output line width to 5.
For I = 0 To 9 ' Loop 10 times.
   Print #1, Chr(48 + I);  ' Prints five characters per line.
Next I
Close #1 ' Close file.
```

Write # Statement Example

This example uses the **Write #** statement to write raw data to a sequential file.

```
Open "TESTFILE" For Output As #1    ' Open file for output.
Write #1, "Hello World", 234 ' Write comma-delimited data.
Write #1,' Write blank line.

Dim MyBool, MyDate, MyNull, MyError
' Assign Boolean, Date, Null, and Error values.
MyBool = False : MyDate = #February 12, 1969# : MyNull = Null
MyError = CVErr(32767)
' Boolean data is written as #TRUE# or #FALSE#. Date literals are
' written in universal date format, for example, #1994-07-13#
 'represents July 13, 1994. Null data is written as #NULL#.
' Error data is written as #ERROR errorcode#.
Write #1, MyBool ; " is a Boolean value"
Write #1, MyDate ; " is a date"
Write #1, MyNull ; " is a null value"
Write #1, MyError ; " is an error value"
Close #1 ' Close file.
```

Close Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmCloseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmCloseX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmCloseS"}

Concludes input/output (I/O) to a file opened using the **Open** statement.

Syntax

Close [*filenumberlist*]

The optional *filenumberlist* argument can be one or more file numbers using the following syntax, where *filenumber* is any valid file number:

[[**#**]*filenumber*] [**,** [**#**]*filenumber*] **. . .**

Remarks

If you omit *filenumberlist*, all active files opened by the **Open** statement are closed.

When you close files that were opened for **Output** or **Append**, the final buffer of output is written to the operating system buffer for that file. All buffer space associated with the closed file is released.

When the **Close** statement is executed, the association of a file with its file number ends.

EOF Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctEOFC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctEOFX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctEOFS"}

Returns an **Integer** containing the **Boolean** value **True** when the end of a file opened for **Random** or sequential **Input** has been reached.

Syntax

EOF(*filenumber***)**

The required *filenumber* argument is an **Integer** containing any valid file number.

Remarks

Use **EOF** to avoid the error generated by attempting to get input past the end of a file.

The **EOF** function returns **False** until the end of the file has been reached. With files opened for **Random** or **Binary** access, **EOF** returns **False** until the last executed **Get** statement is unable to read an entire record.

With files opened for **Binary** access, an attempt to read through the file using the **Input** function until **EOF** returns **True** generates an error. Use the **LOF** and **Loc** functions instead of **EOF** when reading binary files with **Input**, or use **Get** when using the **EOF** function. With files opened for **Output**, **EOF** always returns **True**.

FileAttr Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctFileAttrC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctFileAttrX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctFileAttrS"}

Returns a **Long** representing the file mode for files opened using the **Open** statement.

Syntax

FileAttr(*filenumber*, *returntype*)

The **FileAttr** function syntax has these named arguments:

Part	Description
filenumber	Required; **Integer**. Any valid file number.
returntype	Required; **Integer**. Number indicating the type of information to return. Specify 1 to return a value indicating the file mode. On 16-bit systems only, specify 2 to retrieve an operating system file handle. ***Returntype*** 2 is not supported in 32-bit systems and causes an error.

Return Values

When the ***returntype*** argument is 1, the following return values indicate the file access mode:

Mode	Value
Input	1
Output	2
Random	4
Append	8
Binary	32

FreeFile Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctFreeFileC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctFreeFileX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctFreeFileS"}

Returns an **Integer** representing the next file number available for use by the **Open** statement.

Syntax

FreeFile[(*rangenumber*)]

The optional *rangenumber* argument is a **Variant** that specifies the range from which the next free file number is to be returned. Specify a 0 (default) to return a file number in the range 1 – 255, inclusive. Specify a 1 to return a file number in the range 256 – 511.

Remarks

Use **FreeFile** to supply a file number that is not already in use.

Get Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmGetC"}
{ewc HLP95EN.DLL,DYNALINK,"Example":"vastmGetX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmGetS"}

Reads data from an open disk file into a variable.

Syntax

Get [**#**]*filenumber*, [*recnumber*], *varname*

The **Get** statement syntax has these parts:

Part	Description
filenumber	Required. Any valid file number.
recnumber	Optional. **Variant** (**Long**). Record number (**Random** mode files) or byte number (**Binary** mode files) at which reading begins.
varname	Required. Valid variable name into which data is read.

Remarks

Data read with **Get** is usually written to a file with **Put**.

The first record or byte in a file is at position 1, the second record or byte is at position 2, and so on. If you omit *recnumber*, the next record or byte following the last **Get** or **Put** statement (or pointed to by the last **Seek** function) is read. You must include delimiting commas, for example:

```
Get #4,,FileBuffer
```

For files opened in **Random** mode, the following rules apply:

- If the length of the data being read is less than the length specified in the **Len** clause of the **Open** statement, **Get** reads subsequent records on record-length boundaries. The space between the end of one record and the beginning of the next record is padded with the existing contents of the file buffer. Because the amount of padding data can't be determined with any certainty, it is generally a good idea to have the record length match the length of the data being read.
- If the variable being read into is a variable-length string, **Get** reads a 2-byte descriptor containing the string length and then reads the data that goes into the variable. Therefore, the record length specified by the **Len** clause in the **Open** statement must be at least 2 bytes greater than the actual length of the string.
- If the variable being read into is a **Variant** of numeric type, **Get** reads 2 bytes identifying the **VarType** of the **Variant** and then the data that goes into the variable. For example, when reading a **Variant** of **VarType** 3, **Get** reads 6 bytes: 2 bytes identifying the **Variant** as **VarType** 3 (**Long**) and 4 bytes containing the **Long** data. The record length specified by the **Len** clause in the **Open** statement must be at least 2 bytes greater than the actual number of bytes required to store the variable.

  Note You can use the **Get** statement to read a **Variant** array from disk, but you can't use **Get** to read a scalar **Variant** containing an array. You also can't use **Get** to read objects from disk.
- If the variable being read into is a **Variant** of **VarType** 8 (**String**), **Get** reads 2 bytes identifying the **VarType**, 2 bytes indicating the length of the string, and then reads the string data. The record length specified by the **Len** clause in the **Open** statement must be at least 4 bytes greater than the actual length of the string.
- If the variable being read into is a dynamic array, **Get** reads a descriptor whose length equals 2 plus 8 times the number of dimensions, that is, 2 + 8 * *NumberOfDimensions*. The record length specified by the **Len** clause in the **Open** statement must be greater than or equal to the sum of all the bytes required to read the array data and the array descriptor. For example, the following array declaration requires 118 bytes when the array is written to disk.

```
Dim MyArray(1 To 5,1 To 10) As Integer
```

The 118 bytes are distributed as follows: 18 bytes for the descriptor (2 + 8 * 2), and 100 bytes for the data (5 * 10 * 2).

- If the variable being read into is a fixed-size array, **Get** reads only the data. No descriptor is read.
- If the variable being read into is any other type of variable (not a variable-length string or a **Variant**), **Get** reads only the variable data. The record length specified by the **Len** clause in the **Open** statement must be greater than or equal to the length of the data being read.
- **Get** reads elements of user-defined types as if each were being read individually, except that there is no padding between elements. On disk, a dynamic array in a user-defined type (written with **Put**) is prefixed by a descriptor whose length equals 2 plus 8 times the number of dimensions, that is, 2 + 8 * *NumberOfDimensions*. The record length specified by the **Len** clause in the **Open** statement must be greater than or equal to the sum of all the bytes required to read the individual elements, including any arrays and their descriptors.

For files opened in **Binary** mode, all of the **Random** rules apply, except:

- The **Len** clause in the **Open** statement has no effect. **Get** reads all variables from disk contiguously; that is, with no padding between records.
- For any array other than an array in a user-defined type, **Get** reads only the data. No descriptor is read.
- **Get** reads variable-length strings that aren't elements of user-defined types without expecting the 2-byte length descriptor. The number of bytes read equals the number of characters already in the string. For example, the following statements read 10 bytes from file number 1:

```
VarString = String(10," ")
Get #1,,VarString
```

Input # Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmInputC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmInputX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmInputS"}

Reads data from an open sequential file and assigns the data to variables.

Syntax

Input #*filenumber, varlist*

The **Input #** statement syntax has these parts:

Part	Description
filenumber	Required. Any valid file number.
varlist	Required. Comma-delimited list of variables that are assigned values read from the file—can't be an array or object variable. However, variables that describe an element of an array or user-defined type may be used.

Remarks

Data read with **Input #** is usually written to a file with **Write #**. Use this statement only with files opened in **Input** or **Binary** mode.

When read, standard string or numeric data is assigned to variables without modification. The following table illustrates how other input data is treated:

Data	Value assigned to variable
Delimiting comma or blank line	**Empty**
#NULL#	**Null**
#TRUE# or #FALSE#	**True** or **False**
#*yyyy-mm-dd hh:mm:ss*#	The date and/or time represented by the expression
#ERROR *errornumber*#	*errornumber* (variable is a **Variant** tagged as an error)

Double quotation marks (`" "`) within input data are ignored.

Data items in a file must appear in the same order as the variables in *varlist* and match variables of the same data type. If a variable is numeric and the data is not numeric, a value of zero is assigned to the variable.

If you reach the end of the file while you are inputting a data item, the input is terminated and an error occurs.

Note To be able to correctly read data from a file into variables using **Input #**, use the **Write #** statement instead of the **Print #** statement to write the data to the files. Using **Write #** ensures each separate data field is properly delimited.

Input Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctInputC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctInputX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctInputS"}

Returns **String** containing characters from a file opened in **Input** or **Binary** mode.

Syntax

Input(*number*, [**#**]*filenumber*)

The **Input** function syntax has these parts:

Part	Description
number	Required. Any valid numeric expression specifying the number of characters to return.
filenumber	Required. Any valid file number.

Remarks

Data read with the **Input** function is usually written to a file with **Print #** or **Put**. Use this function only with files opened in **Input** or **Binary** mode.

Unlike the **Input #** statement, the **Input** function returns all of the characters it reads, including commas, carriage returns, linefeeds, quotation marks, and leading spaces.

With files opened for **Binary** access, an attempt to read through the file using the **Input** function until **EOF** returns **True** generates an error. Use the **LOF** and **Loc** functions instead of **EOF** when reading binary files with **Input**, or use **Get** when using the **EOF** function.

Note Use the **InputB** function for byte data contained within text files. With **InputB**, *number* specifies the number of bytes to return rather than the number of characters to return.

Line Input # Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmLineInputC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmLineInputX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmLineInputS"}

Reads a single line from an open sequential file and assigns it to a **String** variable.

Syntax

Line Input #*filenumber*, *varname*

The **Line Input #** statement syntax has these parts:

Part	Description
filenumber	Required. Any valid file number.
varname	Required. Valid **Variant** or **String** variable name.

Remarks

Data read with **Line Input #** is usually written from a file with **Print #**.

The **Line Input #** statement reads from a file one character at a time until it encounters a carriage return (**Chr(**13**)**) or carriage return–linefeed (**Chr(**13**)** + **Chr(**10**)**) sequence. Carriage return–linefeed sequences are skipped rather than appended to the character string.

Loc Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctLocC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctLocX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctLocS"}

Returns a **Long** specifying the current read/write position within an open file.

Syntax

Loc(*filenumber***)**

The required *filenumber* argument is any valid **Integer** file number.

Remarks

The following describes the return value for each file access mode:

Mode	Return Value
Random	Number of the last record read from or written to the file.
Sequential	Current byte position in the file divided by 128. However, information returned by **Loc** for sequential files is neither used nor required.
Binary	Position of the last byte read or written.

Lock, Unlock Statements

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmLockC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmLockX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmLockS"}

Controls access by other processes to all or part of a file opened using the **Open** statement.

Syntax

Lock [**#**]*filenumber*[, *recordrange*]
. . .
Unlock [**#**]*filenumber*[, *recordrange*]

The **Lock** and **Unlock** statement syntax has these parts:

Part	Description
filenumber	Required. Any valid file number.
recordrange	Optional. The range of records to lock or unlock.

Settings

The *recordrange* argument settings are:

recnumber | [*start*] **To** *end*

Setting	Description
recnumber	Record number (**Random** mode files) or byte number (**Binary** mode files) at which locking or unlocking begins.
start	Number of the first record or byte to lock or unlock.
end	Number of the last record or byte to lock or unlock.

Remarks

The **Lock** and **Unlock** statements are used in environments where several processes might need access to the same file.

Lock and **Unlock** statements are always used in pairs. The arguments to **Lock** and **Unlock** must match exactly.

The first record or byte in a file is at position 1, the second record or byte is at position 2, and so on. If you specify just one record, then only that record is locked or unlocked. If you specify a range of records and omit a starting record (*start*), all records from the first record to the end of the range (*end*) are locked or unlocked. Using **Lock** without *recnumber* locks the entire file; using **Unlock** without *recnumber* unlocks the entire file.

If the file has been opened for sequential input or output, **Lock** and **Unlock** affect the entire file, regardless of the range specified by *start* and *end*.

Caution Be sure to remove all locks with an **Unlock** statement before closing a file or quitting your program. Failure to remove locks produces unpredictable results.

LOF Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctLOFC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctLOFX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctLOFS"}

Returns a **Long** representing the size, in bytes, of a file opened using the **Open** statement.

Syntax

LOF(*filenumber***)**

The required *filenumber* argument is an **Integer** containing a valid file number.

Note Use the **FileLen** function to obtain the length of a file that is not open.

Open Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmOpenC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmOpenX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmOpenS"}

Enables input/output (I/O) to a file.

Syntax

Open *pathname* **For** *mode* [**Access** *access*] [*lock*] **As** [**#**]*filenumber* [**Len**=*reclength*]

The **Open** statement syntax has these parts:

Part	Description
pathname	Required. String expression that specifies a file name—may include directory or folder, and drive.
mode	Required. Keyword specifying the file mode: **Append**, **Binary**, **Input**, **Output**, or **Random**. If unspecified, the file is opened for **Random** access.
access	Optional. Keyword specifying the operations permitted on the open file: **Read**, **Write**, or **Read Write**.
lock	Optional. Keyword specifying the operations permitted on the open file by other processes: **Shared**, **Lock Read**, **Lock Write**, and **Lock Read Write**.
filenumber	Required. A valid file number in the range 1 to 511, inclusive. Use the **FreeFile** function to obtain the next available file number.
reclength	Optional. Number less than or equal to 32,767 (bytes). For files opened for random access, this value is the record length. For sequential files, this value is the number of characters buffered.

Remarks

You must open a file before any I/O operation can be performed on it. **Open** allocates a buffer for I/O to the file and determines the mode of access to use with the buffer.

If the file specified by *pathname* doesn't exist, it is created when a file is opened for **Append**, **Binary**, **Output**, or **Random** modes.

If the file is already opened by another process and the specified type of access is not allowed, the **Open** operation fails and an error occurs.

The **Len** clause is ignored if *mode* is **Binary**.

Important In **Binary**, **Input**, and **Random** modes, you can open a file using a different file number without first closing the file. In **Append** and **Output** modes, you must close a file before opening it with a different file number.

Print # Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmPrintC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmPrintX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmPrintS"}

Writes display-formatted data to a sequential file.

Syntax

Print #*filenumber*, [*outputlist*]

The **Print #** statement syntax has these parts:

Part	Description
filenumber	Required. Any valid file number.
outputlist	Optional. Expression or list of expressions to print.

Settings

The *outputlist* argument settings are:

[{**Spc(***n***)** | **Tab**[**(***n***)**]}] [*expression*] [*charpos*]

Setting	Description
Spc(*n***)**	Used to insert space characters in the output, where *n* is the number of space characters to insert.
Tab(*n***)**	Used to position the insertion point to an absolute column number, where *n* is the column number. Use **Tab** with no argument to position the insertion point at the beginning of the next print zone.
expression	Numeric expressions or string expressions to print.
charpos	Specifies the insertion point for the next character. Use a semicolon to position the insertion point immediately after the last character displayed. Use **Tab(***n***)** to position the insertion point to an absolute column number. Use **Tab** with no argument to position the insertion point at the beginning of the next print zone. If *charpos* is omitted, the next character is printed on the next line.

Remarks

Data written with **Print #** is usually read from a file with **Line Input #** or **Input**.

If you omit *outputlist* and include only a list separator after *filenumber*, a blank line is printed to the file. Multiple expressions can be separated with either a space or a semicolon. A space has the same effect as a semicolon.

For **Boolean** data, either `True` or `False` is printed. The **True** and **False** keywords are not translated, regardless of the locale.

Date data is written to the file using the standard short date format recognized by your system. When either the date or the time component is missing or zero, only the part provided gets written to the file.

Nothing is written to the file if *outputlist* data is **Empty**. However, if *outputlist* data is **Null**, **Null** is written to the file.

For **Error** data, the output appears as `Error errorcode`. The **Error** keyword is not translated regardless of the locale.

All data written to the file using **Print #** is internationally aware; that is, the data is properly formatted using the appropriate decimal separator.

Because **Print #** writes an image of the data to the file, you must delimit the data so it prints correctly. If you use **Tab** with no arguments to move the print position to the next print zone, **Print #** also writes the spaces between print fields to the file.

Note If, at some future time, you want to read the data from a file using the **Input #** statement, use the **Write #** statement instead of the **Print #** statement to write the data to the file. Using **Write #** ensures the integrity of each separate data field by properly delimiting it, so it can be read back in using **Input #**. Using **Write #** also ensures it can be correctly read in any locale.

Put Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmPutC"}
{ewc HLP95EN.DLL,DYNALINK,"Example":"vastmPutX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmPutS"}

Writes data from a variable to a disk file.

Syntax

Put [**#**]*filenumber*, [*recnumber*], *varname*

The **Put** statement syntax has these parts:

Part	Description
filenumber	Required. Any valid file number.
recnumber	Optional. **Variant** (**Long**). Record number (**Random** mode files) or byte number (**Binary** mode files) at which writing begins.
varname	Required. Name of variable containing data to be written to disk.

Remarks

Data written with **Put** is usually read from a file with **Get**.

The first record or byte in a file is at position 1, the second record or byte is at position 2, and so on. If you omit *recnumber*, the next record or byte after the last **Get** or **Put** statement or pointed to by the last **Seek** function is written. You must include delimiting commas, for example:

```
Put #4,,FileBuffer
```

For files opened in **Random** mode, the following rules apply:

- If the length of the data being written is less than the length specified in the **Len** clause of the **Open** statement, **Put** writes subsequent records on record-length boundaries. The space between the end of one record and the beginning of the next record is padded with the existing contents of the file buffer. Because the amount of padding data can't be determined with any certainty, it is generally a good idea to have the record length match the length of the data being written. If the length of the data being written is greater than the length specified in the **Len** clause of the **Open** statement, an error occurs.
- If the variable being written is a variable-length string, **Put** writes a 2-byte descriptor containing the string length and then the variable. The record length specified by the **Len** clause in the **Open** statement must be at least 2 bytes greater than the actual length of the string.
- If the variable being written is a **Variant** of a numeric type, **Put** writes 2 bytes identifying the **VarType** of the **Variant** and then writes the variable. For example, when writing a **Variant** of **VarType** 3, **Put** writes 6 bytes: 2 bytes identifying the **Variant** as **VarType** 3 (**Long**) and 4 bytes containing the **Long** data. The record length specified by the **Len** clause in the **Open** statement must be at least 2 bytes greater than the actual number of bytes required to store the variable.

Note You can use the **Put** statement to write a **Variant** array to disk, but you can't use **Put** to write a scalar **Variant** containing an array to disk. You also can't use **Put** to write objects to disk.

- If the variable being written is a **Variant** of **VarType** 8 (**String**), **Put** writes 2 bytes identifying the **VarType**, 2 bytes indicating the length of the string, and then writes the string data. The record length specified by the **Len** clause in the **Open** statement must be at least 4 bytes greater than the actual length of the string.
- If the variable being written is a dynamic array, **Put** writes a descriptor whose length equals 2 plus 8 times the number of dimensions, that is, 2 + 8 * *NumberOfDimensions*. The record length specified by the **Len** clause in the **Open** statement must be greater than or equal to the sum of all

the bytes required to write the array data and the array descriptor. For example, the following array declaration requires 118 bytes when the array is written to disk.

```
Dim MyArray(1 To 5,1 To 10) As Integer
```

- The 118 bytes are distributed as follows: 18 bytes for the descriptor (2 + 8 * 2), and 100 bytes for the data (5 * 10 * 2).
- If the variable being written is a fixed-size array, **Put** writes only the data. No descriptor is written to disk.
- If the variable being written is any other type of variable (not a variable-length string or a **Variant**), **Put** writes only the variable data. The record length specified by the **Len** clause in the **Open** statement must be greater than or equal to the length of the data being written.
- **Put** writes elements of user-defined types as if each were written individually, except there is no padding between elements. On disk, a dynamic array in a user-defined type written with **Put** is prefixed by a descriptor whose length equals 2 plus 8 times the number of dimensions, that is, 2 + 8 * *NumberOfDimensions*. The record length specified by the **Len** clause in the **Open** statement must be greater than or equal to the sum of all the bytes required to write the individual elements, including any arrays and their descriptors.

For files opened in **Binary** mode, all of the **Random** rules apply, except:

- The **Len** clause in the **Open** statement has no effect. **Put** writes all variables to disk contiguously; that is, with no padding between records.
- For any array other than an array in a user-defined type, **Put** writes only the data. No descriptor is written.
- **Put** writes variable-length strings that are not elements of user-defined types without the 2-byte length descriptor. The number of bytes written equals the number of characters in the string. For example, the following statements write 10 bytes to file number 1:

```
VarString$ = String$(10," ")
Put #1,,VarString$
```

Reset Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmResetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmResetX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmResetS"}

Closes all disk files opened using the **Open** statement.

Syntax

Reset

Remarks

The **Reset** statement closes all active files opened by the **Open** statement and writes the contents of all file buffers to disk.

Seek Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctSeekC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctSeekX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctSeekS"}

Returns a **Long** specifying the current read/write position within a file opened using the **Open** statement.

Syntax

Seek(*filenumber***)**

The required *filenumber* argument is an **Integer** containing a valid file number.

Remarks

Seek returns a value between 1 and 2,147,483,647 (equivalent to 2^31 – 1), inclusive.

The following describes the return values for each file access mode.

Mode	Return Value
Random	Number of the next record read or written
Binary, **Output**, **Append**, **Input**	Byte position at which the next operation takes place. The first byte in a file is at position 1, the second byte is at position 2, and so on.

Seek Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmSeekC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmSeekX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmSeekS"}

Sets the position for the next read/write operation within a file opened using the **Open** statement.

Syntax

Seek [**#**]*filenumber*, *position*

The **Seek** statement syntax has these parts:

Part	Description
filenumber	Required. Any valid file number.
position	Required. Number in the range 1 – 2,147,483,647, inclusive, that indicates where the next read/write operation should occur.

Remarks

Record numbers specified in **Get** and **Put** statements override file positioning performed by **Seek**.

Performing a file-write operation after a **Seek** operation beyond the end of a file extends the file. If you attempt a **Seek** operation to a negative or zero position, an error occurs.

Spc Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctSpcC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctSpcX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctSpcS"}

Used with the **Print #** statement or the **Print** method to position output.

Syntax

Spc(*n***)**

The required *n* argument is the number of spaces to insert before displaying or printing the next expression in a list.

Remarks

If *n* is less than the output line width, the next print position immediately follows the number of spaces printed. If *n* is greater than the output line width, **Spc** calculates the next print position using the formula:

currentprintposition + (*n* **Mod** *width*)

For example, if the current print position is 24, the output line width is 80, and you specify **Spc(**90**)**, the next print will start at position 34 (current print position + the remainder of 90/80). If the difference between the current print position and the output line width is less than *n* (or *n* **Mod** *width*), the **Spc** function skips to the beginning of the next line and generates spaces equal to *n* – (*width* – *currentprintposition*).

Note Make sure your tabular columns are wide enough to accommodate wide letters.

When you use the **Print** method with a proportionally spaced font, the width of space characters printed using the **Spc** function is always an average of the width of all characters in the point size for the chosen font. However, there is no correlation between the number of characters printed and the number of fixed-width columns those characters occupy. For example, the uppercase letter W occupies more than one fixed-width column and the lowercase letter i occupies less than one fixed-width column.

Tab Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctTabC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctTabX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctTabS"}

Used with the **Print #** statement or the **Print** method to position output.

Syntax

Tab[(*n***)]**

The optional *n* argument is the column number moved to before displaying or printing the next expression in a list. If omitted, **Tab** moves the insertion point to the beginning of the next print zone. This allows **Tab** to be used instead of a comma in locales where the comma is used as a decimal separator.

Remarks

If the current print position on the current line is greater than *n*, **Tab** skips to the *n*th column on the next output line. If *n* is less than 1, **Tab** moves the print position to column 1. If *n* is greater than the output line width, **Tab** calculates the next print position using the formula:

n **Mod** *width*

For example, if *width* is 80 and you specify **Tab(**90**)**, the next print will start at column 10 (the remainder of 90/80). If *n* is less than the current print position, printing begins on the next line at the calculated print position. If the calculated print position is greater than the current print position, printing begins at the calculated print position on the same line.

The leftmost print position on an output line is always 1. When you use the **Print #** statement to print to files, the rightmost print position is the current width of the output file, which you can set using the **Width #** statement.

Note Make sure your tabular columns are wide enough to accommodate wide letters.

When you use the **Tab** function with the **Print** method, the print surface is divided into uniform, fixed-width columns. The width of each column is an average of the width of all characters in the point size for the chosen font. However, there is no correlation between the number of characters printed and the number of fixed-width columns those characters occupy. For example, the uppercase letter W occupies more than one fixed-width column and the lowercase letter i occupies less than one fixed-width column.

Width # Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmWidthC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmWidthX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmWidthS"}

Assigns an output line width to a file opened using the **Open** statement.

Syntax

Width #*filenumber*, *width*

The **Width #** statement syntax has these parts:

Part	Description
filenumber	Required. Any valid file number.
width	Required. Numeric expression in the range 0–255, inclusive, that indicates how many characters appear on a line before a new line is started. If *width* equals 0, there is no limit to the length of a line. The default value for *width* is 0.

Write # Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmWriteC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmWriteX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmWriteS"}

Writes data to a sequential file.

Syntax

Write #*filenumber*, [*outputlist*]

The **Write #** statement syntax has these parts:

Part	Description
filenumber	Required. Any valid file number.
outputlist	Optional. One or more comma-delimited numeric expressions or string expressions to write to a file.

Remarks

Data written with **Write #** is usually read from a file with **Input #**.

If you omit *outputlist* and include a comma after *filenumber*, a blank line is printed to the file. Multiple expressions can be separated with a space, a semicolon, or a comma. A space has the same effect as a semicolon.

When **Write #** is used to write data to a file, several universal assumptions are followed so the data can always be read and correctly interpreted using **Input #**, regardless of locale:

- Numeric data is always written using the period as the decimal separator.
- For **Boolean** data, either `#TRUE#` or `#FALSE#` is printed. The **True** and **False** keywords are not translated, regardless of locale.
- **Date** data is written to the file using the universal date format. When either the date or the time component is missing or zero, only the part provided gets written to the file.
- Nothing is written to the file if *outputlist* data is **Empty**. However, for **Null** data, `#NULL#` is written.
- If *outputlist* data is **Null** data, `#NULL#` is written to the file.
- For **Error** data, the output appears as `#ERROR errorcode#`. The **Error** keyword is not translated, regardless of locale.

Unlike the **Print #** statement, the **Write #** statement inserts commas between items and quotation marks around strings as they are written to the file. You don't have to put explicit delimiters in the list. **Write #** inserts a newline character, that is, a carriage return–linefeed (**Chr(**13**)** + **Chr(**10**)**), after it has written the final character in *outputlist* to the file.

DDB Function Example

This example uses the **DDB** function to return the depreciation of an asset for a specified period given the initial cost (`InitCost`), the salvage value at the end of the asset's useful life (`SalvageVal`), the total life of the asset in years (`LifeTime`), and the period in years for which the depreciation is calculated (`Depr`).

```
Dim Fmt, InitCost, SalvageVal, MonthLife, LifeTime, DepYear, Depr
Const YRMOS = 12  ' Number of months in a year.
Fmt = "###,##0.00"
InitCost = InputBox("What's the initial cost of the asset?")
SalvageVal = InputBox("Enter the asset's value at end of its life.")
MonthLife = InputBox("What's the asset's useful life in months?")
Do While MonthLife < YRMOS ' Ensure period is >= 1 year.
   MsgBox "Asset life must be a year or more."
   MonthLife = InputBox("What's the asset's useful life in months?")
Loop
LifeTime = MonthLife / YRMOS ' Convert months to years.
If LifeTime <> Int(MonthLife / YRMOS) Then
   LifeTime = Int(LifeTime + 1)    ' Round up to nearest year.
End If
DepYear = CInt(InputBox("Enter year for depreciation calculation."))
Do While DepYear < 1 Or DepYear > LifeTime
   MsgBox "You must enter at least 1 but not more than " & LifeTime
   DepYear = InputBox("Enter year for depreciation calculation.")
Loop
Depr = DDB(InitCost, SalvageVal, LifeTime, DepYear)
MsgBox "The depreciation for year " & DepYear & " is " & _
Format(Depr, Fmt) & "."
```

FV Function Example

This example uses the **FV** function to return the future value of an investment given the percentage rate that accrues per period (`APR / 12`), the total number of payments (`TotPmts`), the payment (`Payment`), the current value of the investment (`PVal`), and a number that indicates whether the payment is made at the beginning or end of the payment period (`PayType`). Note that because `Payment` represents cash paid out, it's a negative number.

```
Dim Fmt, Payment, APR, TotPmts, PayType, PVal, FVal
Const ENDPERIOD = 0, BEGINPERIOD = 1     ' When payments are made.
Fmt = "###,###,##0.00"  ' Define money format.
Payment = InputBox("How much do you plan to save each month?")
APR = InputBox("Enter the expected interest annual percentage rate.")
If APR > 1 Then APR = APR / 100    ' Ensure proper form.
TotPmts = InputBox("For how many months do you expect to save?")
PayType = MsgBox("Do you make payments at the end of month?", vbYesNo)
If PayType = vbNo Then PayType = BEGINPERIOD Else PayType = ENDPERIOD
PVal = InputBox("How much is in this savings account now?")
FVal = FV(APR / 12, TotPmts, -Payment, -PVal, PayType)
MsgBox "Your savings will be worth " & Format(FVal, Fmt) & "."
```

IPmt Function Example

This example uses the **IPmt** function to calculate how much of a payment is interest when all the payments are of equal value. Given are the interest percentage rate per period (`APR / 12`), the payment period for which the interest portion is desired (`Period`), the total number of payments (`TotPmts`), the present value or principal of the loan (`PVal`), the future value of the loan (`FVal`), and a number that indicates whether the payment is due at the beginning or end of the payment period (`PayType`).

```
Dim FVal, Fmt, PVal, APR, TotPmts, PayType, Period, IntPmt, TotInt, Msg
Const ENDPERIOD = 0, BEGINPERIOD = 1     ' When payments are made.
FVal = 0 ' Usually 0 for a loan.
Fmt = "###,###,##0.00"  ' Define money format.
PVal = InputBox("How much do you want to borrow?")
APR = InputBox("What is the annual percentage rate of your loan?")
If APR > 1 Then APR = APR / 100    ' Ensure proper form.
TotPmts = InputBox("How many monthly payments?")
PayType = MsgBox("Do you make payments at end of the month?", vbYesNo)
If PayType = vbNo Then PayType = BEGINPERIOD Else PayType = ENDPERIOD
For Period = 1 To TotPmts  ' Total all interest.
   IntPmt = IPmt(APR / 12, Period, TotPmts, -PVal, FVal, PayType)
   TotInt = TotInt + IntPmt
Next Period
Msg = "You'll pay a total of " & Format(TotInt, Fmt)
Msg = Msg & " in interest for this loan."
MsgBox Msg  ' Display results.
```

IRR Function Example

In this example, the **IRR** function returns the internal rate of return for a series of 5 cash flows contained in the array `Values()`. The first array element is a negative cash flow representing business start-up costs. The remaining 4 cash flows represent positive cash flows for the subsequent 4 years. `Guess` is the estimated internal rate of return.

```
Dim Guess, Fmt, RetRate, Msg
Static Values(5) As Double ' Set up array.
Guess = .1  ' Guess starts at 10 percent.
Fmt = "#0.00"  ' Define percentage format.
Values(0) = -70000' Business start-up costs.
' Positive cash flows reflecting income for four successive years.
Values(1) = 22000 : Values(2) = 25000
Values(3) = 28000 : Values(4) = 31000
RetRate = IRR(Values(), Guess) * 100     ' Calculate internal rate.
Msg = "The internal rate of return for these five cash flows is "
Msg = Msg & Format(RetRate, Fmt) & " percent."
MsgBox Msg  ' Display internal return rate.
```

MIRR Function Example

This example uses the **MIRR** function to return the modified internal rate of return for a series of cash flows contained in the array `Values()`. `LoanAPR` represents the financing interest, and `InvAPR` represents the interest rate received on reinvestment.

```
Dim LoanAPR, InvAPR, Fmt, RetRate, Msg
Static Values(5) As Double ' Set up array.
LoanAPR = .1' Loan rate.
InvAPR = .12' Reinvestment rate.
Fmt = "#0.00"  ' Define money format.
Values(0) = -70000' Business start-up costs.
' Positive cash flows reflecting income for four successive years.
Values(1) = 22000 : Values(2) = 25000
Values(3) = 28000 : Values(4) = 31000
RetRate = MIRR(Values(), LoanAPR, InvAPR)' Calculate internal rate.
Msg = "The modified internal rate of return for these five cash flows is"
Msg = Msg & Format(Abs(RetRate) * 100, Fmt) & "%."
MsgBox Msg  ' Display internal return
      ' rate.
```

NPer Function Example

This example uses the **NPer** function to return the number of periods during which payments must be made to pay off a loan whose value is contained in `PVal`. Also provided are the interest percentage rate per period (`APR / 12`), the payment (`Payment`), the future value of the loan (`FVal`), and a number that indicates whether the payment is due at the beginning or end of the payment period (`PayType`).

```
Dim FVal, PVal, APR, Payment, PayType, TotPmts
Const ENDPERIOD = 0, BEGINPERIOD = 1     ' When payments are made.
FVal = 0 ' Usually 0 for a loan.
PVal = InputBox("How much do you want to borrow?")
APR = InputBox("What is the annual percentage rate of your loan?")
If APR > 1 Then APR = APR / 100    ' Ensure proper form.
Payment = InputBox("How much do you want to pay each month?")
PayType = MsgBox("Do you make payments at the end of month?", vbYesNo)
If PayType = vbNo Then PayType = BEGINPERIOD Else PayType = ENDPERIOD
TotPmts = NPer(APR / 12, -Payment, PVal, FVal, PayType)
If Int(TotPmts) <> TotPmts Then TotPmts = Int(TotPmts) + 1
MsgBox "It will take you " & TotPmts & " months to pay off your loan."
```

NPV Function Example

This example uses the **NPV** function to return the net present value for a series of cash flows contained in the array `Values()`. `RetRate` represents the fixed internal rate of return.

```
Dim Fmt, Guess, RetRate, NetPVal, Msg
Static Values(5) As Double ' Set up array.
Fmt = "###,##0.00"' Define money format.
Guess = .1  ' Guess starts at 10 percent.
RetRate = .0625' Set fixed internal rate.
Values(0) = -70000' Business start-up costs.
' Positive cash flows reflecting income for four successive years.
Values(1) = 22000 : Values(2) = 25000
Values(3) = 28000 : Values(4) = 31000
NetPVal = NPV(RetRate, Values())   ' Calculate net present value.
Msg = "The net present value of these cash flows is "
Msg = Msg & Format(NetPVal, Fmt) & "."
MsgBox Msg  ' Display net present value.
```

Pmt Function Example

This example uses the **Pmt** function to return the monthly payment for a loan over a fixed period. Given are the interest percentage rate per period (`APR / 12`), the total number of payments (`TotPmts`), the present value or principal of the loan (`PVal`), the future value of the loan (`FVal`), and a number that indicates whether the payment is due at the beginning or end of the payment period (`PayType`).

```
Dim Fmt, FVal, PVal, APR, TotPmts, PayType, Payment
Const ENDPERIOD = 0, BEGINPERIOD = 1     ' When payments are made.
Fmt = "###,###,##0.00"  ' Define money format.
FVal = 0 ' Usually 0 for a loan.
PVal = InputBox("How much do you want to borrow?")
APR = InputBox("What is the annual percentage rate of your loan?")
If APR > 1 Then APR = APR / 100    ' Ensure proper form.
TotPmts = InputBox("How many monthly payments will you make?")
PayType = MsgBox("Do you make payments at the end of month?", vbYesNo)
If PayType = vbNo Then PayType = BEGINPERIOD Else PayType = ENDPERIOD
Payment = Pmt(APR / 12, TotPmts, -PVal, FVal, PayType)
MsgBox "Your payment will be " & Format(Payment, Fmt) & " per month."
```

PPmt Function Example

This example uses the **PPmt** function to calculate how much of a payment for a specific period is principal when all the payments are of equal value. Given are the interest percentage rate per period (`APR / 12`), the payment period for which the principal portion is desired (`Period`), the total number of payments (`TotPmts`), the present value or principal of the loan (`PVal`), the future value of the loan (`FVal`), and a number that indicates whether the payment is due at the beginning or end of the payment period (`PayType`).

```
Dim NL, TB, Fmt, FVal, PVal, APR, TotPmts, PayType, Payment, Msg,
MakeChart, Period, P, I
Const ENDPERIOD = 0, BEGINPERIOD = 1     ' When payments are made.
NL = Chr(13) & Chr(10)  ' Define newline.
TB = Chr(9) ' Define tab.
Fmt = "###,###,##0.00"  ' Define money format.
FVal = 0 ' Usually 0 for a loan.
PVal = InputBox("How much do you want to borrow?")
APR = InputBox("What is the annual percentage rate of your loan?")
If APR > 1 Then APR = APR / 100    ' Ensure proper form.
TotPmts = InputBox("How many monthly payments do you have to make?")
PayType = MsgBox("Do you make payments at the end of month?", vbYesNo)
If PayType = vbNo Then PayType = BEGINPERIOD Else PayType = ENDPERIOD
Payment = Abs(-Pmt(APR / 12, TotPmts, PVal, FVal, PayType))
Msg = "Your monthly payment is " & Format(Payment, Fmt) & ". "
Msg = Msg & "Would you like a breakdown of your principal and "
Msg = Msg & "interest per period?"
MakeChart = MsgBox(Msg, vbYesNo)   ' See if chart is desired.
If MakeChart <> vbNo Then
   If TotPmts > 12 Then MsgBox "Only first year will be shown."
   Msg = "Month  Payment  Principal  Interest" & NL
   For Period = 1 To TotPmts
      If Period > 12 Then Exit For ' Show only first 12.
      P = PPmt(APR / 12, Period, TotPmts, -PVal, FVal, PayType)
      P = (Int((P + .005) * 100) / 100)  ' Round principal.
      I = Payment - P
      I = (Int((I + .005) * 100) / 100)  ' Round interest.
      Msg = Msg & Period & TB & Format(Payment, Fmt)
      Msg = Msg & TB & Format(P, Fmt) & TB & Format(I, Fmt) & NL
   Next Period
   MsgBox Msg  ' Display amortization table.
End If
```

PV Function Example

In this example, the **PV** function returns the present value of an $1,000,000 annuity that will provide $50,000 a year for the next 20 years. Provided are the expected annual percentage rate (`APR`), the total number of payments (`TotPmts`), the amount of each payment (`YrIncome`), the total future value of the investment (`FVal`), and a number that indicates whether each payment is made at the beginning or end of the payment period (`PayType`). Note that `YrIncome` is a negative number because it represents cash paid out from the annuity each year.

```
Dim Fmt, APR, TotPmts, YrIncome, FVal, PayType, PVal
Const ENDPERIOD = 0, BEGINPERIOD = 1     ' When payments are made.
Fmt = "###,##0.00"' Define money format.
APR = .0825 ' Annual percentage rate.
TotPmts = 20' Total number of payments.
YrIncome = 50000  ' Yearly income.
FVal = 1000000 ' Future value.
PayType = BEGINPERIOD' Payment at beginning of month.
PVal = PV(APR, TotPmts, -YrIncome, FVal, PayType)
MsgBox "The present value is " & Format(PVal, Fmt) & "."
```

Rate Function Example

This example uses the **Rate** function to calculate the interest rate of a loan given the total number of payments (`TotPmts`), the amount of the loan payment (`Payment`), the present value or principal of the loan (`PVal`), the future value of the loan (`FVal`), a number that indicates whether the payment is due at the beginning or end of the payment period (`PayType`), and an approximation of the expected interest rate (`Guess`).

```
Dim Fmt, FVal, Guess, PVal, Payment, TotPmts, PayType, APR
Const ENDPERIOD = 0, BEGINPERIOD = 1     ' When payments are made.
Fmt = "##0.00" ' Define percentage format.
FVal = 0 ' Usually 0 for a loan.
Guess = .1  ' Guess of 10 percent.
PVal = InputBox("How much did you borrow?")
Payment = InputBox("What's your monthly payment?")
TotPmts = InputBox("How many monthly payments do you have to make?")
PayType = MsgBox("Do you make payments at the end of the month?", _
vbYesNo)
If PayType = vbNo Then PayType = BEGINPERIOD Else PayType = ENDPERIOD
APR = (Rate(TotPmts, -Payment, PVal, FVal, PayType, Guess) * 12) * 100
MsgBox "Your interest rate is " & Format(CInt(APR), Fmt) & " percent."
```

SLN Function Example

This example uses the **SLN** function to return the straight-line depreciation of an asset for a single period given the asset's initial cost (`InitCost`), the salvage value at the end of the asset's useful life (`SalvageVal`), and the total life of the asset in years (`LifeTime`).

```
Dim Fmt, InitCost, SalvageVal, MonthLife, LifeTime, PDepr
Const YEARMONTHS = 12' Number of months in a year.
Fmt = "###,##0.00"' Define money format.
InitCost = InputBox("What's the initial cost of the asset?")
SalvageVal = InputBox("What's the asset's value at the end of its useful
life?")
MonthLife = InputBox("What's the asset's useful life in months?")
Do While MonthLife < YEARMONTHS    ' Ensure period is >= 1 year.
   MsgBox "Asset life must be a year or more."
   MonthLife = InputBox("What's the asset's useful life in months?")
Loop
LifeTime = MonthLife / YEARMONTHS  ' Convert months to years.
If LifeTime <> Int(MonthLife / YEARMONTHS) Then
   LifeTime = Int(LifeTime + 1)    ' Round up to nearest year.
End If
PDepr = SLN(InitCost, SalvageVal, LifeTime)
MsgBox "The depreciation is " & Format(PDepr, Fmt) & " per year."
```

SYD Function Example

This example uses the **SYD** function to return the depreciation of an asset for a specified period given the asset's initial cost (`InitCost`), the salvage value at the end of the asset's useful life (`SalvageVal`), and the total life of the asset in years (`LifeTime`). The period in years for which the depreciation is calculated is `PDepr`.

```
Dim Fmt, InitCost, SalvageVal, MonthLife, LifeTime, DepYear, PDepr
Const YEARMONTHS = 12' Number of months in a year.
Fmt = "###,##0.00"' Define money format.
InitCost = InputBox("What's the initial cost of the asset?")
SalvageVal = InputBox("What's the asset's value at the end of its life?")
MonthLife = InputBox("What's the asset's useful life in months?")
Do While MonthLife < YEARMONTHS    ' Ensure period is >= 1 year.
   MsgBox "Asset life must be a year or more."
   MonthLife = InputBox("What's the asset's useful life in months?")
Loop
LifeTime = MonthLife / YEARMONTHS  ' Convert months to years.
If LifeTime <> Int(MonthLife / YEARMONTHS) Then
   LifeTime = Int(LifeTime + 1)    ' Round up to nearest year.
End If
DepYear = CInt(InputBox("For which year do you want depreciation?"))
Do While DepYear < 1 Or DepYear > LifeTime
   MsgBox "You must enter at least 1 but not more than " & LifeTime
   DepYear = CInt(InputBox("For what year do you want depreciation?"))
Loop
PDepr = SYD(InitCost, SalvageVal, LifeTime, DepYear)
MsgBox "The depreciation for year " & DepYear & " is " & Format(PDepr, Fmt)
& "."
```

DDB Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctDDBC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctDDBX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctDDBS"}

Returns a **Double** specifying the depreciation of an asset for a specific time period using the double-declining balance method or some other method you specify.

Syntax

DDB(*cost*, *salvage*, *life*, *period*[, *factor*])

The **DDB** function has these named arguments:

Part	Description
cost	Required. **Double** specifying initial cost of the asset.
salvage	Required. **Double** specifying value of the asset at the end of its useful life.
life	Required. **Double** specifying length of useful life of the asset.
period	Required. **Double** specifying period for which asset depreciation is calculated.
factor	Optional. **Variant** specifying rate at which the balance declines. If omitted, 2 (double-declining method) is assumed.

Remarks

The double-declining balance method computes depreciation at an accelerated rate. Depreciation is highest in the first period and decreases in successive periods.

The ***life*** and ***period*** arguments must be expressed in the same units. For example, if ***life*** is given in months, ***period*** must also be given in months. All arguments must be positive numbers.

The **DDB** function uses the following formula to calculate depreciation for a given period:

Depreciation / ***period*** = ((***cost*** – ***salvage***) * ***factor***) / ***life***

FV Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctFVC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctFVX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctFVS"}

Returns a **Double** specifying the future value of an annuity based on periodic, fixed payments and a fixed interest rate.

Syntax

FV(*rate*, *nper*, *pmt*[, *pv*[, *type*]])

The **FV** function has these named arguments:

Part	Description
rate	Required. **Double** specifying interest rate per period. For example, if you get a car loan at an annual percentage rate (APR) of 10 percent and make monthly payments, the rate per period is 0.1/12, or 0.0083.
nper	Required. **Integer** specifying total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 * 12 (or 48) payment periods.
pmt	Required. **Double** specifying payment to be made each period. Payments usually contain principal and interest that doesn't change over the life of the annuity.
pv	Optional. **Variant** specifying present value (or lump sum) of a series of future payments. For example, when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make. If omitted, 0 is assumed.
type	Optional. **Variant** specifying when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Remarks

An annuity is a series of fixed cash payments made over a period of time. An annuity can be a loan (such as a home mortgage) or an investment (such as a monthly savings plan).

The ***rate*** and ***nper*** arguments must be calculated using payment periods expressed in the same units. For example, if ***rate*** is calculated using months, ***nper*** must also be calculated using months.

For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

IPmt Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIPmtC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIPmtX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIPmtS"}

Returns a **Double** specifying the interest payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.

Syntax

IPmt(*rate*, *per*, *nper*, *pv*[, *fv*[, *type*]])

The **IPmt** function has these named arguments:

Part	Description
rate	Required. **Double** specifying interest rate per period. For example, if you get a car loan at an annual percentage rate (APR) of 10 percent and make monthly payments, the rate per period is 0.1/12, or 0.0083.
per	Required. **Double** specifying payment period in the range 1 through ***nper***.
nper	Required. **Double** specifying total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 * 12 (or 48) payment periods.
pv	Required. **Double** specifying present value, or value today, of a series of future payments or receipts. For example, when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make.
fv	Optional. **Variant** specifying future value or cash balance you want after you've made the final payment. For example, the future value of a loan is $0 because that's its value after the final payment. However, if you want to save $50,000 over 18 years for your child's education, then $50,000 is the future value. If omitted, 0 is assumed.
type	Optional. **Variant** specifying when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Remarks

An annuity is a series of fixed cash payments made over a period of time. An annuity can be a loan (such as a home mortgage) or an investment (such as a monthly savings plan).

The ***rate*** and ***nper*** arguments must be calculated using payment periods expressed in the same units. For example, if ***rate*** is calculated using months, ***nper*** must also be calculated using months.

For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

IRR Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIRRC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIRRX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIRRS"}

Returns a **Double** specifying the internal rate of return for a series of periodic cash flows (payments and receipts).

Syntax

IRR(*values*()[, ***guess***]**)**

The **IRR** function has these named arguments:

Part	Description
***values*()**	Required. Array of **Double** specifying cash flow values. The array must contain at least one negative value (a payment) and one positive value (a receipt).
guess	Optional. **Variant** specifying value you estimate will be returned by **IRR**. If omitted, ***guess*** is 0.1 (10 percent).

Remarks

The internal rate of return is the interest rate received for an investment consisting of payments and receipts that occur at regular intervals.

The **IRR** function uses the order of values within the array to interpret the order of payments and receipts. Be sure to enter your payment and receipt values in the correct sequence. The cash flow for each period doesn't have to be fixed, as it is for an annuity.

IRR is calculated by iteration. Starting with the value of ***guess***, **IRR** cycles through the calculation until the result is accurate to within 0.00001 percent. If **IRR** can't find a result after 20 tries, it fails.

MIRR Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctMIRRC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctMIRRX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctMIRRS"}

Returns a **Double** specifying the modified internal rate of return for a series of periodic cash flows (payments and receipts).

Syntax

MIRR(*values*(), *finance_rate*, *reinvest_rate*)

The **MIRR** function has these named arguments:

Part	Description
***values*()**	Required. Array of **Double** specifying cash flow values. The array must contain at least one negative value (a payment) and one positive value (a receipt).
finance_rate	Required. **Double** specifying interest rate paid as the cost of financing.
reinvest_rate	Required. **Double** specifying interest rate received on gains from cash reinvestment.

Remarks

The modified internal rate of return is the internal rate of return when payments and receipts are financed at different rates. The **MIRR** function takes into account both the cost of the investment (***finance_rate***) and the interest rate received on reinvestment of cash (***reinvest_rate***).

The ***finance_rate*** and ***reinvest_rate*** arguments are percentages expressed as decimal values. For example, 12 percent is expressed as 0.12.

The **MIRR** function uses the order of values within the array to interpret the order of payments and receipts. Be sure to enter your payment and receipt values in the correct sequence.

NPer Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctNPerC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctNPerX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctNPerS"}

Returns a **Double** specifying the number of periods for an annuity based on periodic, fixed payments and a fixed interest rate.

Syntax

NPer(*rate*, *pmt*, *pv*[, *fv*[, *type*]])

The **NPer** function has these named arguments:

Part	Description
rate	Required. **Double** specifying interest rate per period. For example, if you get a car loan at an annual percentage rate (APR) of 10 percent and make monthly payments, the rate per period is 0.1/12, or 0.0083.
pmt	Required. **Double** specifying payment to be made each period. Payments usually contain principal and interest that doesn't change over the life of the annuity.
pv	Required. **Double** specifying present value, or value today, of a series of future payments or receipts. For example, when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make.
fv	Optional. **Variant** specifying future value or cash balance you want after you've made the final payment. For example, the future value of a loan is $0 because that's its value after the final payment. However, if you want to save $50,000 over 18 years for your child's education, then $50,000 is the future value. If omitted, 0 is assumed.
type	Optional. **Variant** specifying when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Remarks

An annuity is a series of fixed cash payments made over a period of time. An annuity can be a loan (such as a home mortgage) or an investment (such as a monthly savings plan).

For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

NPV Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctNPVC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctNPVX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctNPVS"}

Returns a **Double** specifying the net present value of an investment based on a series of periodic cash flows (payments and receipts) and a discount rate.

Syntax

NPV(*rate*, *values*())

The **NPV** function has these named arguments:

Part	Description
rate	Required. **Double** specifying discount rate over the length of the period, expressed as a decimal.
***values*()**	Required. Array of **Double** specifying cash flow values. The array must contain at least one negative value (a payment) and one positive value (a receipt).

Remarks

The net present value of an investment is the current value of a future series of payments and receipts.

The **NPV** function uses the order of values within the array to interpret the order of payments and receipts. Be sure to enter your payment and receipt values in the correct sequence.

The **NPV** investment begins one period before the date of the first cash flow value and ends with the last cash flow value in the array.

The net present value calculation is based on future cash flows. If your first cash flow occurs at the beginning of the first period, the first value must be added to the value returned by **NPV** and must not be included in the cash flow values of ***values*()**.

The **NPV** function is similar to the **PV** function (present value) except that the **PV** function allows cash flows to begin either at the end or the beginning of a period. Unlike the variable **NPV** cash flow values, **PV** cash flows must be fixed throughout the investment.

Pmt Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctPmtC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctPmtX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctPmtS"}

Returns a **Double** specifying the payment for an annuity based on periodic, fixed payments and a fixed interest rate.

Syntax

Pmt(*rate*, *nper*, *pv*[, *fv*[, *type*]])

The **Pmt** function has these named arguments:

Part	Description
rate	Required. **Double** specifying interest rate per period. For example, if you get a car loan at an annual percentage rate (APR) of 10 percent and make monthly payments, the rate per period is 0.1/12, or 0.0083.
nper	Required. **Integer** specifying total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 * 12 (or 48) payment periods.
pv	Required. **Double** specifying present value (or lump sum) that a series of payments to be paid in the future is worth now. For example, when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make.
fv	Optional. **Variant** specifying future value or cash balance you want after you've made the final payment. For example, the future value of a loan is $0 because that's its value after the final payment. However, if you want to save $50,000 over 18 years for your child's education, then $50,000 is the future value. If omitted, 0 is assumed.
type	Optional. **Variant** specifying when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Remarks

An annuity is a series of fixed cash payments made over a period of time. An annuity can be a loan (such as a home mortgage) or an investment (such as a monthly savings plan).

The ***rate*** and ***nper*** arguments must be calculated using payment periods expressed in the same units. For example, if ***rate*** is calculated using months, ***nper*** must also be calculated using months.

For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

PPmt Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctPPmtC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctPPmtX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctPPmtS"}

Returns a **Double** specifying the principal payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.

Syntax

PPmt(*rate*, *per*, *nper*, *pv*[, *fv*[, *type*]])

The **PPmt** function has these named arguments:

Part	Description
rate	Required. **Double** specifying interest rate per period. For example, if you get a car loan at an annual percentage rate (APR) of 10 percent and make monthly payments, the rate per period is 0.1/12, or 0.0083.
per	Required. **Integer** specifying payment period in the range 1 through ***nper***.
nper	Required. **Integer** specifying total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 * 12 (or 48) payment periods.
pv	Required. **Double** specifying present value, or value today, of a series of future payments or receipts. For example, when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make.
fv	Optional. **Variant** specifying future value or cash balance you want after you've made the final payment. For example, the future value of a loan is $0 because that's its value after the final payment. However, if you want to save $50,000 over 18 years for your child's education, then $50,000 is the future value. If omitted, 0 is assumed.
type	Optional. **Variant** specifying when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Remarks

An annuity is a series of fixed cash payments made over a period of time. An annuity can be a loan (such as a home mortgage) or an investment (such as a monthly savings plan).

The ***rate*** and ***nper*** arguments must be calculated using payment periods expressed in the same units. For example, if ***rate*** is calculated using months, ***nper*** must also be calculated using months.

For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

PV Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctPVC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctPVX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctPVS"}

Returns a **Double** specifying the present value of an annuity based on periodic, fixed payments to be paid in the future and a fixed interest rate.

Syntax

PV(*rate*, *nper*, *pmt*[, *fv*[, *type*]])

The **PV** function has these named arguments:

Part	Description
rate	Required. **Double** specifying interest rate per period. For example, if you get a car loan at an annual percentage rate (APR) of 10 percent and make monthly payments, the rate per period is 0.1/12, or 0.0083.
nper	Required. **Integer** specifying total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 * 12 (or 48) payment periods.
pmt	Required. **Double** specifying payment to be made each period. Payments usually contain principal and interest that doesn't change over the life of the annuity.
fv	Optional. **Variant** specifying future value or cash balance you want after you've made the final payment. For example, the future value of a loan is $0 because that's its value after the final payment. However, if you want to save $50,000 over 18 years for your child's education, then $50,000 is the future value. If omitted, 0 is assumed.
type	Optional. **Variant** specifying when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.

Remarks

An annuity is a series of fixed cash payments made over a period of time. An annuity can be a loan (such as a home mortgage) or an investment (such as a monthly savings plan).

The ***rate*** and ***nper*** arguments must be calculated using payment periods expressed in the same units. For example, if ***rate*** is calculated using months, ***nper*** must also be calculated using months.

For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

Rate Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctRateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctRateX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctRateS"}

Returns a **Double** specifying the interest rate per period for an annuity.

Syntax

Rate(*nper*, *pmt*, *pv*[, *fv*[, *type*[, *guess*]]])

The **Rate** function has these named arguments:

Part	Description
nper	Required. **Double** specifying total number of payment periods in the annuity. For example, if you make monthly payments on a four-year car loan, your loan has a total of 4 * 12 (or 48) payment periods.
pmt	Required. **Double** specifying payment to be made each period. Payments usually contain principal and interest that doesn't change over the life of the annuity.
pv	Required. **Double** specifying present value, or value today, of a series of future payments or receipts. For example, when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make.
fv	Optional. **Variant** specifying future value or cash balance you want after you make the final payment. For example, the future value of a loan is $0 because that's its value after the final payment. However, if you want to save $50,000 over 18 years for your child's education, then $50,000 is the future value. If omitted, 0 is assumed.
type	Optional. **Variant** specifying a number indicating when payments are due. Use 0 if payments are due at the end of the payment period, or use 1 if payments are due at the beginning of the period. If omitted, 0 is assumed.
guess	Optional. **Variant** specifying value you estimate will be returned by **Rate**. If omitted, ***guess*** is 0.1 (10 percent).

Remarks

An annuity is a series of fixed cash payments made over a period of time. An annuity can be a loan (such as a home mortgage) or an investment (such as a monthly savings plan).

For all arguments, cash paid out (such as deposits to savings) is represented by negative numbers; cash received (such as dividend checks) is represented by positive numbers.

Rate is calculated by iteration. Starting with the value of ***guess***, **Rate** cycles through the calculation until the result is accurate to within 0.00001 percent. If **Rate** can't find a result after 20 tries, it fails. If your guess is 10 percent and **Rate** fails, try a different value for ***guess***.

SLN Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctSLNC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctSLNX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctSLNS"}

Returns a **Double** specifying the straight-line depreciation of an asset for a single period.

Syntax

SLN(*cost*, *salvage*, *life*)

The **SLN** function has these named arguments:

Part	Description
cost	Required. **Double** specifying initial cost of the asset.
salvage	Required. **Double** specifying value of the asset at the end of its useful life.
life	Required. **Double** specifying length of the useful life of the asset.

Remarks

The depreciation period must be expressed in the same unit as the ***life*** argument. All arguments must be positive numbers.

SYD Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctSYDC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctSYDX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctSYDS"}

Returns a **Double** specifying the sum-of-years' digits depreciation of an asset for a specified period.

Syntax

SYD(*cost*, *salvage*, *life*, *period*)

The **SYD** function has these named arguments:

Part	Description
cost	Required. **Double** specifying initial cost of the asset.
salvage	Required. **Double** specifying value of the asset at the end of its useful life.
life	Required. **Double** specifying length of the useful life of the asset.
period	Required. **Double** specifying period for which asset depreciation is calculated.

Remarks

The ***life*** and ***period*** arguments must be expressed in the same units. For example, if ***life*** is given in months, ***period*** must also be given in months. All arguments must be positive numbers.

IsArray Function Example

This example uses the **IsArray** function to check if a variable is an array.

```
Dim MyArray(1 To 5) As Integer, YourArray, MyCheck   ' Declare array
variables.
YourArray = Array(1, 2, 3) ' Use Array function.
MyCheck = IsArray(MyArray) ' Returns True.
MyCheck = IsArray(YourArray) ' Returns True.
```

IsDate Function Example

This example uses the **IsDate** function to determine if an expression can be converted to a date.

```
Dim MyDate, YourDate, NoDate, MyCheck
MyDate = "February 12, 1969": YourDate = #2/12/69#: NoDate = "Hello"
MyCheck = IsDate(MyDate)' Returns True.
MyCheck = IsDate(YourDate) ' Returns True.
MyCheck = IsDate(NoDate)' Returns False.
```

IsEmpty Function Example

This example uses the **IsEmpty** function to determine whether a variable has been initialized.

```
Dim MyVar, MyCheck
MyCheck = IsEmpty(MyVar)' Returns True.

MyVar = Null' Assign Null.
MyCheck = IsEmpty(MyVar)' Returns False.

MyVar = Empty  ' Assign Empty.
MyCheck = IsEmpty(MyVar)' Returns True.
```

IsError Function Example

This example uses the **IsError** function to check if a numeric expression is an error value. The **CVErr** function is used to return an **Error Variant** from a user-defined function. Assume `UserFunction` is a user-defined function procedure that returns an error value; for example, a return value assigned with the statement `UserFunction = CVErr(32767)`, where 32767 is a user-defined number.

```
Dim ReturnVal, MyCheck
ReturnVal = UserFunction()
MyCheck = IsError(ReturnVal) ' Returns True.
```

IsMissing Function Example

This example uses the **IsMissing** function to check if an optional argument has been passed to a user-defined procedure. Note that **Optional** arguments can now have default values and types other than **Variant**.

```
Dim ReturnValue
' The following statements call the user-defined function procedure.
ReturnValue = ReturnTwice()' Returns Null.
ReturnValue = ReturnTwice(2) ' Returns 4.

' Function procedure definition.
Function ReturnTwice(Optional A)
   If IsMissing(A) Then
      ' If argument is missing, return a Null.
      ReturnTwice = Null
   Else
      ' If argument is present, return twice the value.
      ReturnTwice = A * 2
   End If
End Function
```

IsNull Function Example

This example uses the **IsNull** function to determine if a variable contains a **Null**.

```
Dim MyVar, MyCheck
MyCheck = IsNull(MyVar) ' Returns False.

MyVar = ""
MyCheck = IsNull(MyVar) ' Returns False.

MyVar = Null
MyCheck = IsNull(MyVar) ' Returns True.
```

IsNumeric Function Example

This example uses the **IsNumeric** function to determine if a variable can be evaluated as a number.

```
Dim MyVar, MyCheck
MyVar = "53"' Assign value.
MyCheck = IsNumeric(MyVar) ' Returns True.

MyVar = "459.95"  ' Assign value.
MyCheck = IsNumeric(MyVar) ' Returns True.

MyVar = "45 Help" ' Assign value.
MyCheck = IsNumeric(MyVar) ' Returns False.
```

IsObject Function Example

This example uses the **IsObject** function to determine if an identifier represents an object variable. `MyObject` and `YourObject` are object variables of the same type. They are generic names used for illustration purposes only.

```
Dim MyInt As Integer, YourObject, MyCheck' Declare variables.
Dim MyObject As Object
Set YourObject = MyObject  ' Assign an object reference.
MyCheck = IsObject(YourObject)     ' Returns True.
MyCheck = IsObject(MyInt)  ' Returns False.
```

TypeName Function Example

This example uses the **TypeName** function to return information about a variable.

```
' Declare variables.
Dim NullVar, MyType, StrVar As String, IntVar As Integer, CurVar As
Currency
Dim ArrayVar (1 To 5) As Integer
NullVar = Null ' Assign Null value.
MyType = TypeName(StrVar)   ' Returns "String".
MyType = TypeName(IntVar)   ' Returns "Integer".
MyType = TypeName(CurVar)   ' Returns "Currency".
MyType = TypeName(NullVar)  ' Returns "Null".
MyType = TypeName(ArrayVar)' Returns "Integer()".
```

VarType Function Example

This example uses the **VarType** function to determine the subtype of a variable.

```
Dim IntVar, StrVar, DateVar, MyCheck
' Initialize variables.
IntVar = 459: StrVar = "Hello World": DateVar = #2/12/69#
MyCheck = VarType(IntVar)   ' Returns 2.
MyCheck = VarType(DateVar)  ' Returns 7.
MyCheck = VarType(StrVar)   ' Returns 8.
```

IsArray Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIsArrayC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIsArrayX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIsArrayS"}

Returns a **Boolean** value indicating whether a variable is an array.

Syntax

IsArray(*varname***)**

The required *varname* argument is an identifier specifying a variable.

Remarks

IsArray returns **True** if the variable is an array; otherwise, it returns **False**. **IsArray** is especially useful with variants containing arrays.

IsDate Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIsDateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIsDateX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIsDateS"}

Returns a **Boolean** value indicating whether an expression can be converted to a date.

Syntax

IsDate(*expression***)**

The required *expression* argument is a **Variant** containing a date expression or string expression recognizable as a date or time.

Remarks

IsDate returns **True** if the expression is a date or can be converted to a valid date; otherwise, it returns **False**. In Microsoft Windows, the range of valid dates is January 1, 100 A.D. through December 31, 9999 A.D.; the ranges vary among operating systems.

IsEmpty Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIsEmptyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIsEmptyX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIsEmptyS"}

Returns a **Boolean** value indicating whether a variable has been initialized.

Syntax

IsEmpty(*expression***)**

The required *expression* argument is a **Variant** containing a numeric or string expression. However, because **IsEmpty** is used to determine if individual variables are initialized, the *expression* argument is most often a single variable name.

Remarks

IsEmpty returns **True** if the variable is uninitialized, or is explicitly set to **Empty**; otherwise, it returns **False**. **False** is always returned if *expression* contains more than one variable. **IsEmpty** only returns meaningful information for variants.

IsError Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIsErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIsErrorX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIsErrorS"}

Returns a **Boolean** value indicating whether an expression is an error value.

Syntax

IsError(*expression***)**

The required *expression* argument must be a **Variant** of **VarType vbError**.

Remarks

Error values are created by converting real numbers to error values using the **CVErr** function. The **IsError** function is used to determine if a numeric expression represents an error. **IsError** returns **True** if the *expression* argument indicates an error; otherwise, it returns **False**. **IsError** only returns meaningful information for variants of **VarType vbError**.

IsMissing Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIsMissingC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIsMissingX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIsMissingS"}

Returns a **Boolean** value indicating whether an optional **Variant** argument has been passed to a procedure.

Syntax

IsMissing(*argname***)**

The required *argname* argument contains the name of an optional **Variant** procedure argument.

Remarks

Use the **IsMissing** function to detect whether or not optional **Variant** arguments have been provided in calling a procedure. **IsMissing** returns **True** if no value has been passed for the specified argument; otherwise, it returns **False**. If **IsMissing** returns **True** for an argument, use of the missing argument in other code may cause a user-defined error. If **IsMissing** is used on a **ParamArray** argument, it always returns **False**. To detect an empty **ParamArray**, test to see if the array's upper bound is less than its lower bound.

IsNull Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIsNullC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIsNullX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIsNullS"}

Returns a **Boolean** value that indicates whether an expression contains no valid data (**Null**).

Syntax

IsNull(*expression***)**

The required *expression* argument is a **Variant** containing a numeric expression or string expression.

Remarks

IsNull returns **True** if *expression* is **Null**; otherwise, **IsNull** returns **False**. If *expression* consists of more than one variable, **Null** in any constituent variable causes **True** to be returned for the entire expression.

The **Null** value indicates that the **Variant** contains no valid data. **Null** is not the same as **Empty**, which indicates that a variable has not yet been initialized. It is also not the same as a zero-length string (""), which is sometimes referred to as a null string.

Important Use the **IsNull** function to determine whether an expression contains a **Null** value. Expressions that you might expect to evaluate to **True** under some circumstances, such as `If Var = Null` and `If Var <> Null`, are always **False**. This is because any expression containing a **Null** is itself **Null** and, therefore, **False**.

IsNumeric Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIsNumericC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIsNumericX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIsNumericS"}

Returns a **Boolean** value indicating whether an expression can be evaluated as a number.

Syntax

IsNumeric(*expression***)**

The required *expression* argument is a **Variant** containing a numeric expression or string expression.

Remarks

IsNumeric returns **True** if the entire *expression* is recognized as a number; otherwise, it returns **False**.

IsNumeric returns **False** if *expression* is a date expression.

IsObject Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIsObjectC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIsObjectX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIsObjectS"}

Returns a **Boolean** value indicating whether an identifier represents an object variable.

Syntax

IsObject(*identifier***)**

The required *identifier* argument is a variable name.

Remarks

IsObject is useful only in determining whether a **Variant** is of **VarType vbObject**. This could occur if the **Variant** actually references (or once referenced) an object, or if it contains **Nothing**.

IsObject returns **True** if *identifier* is a variable declared with **Object** type or any valid class type, or if *identifier* is a **Variant** of **VarType vbObject**, or a user-defined object; otherwise, it returns **False**. **IsObject** returns **True** even if the variable has been set to **Nothing**.

Use error trapping to be sure that an object reference is valid.

TypeName Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctTypeNameC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctTypeNameX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctTypeNameS"}

Returns a **String** that provides information about a variable.

Syntax

TypeName(*varname***)**

The required *varname* argument is a **Variant** containing any variable except a variable of a user-defined type.

Remarks

The string returned by **TypeName** can be any one of the following:

String returned	Variable
object type	An object whose type is *objecttype*
Byte	Byte value
Integer	Integer
Long	Long integer
Single	Single-precision floating-point number
Double	Double-precision floating-point number
Currency	Currency value
Decimal	Decimal value
Date	Date value
String	String
Boolean	Boolean value
Error	An error value
Empty	Uninitialized
Null	No valid data
Object	An object
Unknown	An object whose type is unknown
Nothing	Object variable that doesn't refer to an object

If *varname* is an array, the returned string can be any one of the possible returned strings (or **Variant**) with empty parentheses appended. For example, if *varname* is an array of integers, **TypeName** returns "`Integer()`".

VarType Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctVarTypeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctVarTypeX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctVarTypeS"}

Returns an **Integer** indicating the subtype of a variable.

Syntax

VarType(*varname***)**

The required *varname* argument is a **Variant** containing any variable except a variable of a user-defined type.

Return Values

Constant	Value	Description
vbEmpty	0	**Empty** (uninitialized)
vbNull	1	**Null** (no valid data)
vbInteger	2	Integer
vbLong	3	Long integer
vbSingle	4	Single-precision floating-point number
vbDouble	5	Double-precision floating-point number
vbCurrency	6	Currency value
vbDate	7	Date value
vbString	8	String
vbObject	9	Object
vbError	10	Error value
vbBoolean	11	Boolean value
vbVariant	12	**Variant** (used only with arrays of variants)
vbDataObject	13	A data access object
vbDecimal	14	Decimal value
vbByte	17	Byte value
vbArray	8192	Array

Note These constants are specified by Visual Basic for Applications. The names can be used anywhere in your code in place of the actual values.

Remarks

The **VarType** function never returns the value for **vbArray** by itself. It is always added to some other value to indicate an array of a particular type. The constant **vbVariant** is only returned in conjunction with **vbArray** to indicate that the argument to the **VarType** function is an array of type **Variant**. For example, the value returned for an array of integers is calculated as **vbInteger** + **vbArray**, or 8194. If an object has a default property, **VarType (***object***)** returns the type of the object's default property.

Empty

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vakeyEmptyC"}

The **Empty** keyword is used as a **Variant** subtype. It indicates an uninitialized variable value.

False

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vakeyFalseC"}

The **False** keyword has a value equal to 0.

Me

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vakeyMeC"}

The **Me** keyword behaves like an implicitly declared variable. It is automatically available to every procedure in a class module. When a class can have more than one instance, **Me** provides a way to refer to the specific instance of the class where the code is executing. Using **Me** is particularly useful for passing information about the currently executing instance of a class to a procedure in another module. For example, suppose you have the following procedure in a module:

```
Sub ChangeFormColor(FormName As Form)
   FormName.BackColor = RGB(Rnd * 256, Rnd * 256, Rnd * 256)
End Sub
```

You can call this procedure and pass the current instance of the Form class as an argument using the following statement:

```
ChangeFormColor Me
```

Nothing

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vakeyNothingC"}

The **Nothing** keyword is used to disassociate an object variable from an actual object. Use the **Set** statement to assign **Nothing** to an object variable. For example:

```
Set MyObject = Nothing
```

Several object variables can refer to the same actual object. When **Nothing** is assigned to an object variable, that variable no longer refers to an actual object. When several object variables refer to the same object, memory and system resources associated with the object to which the variables refer are released only after all of them have been set to **Nothing**, either explicitly using **Set**, or implicitly after the last object variable set to **Nothing** goes out of scope.

Null

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vakeyNullC"}

The **Null** keyword is used as a **Variant** subtype. It indicates that a variable contains no valid data.

True

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vakeyTrueC"}

The **True** keyword has a value equal to -1.

Abs Function Example

This example uses the **Abs** function to compute the absolute value of a number.

```
Dim MyNumber
MyNumber = Abs(50.3)  ' Returns 50.3.
MyNumber = Abs(-50.3)' Returns 50.3.
```

Atn Function Example

This example uses the **Atn** function to calculate the value of pi.

```
Dim pi
pi = 4 * Atn(1)' Calculate the value of pi.
```

Cos Function Example

This example uses the **Cos** function to return the cosine of an angle.

```
Dim MyAngle, MySecant
MyAngle = 1.3  ' Define angle in radians.
MySecant = 1 / Cos(MyAngle)' Calculate secant.
```

Exp Function Example

This example uses the **Exp** function to return *e* raised to a power.

```
Dim MyAngle, MyHSin
' Define angle in radians.
MyAngle = 1.3
' Calculate hyperbolic sine.
MyHSin = (Exp(MyAngle) - Exp(-1 * MyAngle)) / 2
```

Int Function, Fix Function Example

This example illustrates how the **Int** and **Fix** functions return integer portions of numbers. In the case of a negative number argument, the **Int** function returns the first negative integer less than or equal to the number; the **Fix** function returns the first negative integer greater than or equal to the number.

```
Dim MyNumber
MyNumber = Int(99.8)  ' Returns 99.
MyNumber = Fix(99.2)  ' Returns 99.

MyNumber = Int(-99.8)' Returns -100.
MyNumber = Fix(-99.8)' Returns -99.

MyNumber = Int(-99.2)' Returns -100.
MyNumber = Fix(-99.2)' Returns -99.
```

Log Function Example

This example uses the **Log** function to return the natural logarithm of a number.

```
Dim MyAngle, MyLog
' Define angle in radians.
MyAngle = 1.3
' Calculate inverse hyperbolic sine.
MyLog = Log(MyAngle + Sqr(MyAngle * MyAngle + 1))
```

Randomize Statement Example

This example uses the **Randomize** statement to initialize the random-number generator. Because the number argument has been omitted, **Randomize** uses the return value from the **Timer** function as the new seed value.

```
Dim MyValue
Randomize' Initialize random-number generator.

MyValue = Int((6 * Rnd) + 1) ' Generate random value between 1 and 6.
```

Rnd Function Example

This example uses the **Rnd** function to generate a random integer value from 1 to 6.

```
Dim MyValue
MyValue = Int((6 * Rnd) + 1) ' Generate random value between 1 and 6.
```

Sgn Function Example

This example uses the **Sgn** function to determine the sign of a number.

```
Dim MyVar1, MyVar2, MyVar3, MySign
MyVar1 = 12: MyVar2 = -2.4: MyVar3 = 0
MySign = Sgn(MyVar1)  ' Returns 1.
MySign = Sgn(MyVar2)  ' Returns -1.
MySign = Sgn(MyVar3)  ' Returns 0.
```

Sin Function Example

This example uses the **Sin** function to return the sine of an angle.

```
Dim MyAngle, MyCosecant
MyAngle = 1.3  ' Define angle in radians.
MyCosecant = 1 / Sin(MyAngle)' Calculate cosecant.
```

Sqr Function Example

This example uses the **Sqr** function to calculate the square root of a number.

```
Dim MySqr
MySqr = Sqr(4)  ' Returns 2.
MySqr = Sqr(23)' Returns 4.79583152331272.
MySqr = Sqr(0)  ' Returns 0.
MySqr = Sqr(-4)' Generates a run-time error.
```

Tan Function Example

This example uses the **Tan** function to return the tangent of an angle.

```
Dim MyAngle, MyCotangent
MyAngle = 1.3  ' Define angle in radians.
MyCotangent = 1 / Tan(MyAngle)     ' Calculate cotangent.
```

Math Functions

Abs Function

Atn Function

Cos Function

Exp Function

Fix Function

Int Function

Log Function

Rnd Function

Sgn Function

Sin Function

Sqr Function

Tan Function

Derived Math Functions

Abs Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctAbsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctAbsX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctAbsS"}

Returns a value of the same type that is passed to it specifying the absolute value of a number.

Syntax

Abs(*number***)**

The required *number* argument can be any valid numeric expression. If *number* contains **Null**, **Null** is returned; if it is an uninitialized variable, zero is returned.

Remarks

The absolute value of a number is its unsigned magnitude. For example, `ABS(-1)` and `ABS(1)` both return `1`.

Atn Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctAtnC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctAtnX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctAtnS"}

Returns a **Double** specifying the arctangent of a number.

Syntax

Atn(*number***)**

The required *number* argument is a **Double** or any valid numeric expression.

Remarks

The **Atn** function takes the ratio of two sides of a right triangle (*number*) and returns the corresponding angle in radians. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

The range of the result is **-**pi/2 to pi/2 radians.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Note **Atn** is the inverse trigonometric function of **Tan**, which takes an angle as its argument and returns the ratio of two sides of a right triangle. Do not confuse **Atn** with the cotangent, which is the simple inverse of a tangent (1/tangent).

Cos Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctCosC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctCosX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctCosS"}

Returns a **Double** specifying the cosine of an angle.

Syntax

Cos(*number***)**

The required *number* argument is a **Double** or any valid numeric expression that expresses an angle in radians.

Remarks

The **Cos** function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse.

The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Exp Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctExpC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctExpX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctExpS"}

Returns a **Double** specifying *e* (the base of natural logarithms) raised to a power.

Syntax

Exp(*number***)**

The required *number* argument is a **Double** or any valid numeric expression.

Remarks

If the value of *number* exceeds 709.782712893, an error occurs. The constant *e* is approximately 2.718282.

Note The **Exp** function complements the action of the **Log** function and is sometimes referred to as the antilogarithm.

Int, Fix Functions

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIntC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIntX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIntS"}

Returns a value of the type passed to it containing the integer portion of a number.

Syntax

Int(*number***)**
Fix(*number***)**

The required *number* argument is a **Double** or any valid numeric expression. If *number* contains **Null**, **Null** is returned.

Remarks

Both **Int** and **Fix** remove the fractional part of *number* and return the resulting integer value.

The difference between **Int** and **Fix** is that if *number* is negative, **Int** returns the first negative integer less than or equal to *number,* whereas **Fix** returns the first negative integer greater than or equal to *number.* For example, **Int** converts -8.4 to -9, and **Fix** converts -8.4 to -8.

Fix(*number***)** is equivalent to:

```
Sgn(number) * Int(Abs(number))
```

Log Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctLogC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctLogX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctLogS"}

Returns a **Double** specifying the natural logarithm of a number.

Syntax

Log(*number***)**

The required *number* argument is a **Double** or any valid numeric expression greater than zero.

Remarks

The natural logarithm is the logarithm to the base *e*. The constant *e* is approximately 2.718282.

You can calculate base-*n* logarithms for any number *x* by dividing the natural logarithm of *x* by the natural logarithm of *n* as follows:

Log*n(x)* = **Log**(*x*) / **Log**(*n*)

The following example illustrates a custom **Function** that calculates base-10 logarithms:

```
Static Function Log10(X)
   Log10 = Log(X) / Log(10#)
End Function
```

Randomize Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmRandomizeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmRandomizeX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmRandomizeS"}

Initializes the random-number generator.

Syntax

Randomize [*number*]

The optional *number* argument is a **Variant** or any valid numeric expression.

Remarks

Randomize uses *number* to initialize the **Rnd** function's random-number generator, giving it a new seed value. If you omit *number,* the value returned by the system timer is used as the new seed value.

If **Randomize** is not used, the **Rnd** function (with no arguments) uses the same number as a seed the first time it is called, and thereafter uses the last generated number as a seed value.

Note To repeat sequences of random numbers, call **Rnd** with a negative argument immediately before using **Randomize** with a numeric argument. Using **Randomize** with the same value for *number* does not repeat the previous sequence.

Rnd Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctRndC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctRndX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctRndS"}

Returns a **Single** containing a random number.

Syntax

Rnd[**(***number***)**]

The optional *number* argument is a **Single** or any valid numeric expression.

Return Values

If *number* is	**Rnd generates**
Less than zero	The same number every time, using *number* as the seed.
Greater than zero	The next random number in the sequence.
Equal to zero	The most recently generated number.
Not supplied	The next random number in the sequence.

Remarks

The **Rnd** function returns a value less than 1 but greater than or equal to zero.

The value of *number* determines how **Rnd** generates a random number:

For any given initial seed, the same number sequence is generated because each successive call to the **Rnd** function uses the previous number as a seed for the next number in the sequence.

Before calling **Rnd**, use the **Randomize** statement without an argument to initialize the random-number generator with a seed based on the system timer.

To produce random integers in a given range, use this formula:

```
Int((upperbound - lowerbound + 1) * Rnd + lowerbound)
```

Here, *upperbound* is the highest number in the range, and *lowerbound* is the lowest number in the range.

Note To repeat sequences of random numbers, call **Rnd** with a negative argument immediately before using **Randomize** with a numeric argument. Using **Randomize** with the same value for *number* does not repeat the previous sequence.

Sgn Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctSgnC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctSgnX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctSgnS"}

Returns a **Variant** (**Integer**) indicating the sign of a number.

Syntax

Sgn(*number***)**

The required *number* argument can be any valid numeric expression.

Return Values

If *number* is	**Sgn returns**
Greater than zero	1
Equal to zero	0
Less than zero	-1

Remarks

The sign of the *number* argument determines the return value of the **Sgn** function.

Sin Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctSinC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctSinX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctSinS"}

Returns a **Double** specifying the sine of an angle.

Syntax

Sin(*number***)**

The required *number* argument is a **Double** or any valid numeric expression that expresses an angle in radians.

Remarks

The **Sin** function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse.

The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Sqr Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctSqrC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctSqrX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctSqrS"}

Returns a **Double** specifying the square root of a number.

Syntax

Sqr(*number***)**

The required *number* argument is a **Double** or any valid numeric expression greater than or equal to zero.

Tan Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctTanC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctTanX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctTanS"}

Returns a **Double** specifying the tangent of an angle.

Syntax

Tan(*number***)**

The required *number* argument is a **Double** or any valid numeric expression that expresses an angle in radians.

Remarks

Tan takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle.

To convert degrees to radians, multiply degrees by pi/180. To convert radians to degrees, multiply radians by 180/pi.

Derived Math Functions

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vagrpderivedmathc"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vagrpDerivedMathS"}

The following is a list of nonintrinsic math functions that can be derived from the intrinsic math functions:

Function	Derived equivalents
Secant	Sec(X) = 1 / Cos(X)
Cosecant	Cosec(X) = 1 / Sin(X)
Cotangent	Cotan(X) = 1 / Tan(X)
Inverse Sine	Arcsin(X) = Atn(X / Sqr(-X * X + 1))
Inverse Cosine	Arccos(X) = Atn(-X / Sqr(-X * X + 1)) + 2 * Atn(1)
Inverse Secant	Arcsec(X) = Atn(X / Sqr(X * X – 1)) + Sgn((X) – 1) * (2 * Atn(1))
Inverse Cosecant	Arccosec(X) = Atn(X / Sqr(X * X - 1)) + (Sgn(X) – 1) * (2 * Atn(1))
Inverse Cotangent	Arccotan(X) = Atn(X) + 2 * Atn(1)
Hyperbolic Sine	HSin(X) = (Exp(X) – Exp(-X)) / 2
Hyperbolic Cosine	HCos(X) = (Exp(X) + Exp(-X)) / 2
Hyperbolic Tangent	HTan(X) = (Exp(X) – Exp(-X)) / (Exp(X) + Exp(-X))
Hyperbolic Secant	HSec(X) = 2 / (Exp(X) + Exp(-X))
Hyperbolic Cosecant	HCosec(X) = 2 / (Exp(X) – Exp(-X))
Hyperbolic Cotangent	HCotan(X) = (Exp(X) + Exp(-X)) / (Exp(X) – Exp(-X))
Inverse Hyperbolic Sine	HArcsin(X) = Log(X + Sqr(X * X + 1))
Inverse Hyperbolic Cosine	HArccos(X) = Log(X + Sqr(X * X – 1))
Inverse Hyperbolic Tangent	HArctan(X) = Log((1 + X) / (1 – X)) / 2
Inverse Hyperbolic Secant	HArcsec(X) = Log((Sqr(-X * X + 1) + 1) / X)
Inverse Hyperbolic Cosecant	HArccosec(X) = Log((Sgn(X) * Sqr(X * X + 1) + 1) / X)
Inverse Hyperbolic Cotangent	HArccotan(X) = Log((X + 1) / (X – 1)) / 2
Logarithm to base N	LogN(X) = Log(X) / Log(N)

Add Method Example

This example uses the **Add** method to add `Inst` objects (instances of a class called `Class1` containing a **Public** variable `InstanceName`) to a collection called `MyClasses`. To see how this works, insert a class module and declare a public variable called `InstanceName` at module level of `Class1` (type **Public** `InstanceName`) to hold the names of each instance. Leave the default name as `Class1`. Copy and paste the following code into the Form_Load event procedure of a form module.

```
Dim MyClasses As New Collection    ' Create a Collection object.
Dim Num As Integer' Counter for individualizing keys.
Dim Msg
Dim TheName ' Holder for names user enters.
Do
   Dim Inst As New Class1  ' Create a new instance of Class1.
   Num = Num + 1  ' Increment Num, then get a name.
   Msg = "Please enter a name for this object." & Chr(13) _
    & "Press Cancel to see names in collection."
   TheName = InputBox(Msg, "Name the Collection Items")
   Inst.InstanceName = TheName     ' Put name in object instance.
   ' If user entered name, add it to the collection.
   If Inst.InstanceName <> "" Then
      ' Add the named object to the collection.
      MyClasses.Add item := Inst, key := CStr(Num)
   End If
   ' Clear the current reference in preparation for next one.
   Set Inst = Nothing
Loop Until TheName = ""
For Each x In MyClasses
   MsgBox x.instancename, , "Instance Name"
Next
```

Clear Method Example

This example uses the **Err** object's **Clear** method to reset the numeric properties of the **Err** object to zero and its string properties to zero-length strings. If **Clear** were omitted from the following code, the error message dialog box would be displayed on every iteration of the loop (after an error occurs) whether or not a successive calculation generated an error. You can single-step through the code to see the effect.

```
Dim Result(10) As Integer  ' Declare array whose elements
         ' will overflow easily.
Dim indx
On Error Resume Next ' Defer error trapping.
Do Until indx = 10
   ' Generate an occasional error or store result if no error.
   Result(indx) = Rnd * indx * 20000
   If Err.Number <> 0 Then
      MsgBox Err, , "Error Generated: ", Err.HelpFile, Err.HelpContext
      Err.Clear   ' Clear Err object properties.
   End If
   indx = indx + 1
Loop
```

Item Method Example

This example uses the **Item** method to retrieve a reference to an object in a collection. Assuming `Birthdays` is a **Collection** object, the following code retrieves from the collection references to the objects representing Bill Smith's birthday and Adam Smith's birthday, using the keys "SmithBill" and "SmithAdam" as the *index* arguments. Note that the first call explicitly specifies the **Item** method, but the second does not. Both calls work because the **Item** method is the default for a **Collection** object. The references, assigned to `SmithBillBD` and `SmithAdamBD` using **Set**, can be used to access the properties and methods of the specified objects. To run this code, create the collection and populate it with at least the two referenced members.

```
Dim SmithBillBD As Object
Dim SmithAdamBD As Object
Dim Birthdays
Set SmithBillBD = Birthdays.Item("SmithBill")
Set SmithAdamBD = Birthdays("SmithAdam")
```

Print Method Example

Using the **Print** method, this example displays the value of the variable `MyVar` in the **Immediate** pane of the **Debug** window. Note that the **Print** method only applies to objects that can display text.

```
Dim MyVar
MyVar = "Come see me in the Immediate pane."
Debug.Print MyVar
```

Raise Method Example

This example uses the **Err** object's **Raise** method to generate an error within an Automation object written in Visual Basic. It has the programmatic ID `MyProj.MyObject`.

```
Const MyContextID = 1010407' Define a constant for contextID.
Function TestName(CurrentName, NewName)
   If Instr(NewName, "bob") Then   ' Test the validity of NewName.
      ' Raise the exception
      Err.Raise vbObjectError + 27, "MyProj.MyObject", _
      "No ""bob"" allowed in your name", "c:\MyProj\MyHelp.Hlp", _
      MyContextID
   End If
End Function
```

Remove Method Example

This example illustrates the use of the **Remove** method to remove objects from a **Collection** object, `MyClasses`. This code removes the object whose index is 1 on each iteration of the loop.

```
Dim Num, MyClasses
For Num = 1 To MyClasses.Count
   MyClasses.Remove 1   ' Remove the first object each time
         ' through the loop until there are
         ' no objects left in the collection.
Next Num
```

Add Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamthAddC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vamthAddX":1}
{ewc HLP95EN.DLL,DYNALINK,"Applies To":"vamthAddA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamthAddS"}

Adds a member to a **Collection** object.

Syntax

object.**Add** ***item*, *key*, *before*, *after***

The **Add** method syntax has the following object qualifier and named arguments:

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
item	Required. An expression of any type that specifies the member to add to the collection.
key	Optional. A unique string expression that specifies a key string that can be used, instead of a positional index, to access a member of the collection.
before	Optional. An expression that specifies a relative position in the collection. The member to be added is placed in the collection before the member identified by the ***before*** argument. If a numeric expression, ***before*** must be a number from 1 to the value of the collection's **Count** property. If a string expression, ***before*** must correspond to the ***key*** specified when the member being referred to was added to the collection. You can specify a ***before*** position or an ***after*** position, but not both.
after	Optional. An expression that specifies a relative position in the collection. The member to be added is placed in the collection after the member identified by the ***after*** argument. If numeric, ***after*** must be a number from 1 to the value of the collection's **Count** property. If a string, ***after*** must correspond to the ***key*** specified when the member referred to was added to the collection. You can specify a ***before*** position or an ***after*** position, but not both.

Remarks

Whether the ***before*** or ***after*** argument is a string expression or numeric expression, it must refer to an existing member of the collection, or an error occurs.

An error also occurs if a specified ***key*** duplicates the ***key*** for an existing member of the collection.

Clear Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamthClearC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vamthClearX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vamthClearA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamthClearS"}

Clears all property settings of the **Err** object.

Syntax

object.**Clear**

The *object* is always the **Err** object.

Remarks

Use **Clear** to explicitly clear the **Err** object after an error has been handled, for example, when you use deferred error handling with **On Error Resume Next**. The **Clear** method is called automatically whenever any of the following statements is executed:

- Any type of **Resume** statement
- **Exit Sub**, **Exit Function**, **Exit Property**
- Any **On Error** statement

Note The **On Error Resume Next** construct may be preferable to **On Error GoTo** when handling errors generated during access to other objects. Checking **Err** after each interaction with an object removes ambiguity about which object was accessed by the code. You can be sure which object placed the error code in **Err.Number**, as well as which object originally generated the error (the object specified in **Err.Source**).

Item Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamthItemC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vamthItemX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamthItemS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vamthItemA"}

Returns a specific member of a **Collection** object either by position or by key.

Syntax

object.**Item(***index***)**

The **Item** method syntax has the following object qualifier and part:

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
index	Required. An expression that specifies the position of a member of the collection. If a numeric expression, *index* must be a number from 1 to the value of the collection's **Count** property. If a string expression, *index* must correspond to the ***key*** argument specified when the member referred to was added to the collection.

Remarks

If the value provided as *index* doesn't match any existing member of the collection, an error occurs.

The **Item** method is the default method for a collection. Therefore, the following lines of code are equivalent:

```
Print MyCollection(1)
Print MyCollection.Item(1)
```

Print Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamthPrintC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vamthPrintX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamthprintS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vamthPrintA"}

Prints text in the **Immediate** pane of the **Debug** window.

Syntax

object.**Print** [*outputlist*]

The **Print** method syntax has the following object qualifier and part:

Part	Description
object	Optional. An object expression that evaluates to an object in the Applies To list.
outputlist	Optional. Expression or list of expressions to print. If omitted, a blank line is printed.

The *outputlist* argument has the following syntax and parts:

{**Spc(***n***)** | **Tab(***n***)**} *expression charpos*

Part	Description
Spc(*n***)**	Optional. Used to insert space characters in the output, where *n* is the number of space characters to insert.
Tab(*n***)**	Optional. Used to position the insertion point at an absolute column number where *n* is the column number. Use **Tab** with no argument to position the insertion point at the beginning of the next print zone.
expression	Optional. Numeric expression or string expression to print.
charpos	Optional. Specifies the insertion point for the next character. Use a semicolon (**;**) to position the insertion point immediately following the last character displayed. Use **Tab(***n***)** to position the insertion point at an absolute column number. Use **Tab** with no argument to position the insertion point at the beginning of the next print zone. If *charpos* is omitted, the next character is printed on the next line.

Remarks

Multiple expressions can be separated with either a space or a semicolon.

All data printed to the **Immediate** window is properly formatted using the decimal separator for the locale settings specified for your system. The keywords are output in the appropriate language for the host application.

For **Boolean** data, either `True` or `False` is printed. The **True** and **False** keywords are translated according to the locale setting for the host application.

Date data is written using the standard short date format recognized by your system. When either the date or the time component is missing or zero, only the data provided is written.

Nothing is written if *outputlist* data is **Empty**. However, if *outputlist* data is **Null**, `Null` is output. The **Null** keyword is appropriately translated when it is output.

For error data, the output is written as `Error errorcode`. The **Error** keyword is appropriately translated when it is output.

The *object* is required if the method is used outside a module having a default display space. For

example an error occurs if the method is called in a standard module without specifying an *object*, but if called in a form module, *outputlist* is displayed on the form.

Note Because the **Print** method typically prints with proportionally-spaced characters, there is no correlation between the number of characters printed and the number of fixed-width columns those characters occupy. For example, a wide letter, such as a "W", occupies more than one fixed-width column, and a narrow letter, such as an "i", occupies less. To allow for cases where wider than average characters are used, your tabular columns must be positioned far enough apart. Alternatively, you can print using a fixed-pitch font (such as Courier) to ensure that each character uses only one column.

Raise Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamthRaiseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vamthRaiseX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamthRaiseS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vamthRaiseA"}

Generates a run-time error.

Syntax

object.**Raise** ***number*, *source*, *description*, *helpfile*, *helpcontext***

The **Raise** method has the following object qualifier and named arguments:

Argument	Description
object	Required. Always the **Err** object.
number	Required. **Long** integer that identifies the nature of the error. Visual Basic errors (both Visual Basic-defined and user-defined errors) are in the range 0–65535. When setting the **Number** property to your own error code in a class module, you add your error code number to the **vbObjectError** constant. For example, to generate the error number 1050, assign **vbObjectError** + 1050 to the **Number** property.
source	Optional. String expression naming the object or application that generated the error. When setting this property for an object, use the form *project.class*. If *source* is not specified, the programmatic ID of the current Visual Basic project is used.
description	Optional. String expression describing the error. If unspecified, the value in **Number** is examined. If it can be mapped to a Visual Basic run-time error code, the string that would be returned by the **Error** function is used as **Description**. If there is no Visual Basic error corresponding to **Number**, the "Application-defined or object-defined error" message is used.
helpfile	Optional. The fully qualified path to the Microsoft Windows Help file in which help on this error can be found. If unspecified, Visual Basic uses the fully qualified drive, path, and file name of the Visual Basic Help file.
helpcontext	Optional. The context ID identifying a topic within ***helpfile*** that provides help for the error. If omitted, the Visual Basic Help file context ID for the error corresponding to the **Number** property is used, if it exists.

Remarks

All of the arguments are optional except ***number***. If you use **Raise** without specifying some arguments, and the property settings of the **Err** object contain values that have not been cleared, those values serve as the values for your error.

Raise is used for generating run-time errors and can be used instead of the **Error** statement. **Raise** is useful for generating errors when writing class modules, because the **Err** object gives richer information than is possible if you generate errors with the **Error** statement. For example, with the **Raise** method, the source that generated the error can be specified in the **Source** property, online Help for the error can be referenced, and so on.

Remove Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamthRemoveC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vamthRemoveX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vamthRemoveA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamthRemoveS"}

Removes a member from a **Collection** object.

Syntax

object.**Remove** *index*

The **Remove** method syntax has the following object qualifier and part:

Part	Description
object	Required. An object expression that evaluates to an object in the Applies To list.
index	Required. An expression that specifies the position of a member of the collection. If a numeric expression, *index* must be a number from 1 to the value of the collection's **Count** property. If a string expression, *index* must correspond to the ***key*** argument specified when the member referred to was added to the collection.

Remarks

If the value provided as *index* doesn't match an existing member of the collection, an error occurs.

AppActivate Statement Example

This example illustrates various uses of the **AppActivate** statement to activate an application window. The **Shell** statements assume the applications are in the paths specified.

```
Dim MyAppID, ReturnValue
AppActivate "Microsoft Word" ' Activate Microsoft
    ' Word.

' AppActivate can also use the return value of the Shell function.
MyAppID = Shell("C:\WORD\WINWORD.EXE", 1)' Run Microsoft Word.
AppActivate MyAppID  ' Activate Microsoft
    ' Word.


' You can also use the return value of the Shell function.
ReturnValue = Shell("c:\EXCEL\EXCEL.EXE",1)    ' Run Microsoft Excel.
AppActivate ReturnValue ' Activate Microsoft
    ' Excel.
```

Beep Statement Example

This example uses the **Beep** statement to sound three consecutive tones through the computer's speaker.

```
Dim I
For I = 1 To 3 ' Loop 3 times.
   Beep  ' Sound a tone.
Next I
```

Command Function Example

This example uses the **Command** function to get the command line arguments in a function that returns them in a **Variant** containing an array.

```
Function GetCommandLine(Optional MaxArgs)
   'Declare variables.
   Dim C, CmdLine, CmdLnLen, InArg, I, NumArgs
   'See if MaxArgs was provided.
   If IsMissing(MaxArgs) Then MaxArgs = 10
   'Make array of the correct size.
   ReDim ArgArray(MaxArgs)
   NumArgs = 0: InArg = False
   'Get command line arguments.
   CmdLine = Command()
   CmdLnLen = Len(CmdLine)
   'Go thru command line one character
   'at a time.
   For I = 1 To CmdLnLen
      C = Mid(CmdLine, I, 1)
      'Test for space or tab.
      If (C <> " " And C <> vbTab) Then
         'Neither space nor tab.
         'Test if already in argument.
         If Not InArg Then
         'New argument begins.
         'Test for too many arguments.
            If NumArgs = MaxArgs Then Exit For
            NumArgs = NumArgs + 1
            InArg = True
         End If
         'Concatenate character to current argument.
         ArgArray(NumArgs) = ArgArray(NumArgs) & C
      Else
         'Found a space or tab.
         'Set InArg flag to False.
         InArg = False
      End If
   Next I
   'Resize array just enough to hold arguments.
   ReDim Preserve ArgArray(NumArgs)
   'Return Array in Function name.
   GetCommandLine = ArgArray()
End Function
```

InputBox Function Example

This example shows various ways to use the **InputBox** function to prompt the user to enter a value. If the x and y positions are omitted, the dialog box is automatically centered for the respective axes. The variable `MyValue` contains the value entered by the user if the user clicks **OK** or presses the ENTER key . If the user clicks **Cancel**, a zero-length string is returned.

```
Dim Message, Title, Default, MyValue
Message = "Enter a value between 1 and 3"' Set prompt.
Title = "InputBox Demo" ' Set title.
Default = "1"   ' Set default.
' Display message, title, and default value.
MyValue = InputBox(Message, Title, Default)

' Use Helpfile and context. The Help button is added automatically.
MyValue = InputBox(Message, Title, , , , "DEMO.HLP", 10)

' Display dialog box at position 100, 100.
MyValue = InputBox(Message, Title, Default, 100, 100)
```

MsgBox Function Example

This example uses the **MsgBox** function to display a critical-error message in a dialog box with Yes and No buttons. The No button is specified as the default response. The value returned by the **MsgBox** function depends on the button chosen by the user. This example assumes that `DEMO.HLP` is a Help file that contains a topic with a Help context number equal to `1000`.

```
Dim Msg, Style, Title, Help, Ctxt, Response, MyString
Msg = "Do you want to continue ?"  ' Define message.
Style = vbYesNo + vbCritical + vbDefaultButton2' Define buttons.
Title = "MsgBox Demonstration"     ' Define title.
Help = "DEMO.HLP" ' Define Help file.
Ctxt = 1000 ' Define topic
      ' context.
      ' Display message.
Response = MsgBox(Msg, Style, Title, Help, Ctxt)
If Response = vbYes Then' User chose Yes.
   MyString = "Yes"  ' Perform some action.
Else  ' User chose No.
   MyString = "No"   ' Perform some action.
End If
```

SendKeys Statement Example

This example uses the **Shell** function to run the Calculator application included with Microsoft Windows.It uses the **SendKeys** statement to send keystrokes to add some numbers, and then quit the Calculator. (To see the example, paste it into a procedure, then run the procedure. Because **AppActivate** changes the focus to the Calculator application, you can't single step through the code.)

```
Dim ReturnValue, I
ReturnValue = Shell("CALC.EXE", 1) ' Run Calculator.
AppActivate ReturnValue ' Activate the Calculator.
For I = 1 To 100  ' Set up counting loop.
   SendKeys I & "{+}", True  ' Send keystrokes to Calculator
Next I' to add each value of I.
SendKeys "=", True' Get grand total.
SendKeys "%{F4}", True  ' Send ALT+F4 to close Calculator.
```

AppActivate Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmAppActivateC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmAppActivateX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmAppActivateS"}

Activates an application window.

Syntax

AppActivate ***title***[, ***wait***]

The **AppActivate** statement syntax has these named arguments:

Part	Description
title	Required. String expression specifying the title in the title bar of the application window you want to activate. The task ID returned by the **Shell** function can be used in place of ***title*** to activate an application.
wait	Optional. **Boolean** value specifying whether the calling application has the focus before activating another. If **False** (default), the specified application is immediately activated, even if the calling application does not have the focus. If **True**, the calling application waits until it has the focus, then activates the specified application.

Remarks

The **AppActivate** statement changes the focus to the named application or window but does not affect whether it is maximized or minimized. Focus moves from the activated application window when the user takes some action to change the focus or close the window. Use the **Shell** function to start an application and set the window style.

In determining which application to activate, ***title*** is compared to the title string of each running application. If there is no exact match, any application whose title string begins with ***title*** is activated. If there is more than one instance of the application named by ***title***, one instance is arbitrarily activated.

Beep Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmBeepC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmBeepX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmBeepS"}

Sounds a tone through the computer's speaker.

Syntax

Beep

Remarks

The frequency and duration of the beep depend on your hardware and system software, and vary among computers.

Command Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctCommandC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctCommandX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctCommandS"}

Returns the argument portion of the command line used to launch Microsoft Visual Basic or an executable program developed with Visual Basic.

Syntax

Command

Remarks

When Visual Basic is launched from the command line, any portion of the command line that follows `/cmd` is passed to the program as the command-line argument. In the following example, `cmdlineargs` represents the argument information returned by the **Command** function.

```
VB /cmd cmdlineargs
```

For applications developed with Visual Basic and compiled to an .exe file, **Command** returns any arguments that appear after the name of the application on the command line. For example:

```
MyApp cmdlineargs
```

To find how command line arguments can be changed in the user interface of the application you're using, search Help for "command line arguments."

InputBox Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctInputBoxC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"VAFCTInputBoxX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctInputBoxS"}

Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns a **String** containing the contents of the text box.

Syntax

InputBox(*prompt*[, *title*] [, *default*] [, *xpos*] [, *ypos*] [, *helpfile*, *context*])

The **InputBox** function syntax has these named arguments:

Part	Description
prompt	Required. String expression displayed as the message in the dialog box. The maximum length of ***prompt*** is approximately 1024 characters, depending on the width of the characters used. If ***prompt*** consists of more than one line, you can separate the lines using a carriage return character (**Chr(**13**)**), a linefeed character (**Chr(**10**)**), or carriage return–linefeed character combination (**Chr(**13**)** & **Chr(**10**)**) between each line.
title	Optional. String expression displayed in the title bar of the dialog box. If you omit ***title***, the application name is placed in the title bar.
default	Optional. String expression displayed in the text box as the default response if no other input is provided. If you omit ***default***, the text box is displayed empty.
xpos	Optional. Numeric expression that specifies, in twips, the horizontal distance of the left edge of the dialog box from the left edge of the screen. If ***xpos*** is omitted, the dialog box is horizontally centered.
ypos	Optional. Numeric expression that specifies, in twips, the vertical distance of the upper edge of the dialog box from the top of the screen. If ***ypos*** is omitted, the dialog box is vertically positioned approximately one-third of the way down the screen.
helpfile	Optional. String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If ***helpfile*** is provided, ***context*** must also be provided.
context	Optional. Numeric expression that is the Help context number assigned to the appropriate Help topic by the Help author. If ***context*** is provided, ***helpfile*** must also be provided.

Remarks

When both ***helpfile*** and ***context*** are provided, the user can press F1 to view the Help topic corresponding to the **context**. Some host applications, for example, Microsoft Excel, also automatically add a **Help** button to the dialog box. If the user clicks **OK** or presses ENTER, the **InputBox** function returns whatever is in the text box. If the user clicks **Cancel**, the function returns a zero-length string ("").

Note To specify more than the first named argument, you must use **InputBox** in an expression. To omit some positional arguments, you must include the corresponding comma delimiter.

MsgBox Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctMsgBoxC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctMsgBoxX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctMsgBoxS"}

Displays a message in a dialog box, waits for the user to click a button, and returns an **Integer** indicating which button the user clicked.

Syntax

MsgBox(*prompt*[, *buttons*] [, *title*] [, *helpfile*, *context*])

The **MsgBox** function syntax has these named arguments:

Part	Description
prompt	Required. String expression displayed as the message in the dialog box. The maximum length of ***prompt*** is approximately 1024 characters, depending on the width of the characters used. If ***prompt*** consists of more than one line, you can separate the lines using a carriage return character (**Chr(**13**)**), a linefeed character (**Chr(**10**)**), or carriage return – linefeed character combination (**Chr(**13**)** & **Chr(**10**)**) between each line.
buttons	Optional. Numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If omitted, the default value for ***buttons*** is 0.
title	Optional. String expression displayed in the title bar of the dialog box. If you omit ***title***, the application name is placed in the title bar.
helpfile	Optional. String expression that identifies the Help file to use to provide context-sensitive Help for the dialog box. If ***helpfile*** is provided, ***context*** must also be provided.
context	Optional. Numeric expression that is the Help context number assigned to the appropriate Help topic by the Help author. If ***context*** is provided, ***helpfile*** must also be provided.

Settings

The ***buttons*** argument settings are:

Constant	Value	Description
vbOKOnly	0	Display **OK** button only.
vbOKCancel	1	Display **OK** and **Cancel** buttons.
vbAbortRetryIgnore	2	Display **Abort**, **Retry**, and **Ignore** buttons.
vbYesNoCancel	3	Display **Yes**, **No**, and **Cancel** buttons.
vbYesNo	4	Display **Yes** and **No** buttons.
vbRetryCancel	5	Display **Retry** and **Cancel** buttons.
vbCritical	16	Display **Critical Message** icon.
vbQuestion	32	Display **Warning Query** icon.
vbExclamation	48	Display **Warning Message** icon.
vbInformation	64	Display **Information Message**

		icon.
vbDefaultButton1	0	First button is default.
vbDefaultButton2	256	Second button is default.
vbDefaultButton3	512	Third button is default.
vbDefaultButton4	768	Fourth button is default.
vbApplicationModal	0	Application modal; the user must respond to the message box before continuing work in the current application.
vbSystemModal	4096	System modal; all applications are suspended until the user responds to the message box.
vbMsgBoxHelpButton	16384	Adds Help button to the message box
VbMsgBoxSetForeground	65536	Specifies the message box window as the foreground window
vbMsgBoxRight	524288	Text is right aligned
vbMsgBoxRtlReading	1048576	Specifies text should appear as right-to-left reading on Hebrew and Arabic systems

The first group of values (0–5) describes the number and type of buttons displayed in the dialog box; the second group (16, 32, 48, 64) describes the icon style; the third group (0, 256, 512) determines which button is the default; and the fourth group (0, 4096) determines the modality of the message box. When adding numbers to create a final value for the ***buttons*** argument, use only one number from each group.

Note These constants are specified by Visual Basic for Applications. As a result, the names can be used anywhere in your code in place of the actual values.

Return Values

Constant	Value	Description
vbOK	1	**OK**
vbCancel	2	**Cancel**
vbAbort	3	**Abort**
vbRetry	4	**Retry**
vbIgnore	5	**Ignore**
vbYes	6	**Yes**
vbNo	7	**No**

Remarks

When both ***helpfile*** and ***context*** are provided, the user can press F1 to view the Help topic corresponding to the **context**. Some host applications, for example, Microsoft Excel, also automatically add a **Help** button to the dialog box.

If the dialog box displays a **Cancel** button, pressing the ESC key has the same effect as clicking **Cancel**. If the dialog box contains a **Help** button, context-sensitive Help is provided for the dialog box. However, no value is returned until one of the other buttons is clicked.

Note To specify more than the first named argument, you must use **MsgBox** in an expression. To omit some positional arguments, you must include the corresponding comma delimiter.

SendKeys Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmSendKeysC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmSendKeysX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmSendKeysS"}

Sends one or more keystrokes to the active window as if typed at the keyboard.

Syntax

SendKeys ***string***[, ***wait***]

The **SendKeys** statement syntax has these named arguments:

Part	Description
string	Required. String expression specifying the keystrokes to send.
Wait	Optional. **Boolean** value specifying the wait mode. If **False** (default), control is returned to the procedure immediately after the keys are sent. If **True**, keystrokes must be processed before control is returned to the procedure.

Remarks

Each key is represented by one or more characters. To specify a single keyboard character, use the character itself. For example, to represent the letter A, use `"A"` for ***string***. To represent more than one character, append each additional character to the one preceding it. To represent the letters A, B, and C, use `"ABC"` for ***string***.

The plus sign (**+**), caret (**^**), percent sign (**%**), tilde (**~**), and parentheses **( )** have special meanings to **SendKeys**. To specify one of these characters, enclose it within braces (`{}`). For example, to specify the plus sign, use `{+}`. Brackets ([ ]) have no special meaning to **SendKeys**, but you must enclose them in braces. In other applications, brackets do have a special meaning that may be significant when dynamic data exchange (DDE) occurs. To specify brace characters, use `{{}` and `{}}`.

To specify characters that aren't displayed when you press a key, such as ENTER or TAB, and keys that represent actions rather than characters, use the codes shown below:

Key	Code
BACKSPACE	`{BACKSPACE}`, `{BS}`, or `{BKSP}`
BREAK	`{BREAK}`
CAPS LOCK	`{CAPSLOCK}`
DEL or DELETE	`{DELETE}` or `{DEL}`
DOWN ARROW	`{DOWN}`
END	`{END}`
ENTER	`{ENTER}`or ~
ESC	`{ESC}`
HELP	`{HELP}`
HOME	`{HOME}`
INS or INSERT	`{INSERT}` or `{INS}`
LEFT ARROW	`{LEFT}`
NUM LOCK	`{NUMLOCK}`
PAGE DOWN	`{PGDN}`
PAGE UP	`{PGUP}`
PRINT SCREEN	`{PRTSC}`
RIGHT ARROW	`{RIGHT}`

SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}

To specify keys combined with any combination of the SHIFT, CTRL, and ALT keys, precede the key code with one or more of the following codes:

Key	Code
SHIFT	+
CTRL	^
ALT	%

To specify that any combination of SHIFT, CTRL, and ALT should be held down while several other keys are pressed, enclose the code for those keys in parentheses. For example, to specify to hold down SHIFT while E and C are pressed, use "+(EC)". To specify to hold down SHIFT while E is pressed, followed by C without SHIFT, use "+EC".

To specify repeating keys, use the form {key number}. You must put a space between key and number. For example, {LEFT 42} means press the LEFT ARROW key 42 times; {h 10} means press H 10 times.

Note You can't use **SendKeys** to send keystrokes to an application that is not designed to run in Microsoft Windows. **Sendkeys** also can't send the PRINT SCREEN key {PRTSC} to any application.

Character Set (0 – 127)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscANSITableC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscANSITableS"}

0	·	32	[space]	64	@	96	`
1	·	33	!	65	A	97	a
2	·	34	"	66	B	98	b
3	·	35	#	67	C	99	c
4	·	36	$	68	D	100	d
5	·	37	%	69	E	101	e
6	·	38	&	70	F	102	f
7	·	39	'	71	G	103	g
8	* *	40	(	72	H	104	h
9	* *	41	)	73	I	105	i
10	* *	42	*	74	J	106	j
11	·	43	+	75	K	107	k
12	·	44	,	76	L	108	l
13	* *	45	-	77	M	109	m
14	·	46	.	78	N	110	n
15	·	47	/	79	O	111	o
16	·	48	0	80	P	112	p
17	·	49	1	81	Q	113	q
18	·	50	2	82	R	114	r
19	·	51	3	83	S	115	s
20	·	52	4	84	T	116	t
21	·	53	5	85	U	117	u
22	·	54	6	86	V	118	v
23	·	55	7	87	W	119	w
24	·	56	8	88	X	120	x
25	·	57	9	89	Y	121	y
26	·	58	:	90	Z	122	z
27	·	59	;	91	[	123	{
28	·	60	<	92	\	124	|
29	·	61	=	93	]	125	}
30	·	62	>	94	^	126	~
31	·	63	?	95	_	127	•

- These characters aren't supported by Microsoft Windows.

* * Values 8, 9, 10, and 13 convert to backspace, tab, linefeed, and carriage return characters, respectively. They have no graphical representation but, depending on the application, can affect the visual display of text.

IMEStatus Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctIMEStatusC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctIMEStatusX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctIMEStatusS"}

Returns an **Integer** specifying the current Input Method Editor (IME) mode of Microsoft Windows; available in Far East versions only.

Syntax

IMEStatus

Return Values

The return values for the Japanese locale are as follows:

Constant	Value	Description
vbIMENoOP	0	No IME installed
vbIMEOn	1	IME on
vbIMEOff	2	IME off
vbIMEDisable	3	IME disabled
vbIMEHiragana	4	Hiragana double-byte characters (DBC)
vbIMEKatakanaDbl	5	Katakana DBC
vbIMEKatakanaSng	6	Katakana single-byte characters (SBC)
vbIMEAlphaDbl	7	Alphanumeric DBC
vbIMEAlphaSng	8	Alphanumeric SBC

The return values for the Chinese (traditional and simplified) locale are as follows:

Constant	Value	Description
vbIMENoOP	0	No IME installed
vbIMEOn	1	IME on
vbIMEOff	2	IME off

For the Korean locale, the first five bits of the return are set as follows:

Bit	Value	Description	Value	Description
0	0	No IME installed	1	IME installed
1	0	IME disabled	1	IME enabled
2	0	IME English mode	1	Hangeul mode
3	0	Banja mode (SB)	1	Junja mode (DB)
4	0	Normal mode	1	Hanja conversion mode

Can't execute code in break mode

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantExecCodeInBreakModeS"}

You enter break mode when you suspend execution of code. This error has the following causes and solutions:

- You tried to run code from the **Macro** dialog box. However, Visual Basic was already running code, although the code was suspended in break mode.

  You may have entered break mode without knowing it, for example, if a syntax error or run-time error occurred. Continue running the suspended code, or terminate its execution before you run code from the **Macro** dialog box. You can fix the error and choose **Continue**, or you can return to the **Macro** dialog box and restart the macro.

For additional information, select the item in question and press F1.

Can't call Friend procedure on an object that isn't an instance of the defining class (Error 97)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadCallToFriendFunctionS"}

A **Friend** procedure is callable from a module that is outside the class, but part of the project within which the class is defined. This error has the following causes and solutions:

- You tried to call the **Friend** procedure of a class. Although your reference variable is of the proper type, the variable points to an instance that isn't an instance of the class.

  For example, this can occur if there are two classes, class*ics* and class*y* (that implements class*y*), but you mistakenly assign the instance of *classy* to the instance of class*ics*.

For additional information, select the item in question and press F1.

Can't exit design mode because control can't be created

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCannotExitDesignModeS"}

All controls must be instantiated before you can exit design mode. This error has the following causes and solutions:

- The control specified in the error message dialog box could not be created.

  Code can only run after all controls are instantiated and properly connected. Make sure every file needed for the control is available before trying again.

For additional information, select the item in question and press F1.

Object does not source Automation events

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoDefaultSourceS"}

An object must provide a default source interface so that you can write event procedures for its events. This error has the following causes and solutions:

- You tried to write an event procedure for an event of an object, but that event isn't available outside the object.

  See your object’s documentation for suggestions on less direct ways to deal with the event you are interested in.

For additional information, select the item in question and press F1.

Operation not allowed in DLL

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgOperationNotAllowedInDLLC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgOperationNotAllowedInDLLS"}

Not all Visual Basic statements are allowed within a dynamic-link library (DLL). This error has the following causes and solutions:

- You tried to create a DLL from a class that contains a statement that can't be executed from a DLL.

  Check your class and remove any statements that can't be executed within a DLL, for example, the **End statement**.

For additional information, select the item in question and press F1.

Invalid outside Enum

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgInvOutsideEnumC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvOutsideEnumS"}

An **Enum** is a data type that can be used to create groups of related constants (called enumerations). This error has the following causes and solutions:

- You used an **End Enum** where it wasn't part of an **Enum** definition.

  Check for text that may appear between the body of the **Enum** definition and the **End Enum** statement.

For additional information, select the item in question and press F1.

This document was opened with Macros Disabled

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantExitVirusDesignModeS"}

A host application may disable or enable macros. This error has the following causes and solutions:

- You opened the document with **Macros Disabled**.

Close the document, and then reopen it with **Enable Macros**.

For additional information, select the item in question and press F1.

The library containing this symbol is not referenced by the current project, so the symbol is undefined. Would you like to add a reference to the containing library now?

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLibNotReferencedS"}

Type and object information is contained in libraries. This error has the following causes and solutions:

- A definition for this symbol exists in a type library.

  If you want to define the symbol as contained in the library mentioned, click **OK** to add the reference to the library.

For additional information, select the item in question and press F1.

Object does not have a Property Let procedure

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgLetNotSupportedS"}

You can't assign a value to a property unless it has exposed a **Property Let** method. This error has the following causes and solutions:

- You tried to assign a value to a property that hasn't exposed a **Property Let** method.

  You can't directly assign a value to this property. If you created the class, you can modify the interface by exposing a **Property Let** method. Otherwise, check the component's documentation to determine if there is an indirect method for assigning the value.

For additional information, select the item in question and press F1.

No Help available

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNoHelpS"}

Not all errors have an associated help topic. This error has the following causes and solutions:

- You generated an error for which no Help exists.
  Check the Readme file. Help for late-breaking errors is often available through the Readme file.

Can't sink this object's events because it's already firing events to the maximum number of supported event recipients (Error 96)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgAdviseLimitS"}

Each object specifies the maximum number of simultaneous recipients to which it can fire events. This error has the following causes and solutions:

- You tried to use **Set** for a **WithEvents** variable for a control on a form.

  You can't use **Set** for a **WithEvents** variable because the number of recipients allowed for these events.

For additional information, select the item in question and press F1.

Can't find DLL entry point in specified DLL

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantFindDLLEntryPointS"}

An entry point is the name of a DLL procedure or the ordinal representing the procedure. This error has the following causes and solutions:

- A type library incorrectly described the entry point, perhaps misspelling the name or specifying the ordinal incorrectly.

Contact the vendor for a corrected type library.

For additional information, select the item in question and press F1.

Could not execute specified program

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantExecuteProgramS"}

When you create a native-code executable file, some extra programs must run. This error has the following causes and solutions:

- Memory or system resources were insufficient to run the code generator or linker.
  Close as many running applications as possible to free memory and other system resources.
- Visual Basic was installed incorrectly.

Reinstall Visual Basic.

For additional information, select the item in question and press F1.

Duplicate resources with same type and name

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgDuplicateResourceS"}

A Microsoft Windows resource file typically contains bitmaps, text strings, and other similar data used by an application. This error has the following causes and solutions:

- The Windows resource file you are trying to use in your project contains two or more resources with the same type and name, or the file contains a resource that Visual Basic automatically creates.

  Use another resource file, or recreate the invalid resource file and delete one of the duplicate resources.

For additional information, select the item in question and press F1.

A property or method call cannot include a reference to a private object, either as an argument or as a return value (Error 98)

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantPassPrivateObjectS"}

Private objects should never be passed outside a project. The following, all of which are prohibited, are possible causes for the error:

- A client invoked a property or method of an out-of-process component and attempted to pass a reference to a private object as one of the arguments. A client invoked a property or method of an out-of-process component and the component attempted to return a reference to a private object, or to assign such a reference to a **ByRef** argument.
- An out-of-process component has invoked a call-back method on its client and attempted to pass a reference to a private object
- An out-of-process component attempted to pass a reference to a private object as an argument of an event it was raising.
- A client attempted to assign a private object reference to a **ByRef** argument of an event it was handling.

Note that although Visual Basic prevents you from passing references to nonvisual private objects across processes, there are some cases in which Visual Basic can't detect this error and thus can't prevent it. Private objects are not designed to be used outside your project. If you pass them to a client, you may jeopardize program stability and cause incompatibility with future versions of Visual Basic. If you need to pass a private class of your own to a client, set the **Instancing** property to a value other than **Private**.

For additional information, select the item in question and press F1.

Assignment to constant not permitted

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantAssignToConstS"}

A constant represents a read-only value. This error has the following causes and solutions:

- You tried to assign a new value to a variable declared with **Const**, or to a type library constant.

  If you need to assign a new value, declare an ordinary variable of the type desired and assign your value to that variable. If you need a variable with a restricted set of values, you can declare an enumeration, using the **Enum** statement.

For additional information, select the item in question and press F1.

Can't have paramarrays with optional arguments

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgParamArrayWithOptArgsS"}

You can use a **ParamArray** to define procedures that accept variable-length argument lists. An **Optional** argument is one that the caller can supply or omit without generating an error. This error has the following causes and solutions:

- You used both the **ParamArray** and **Optional** keywords in the same parameter list.

  Remove either the **Optional** or **ParamArray** parameter. You cannot use both keywords in the same parameter list. Either approach can be used for the type of parameter list you want.

For additional information, select the item in question and press F1.

File specified was not found

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgFileNotFoundWithNameS"}

Check the message for the name of the file that couldn't be found. This error has the following causes and solutions:

- The file named in the error message could not be found as specified.
  Make sure the drive name and path, as well as the filename, are specified correctly.

For additional information, select the item in question and press F1.

Invalid optional parameter type

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadOptParamTypeS"}

Not all Visual Basic types are permitted for **Optional** parameters. This error has the following causes and solutions:

- A parameter is defined with an invalid data type.

  For example, you can't have an **Optional** parameter of user-defined type. You may be able to accomplish your goal using an array of **Variant** type, since the elements of an array of **Variant** can store data of different types.

For additional information, select the item in question and press F1.

Invalid Template

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidTemplateS"}

A derived class of a template can't be a template. This error has the following causes and solutions:

- You tried to create an template component, but tried to derive it from a template.
  You can't derive a template class from a template.

For additional information, select the item in question and press F1.

Only valid in object module

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgClassModuleOnlyS"}

Some Visual Basic statements or constructs are valid only in object modules (forms, class modules, etc.). This error has the following causes and solutions:

- You tried to use an invalid statement or construct (for example, the **Implements** statement, **WithEvents** keyword, or an event sink) in a non-class module.

  Place statements that generate this error in an object module.

For additional information, select the item in question and press F1.

Requested type library or wizard is not a VBA project

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgNotAProjectS"}

Not all elements available in Visual Basic are able to be both read from and written to. This error has the following causes and solutions:

- You tried to write to an Addin or a type library from within a Visual Basic project.
  You can't directly write to a prepackaged Addin or a type library as part of a Visual Basic project.

For additional information, select the item in question and press F1.

This edit requires a Reset

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgMustResetS"}

In most cases, you can edit code and continue running the code from the point of the edit. This error has the following causes and solutions:

- You made a change to your code that requires you to restart the code from the initialization point.

  This might occur if you broke up the code in a procedure and the current execution line is no longer contained in the same procedural context as it was when execution was suspended.

For additional information, select the item in question and press F1.

Translation failed. Please check the trnslate.log file for more information.

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgTranslationErrS"}

Current versions of Visual Basic only perform translation for backward compatibility. This error has the following causes and solutions:

- The translation failed unexpectedly.

  This usually occurs because of a broken type library reference. More details should be available in the specified translation log.

For additional information, select the item in question and press F1.

Wizards can't reference projects

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgWizCantRefProjS"}

A Wizard can reference another Wizard, but cannot reference a project. This error has the following causes and solutions:

- During a Make Wizard command, a reference to a project was found.
  Remove the reference to the project.

For additional information, select the item in question and press F1.

Unexpected compile error

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgUnexpectedCompileErrorS"}

This error occurs when a completely unanticipated error occurs during compilation.

Invalid use of base class name

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgBadBaseUsageS"}

You cannot use the name of a base class by itself. This error has the following causes and solutions:

- You tried to use the name of a base class by itself without making clear that you were trying to access the base class' default member.

  Place the base-class name within parentheses to indicate you want to access the default member.
- You used the base-class name in an expression but the member you were trying to access was ambiguously specified.

  Use a disambiguator (for example, an exclamation point) between the base-class name and the member you are interested in.
- You used the base-class name in a **Set** statement as though it contained a reference to the class.

  Use the base-class name to retrieve a reference for example, using GetObject.

For additional information, select the item in question and press F1.

Invalid event name

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvalidEventNameS"}

Since an event procedure names are constructed by joining the object name to the event name with an underscore. This error has the following causes and solutions:

- You used an underscore as part of the event name.
  Remove the underscore from the event name.

For additional information, select the item in question and press F1.

Event not found

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEventNotFoundS"}

An event specified in a **RaiseEvent** statement must correspond to a defined event. This error has the following causes and solutions:

- You specified a name in a **RaiseEvent** statement, but the event definintion cannot be found. Make sure the event name is spelled correctly.

For additional information, select the item in question and press F1.

Search string must be specified

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgEmptyObjbrwSearchTextS"}

You have to enter a string when searching in the object browser. This error has the following causes and solutions:

- You initiated a search in the Object Browser, but didn't specify text to search for.
  Enter a search string.

For additional information, select the item in question and press F1.

Cannot display specified name because it is hidden

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantShowHiddenObjbrwMemS"}

Some names exist in a type library, but are marked as hidden. This error has the following causes and solutions:

- You specified a name that is in the type library, but it is marked as hidden.

  You cannot normally view hidden type library members. Choose Show Hidden Members on the object browser context menu to make hidden members visible. You can then view the member information.

For additional information, select the item in question and press F1.

Cannot jump to specified type because it is in the specified library, which is not currently referenced

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgCantShowUnrefedObjbrwMemS"}

This error has the following causes and solutions:

- You tried to specify a type in a library that isn't reference within the project.
  Set a reference to the type library through the References dialog.

For additional information, select the item in question and press F1.

Invalid inside Enum

{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgInvInsideEnumS"}

Not all types are valid within an enumeration definition. This error has the following causes and solutions:

- You tried to specify a string or some other invalid type as the value of an **Enum** member.

  The constant expression used to specify an **Enum** member must evaluate to type **Long** or anothe **Enum** type.

For additional information, select the item in question and press F1.

Keyword Not Found

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscKeywordNotFoundC"}

The keyword you selected can't be found in Visual Basic Help. You may have misspelled the keyword, selected too much or too little text, or asked for help on a word that isn't a valid Visual Basic keyword.

They keyword you want help on may be contained within an object library that is not referenced. Make sure references are set to the appropriate object libraries for all objects used in your code.

The easiest way to get help on a specific keyword is to position the insertion point anywhere within the keyword and press F1. You don't have to select the keyword. In fact, if you select only a portion of the keyword, or more than a single word, Help won't find what you're looking for.

The **Value** property topic is displayed when you press F1 with the insertion point between the "a" and the "l" in the **Value** keyword as shown in the following example.

```
Worksheets(1).Range ("A2").value=3.14159
```

To use the built-in Help **Search** dialog box, click **Contents and Index** on the **Help** menu.

Collection (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscCollectionModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscCollectionModuleS"}

The **Collection** module contains procedures used to perform operations on the **Collection** object. These constants can be used anywhere in your code.

To get Help on a particular procedure

1. Select the procedure from the **Members Of 'Collection'** list.
2. Click the  button.

ColorConstants (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscColorConstantsBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscColorConstantsBrowserStringS"}

The **ColorConstants** module contains predefined color constants. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members Of 'ColorConstants'** list.
2. Click the  button.

Constants (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscConstantsModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscConstantsModuleS"}

The **Constants** module contains miscellaneous constants. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members Of 'Constants'** list.
2. Click the  button.

Conversion (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscConversionModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscConversionModuleS"}

The **Conversion** module contains the procedures used to perform various conversion operations. These constants can be used anywhere in your code.

To get Help on a particular procedure

1. Select the procedure from the **Members Of 'Conversion'** list.
2. Click the  button.

Note When you use **Variant** variables, explicit data-type conversions are unnecessary.

DateTime (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscDateTimeModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscDateTimeModuleS"}

The **DateTime** module contains the procedures and properties used in date and time operations. These constants can be used anywhere in your code.

To get Help on a particular procedure or property

1. Select the procedure from the **Members Of 'DateTime'** list.
2. Click the  button.

ErrObject (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscErrObjectModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscErrObjectModuleS"}

The **ErrObject** module contains properties and procedures used to identify and handle run-time errors using the **Err** object. These constants can be used anywhere in your code.

To get Help on a particular property or procedure

1. Select the property or procedure from the **Members of 'ErrObject'** list.
2. Click the  button.

FileSystem (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscFileSystemModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscFileSystemModuleS"}

The **FileSystem** module contains the procedures used to perform file, directory or folder, and system operations. These constants can be used anywhere in your code.

To get Help on a particular procedure

1. Select the procedure from the **Members of 'FileSystem'** list.
2. Click the  button.

Financial (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscFinancialModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscFinancialModuleS"}

The **Financial** module contains procedures used to perform financial operations. These constants can be used anywhere in your code.

To get Help on a particular procedure

1. Select the procedure from the **Members of 'Financial'** list.
2. Click the  button.

Global (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscGlobalModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscGlobalModuleS"}

The **Global** module contains procedures and properties used to perform operations on the **UserForm** object. These constants can be used anywhere in your code.

To get Help on a particular property or procedure

1. Select the procedure from the **Members Of 'Global'** list.
2. Click the ▶ button.

Information (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscInformationModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscInformationModuleS"}

The **Information** module contains the procedures used to return, test for, or verify information. These constants can be used anywhere in your code.

To get Help on a particular procedure

1. Select the procedure from the **Members Of 'Information'** list.
2. Click the ▶ button.

Interaction (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscInteractionModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscInteractionModuleS"}

The **Interaction** module contains procedures used to interact with objects, applications, and systems. These constants can be used anywhere in your code.

To get Help on a particular procedure

1. Select the procedure from the **Members of 'Interaction'** list.
2. Click the ▶ button.

KeycodeConstants (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscKeyCodeConstantsBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscKeyCodeConstantsBrowserStringS"}

The **KeyCodeConstants** module contains predefined keycode constants that can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'KeyCodeConstants'** list.
2. Click the ▶ button.

Math (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscMathModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscMathModuleS"}

The **Math** module contains procedures used to perform mathematical operations. These constants can be used anywhere in your code.

To get Help on a particular procedure

1. Select the procedure from the **Members of 'Math'** list.
2. Click the ▶ button.

String (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscStringModuleC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscStringModuleS"}

The **String** module contains procedures used to perform string operations. These constants can be used anywhere in your code.

To get Help on a particular procedure

1. Select the procedure from the **Members of 'String'** list.
2. Click the ▶ button.

SystemColorConstants (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscSystemColorConstantsBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscSystemColorConstantsBrowserStringS"}

The **SystemColorConstants** module contains constants that identify various parts of the graphical user interface. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'SystemColorConstants'** list.
2. Click the ▶ button.

VbAppWinStyle (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVbAppWinStyleBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVbAppWinStyleBrowserStringS"}

The **VbAppWinStyle** enumeration contains constants used by the **Shell** function to control the style of an application window. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'VbAppWinStyle'** list.
2. Click the ▶ button.

VbCalendar (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVbCalendarBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVbCalendarBrowserStringS"}

The **VbCalendar** enumeration contains constants used to determine the type of calendar used by Visual Basic. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'VbCalendar'** list.
2. Click the ▶ button.

VbCompareMethod (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVbCompareMethodBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVbCompareMethodBrowserStringS"}

The **VbCompareMethod** enumeration contains constants used to determine the way strings are compared when using the **Instr** and **StrComp** functions. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'VbCompareMethod'** list.
2. Click the ▶ button.

VbDayOfWeek (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVbDayOfWeekBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVbDayOfWeekBrowserStringS"}

The **VbDayOfWeek** enumeration contains constants used to identify specific days of the week when using the **DateDiff**, **DatePart**, and **Weekday** functions. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'VbDayOfWeek'** list.
2. Click the ▶ button.

VbFileAttribute (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVbFileAttributeBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVbFileAttributeBrowserStringS"}

The **VbFileAttribute** enumeration contains constants used to identify file attributes used in the **Dir**, **GetAttr**, and **SetAttr** functions. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'VbFileAttribute'** list.
2. Click the ▶ button.

VbFirstWeekOfYear (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVbFirstWeekOfYearBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVbFirstWeekOfYearBrowserStringS"}

The **VbFirstWeekOfYear** enumeration contains constants used to identify how the first week of a year is determined when using the **DateDiff** and **DatePart** functions. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'VbFirstWeekOfYear'** list.
2. Click the ▶ button.

VbIMEStatus (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVbIMEStatusBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVbIMEStatusBrowserStringS"}

Available only in Far East versions, the **VbIMEStatus** enumeration contains constants used to identify the Input Method Editor (IME) when using the **IMEStatus** function. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'VbIMEStatus'** list.
2. Click the ▶ button.

VbMsgBoxResult (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVbMsgBoxResultBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVbMsgBoxResultBrowserStringS"}

The **VbMsgBoxResult** enumeration contains constants used to identify which button was pressed on a message box displayed using the **MsgBox** function. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'VbMsgBoxResult'** list.
2. Click the ▶ button.

VbMsgBoxStyle (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVbMsgBoxStyleBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVbMsgBoxStyleBrowserStringS"}

The **VbMsgBoxStyle** enumeration contains constants used to specify the behavior of a message box, along with symbols and buttons that appear on it, when displayed using the **MsgBox** function. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'VbMsgBoxStyle'** list.
2. Click the ▶ button.

VbQueryClose (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVbQueryCloseBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVbQueryCloseBrowserStringS"}

The **VbQueryClose** enumeration contains constants used to identify what caused the **QueryClose** event to occur. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'VbQueryClose'** list.
2. Click the ▶ button.

VbStrConv (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVbStrConvBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVbStrConvBrowserStringS"}

The **VbStrConv** enumeration contains constants used to identify the type of string conversion to be performed by the **StrConv** function. These constants can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'VbStrConv'** list.
2. Click the ▶ button.

VbVarType (Object Browser)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscVbVarTypeBrowserStringC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscVbVarTypeBrowserStringS"}

The **VbVarType** enumeration contains constants used to identify the various types of data that can be contained in a **Variant**. These constants match the return values of the **VarType** function and can be used anywhere in your code.

To get Help on a particular constant

1. Select the constant from the **Members of 'VbVarType'** list.
2. Click the ▶ button.

Collection Object Example

This example creates a **Collection** object (`MyClasses`), and then creates a dialog box in which users can add objects to the collection. To see how this works, choose the **Class Module** command from the **Insert** menu and declare a public variable called `InstanceName` at module level of Class1 (type **Public** `InstanceName)` to hold the names of each instance. Leave the default name as Class1. Copy and paste the following code into the General section of another module, and then start it with the statement `ClassNamer` in another procedure. (This example only works with host applications that support classes.)

```
Sub ClassNamer()
   Dim MyClasses As New Collection ' Create a Collection object.
   Dim Num  ' Counter for individualizing keys.
   Dim Msg As String ' Variable to hold prompt string.
   Dim TheName, MyObject, NameList ' Variants to hold information.
   Do
      Dim Inst As New Class1 ' Create a new instance of Class1.
      Num = Num + 1  ' Increment Num, then get a name.
      Msg = "Please enter a name for this object." & Chr(13) _
       & "Press Cancel to see names in collection."
      TheName = InputBox(Msg, "Name the Collection Items")
      Inst.InstanceName = TheName  ' Put name in object instance.
      ' If user entered name, add it to the collection.
      If Inst.InstanceName <> "" Then
         ' Add the named object to the collection.
         MyClasses.Add item := Inst, key := CStr(Num)
      End If
      ' Clear the current reference in preparation for next one.
      Set Inst = Nothing
   Loop Until TheName = ""
   For Each MyObject In MyClasses  ' Create list of names.
      NameList = NameList & MyObject.InstanceName & Chr(13)
   Next MyObject
   ' Display the list of names in a message box.
   MsgBox NameList, , "Instance Names In MyClasses Collection"

   For Num = 1 To MyClasses.Count  ' Remove name from the collection.
      MyClasses.Remove 1   ' Since collections are reindexed
            ' automatically, remove the first
   Next     ' member on each iteration.
End Sub
```

Err Object Example

This example uses the properties of the **Err** object in constructing an error-message dialog box. Note that if you use the **Clear** method first, when you generate a Visual Basic error with the **Raise** method, Visual Basic's default values become the properties of the **Err** object.

```
Dim Msg
' If an error occurs, construct an error message
On Error Resume Next ' Defer error handling.
Err.Clear
Err.Raise 6 ' Generate an "Overflow" error.
' Check for error, then show message.
If Err.Number <> 0 Then
   Msg = "Error # " & Str(Err.Number) & " was generated by " _
         & Err.Source & Chr(13) & Err.Description
   MsgBox Msg, , "Error", Err.Helpfile, Err.HelpContext
End If
```

Collection Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaobjCollectionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaobjCollectionX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"vaobjCollectionP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"vaobjCollectionM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"vaobjCollectionE"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaobjCollectionS"}

A **Collection** object is an ordered set of items that can be referred to as a unit.

Remarks

The **Collection** object provides a convenient way to refer to a related group of items as a single object. The items, or members, in a collection need only be related by the fact that they exist in the collection. Members of a collection don't have to share the same data type.

A collection can be created the same way other objects are created. For example:

```
Dim X As New Collection
```

Once a collection is created, members can be added using the **Add** method and removed using the **Remove** method. Specific members can be returned from the collection using the **Item** method, while the entire collection can be iterated using the **For Each...Next** statement.

Debug Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaobjDebugC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaobjDebugX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"vaobjDebugP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"vaobjDebugM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"vaobjDebugE"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaobjDebugS"}

The **Debug** object sends output to the **Immediate** window at run time.

Err Object

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaobjErrC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaobjErrX":1}
{ewc HLP95EN.DLL,DYNALINK,"Properties":"vaobjErrP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"vaobjErrM"}
{ewc HLP95EN.DLL,DYNALINK,"Events":"vaobjErrE"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaobjErrS"}

Contains information about run-time errors.

Remarks

The properties of the **Err** object are set by the generator of an error—Visual Basic, an object, or the Visual Basic programmer.

The default property of the **Err** object is **Number**. Because the default property can be represented by the object name **Err**, earlier code written using the **Err** function or **Err** statement doesn't have to be modified.

When a run-time error occurs, the properties of the **Err** object are filled with information that uniquely identifies the error and information that can be used to handle it. To generate a run-time error in your code, use the **Raise** method.

The **Err** object's properties are reset to zero or zero-length strings ("") after any form of the **Resume** or **On Error** statement and after an **Exit Sub**, **Exit Function**, or **Exit Property** statement within an error-handling routine. The **Clear** method can be used to explicitly reset **Err**.

Use the **Raise** method, rather than the **Error** statement, to generate run-time errors for a class module. Using the **Raise** method in other code depends on the richness of the information you want to return. In code that uses **Error** statements instead of the **Raise** method to generate errors, the properties of the **Err** object are assigned the following default values when **Error** is executed:

Property	Value
Number	Value specified as argument to **Error** statement. Can be any valid error number.
Source	Name of the current Visual Basic project.
Description	A string corresponding to the return of the **Error** function for the specified **Number**, if this string exists. If the string doesn't exist, **Description** contains "Application-defined or object-defined error".
HelpFile	The fully qualified drive, path, and file name of the Visual Basic Help file.
HelpContext	The Visual Basic Help file context ID for the error corresponding to the **Number** property.
LastDLLError	On 32-bit Microsoft Windows operating systems only, contains the system error code for the last call to a dynamic-link library (DLL). The **LastDLLError** property is read-only.

You don't have to change existing code that uses the **Err** object and the **Error** statement. However, using both the **Err** object and the **Error** statement can result in unintended consequences. For example, even if you fill in properties for the **Err** object, they are reset to the default values indicated in the preceding table as soon as the **Error** statement is executed. Although you can still use the **Error** statement to generate Visual Basic run-time errors, it is retained principally for compatibility with existing code. Use the **Err** object, the **Raise** method, and the **Clear** method for system errors and in new code, especially for class modules.

The **Err** object is an intrinsic object with global scope. There is no need to create an instance of it in your code.

^ Operator Example

This example uses the **^** operator to raise a number to the power of an exponent.

```
Dim MyValue
MyValue = 2 ^ 2' Returns 4.
MyValue = 3 ^ 3 ^ 3  ' Returns 19683.
MyValue = (-5) ^ 3' Returns -125.
```

+ Operator Example

This example uses the **+** operator to sum numbers. The **+** operator can also be used to concatenate strings. However, to eliminate ambiguity, you should use the **&** operator instead. If the components of an expression created with the **+** operator include both strings and numerics, the arithmetic result is assigned. If the components are exclusively strings, the strings are concatenated.

```
Dim MyNumber, Var1, Var2
MyNumber = 2 + 2  ' Returns 4.
MyNumber = 4257.04 + 98112 ' Returns 102369.04.

Var1 = "34": Var2 = 6' Initialize mixed variables.
MyNumber = Var1 + Var2  ' Returns 40.

Var1 = "34": Var2 = "6" ' Initialize variables with strings.
MyNumber = Var1 + Var2  ' Returns "346" (string concatenation).
```

- Operator Example

This example uses the **-** operator to calculate the difference between two numbers.

```
Dim MyResult
MyResult = 4 - 2   ' Returns 2.
MyResult = 459.35 - 334.90 ' Returns 124.45.
```

* Operator Example

This example uses the * operator to multiply two numbers.

```
Dim MyValue
MyValue = 2 * 2' Returns 4.
MyValue = 459.35 * 334.90  ' Returns 153836.315.
```

/ Operator Example

This example uses the **/** operator to perform floating-point division.

```
Dim MyValue
MyValue = 10 / 4   ' Returns 2.5.
MyValue = 10 / 3   ' Returns 3.333333.
```

\ Operator Example

This example uses the \ operator to perform integer division.

```
Dim MyValue
MyValue = 11 \ 4  ' Returns 2.
MyValue = 9 \ 3' Returns 3.
MyValue = 100 \ 3 ' Returns 33.
```

Mod Operator Example

This example uses the **Mod** operator to divide two numbers and return only the remainder. If either number is a floating-point number, it is first rounded to an integer.

```
Dim MyResult
MyResult = 10 Mod 5   ' Returns 0.
MyResult = 10 Mod 3   ' Returns 1.
MyResult = 12 Mod 4.3' Returns 0.
MyResult = 12.6 Mod 5' Returns 3.
```

& Operator Example

This example uses the **&** operator to force string concatenation.

```
Dim MyStr
MyStr = "Hello" & " World" ' Returns "Hello World".
MyStr = "Check " & 123 & " Check"  ' Returns "Check 123 Check".
```

Comparison Operators Example

This example shows various uses of comparison operators, which you use to compare expressions.

```
Dim MyResult, Var1, Var2
MyResult = (45 < 35) ' Returns False.
MyResult = (45 = 45) ' Returns True.
MyResult = (4 <> 3)  ' Returns True.
MyResult = ("5" > "4")  ' Returns True.

Var1 = "5": Var2 = 4 ' Initialize variables.
MyResult = (Var1 > Var2)' Returns True.

Var1 = 5: Var2 = Empty
MyResult = (Var1 > Var2)' Returns True.

Var1 = 0: Var2 = Empty
MyResult = (Var1 = Var2)' Returns True.
```

And Operator Example

This example uses the **And** operator to perform a logical conjunction on two expressions.

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null     ' Initialize variables.
MyCheck = A > B And B > C  ' Returns True.
MyCheck = B > A And B > C  ' Returns False.
MyCheck = A > B And B > D  ' Returns Null.
MyCheck = A And B ' Returns 8 (bitwise comparison).
```

Eqv Operator Example

This example uses the **Eqv** operator to perform logical equivalence on two expressions.

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null     ' Initialize variables.
MyCheck = A > B Eqv B > C  ' Returns True.
MyCheck = B > A Eqv B > C  ' Returns False.
MyCheck = A > B Eqv B > D  ' Returns Null.
MyCheck = A Eqv B ' Returns -3 (bitwise comparison).
```

Imp Operator Example

This example uses the **Imp** Operator to perform logical implication on two expressions.

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null     ' Initialize variables.
MyCheck = A > B Imp B > C  ' Returns True.
MyCheck = A > B Imp C > B  ' Returns False.
MyCheck = B > A Imp C > B  ' Returns True.
MyCheck = B > A Imp C > D  ' Returns True.
MyCheck = C > D Imp B > A  ' Returns Null.
MyCheck = B Imp A ' Returns -1 (bitwise comparison).
```

Not Operator Example

This example uses the **Not** operator to perform logical negation on an expression.

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null     ' Initialize variables.
MyCheck = Not(A > B)  ' Returns False.
MyCheck = Not(B > A)  ' Returns True.
MyCheck = Not(C > D)  ' Returns Null.
MyCheck = Not A' Returns -11 (bitwise comparison).
```

Or Operator Example

This example uses the **Or** operator to perform logical disjunction on two expressions.

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null     ' Initialize variables.
MyCheck = A > B Or B > C' Returns True.
MyCheck = B > A Or B > C' Returns True.
MyCheck = A > B Or B > D' Returns True.
MyCheck = B > D Or B > A' Returns Null.
MyCheck = A Or B  ' Returns 10 (bitwise comparison).
```

Xor Operator Example

This example uses the **Xor** operator to perform logical exclusion on two expressions.

```
Dim A, B, C, D, MyCheck
A = 10: B = 8: C = 6: D = Null     ' Initialize variables.
MyCheck = A > B Xor B > C  ' Returns False.
MyCheck = B > A Xor B > C  ' Returns True.
MyCheck = B > A Xor C > B  ' Returns False.
MyCheck = B > D Xor A > B  ' Returns Null.
MyCheck = A Xor B ' Returns 2 (bitwise comparison).
```

Like Operator Example

This example uses the **Like** operator to compare a string to a pattern.

```
Dim MyCheck
MyCheck = "aBBBa" Like "a*a" ' Returns True.
MyCheck = "F" Like "[A-Z]" ' Returns True.
MyCheck = "F" Like "[!A-Z]"' Returns False.
MyCheck = "a2a" Like "a#a" ' Returns True.
MyCheck = "aM5b" Like "a[L-P]#[!c-e]"    ' Returns True.
MyCheck = "BAT123khg" Like "B?T*"  ' Returns True.
MyCheck = "CAT123khg" Like "B?T*"  ' Returns False.
```

Is Operator Example

This example uses the **Is** operator to compare two object references. The object variable names are generic and used for illustration purposes only.

```
Dim MyObject, YourObject, ThisObject, OtherObject, ThatObject, MyCheck
Set YourObject = MyObject  ' Assign object references.
Set ThisObject = MyObject
Set ThatObject = OtherObject
MyCheck = YourObject Is ThisObject ' Returns True.
MyCheck = ThatObject Is ThisObject ' Returns False.
' Assume MyObject <> OtherObject
MyCheck = MyObject Is ThatObject   ' Returns False.
```

Operator Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vagrpOperatorSummaryC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vagrpOperatorSummaryS"}

Operators	Description
Arithmetic Operators	Operators used to perform mathematical calculations.
Comparison Operators	Operators used to perform comparisons.
Concatenation Operators	Operators used to combine strings.
Logical Operators	Operators used to perform logical operations.

Operator Precedence

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vagrpOperatorPrecedenceC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vagrpOperatorPrecedenceX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vagrpOperatorPrecedenceS"}

When several operations occur in an expression, each part is evaluated and resolved in a predetermined order called operator precedence.

When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last. Comparison operators all have equal precedence; that is, they are evaluated in the left-to-right order in which they appear. Arithmetic and logical operators are evaluated in the following order of precedence:

Arithmetic	Comparison	Logical
Exponentiation (**^**)	Equality (**=**)	**Not**
Negation (**–**)	Inequality (**<>**)	**And**
Multiplication and division (***, */*)	Less than (**<**)	**Or**
Integer division (\)	Greater than (**>**)	**Xor**
Modulus arithmetic (**Mod**)	Less than or equal to (**<=**)	**Eqv**
Addition and subtraction (**+**, **–**)	Greater than or equal to (**>=**)	**Imp**
String concatenation (**&**)	**Like**	
	Is	

When multiplication and division occur together in an expression, each operation is evaluated as it occurs from left to right. When addition and subtraction occur together in an expression, each operation is evaluated in order of appearance from left to right. Parentheses can be used to override the order of precedence and force some parts of an expression to be evaluated before others. Operations within parentheses are always performed before those outside. Within parentheses, however, operator precedence is maintained.

The string concatenation operator (**&**) is not an arithmetic operator, but in precedence, it does follow all arithmetic operators and precede all comparison operators.

The **Like** operator is equal in precedence to all comparison operators, but is actually a pattern-matching operator.

The **Is** operator is an object reference comparison operator. It does not compare objects or their values; it checks only to determine if two object references refer to the same object.

Arithmetic Operators

^ Operator

***** Operator

/ Operator

\ Operator

Mod Operator

+ Operator

- Operator

Concatenation Operators

& Operator

+ Operator

Logical Operators

And Operator

Eqv Operator

Imp Operator

Not Operator

Or Operator

Xor Operator

^ Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprExponentiationC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprExponentiationX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprExponentiationS"}

Used to raise a number to the power of an exponent.

Syntax

result **=** *number***^***exponent*

The **^** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
number	Required; any numeric expression.
exponent	Required; any numeric expression.

Remarks

A *number* can be negative only if *exponent* is an integer value. When more than one exponentiation is performed in a single expression, the **^** operator is evaluated as it is encountered from left to right.

Usually, the data type of *result* is a **Double** or a **Variant** containing a **Double**. However, if either *number* or *exponent* is a **Null** expression, *result* is **Null**.

+ Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprAddC"}
{ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprAddX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprAddS"}

Used to sum two numbers.

Syntax

result **=** *expression1*+*expression2*

The **+** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

When you use the **+** operator, you may not be able to determine whether addition or string concatenation will occur. Use the **&** operator for concatenation to eliminate ambiguity and provide self-documenting code.

If at least one expression is not a **Variant**, the following rules apply:

If	Then
Both expressions are numeric data types (**Byte**, **Boolean**, **Integer**, **Long**, **Single**, **Double**, **Date**, **Currency**, or **Decimal**)	Add.
Both expressions are **String**	Concatenate.
One expression is a numeric data type and the other is any **Variant** except **Null**	Add.
One expression is a **String** and the other is any **Variant** except **Null**	Concatenate.
One expression is an **Empty** **Variant**	Return the remaining expression unchanged as *result*.
One expression is a numeric data type and the other is a **String**	A `Type mismatch` error occurs.
Either expression is **Null**	*result* is **Null**.

If both expressions are **Variant** expressions, the following rules apply:

If	Then
Both **Variant** expressions are numeric	Add.
Both **Variant** expressions are strings	Concatenate.
One **Variant** expression is numeric and the other is a string	Add.

For simple arithmetic addition involving only expressions of numeric data types, the data type of *result* is usually the same as that of the most precise expression. The order of precision, from least to most precise, is **Byte**, **Integer**, **Long**, **Single**, **Double**, **Currency**, and **Decimal**. The following are exceptions to this order:

If	Then *result* is

A **Single** and a **Long** are added,	a **Double**.
The data type of *result* is a **Long**, **Single**, or **Date** variant that overflows its legal range,	converted to a **Double** variant.
The data type of *result* is a **Byte** variant that overflows its legal range,	converted to an **Integer** variant.
The data type of *result* is an **Integer** variant that overflows its legal range,	converted to a **Long** variant.
A **Date** is added to any data type,	a **Date**.

If one or both expressions are **Null** expressions, *result* is **Null**. If both expressions are **Empty**, *result* is an **Integer**. However, if only one expression is **Empty**, the other expression is returned unchanged as *result*.

Note The order of precision used by addition and subtraction is not the same as the order of precision used by multiplication.

- Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprSubtractC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprSubtractX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprSubtractS"}

Used to find the difference between two numbers or to indicate the negative value of a numeric expression.

Syntax 1

result **=** *number1–number2*

Syntax 2

–*number*

The **–** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
number	Required; any numeric expression.
number1	Required; any numeric expression.
number2	Required; any numeric expression.

Remarks

In Syntax 1, the **–** operator is the arithmetic subtraction operator used to find the difference between two numbers. In Syntax 2, the **–** operator is used as the unary negation operator to indicate the negative value of an expression.

The data type of *result* is usually the same as that of the most precise expression. The order of precision, from least to most precise, is **Byte**, **Integer**, **Long**, **Single**, **Double**, **Currency**, and **Decimal**. The following are exceptions to this order:

If	Then *result* is
Subtraction involves a **Single** and a **Long**,	converted to a **Double**.
The data type of *result* is a **Long**, **Single**, or **Date** variant that overflows its legal range,	converted to a **Variant** containing a **Double**.
The data type of *result* is a **Byte** variant that overflows its legal range,	converted to an **Integer** variant.
The data type of *result* is an **Integer** variant that overflows its legal range,	converted to a **Long** variant.
Subtraction involves a **Date** and any other data type,	a **Date**.
Subtraction involves two **Date** expressions,	a **Double**.

One or both expressions are **Null** expressions, *result* is **Null**. If an expression is **Empty**, it is treated as 0.

Note The order of precision used by addition and subtraction is not the same as the order of precision used by multiplication.

* Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprMultiplyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprMultiplyX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprMultiplyS"}

Used to multiply two numbers.

Syntax

result **=** *number1*number2*

The * operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
number1	Required; any numeric expression.
number2	Required; any numeric expression.

Remarks

The data type of *result* is usually the same as that of the most precise expression. The order of precision, from least to most precise, is **Byte**, **Integer**, **Long**, **Single**, **Currency**, **Double**, and **Decimal**. The following are exceptions to this order:

If	Then *result* is
Multiplication involves a **Single** and a **Long**,	converted to a **Double**.
The data type of *result* is a **Long**, **Single**, or **Date** variant that overflows its legal range,	converted to a **Variant** containing a **Double**.
The data type of *result* is a **Byte** variant that overflows its legal range,	converted to an **Integer** variant.
the data type of *result* is an **Integer** variant that overflows its legal range,	converted to a **Long** variant.

If one or both expressions are **Null** expressions, *result* is **Null**. If an expression is **Empty**, it is treated as 0.

Note The order of precision used by multiplication is not the same as the order of precision used by addition and subtraction.

/ Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprDivideC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprDivideX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprDivideS"}

Used to divide two numbers and return a floating-point result.

Syntax

result **=** *number1***/***number2*

The **/** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
number1	Required; any numeric expression.
number2	Required; any numeric expression.

Remarks

The data type of *result* is usually a **Double** or a **Double** variant. The following are exceptions to this rule:

If	Then *result* is
Both expressions are **Byte**, **Integer**, or **Single** expressions,	a **Single** unless it overflows its legal range; in which case, an error occurs.
Both expressions are **Byte**, **Integer**, or **Single** variants,	a **Single** variant unless it overflows its legal range; in which case, *result* is a **Variant** containing a **Double**.
Division involves a **Decimal** and any other data type,	a **Decimal** data type.

One or both expressions are **Null** expressions, *result* is **Null**. Any expression that is **Empty** is treated as 0.

\ Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprIntegerDivideC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprIntegerDivideX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprIntegerDivideS"}

Used to divide two numbers and return an integer result.

Syntax

result **=** *number1******number2*

The **** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
number1	Required; any numeric expression.
number2	Required; any numeric expression.

Remarks

Before division is performed, the numeric expressions are rounded to **Byte**, **Integer**, or **Long** expressions.

Usually, the data type of *result* is a **Byte**, **Byte** variant, **Integer**, **Integer** variant, **Long**, or **Long** variant, regardless of whether *result* is a whole number. Any fractional portion is truncated. However, if any expression is **Null**, *result* is **Null**. Any expression that is **Empty** is treated as 0.

Mod Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprModC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprModX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprModS"}

Used to divide two numbers and return only the remainder.

Syntax

result **=** *number1* **Mod** *number2*

The **Mod** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
number1	Required; any numeric expression.
number2	Required; any numeric expression.

Remarks

The modulus, or remainder, operator divides *number1* by *number2* (rounding floating-point numbers to integers) and returns only the remainder as *result*. For example, in the following expression, A (*result*) equals 5.

```
A = 19 Mod 6.7
```

Usually, the data type of *result* is a **Byte**, **Byte** variant, **Integer**, **Integer** variant, **Long**, or **Variant** containing a **Long**, regardless of whether or not *result* is a whole number. Any fractional portion is truncated. However, if any expression is **Null**, *result* is **Null**. Any expression that is **Empty** is treated as 0.

& Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprConcatenationC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprConcatenationX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprConcatenationS"}

Used to force string concatenation of two expressions.

Syntax

result **=** *expression1* **&** *expression2*

The **&** operator syntax has these parts:

Part	Description
result	Required; any **String** or **Variant** variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If an *expression* is not a string, it is converted to a **String** variant. The data type of *result* is **String** if both expressions are string expressions; otherwise, *result* is a **String** variant. If both expressions are **Null**, *result* is **Null**. However, if only one *expression* is **Null**, that expression is treated as a zero-length string ("") when concatenated with the other expression. Any expression that is **Empty** is also treated as a zero-length string.

Comparison Operators

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vagrpComparisonC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vagrpComparisonX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vagrpComparisonS"}

Used to compare expressions.

Syntax

result **=** *expression1 comparisonoperator expression2*
result **=** *object1* **Is** *object2*
result **=** *string* **Like** *pattern*

Comparison operators have these parts:

Part	Description
result	Required; any numeric variable.
expression	Required; any expression.
comparisonoperator	Required; any comparison operator.
object	Required; any object name.
string	Required; any string expression.
pattern	Required; any string expression or range of characters.

Remarks

The following table contains a list of the comparison operators and the conditions that determine whether *result* is **True**, **False**, or **Null**:

Operator	True if	False if	Null if
< (Less than)	*expression1* < *expression2*	*expression1* >= *expression2*	*expression1* or *expression2* = **Null**
<= (Less than or equal to)	*expression1* <= *expression2*	*expression1* > *expression2*	*expression1* or *expression2* = **Null**
> (Greater than)	*expression1* > *expression2*	*expression1* <= *expression2*	*expression1* or *expression2* = **Null**
>= (Greater than or equal to)	*expression1* >= *expression2*	*expression1* < *expression2*	*expression1* or *expression2* = **Null**
= (Equal to)	*expression1* = *expression2*	*expression1* <> *expression2*	*expression1* or *expression2* = **Null**
<> (Not equal to)	*expression1* <> *expression2*	*expression1* = *expression2*	*expression1* or *expression2* = **Null**

Note The **Is** and **Like** operators have specific comparison functionality that differs from the operators in the table.

When comparing two expressions, you may not be able to easily determine whether the expressions are being compared as numbers or as strings. The following table shows how the expressions are compared or the result when either expression is not a **Variant**:

If	Then
Both expressions are numeric data types (**Byte**, **Boolean**, **Integer**, **Long**, **Single**, **Double**, **Date**, **Currency**, or **Decimal**)	Perform a numeric comparison.

Both expressions are **String**	Perform a string comparison.
One expression is a numeric data type and the other is a **Variant** that is, or can be, a number	Perform a numeric comparison.
One expression is a numeric data type and the other is a string **Variant** that can't be converted to a number	A `Type Mismatch` error occurs.
One expression is a **String** and the other is any **Variant** except a **Null**	Perform a string comparison.
One expression is **Empty** and the other is a numeric data type	Perform a numeric comparison, using 0 as the **Empty** expression.
One expression is **Empty** and the other is a **String**	Perform a string comparison, using a zero-length string ("") as the **Empty** expression.

If *expression1* and *expression2* are both **Variant** expressions, their underlying type determines how they are compared. The following table shows how the expressions are compared or the result from the comparison, depending on the underlying type of the **Variant**:

If	Then
Both **Variant** expressions are numeric	Perform a numeric comparison.
Both **Variant** expressions are strings	Perform a string comparison.
One **Variant** expression is numeric and the other is a string	The numeric expression is less than the string expression.
One **Variant** expression is **Empty** and the other is numeric	Perform a numeric comparison, using 0 as the **Empty** expression.
One **Variant** expression is **Empty** and the other is a string	Perform a string comparison, using a zero-length string ("") as the **Empty** expression.
Both **Variant** expressions are **Empty**	The expressions are equal.

When a **Single** is compared to a **Double**, the **Double** is rounded to the precision of the **Single**.

If a **Currency** is compared with a **Single** or **Double**, the **Single** or **Double** is converted to a **Currency**. Similarly, when a **Decimal** is compared with a **Single** or **Double**, the **Single** or **Double** is converted to a **Decimal**. For **Currency**, any fractional value less than .0001 may be lost; for **Decimal**, any fractional value less than 1E-28 may be lost, or an overflow error can occur. Such fractional value loss may cause two values to compare as equal when they are not.

And Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprAndC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprAndX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprAndS"}

Used to perform a logical conjunction on two expressions.

Syntax

result **=** *expression1* **And** *expression2*

The **And** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If both expressions evaluate to **True**, *result* is **True**. If either expression evaluates to **False**, *result* is **False.** The following table illustrates how *result* is determined:

If *expression1* is	And *expression2* is	The *result* is
True	**True**	**True**
True	**False**	**False**
True	**Null**	**Null**
False	**True**	**False**
False	**False**	**False**
False	**Null**	**False**
Null	**True**	**Null**
Null	**False**	**False**
Null	**Null**	**Null**

The **And** operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	The *result* is
0	0	0
0	1	0
1	0	0
1	1	1

Eqv Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprEqvC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprEqvX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprEqvS"}

Used to perform a logical equivalence on two expressions.

Syntax

result **=** *expression1* **Eqv** *expression2*

The **Eqv** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If either expression is **Null**, *result* is also **Null**. When neither expression is **Null**, *result* is determined according to the following table:

If *expression1* is	And *expression2* is	The *result* is
True	**True**	**True**
True	**False**	**False**
False	**True**	**False**
False	**False**	**True**

The **Eqv** operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	The *result* is
0	0	1
0	1	0
1	0	0
1	1	1

Imp Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprImpC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprImpX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprImpS"}

Used to perform a logical implication on two expressions.

Syntax

result **=** *expression1* **Imp** *expression2*

The **Imp** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

The following table illustrates how *result* is determined:

If *expression1* is	And *expression2* is	The *result* is
True	**True**	**True**
True	**False**	**False**
True	**Null**	**Null**
False	**True**	**True**
False	**False**	**True**
False	**Null**	**True**
Null	**True**	**True**
Null	**False**	**Null**
Null	**Null**	**Null**

The **Imp** operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in *expression1* is	And bit in *expression2* is	The *result* is
0	0	1
0	1	1
1	0	0
1	1	1

Not Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprNotC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprNotX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprNotS"}

Used to perform logical negation on an expression.

Syntax

result **=** **Not** *expression*

The **Not** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
expression	Required; any expression.

Remarks

The following table illustrates how *result* is determined:

If ***expression*** **is**	Then ***result*** **is**
True	**False**
False	**True**
Null	**Null**

In addition, the **Not** operator inverts the bit values of any variable and sets the corresponding bit in *result* according to the following table:

If bit in ***expression*** **is**	Then bit in ***result*** **is**
0	1
1	0

Or Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprOrC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprOrX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprOrS"}

Used to perform a logical disjunction on two expressions.

Syntax

result **=** *expression1* **Or** *expression2*

The **Or** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If either or both expressions evaluate to **True**, *result* is **True**. The following table illustrates how *result* is determined:

If ***expression1*** **is**	And ***expression2*** **is**	Then ***result*** **is**
True	**True**	**True**
True	**False**	**True**
True	**Null**	**True**
False	**True**	**True**
False	**False**	**False**
False	**Null**	**Null**
Null	**True**	**True**
Null	**False**	**Null**
Null	**Null**	**Null**

The **Or** operator also performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in *result* according to the following table:

If bit in ***expression1*** **is**	And bit in ***expression2*** **is**	Then ***result*** **is**
0	0	0
0	1	1
1	0	1
1	1	1

Xor Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprXorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprXorX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprXorS"}

Used to perform a logical exclusion on two expressions.

Syntax

[*result* **=**] *expression1* **Xor** *expression2*

The **Xor** operator syntax has these parts:

Part	Description
result	Optional; any numeric variable.
expression1	Required; any expression.
expression2	Required; any expression.

Remarks

If one, and only one, of the expressions evaluates to **True**, *result* is **True**. However, if either expression is **Null**, *result* is also **Null**. When neither expression is **Null**, *result* is determined according to the following table:

If ***expression1*** **is**	And ***expression2*** **is**	Then ***result*** **is**
True	**True**	**False**
True	**False**	**True**
False	**True**	**True**
False	**False**	**False**

The **Xor** operator performs as both a logical and bitwise operator. A bit-wise comparison of two expressions using exclusive-or logic to form the result, as shown in the following table:

If bit in *expression1* is	**And bit in *expression2* is**	**Then *result* is**
0	0	0
0	1	1
1	0	1
1	1	0

Is Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprIsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprIsX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprIsS"}

Used to compare two object reference variables.

Syntax

result **=** *object1* **Is** *object2*

The **Is** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
object1	Required; any object name.
object2	Required; any object name.

Remarks

If *object1* and *object2* both refer to the same object, *result* is **True**; if they do not, *result* is **False**. Two variables can be made to refer to the same object in several ways.

In the following example, A has been set to refer to the same object as B:

```
Set A = B
```

The following example makes A and B refer to the same object as C:

```
Set A = C
Set B = C
```

Like Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprLikeC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprLikeX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprLikeS"}

Used to compare two strings.

Syntax

result **=** *string* **Like** *pattern*

The **Like** operator syntax has these parts:

Part	Description
result	Required; any numeric variable.
string	Required; any string expression.
pattern	Required; any string expression conforming to the pattern-matching conventions described in Remarks.

Remarks

If *string* matches *pattern*, *result* is **True**; if there is no match, *result* is **False**. If either *string* or *pattern* is **Null**, *result* is **Null**.

The behavior of the **Like** operator depends on the **Option Compare** statement. The default string-comparison method for each module is **Option Compare Binary**.

Option Compare Binary results in string comparisons based on a sort order derived from the internal binary representations of the characters. In Microsoft Windows, sort order is determined by the code page. In the following example, a typical binary sort order is shown:

A < B < E < Z < a < b < e < z < À < Ê < Ø < à < ê < ø

Option Compare Text results in string comparisons based on a case-insensitive, textual sort order determined by your system's locale. When you sort The same characters using **Option Compare Text**, the following text sort order is produced:

(A=a) < (À=à) < (B=b) < (E=e) < (Ê=ê) < (Z=z) < (Ø=ø)

Built-in pattern matching provides a versatile tool for string comparisons. The pattern-matching features allow you to use wildcard characters, character lists, or character ranges, in any combination, to match strings. The following table shows the characters allowed in *pattern* and what they match:

Characters in *pattern*	Matches in *string*
?	Any single character.
*	Zero or more characters.
#	Any single digit (0–9).
[*charlist*]	Any single character in *charlist*.
[**!***charlist*]	Any single character not in *charlist*.

A group of one or more characters (*charlist*) enclosed in brackets (**[ ]**) can be used to match any single character in *string* and can include almost any character code, including digits.

Note To match the special characters left bracket (**[**), question mark (**?**), number sign (**#**), and asterisk (*), enclose them in brackets. The right bracket (**]**) can't be used within a group to match itself, but it can be used outside a group as an individual character.

By using a hyphen (**–**) to separate the upper and lower bounds of the range, *charlist* can specify a range of characters. For example, `[A-Z]` results in a match if the corresponding character position in

string contains any uppercase letters in the range A–Z. Multiple ranges are included within the brackets without delimiters.

The meaning of a specified range depends on the character ordering valid at run time (as determined by **Option Compare** and the locale setting of the system the code is running on). Using the **Option Compare Binary** example, the range `[A-E]` matches A, B and E. With **Option Compare Text**, `[A-E]` matches A, a, À, à, B, b, E, e. The range does not match Ê or ê because accented characters fall after unaccented characters in the sort order.

Other important rules for pattern matching include the following:

- An exclamation point (**!**) at the beginning of *charlist* means that a match is made if any character except the characters in *charlist* is found in *string*. When used outside brackets, the exclamation point matches itself.
- A hyphen (**–**) can appear either at the beginning (after an exclamation point if one is used) or at the end of *charlist* to match itself. In any other location, the hyphen is used to identify a range of characters.
- When a range of characters is specified, they must appear in ascending sort order (from lowest to highest). `[A-Z]` is a valid pattern, but `[Z-A]` is not.
- The character sequence `[]` is considered a zero-length string ("").

In some languages, there are special characters in the alphabet that represent two separate characters. For example, several languages use the character "æ" to represent the characters "a" and "e" when they appear together. The **Like** operator recognizes that the single special character and the two individual characters are equivalent.

When a language that uses a special character is specified in the system locale settings, an occurrence of the single special character in either *pattern* or *string* matches the equivalent 2-character sequence in the other string. Similarly, a single special character in *pattern* enclosed in brackets (by itself, in a list, or in a range) matches the equivalent 2-character sequence in *string*.

Count Property Example

This example uses the **Collection** object's **Count** property to specify how many iterations are required to remove all the elements of the collection called `MyClasses`. When collections are numerically indexed, the base is 1 by default. Since collections are reindexed automatically when a removal is made, the following code removes the first member on each iteration.

```
Dim Num, MyClasses
For Num = 1 To MyClasses.Count     ' Remove name from the collection.
   MyClasses.Remove 1   ' Default collection numeric indexes
Next  ' begin at 1.
```

Description Property Example

This example assigns a user-defined message to the **Description** property of the **Err** object.

```
Err.Description = "It was not possible to access an object necessary " _
& "for this operation."
```

HelpContext Property Example

This example uses the **HelpContext** property of the **Err** object to show the Visual Basic Help topic for the `Overflow` error.

```
Dim Msg
Err.Clear
On Error Resume Next
Err.Raise 6 ' Generate "Overflow" error.
If Err.Number <> 0 Then
   Msg = "Press F1 or Help to see " & Err.HelpFile & " topic for" & _
   " the following HelpContext: " & Err.HelpContext
   MsgBox Msg, , "Error: " & Err.Description, Err.HelpFile, _
Err.HelpContext
End If
```

HelpFile Property Example

This example uses the **HelpFile** property of the **Err** object to start the Microsoft Windows Help system. By default, the **HelpFile** property contains the name of the Visual Basic Help file.

```
Dim Msg
Err.Clear
On Error Resume Next ' Suppress errors for demonstration purposes.
Err.Raise 6 ' Generate "Overflow" error.
Msg = "Press F1 or Help to see " & Err.HelpFile & _
" topic for this error"
MsgBox Msg, , "Error: " & Err.Description,Err.HelpFile, Err.HelpContext
```

LastDLLError Property Example

When pasted into a **UserForm** module, the following code causes an attempt to call a DLL function. The call fails because the argument that is passed in (a null pointer) generates an error, and in any event, SQL can't be cancelled if it isn't running. The code following the call checks the return of the call, and then prints at the **LastDLLError** property of the **Err** object to reveal the error code.

```
 Private Declare Function SQLCancel Lib "ODBC32.dll" _
 (ByVal hstmt As Long) As Integer

Private Sub UserForm_Click()
   Dim RetVal
   ' Call with invalid argument.
   RetVal = SQLCancel(myhandle&)
   ' Check for SQL error code.
   If RetVal = -2 Then
      'Display the information code.
      MsgBox "Error code is :" & Err.LastDllError
    End If
End Sub
```

Number Property Example

The first example illustrates a typical use of the **Number** property in an error-handling routine. The second example examines the **Number** property of the **Err** object to determine whether an error returned by an Automation object was defined by the object, or whether it was mapped to an error defined by Visual Basic. Note that the constant **vbObjectError** is a very large negative number that an object adds to its own error code to indicate that the error is defined by the server. Therefore, subtracting it from **Err.Number** strips it out of the result. If the error is object-defined, the base number is left in `MyError`, which is displayed in a message box along with the original source of the error. If **Err.Number** represents a Visual Basic error, then the Visual Basic error number is displayed in the message box.

```
' Typical use of Number property
Sub test()
   On Error GoTo out

   Dim x, y
   x = 1 / y   ' Create division by zero error
   Exit Sub
   out:
   MsgBox Err.Number
   MsgBox Err.Description
   ' Check for division by zero error
   If Err.Number = 11 Then
      y = y + 1
   End If
   Resume
End Sub
```

```
' Using Number property with an error from an
' Automation object
Dim MyError, Msg
' First, strip off the constant added by the object to indicate one
' of its own errors.
MyError = Err.Number - vbObjectError
' If you subtract the vbObjectError constant, and the number is still
' in the range 0-65,535, it is an object-defined error code.
If MyError > 0 And MyError < 65535 Then
   Msg = "The object you accessed assigned this number to the error: " _
         & MyError & ". The originator of the error was: " _
         & Err.Source & ". Press F1 to see originator's Help topic."
' Otherwise it is a Visual Basic error number.
Else
   Msg = "This error (# " & Err.Number & ") is a Visual Basic error" & _
         " number. Press Help button or F1 for the Visual Basic Help" _
         & " topic for this error."
End If
   MsgBox Msg, , "Object Error", Err.HelpFile, Err.HelpContext
```

Source Property Example

This example assigns the Programmatic ID of an Automation object created in Visual Basic to the variable `MyObjectID`, and then assigns that to the **Source** property of the **Err** object when it generates an error with the **Raise** method. When handling errors, you should not use the **Source** property (or any **Err** properties other than **Number**) programatically. The only valid use of properties other than **Number** is for displaying rich information to an end user in cases where you can't handle an error. The example assumes that `App` and `MyClass` are valid references.

```
Dim MyClass, MyObjectID, MyHelpFile, MyHelpContext
' An object of type MyClass generates an error and fills all Err object
' properties, including Source, which receives MyObjectID, which is a
' combination of the Title property of the App object and the Name
' property of the MyClass object.
MyObjectID = App.Title & "." & MyClass.Name
Err.RaiseNumber := vbObjectError + 894, Source := MyObjectID, _
            Description := "Was not able to complete your task", _
            HelpFile := MyHelpFile, HelpContext := MyHelpContext
```

Calendar Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaproCalendarC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaproCalendarX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vaproCalendarA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaproCalendarS"}

Returns or sets a value specifying the type of calendar to use with your project.

You can use one of two settings for **Calendar**:

Setting	Value	Description
vbCalGreg	0	Use Gregorian calendar (default).
vbCalHijri	1	Use Hijri calendar.

Remarks

You can only set the **Calendar** property programmatically. For example, to use the Hijri calendar, use:

```
Calendar = vbCalHijri
```

Count Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaproCountC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaproCountX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vaproCountA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaproCountS"}

Returns a **Long** (long integer) containing the number of objects in a collection. Read-only.

Description Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaproDescriptionC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaproDescriptionX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vaproDescriptionA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaproDescriptionS"}

Returns or sets a string expression containing a descriptive string associated with an object. Read/write.

For the **Err** object, returns or sets a descriptive string associated with an error.

Remarks

The **Description** property setting consists of a short description of the error. Use this property to alert the user to an error that you either can't or don't want to handle. When generating a user-defined error, assign a short description of your error to the **Description** property. If **Description** isn’t filled in, and the value of **Number** corresponds to a Visual Basic run-time error, the string returned by the **Error** function is placed in **Description** when the error is generated.

HelpContext Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaproHelpContextC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaproHelpContextX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vaproHelpContextA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaproHelpContextS"}

Returns or sets a string expression containing the context ID for a topic in a Microsoft Windows Help file. Read/write.

Remarks

The **HelpContext** property is used to automatically display the Help topic specified in the **HelpFile** property. If both **HelpFile** and **HelpContext** are empty, the value of **Number** is checked. If **Number** corresponds to a Visual Basic run-time error value, then the Visual Basic Help context ID for the error is used. If the **Number** value doesn't correspond to a Visual Basic error, the contents screen for the Visual Basic Help file is displayed.

Note You should write routines in your application to handle typical errors. When programming with an object, you can use the object's Help file to improve the quality of your error handling, or to display a meaningful message to your user if the error isn't recoverable.

HelpFile Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaproHelpFileC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaproHelpFileX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vaproHelpFileA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaproHelpFileS"}

Returns or sets a string expression the fully qualified path to a Microsoft Windows Help file. Read/write.

Remarks

If a Help file is specified in **HelpFile**, it is automatically called when the user presses the **Help** button (or the F1 key) in the error message dialog box. If the **HelpContext** property contains a valid context ID for the specified file, that topic is automatically displayed. If no **HelpFile** is specified, the Visual Basic Help file is displayed.

Note You should write routines in your application to handle typical errors. When programming with an object, you can use the object's Help file to improve the quality of your error handling, or to display a meaningful message to your user if the error isn’t recoverable.

LastDLLError Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaproLastDLLErrorC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaproLastDLLErrorX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vaproLastDLLErrorA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaproLastDLLErrorS"}

Returns a system error code produced by a call to a dynamic-link library (DLL). Read-only.

Remarks

The **LastDLLError** property applies only to DLL calls made from Visual Basic code. When such a call is made, the called function usually returns a code indicating success or failure, and the **LastDLLError** property is filled. Check the documentation for the DLL's functions to determine the return values that indicate success or failure. Whenever the failure code is returned, the Visual Basic application should immediately check the **LastDLLError** property. No exception is raised when the **LastDLLError** property is set.

Number Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaproNumberC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaproNumberX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vaproNumberA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaproNumberS"}

Returns or sets a numeric value specifying an error. **Number** is the **Err** object's default property. Read/write.

Remarks

When returning a user-defined error from an object, set **Err.Number** by adding the number you selected as an error code to the **vbObjectError** constant. For example, you use the following code to return the number 1051 as an error code:

```
Err.Raise Number := vbObjectError + 1051, Source:= "SomeClass"
```

Source Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaproSourceC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaproSourceX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vaproSourceA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaproSourceS"}

Returns or sets a string expression specifying the name of the object or application that originally generated the error. Read/write.

Remarks

The **Source** property specifies a string expression representing the object that generated the error; the expression is usually the object's class name or programmatic ID. Use **Source** to provide information when your code is unable to handle an error generated in an accessed object. For example, if you access Microsoft Excel and it generates a `Division by zero` error, Microsoft Excel sets **Err.Number** to its error code for that error and sets **Source** to Excel.Application.

When generating an error from code, **Source** is your application's programmatic ID. For class modules, **Source** should contain a name having the form *project.class*. When an unexpected error occurs in your code, the **Source** property is automatically filled in. For errors in a standard module, **Source** contains the project name. For errors in a class module, **Source** contains a name with the *project.class* form.

DeleteSetting Statement Example

The following example first uses the **SaveSetting** statement to make entries in the Windows registry (or .ini file on 16-bit Windows platforms) for the `MyApp` application, and then uses the **DeleteSetting** statement to remove them. Because no ***key*** argument is specified, the whole section is deleted, including the section name and all its keys.

```
' Place some settings in the registry.
SaveSetting appname := "MyApp", section := "Startup", _
            key := "Top", setting := 75
SaveSetting "MyApp","Startup", "Left", 50
' Remove section and all its settings from registry.
DeleteSetting "MyApp", "Startup"
```

GetAllSettings Function Example

This example first uses the **SaveSetting** statement to make entries in the Windows registry (or .ini file on 16-bit Windows platforms) for the application specified as ***appname***, then uses the **GetAllSettings** function to display the settings. Note that application names and ***section*** names can't be retrieved with **GetAllSettings**. Finally, the **DeleteSetting** statement removes the application's entries.

```
' Variant to hold 2-dimensional array returned by GetAllSettings
' Integer to hold counter.
Dim MySettings As Variant, intSettings As Integer
' Place some settings in the registry.
SaveSetting appname := "MyApp", section := "Startup", _
key := "Top", setting := 75
SaveSetting "MyApp","Startup", "Left", 50
' Retrieve the settings.
MySettings = GetAllSettings(appname := "MyApp", section := "Startup")
   For intSettings = LBound(MySettings, 1) To UBound(MySettings, 1)
      Debug.Print MySettings(intSettings, 0), MySettings(intSettings, 1)
   Next intSettings
DeleteSetting "MyApp", "Startup"
```

GetSetting Function Example

This example first uses the **SaveSetting** statement to make entries in the Windows registry (or .ini file on 16-bit Windows platforms) for the application specified as ***appname***, and then uses the **GetSetting** function to display one of the settings. Because the ***default*** argument is specified, some value is guaranteed to be returned. Note that ***section*** names can't be retrieved with **GetSetting**. Finally, the **DeleteSetting** statement removes all the application's entries.

```
' Variant to hold 2-dimensional array returned by GetSetting.
Dim MySettings As Variant
' Place some settings in the registry.
SaveSetting "MyApp","Startup", "Top", 75
SaveSetting "MyApp","Startup", "Left", 50

Debug.Print GetSetting(appname := "MyApp", section := "Startup", _
                       key := "Left", default := "25")

DeleteSetting "MyApp", "Startup"
```

SaveSetting Statement Example

The following example first uses the **SaveSetting** statement to make entries in the Windows registry (or .ini file on 16-bit Windows platforms) for the `MyApp` application, and then uses the **DeleteSetting** statement to remove them.

```
' Place some settings in the registry.
SaveSetting appname := "MyApp", section := "Startup", _
            key := "Top", setting := 75
SaveSetting "MyApp","Startup", "Left", 50
' Remove section and all its settings from registry.
DeleteSetting "MyApp", "Startup"
```

DeleteSetting Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmDeleteSettingC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmDeleteSettingX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmDeleteSettingS"}

Deletes a section or key setting from an application's entry in the Windows registry.

Syntax

DeleteSetting *appname*, *section*[, *key*]

The **DeleteSetting** statement syntax has these named arguments:

Part	Description
appname	Required. String expression containing the name of the application or project to which the section or key setting applies.
section	Required. String expression containing the name of the section where the key setting is being deleted. If only ***appname*** and ***section*** are provided, the specified section is deleted along with all related key settings.
key	Optional. String expression containing the name of the key setting being deleted.

Remarks

If all arguments are provided, the specified key setting is deleted. However, the **DeleteSetting** statement does nothing if the specified section or key setting does not exist.

GetAllSettings Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctGetAllSettingsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctGetAllSettingsX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctGetAllSettingsS"}

Returns a list of key settings and their respective values (originally created with **SaveSetting**) from an application's entry in the Windows registry.

Syntax

GetAllSettings(*appname, section*)

The **GetAllSettings** function syntax has these named arguments:

Part	Description
appname	Required. String expression containing the name of the application or project whose key settings are requested.
section	Required. String expression containing the name of the section whose key settings are requested. **GetAllSettings** returns a **Variant** whose contents is a two-dimensional array of strings containing all the key settings in the specified section and their corresponding values.

Remarks

GetAllSettings returns an uninitialized **Variant** if either ***appname*** or ***section*** does not exist.

GetSetting Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctGetSettingC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctGetSettingX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctGetSettingS"}

Returns a key setting value from an application's entry in the Windows registry.

Syntax

GetSetting(*appname*, *section*, *key*[, *default*])

The **GetSetting** function syntax has these named arguments:

Part	Description
appname	Required. String expression containing the name of the application or project whose key setting is requested.
section	Required. String expression containing the name of the section where the key setting is found.
key	Required. String expression containing the name of the key setting to return.
default	Optional. Expression containing the value to return if no value is set in the key setting. If omitted, ***default*** is assumed to be a zero-length string ("").

Remarks

If any of the items named in the **GetSetting** arguments do not exist, **GetSetting** returns the value of ***default***.

SaveSetting Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmSaveSettingC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmSaveSettingX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmSaveSettingS"}

Saves or creates an application entry in the Windows registry.

Syntax

SaveSetting *appname*, *section, key, setting*

The **SaveSetting** statement syntax has these named arguments:

Part	Description
appname	Required. String expression containing the name of the application or project to which the setting applies.
section	Required. String expression containing the name of the section where the key setting is being saved.
key	Required. String expression containing the name of the key setting being saved.
setting	Required. Expression containing the value that ***key*** is being set to.

Remarks

An error occurs if the key setting can’t be saved for any reason.

Chr Function Example

This example uses the **Chr** function to return the character associated with the specified character code.

```
Dim MyChar
MyChar = Chr(65)   ' Returns A.
MyChar = Chr(97)   ' Returns a.
MyChar = Chr(62)   ' Returns >.
MyChar = Chr(37)   ' Returns %.
```

Format Function Example

This example shows various uses of the **Format** function to format values using both named formats and user-defined formats. For the date separator (*/*), time separator (**:**), and AM/ PM literal, the actual formatted output displayed by your system depends on the locale settings on which the code is running. When times and dates are displayed in the development environment, the short time format and short date format of the code locale are used. When displayed by running code, the short time format and short date format of the system locale are used, which may differ from the code locale. For this example, English/U.S. is assumed.

`MyTime` and `MyDate` are displayed in the development environment using current system short time setting and short date setting.

```
Dim MyTime, MyDate, MyStr
MyTime = #17:04:23#
MyDate = #January 27, 1993#

' Returns current system time in the system-defined long time format.
MyStr = Format(Time, "Long Time")

' Returns current system date in the system-defined long date format.
MyStr = Format(Date, "Long Date")

MyStr = Format(MyTime, "h:m:s")     ' Returns "17:4:23".
MyStr = Format(MyTime, "hh:mm:ss AMPM")  ' Returns "05:04:23 PM".
MyStr = Format(MyDate, "dddd, mmm d yyyy")      ' Returns "Wednesday,
   ' Jan 27 1993".
' If format is not supplied, a string is returned.
MyStr = Format(23)' Returns "23".

' User-defined formats.
MyStr = Format(5459.4, "##,##0.00")' Returns "5,459.40".
MyStr = Format(334.9, "###0.00")    ' Returns "334.90".
MyStr = Format(5, "0.00%") ' Returns "500.00%".
MyStr = Format("HELLO", "<") ' Returns "hello".
MyStr = Format("This is it", ">")  ' Returns "THIS IS IT".
```

Hex Function Example

This example uses the **Hex** function to return the hexadecimal value of a number.

```
Dim MyHex
MyHex = Hex(5)  ' Returns 5.
MyHex = Hex(10)' Returns A.
MyHex = Hex(459)   ' Returns 1CB.
```

InStr Function Example

This example uses the **InStr** function to return the position of the first occurrence of one string within another.

```
Dim SearchString, SearchChar, MyPos
SearchString ="XXpXXpXXPXXP" ' String to search in.
SearchChar = "P"   ' Search for "P".

' A textual comparison starting at position 4. Returns 6.
MyPos = Instr(4, SearchString, SearchChar, 1)

' A binary comparison starting at position 1. Returns 9.
MyPos = Instr(1, SearchString, SearchChar, 0)

' Comparison is binary by default (last argument is omitted).
MyPos = Instr(SearchString, SearchChar)   ' Returns 9.

MyPos = Instr(1, SearchString, "W")' Returns 0.
```

LCase Function Example

This example uses the **LCase** function to return a lowercase version of a string.

```
Dim UpperCase, LowerCase
Uppercase = "Hello World 1234"     ' String to convert.
Lowercase = Lcase(UpperCase) ' Returns "hello world 1234".
```

Left Function Example

This example uses the **Left** function to return a specified number of characters from the left side of a string.

```
Dim AnyString, MyStr
AnyString = "Hello World"   ' Define string.
MyStr = Left(AnyString, 1)  ' Returns "H".
MyStr = Left(AnyString, 7)  ' Returns "Hello W".
MyStr = Left(AnyString, 20)' Returns "Hello World".
```

Len Function Example

This example uses the **Len** function to return the number of characters in a string or the number of bytes required to store a variable. The **Type...End Type** block defining `CustomerRecord` must be preceded by the keyword **Private** if it appears in a class module. In a standard module, a **Type** statement can be **Public**.

```
Type CustomerRecord  ' Define user-defined type.
   ID As Integer  ' Place this definition in a
   Name As String * 10  ' standard module.
   Address As String * 30
End Type

Dim Customer As CustomerRecord     ' Declare variables.
Dim MyInt As Integer, MyCur As Currency
Dim MyString, MyLen
MyString = "Hello World"' Initialize variable.
MyLen = Len(MyInt)' Returns 2.
MyLen = Len(Customer)' Returns 42.
MyLen = Len(MyString)' Returns 11.
MyLen = Len(MyCur)' Returns 8.
```

LSet Statement Example

This example uses the **LSet** statement to left align a string within a string variable. Although **LSet** can also be used to copy a variable of one user-defined type to another variable of a different, but compatible, user-defined type, this practice is not recommended. Due to the varying implementations of data structures among platforms, such a use of **LSet** can't be guaranteed to be portable.

```
Dim MyString
MyString = "0123456789" ' Initialize string.
Lset MyString = "<-Left"' MyString contains "<-Left    ".
```

LTrim, RTrim, and Trim Functions Example

This example uses the **LTrim** function to strip leading spaces and the **RTrim** function to strip trailing spaces from a string variable. It uses the **Trim** function to strip both types of spaces.

```
Dim MyString, TrimString
MyString = "  <-Trim->  "  ' Initialize string.
TrimString = LTrim(MyString) ' TrimString = "<-Trim->  ".
TrimString = RTrim(MyString) ' TrimString = "  <-Trim->".
TrimString = LTrim(RTrim(MyString))' TrimString = "<-Trim->".
' Using the Trim function alone achieves the same result.
TrimString = Trim(MyString)' TrimString = "<-Trim->".
```

Mid Function Example

This example uses the **Mid** function to return a specified number of characters from a string.

```
Dim MyString, FirstWord, LastWord, MidWords
MyString = "Mid Function Demo"     ' Create text string.
FirstWord = Mid(MyString, 1, 3)    ' Returns "Mid".
LastWord = Mid(MyString, 14, 4)    ' Returns "Demo".
MidWords = Mid(MyString, 5)' Returns "Function Demo".
```

Mid Statement Example

This example uses the **Mid** statement to replace a specified number of characters in a string variable with characters from another string.

```
Dim MyString
MyString = "The dog jumps" ' Initialize string.
Mid(MyString, 5, 3) = "fox"' MyString = "The fox jumps".
Mid(MyString, 5) = "cow"' MyString = "The cow jumps".
Mid(MyString, 5) = "cow jumped over"     ' MyString = "The cow jumpe".
Mid(MyString, 5, 3) = "duck" ' MyString = "The duc jumpe".
```

Oct Function Example

This example uses the **Oct** function to return the octal value of a number.

```
Dim MyOct
MyOct = Oct(4) ' Returns 4.
MyOct = Oct(8) ' Returns 10.
MyOct = Oct(459)  ' Returns 713.
```

Right Function Example

This example uses the **Right** function to return a specified number of characters from the right side of a string.

```
Dim AnyString, MyStr
AnyString = "Hello World"  ' Define string.
MyStr = Right(AnyString, 1)' Returns "d".
MyStr = Right(AnyString, 6)' Returns " World".
MyStr = Right(AnyString, 20) ' Returns "Hello World".
```

RSet Statement Example

This example uses the **RSet** statement to right align a string within a string variable.

```
Dim MyString
MyString = "0123456789" ' Initialize string.
Rset MyString = "Right->"  ' MyString contains "   Right->".
```

Space Function Example

This example uses the **Space** function to return a string consisting of a specified number of spaces.

```
Dim MyString
' Returns a string with 10 spaces.
MyString = Space(10)

' Insert 10 spaces between two strings.
MyString = "Hello" & Space(10) & "World"
```

Str Function Example

This example uses the **Str** function to return a string representation of a number. When a number is converted to a string, a leading space is always reserved for its sign.

```
Dim MyString
MyString = Str(459)   ' Returns " 459".
MyString = Str(-459.65)  ' Returns "-459.65".
MyString = Str(459.001)  ' Returns " 459.001".
```

StrComp Function Example

This example uses the **StrComp** function to return the results of a string comparison. If the third argument is 1, a textual comparison is performed; if the third argument is 0 or omitted, a binary comparison is performed.

```
Dim MyStr1, MyStr2, MyComp
MyStr1 = "ABCD": MyStr2 = "abcd"   ' Define variables.
MyComp = StrComp(MyStr1, MyStr2, 1)' Returns 0.
MyComp = StrComp(MyStr1, MyStr2, 0)' Returns -1.
MyComp = StrComp(MyStr2, MyStr1)   ' Returns 1.
```

String Function Example

This example uses the **String** function to return repeating character strings of the length specified.

```
Dim MyString
MyString = String(5, "*")   ' Returns "*****".
MyString = String(5, 42)' Returns "*****".
MyString = String(10, "ABC") ' Returns "AAAAAAAAAA".
```

UCase Function Example

This example uses the **UCase** function to return an uppercase version of a string.

```
Dim LowerCase, UpperCase
LowerCase = "Hello World 1234"     ' String to convert.
UpperCase = UCase(LowerCase) ' Returns "HELLO WORLD 1234".
```

Chr Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctChrC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctChrX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctChrS"}

Returns a **String** containing the character associated with the specified character code.

Syntax

Chr(*charcode***)**

The required *charcode* argument is a **Long** that identifies a character.

Remarks

Numbers from 0 – 31 are the same as standard, nonprintable ASCII codes. For example, **Chr(**10**)** returns a linefeed character. The normal range for *charcode* is 0 – 255. However, on DBCS systems, the actual range for *charcode* is -32768 to 65536.

Note The **ChrB** function is used with byte data contained in a **String**. Instead of returning a character, which may be one or two bytes, **ChrB** always returns a single byte. The **ChrW** function returns a **String** containing the Unicode character except on platforms where Unicode is not supported, in which case, the behavior is identical to the **Chr** function.

Format Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctFormatC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctFormatX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctFormatS"}

Returns a **Variant** (**String**) containing an expression formatted according to instructions contained in a format expression.

Syntax

Format(*expression*[, *format*[, *firstdayofweek*[, *firstweekofyear*]]])

The **Format** function syntax has these parts:

Part	Description
expression	Required. Any valid expression.
format	Optional. A valid named or user-defined format expression.
firstdayofweek	Optional. A constant that specifies the first day of the week.
firstweekofyear	Optional. A constant that specifies the first week of the year.

Settings

The *firstdayofweek* argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use NLS API setting.
VbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

The *firstweekofyear* argument has these settings:

Constant	Value	Description
vbUseSystem	0	Use NLS API setting.
vbFirstJan1	1	Start with week in which January 1 occurs (default).
vbFirstFourDays	2	Start with the first week that has at least four days in the year.
vbFirstFullWeek	3	Start with the first full week of the year.

Remarks

To Format	Do This
Numbers	Use predefined named numeric formats or create user-defined numeric formats.
Dates and times	Use predefined named date/time formats or create user-defined date/time formats.
Date and time	Use date and time formats or numeric formats.

serial numbers	
Strings	Create your own user-defined string formats.

If you try to format a number without specifying *format*, **Format** provides functionality similar to the **Str** function, although it is internationally aware. However, positive numbers formatted as strings using **Format** don't include a leading space reserved for the sign of the value; those converted using **Str** retain the leading space.

Named Numeric Formats (Format Function)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafmtNamedNumericFormatsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafmtNamedNumericFormatsX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafmtNamedNumericFormatsS"}

The following table identifies the predefined numeric format names:

Format name	Description
General Number	Display number with no thousand separator.
Currency	Display number with thousand separator, if appropriate; display two digits to the right of the decimal separator. Output is based on system locale settings.
Fixed	Display at least one digit to the left and two digits to the right of the decimal separator.
Standard	Display number with thousand separator, at least one digit to the left and two digits to the right of the decimal separator.
Percent	Display number multiplied by 100 with a percent sign (**%**) appended to the right; always display two digits to the right of the decimal separator.
Scientific	Use standard scientific notation.
Yes/No	Display No if number is 0; otherwise, display Yes.
True/False	Display **False** if number is 0; otherwise, display **True**.
On/Off	Display Off if number is 0; otherwise, display On.

User-Defined Numeric Formats (Format Function)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafmtUserDefinedNumericFormatsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafmtUserDefinedNumericFormatsX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafmtUserDefinedNumericFormatsS"}

The following table identifies characters you can use to create user-defined number formats:

Character	Description
None	Display the number with no formatting.
(**0**)	Digit placeholder. Display a digit or a zero. If the expression has a digit in the position where the 0 appears in the format string, display it; otherwise, display a zero in that position. If the number has fewer digits than there are zeros (on either side of the decimal) in the format expression, display leading or trailing zeros. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, round the number to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, display the extra digits without modification.
(**#**)	Digit placeholder. Display a digit or nothing. If the expression has a digit in the position where the # appears in the format string, display it; otherwise, display nothing in that position. This symbol works like the 0 digit placeholder, except that leading and trailing zeros aren't displayed if the number has the same or fewer digits than there are # characters on either side of the decimal separator in the format expression.
(**.**)	Decimal placeholder. In some locales, a comma is used as the decimal separator. The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only number signs to the left of this symbol, numbers smaller than 1 begin with a decimal separator. To display a leading zero displayed with fractional numbers, use 0 as the first digit placeholder to the left of the decimal separator. The actual character used as a decimal placeholder in the formatted output depends on the Number Format recognized by your system.
(**%**)	Percentage placeholder. The expression is multiplied by 100. The percent character (**%**) is inserted in the position where it appears in the format string.
(**,**)	Thousand separator. In some locales, a period is used as a thousand separator. The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator. Standard use of the thousand separator is specified if the format contains a thousand separator surrounded by digit placeholders (**0** or **#**). Two adjacent thousand separators or a thousand separator immediately to the left of the decimal separator (whether or not a decimal is specified) means "scale the number by dividing it by 1000, rounding as needed." For example, you can use the format string "##0,," to represent 100 million as 100. Numbers smaller than 1 million are displayed as 0. Two adjacent thousand separators in any position other than

	immediately to the left of the decimal separator are treated simply as specifying the use of a thousand separator. The actual character used as the thousand separator in the formatted output depends on the Number Format recognized by your system.
(**:**)	Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by your system settings.
(*/*)	Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by your system settings.
(**E- E+ e- e+**)	Scientific format. If the format expression contains at least one digit placeholder (**0** or **#**) to the right of E-, E+, e-, or e+, the number is displayed in scientific format and E or e is inserted between the number and its exponent. The number of digit placeholders to the right determines the number of digits in the exponent. Use E- or e- to place a minus sign next to negative exponents. Use E+ or e+ to place a minus sign next to negative exponents and a plus sign next to positive exponents.
- + $ ( )	Display a literal character. To display a character other than one of those listed, precede it with a backslash (\) or enclose it in double quotation marks (" ").
(\)	Display the next character in the format string. To display a character that has special meaning as a literal character, precede it with a backslash (\). The backslash itself isn't displayed. Using a backslash is the same as enclosing the next character in double quotation marks. To display a backslash, use two backslashes (\\). Examples of characters that can't be displayed as literal characters are the date-formatting and time-formatting characters (a, c, d, h, m, n, p, q, s, t, w, y, / and :), the numeric-formatting characters (#, 0, %, E, e, comma, and period), and the string-formatting characters (@, &, <, >, and !).
("ABC")	Display the string inside the double quotation marks (" "). To include a string in ***format*** from within code, you must use **Chr(**34**)** to enclose the text (34 is the character code for a quotation mark ("")).

Different Formats for Different Numeric Values (Format Function)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafmtMultipleNumSectionsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafmtMultipleNumSectionsS"}

A user-defined format expression for numbers can have from one to four sections separated by semicolons. If the *format* argument contains one of the named numeric formats, only one section is allowed.

If you use	The result is
One section only	The format expression applies to all values.
Two sections	The first section applies to positive values and zeros, the second to negative values.
Three sections	The first section applies to positive values, the second to negative values, and the third to zeros.
Four sections	The first section applies to positive values, the second to negative values, the third to zeros, and the fourth to **Null** values.

The following example has two sections: the first defines the format for positive values and zeros; the second section defines the format for negative values.

```
"$#,##0;($#,##0)"
```

If you include semicolons with nothing between them, the missing section is printed using the format of the positive value. For example, the following format displays positive and negative values using the format in the first section and displays "Zero" if the value is zero.

```
"$#,##0;;\Z\e\r\o"
```

User-Defined Numeric Format Expressions Example

The following table contains some sample format expressionss for numbers. (These examples all assume that your system's locale setting is English-U.S.) The first column contains the format strings; the other columns contain the resulting output if the formatted data has the value given in the column headings.

Format (*format*)	**Positive 5**	**Negative 5**	**Decimal .5**	**Null**
Zero-length string ("")	5	-5	0.5	
0	5	-5	1	
0.00	5.00	-5.00	0.50	
#,##0	5	-5	1	
#,##0.00;;;Nil	5.00	-5.00	0.50	Nil
$#,##0;($#,##0)	$5	($5)	$1	
$#,##0.00;($#,##0.00)	$5.00	($5.00)	$0.50	
0%	500%	-500%	50%	
0.00%	500.00%	-500.00%	50.00%	
0.00E+00	5.00E+00	-5.00E+00	5.00E-01	
0.00E-00	5.00E00	-5.00E00	5.00E-01	

Named Date/Time Formats (Format Function)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafmtNamedDateFormatsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafmtNamedDateFormatsX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafmtNamedDateFormatsS"}

The following table identifies the predefined date and time format names:

Format Name	Description
General Date	Display a date and/or time. For real numbers, display a date and time, for example, 4/3/93 05:34 PM. If there is no fractional part, display only a date, for example, 4/3/93. If there is no integer part, display time only, for example, 05:34 PM. Date display is determined by your system settings.
Long Date	Display a date according to your system's long date format.
Medium Date	Display a date using the medium date format appropriate for the language version of the host application.
Short Date	Display a date using your system's short date format.
Long Time	Display a time using your system's long time format; includes hours, minutes, seconds.
Medium Time	Display time in 12-hour format using hours and minutes and the AM/PM designator.
Short Time	Display a time using the 24-hour format, for example, 17:45.

User-Defined Date/Time Formats (Format Function)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafmtUserDefinedDateFormatsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafmtUserDefinedDateFormatsX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafmtUserDefinedDateFormatsS"}

The following table identifies characters you can use to create user-defined date/time formats:

Character	Description
(**:**)	Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by your system settings.
(*/*)	Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by your system settings.
c	Display the date as `ddddd` and display the time as `ttttt`, in that order. Display only date information if there is no fractional part to the date serial number; display only time information if there is no integer portion.
d	Display the day as a number without a leading zero (1 – 31).
dd	Display the day as a number with a leading zero (01 – 31).
ddd	Display the day as an abbreviation (Sun – Sat).
dddd	Display the day as a full name (Sunday – Saturday).
ddddd	Display the date as a complete date (including day, month, and year), formatted according to your system's short date format setting. For Microsoft Windows, the default short date format is `m/d/yy`.
dddddd	Display a date serial number as a complete date (including day, month, and year) formatted according to the long date setting recognized by your system. For Microsoft Windows, the default long date format is `mmmm dd, yyyy`.
w	Display the day of the week as a number (1 for Sunday through 7 for Saturday).
ww	Display the week of the year as a number (1 – 54).
m	Display the month as a number without a leading zero (1 – 12). If `m` immediately follows `h` or `hh`, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01 – 12). If `m` immediately follows `h` or `hh`, the minute rather than the month is displayed.
mmm	Display the month as an abbreviation (Jan – Dec).
mmmm	Display the month as a full month name (January – December).
q	Display the quarter of the year as a number (1 – 4).
y	Display the day of the year as a number (1 – 366).
yy	Display the year as a 2-digit number (00 – 99).
yyyy	Display the year as a 4-digit number (100 – 9999).
h	Display the hour as a number without leading zeros (0 – 23).

hh	Display the hour as a number with leading zeros (00 – 23).
n	Display the minute as a number without leading zeros (0 – 59).
nn	Display the minute as a number with leading zeros (00 – 59).
s	Display the second as a number without leading zeros (0 – 59).
ss	Display the second as a number with leading zeros (00 – 59).
t t t t t	Display a time as a complete time (including hour, minute, and second), formatted using the time separator defined by the time format recognized by your system. A leading zero is displayed if the leading zero option is selected and the time is before 10:00 A.M. or P.M. For Microsoft Windows, the default time format is `h:mm:ss`.
AM/PM	Use the 12-hour clock and display an uppercase AM with any hour before noon; display an uppercase PM with any hour between noon and 11:59 P.M.
am/pm	Use the 12-hour clock and display a lowercase AM with any hour before noon; display a lowercase PM with any hour between noon and 11:59 P.M.
A/P	Use the 12-hour clock and display an uppercase A with any hour before noon; display an uppercase P with any hour between noon and 11:59 P.M.
a/p	Use the 12-hour clock and display a lowercase A with any hour before noon; display a lowercase P with any hour between noon and 11:59 P.M.
AMPM	Use the 12-hour clock and display the AM string literal as defined by your system with any hour before noon; display the PM string literal as defined by your system with any hour between noon and 11:59 P.M. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as defined by your system settings. For Microsoft Windows, the default format is AM/PM.

User-Defined Date/Time Formats Example

The following are examples of user-defined date and time formats for December 7, 1958:

Format	Display
`m/d/yy`	12/7/58
`d-mmm`	7-Dec
`d-mmmm-yy`	7-December-58
`d mmmm`	7 December
`mmmm yy`	December 58
`hh:mm AM/PM`	08:50 PM
`h:mm:ss a/p`	8:50:35 p
`h:mm`	20:50
`h:mm:ss`	20:50:35
`m/d/yy h:mm`	12/7/58 20:50

User-Defined String Formats (Format Function)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafmtStringFormatsC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafmtStringFormatsX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafmtStringFormatsS"}

You can use any of the following characters to create a format expression for strings:

Character	Description
@	Character placeholder. Display a character or a space. If the string has a character in the position where the at symbol (@) appears in the format string, display it; otherwise, display a space in that position. Placeholders are filled from right to left unless there is an exclamation point character (!) in the format string.
&	Character placeholder. Display a character or nothing. If the string has a character in the position where the ampersand (&) appears, display it; otherwise, display nothing. Placeholders are filled from right to left unless there is an exclamation point character (!) in the format string.
<	Force lowercase. Display all characters in lowercase format.
>	Force uppercase. Display all characters in uppercase format.
!	Force left to right fill of placeholders. The default is to fill placeholders from right to left.

Different Formats for Different String Values (Format Function)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafmtMultipleStringSectionsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafmtMultipleStringSectionsS"}

A format expression for strings can have one section or two sections separated by a semicolon (**;**).

If you use	The result is
One section only	The format applies to all string data.
Two sections	The first section applies to string data, the second to **Null** values and zero-length strings ("").

Hex Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctHexC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctHexX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctHexS"}

Returns a **String** representing the hexadecimal value of a number.

Syntax

Hex(*number*)

The required *number* argument is any valid numeric expression or string expression.

Remarks

If *number* is not already a whole number, it is rounded to the nearest whole number before being evaluated.

If *number* is	Hex returns
Null	Null
Empty	Zero (0)
Any other number	Up to eight hexadecimal characters

You can represent hexadecimal numbers directly by preceding numbers in the proper range with `&H`. For example, `&H10` represents decimal 16 in hexadecimal notation.

InStr Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctInstrC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctInStrX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctInStrS"}

Returns a **Variant** (**Long**) specifying the position of the first occurrence of one string within another.

Syntax

InStr([*start*,]*string1*, *string2*[, *compare*])

The **InStr** function syntax has these arguments:

Part	Description
start	Optional. Numeric expression that sets the starting position for each search. If omitted, search begins at the first character position. If *start* contains **Null**, an error occurs. The *start* argument is required if *compare* is specified.
string1	Required. String expression being searched.
string2	Required. String expression sought.
compare	Optional. Specifies the type of string comparison. The *compare* argument can be omitted, or it can be 0, 1or 2. Specify **0** (default) to perform a binary comparison. Specify **1** to perform a textual, noncase-sensitive comparison. For Microsoft Access only, specify **2** to perform a comparison based on information contained in your database. If *compare* is Null, an error occurs. If *compare* is omitted, the **Option Compare** setting determines the type of comparison.

Return Values

If	InStr returns
string1 is zero-length	0
string1 is **Null**	Null
string2 is zero-length	*start*
string2 is **Null**	Null
string2 is not found	0
string2 is found within *string1*	Position at which match is found
start > *string2*	0

Remarks

The **InStrB** function is used with byte data contained in a string. Instead of returning the character position of the first occurrence of one string within another, **InStrB** returns the byte position.

LCase Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctLCaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctLCaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctLCaseS"}

Returns a **String** that has been converted to lowercase.

Syntax

LCase(*string*)

The required *string* argument is any valid string expression. If *string* contains **Null**, Null is returned.

Remarks

Only uppercase letters are converted to lowercase; all lowercase letters and nonletter characters remain unchanged.

Left Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctLeftC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctLeftX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctLeftS"}

Returns a **Variant** (**String**) containing a specified number of characters from the left side of a string.

Syntax

Left(***string***, ***length***)

The **Left** function syntax has these named arguments:

Part	Description
string	Required. String expression from which the leftmost characters are returned. If ***string*** contains **Null**, Null is returned.
length	Required; **Variant** (**Long**). Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in ***string***, the entire string is returned.

Remarks

To determine the number of characters in ***string***, use the **Len** function.

Note Use the **LeftB** function with byte data contained in a string. Instead of specifying the number of characters to return, ***length*** specifies the number of bytes.

Len Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctLenC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctLenX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctLenS"}

Returns a **Long** containing the number of characters in a string or the number of bytes required to store a variable.

Syntax

Len(*string* | *varname*)

The **Len** function syntax has these parts:

Part	Description
string	Any valid string expression. If *string* contains **Null**, Null is returned.
Varname	Any valid variable name. If *varname* contains **Null**, **Null** is returned. If *varname* is a **Variant**, **Len** treats it the same as a **String** and always returns the number of characters it contains.

Remarks

One (and only one) of the two possible arguments must be specified. With user-defined types, **Len** returns the size as it will be written to the file.

Note Use the **LenB** function with byte data contained in a string. Instead of returning the number of characters in a string, **LenB** returns the number of bytes used to represent that string. With user-defined types, **LenB** returns the in-memory size, including any padding between elements.

Note **Len** may not be able to determine the actual number of storage bytes required when used with variable-length strings in user-defined data types.

LSet Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmLSetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmLSetX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmLSetS"}

Left aligns a string within a string variable, or copies a variable of one user-defined type to another variable of a different user-defined type.

Syntax

LSet *stringvar* **=** *string*
LSet *varname1* **=** *varname2*

The **LSet** statement syntax has these parts:

Part	Description
stringvar	Required. Name of string variable.
string	Required. String expression to be left-aligned within *stringvar.*
varname1	Required. Variable name of the user-defined type being copied to.
varname2	Required. Variable name of the user-defined type being copied from.

Remarks

LSet replaces any leftover characters in *stringvar* with spaces.

If *string* is longer than *stringvar*, **LSet** places only the leftmost characters, up to the length of the *stringvar,* in *stringvar*.

Warning Using **LSet** to copy a variable of one user-defined type into a variable of a different user-defined type is not recommended. Copying data of one data type into space reserved for a different data type can cause unpredictable results.

When you copy a variable from one user-defined type to another, the binary data from one variable is copied into the memory space of the other, without regard for the data types specified for the elements.

LTrim, RTrim, and Trim Functions

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctLTrimC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctLTrimX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctLTrimS"}

Returns a **Variant** (**String**) containing a copy of a specified string without leading spaces (**LTrim**), trailing spaces (**RTrim**), or both leading and trailing spaces (**Trim**).

Syntax

LTrim(*string*)
RTrim(*string*)
Trim(*string*)

The required *string* argument is any valid string expression. If *string* contains **Null**, **Null** is returned.

Mid Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctMidC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctMidX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctMidS"}

Returns a **Variant** (**String**) containing a specified number of characters from a string.

Syntax

Mid(***string***, ***start***[, ***length***])

The **Mid** function syntax has these named arguments:

Part	Description
string	Required. String expression from which characters are returned. If ***string*** contains **Null**, **Null** is returned.
start	Required; **Long**. Character position in ***string*** at which the part to be taken begins. If ***start*** is greater than the number of characters in ***string***, **Mid** returns a zero-length string ("").
length	Optional; **Variant** (**Long**). Number of characters to return. If omitted or if there are fewer than ***length*** characters in the text (including the character at ***start***), all characters from the ***start*** position to the end of the string are returned.

Remarks

To determine the number of characters in ***string***, use the **Len** function.

Note Use the **MidB** function with byte data contained in a string. Instead of specifying the number of characters, the arguments specify numbers of bytes.

Mid Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmMidC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmMidX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmMidS"}

Replaces a specified number of characters in a **Variant** (**String**) variable with characters from another string.

Syntax

Mid(*stringvar*, *start*[, *length*]) **=** *string*

The **Mid** statement syntax has these parts:

Part	Description
stringvar	Required. Name of string variable to modify.
start	Required; **Variant** (**Long**). Character position in *stringvar* where the replacement of text begins.
length	Optional; **Variant** (**Long**). Number of characters to replace. If omitted, all of *string* is used.
string	Required. String expression that replaces part of *stringvar*.

Remarks

The number of characters replaced is always less than or equal to the number of characters in *stringvar*.

Note Use the **MidB** statement with byte data contained in a string. In the **MidB** statement, *start* specifies the byte position within *stringvar* where replacement begins and *length* specifies the numbers of bytes to replace.

Oct Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctOctC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctOctX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctOctS"}

Returns a **Variant** (**String**) representing the octal value of a number.

Syntax

Oct(*number*)

The required *number* argument is any valid numeric expression or string expression.

Remarks

If *number* is not already a whole number, it is rounded to the nearest whole number before being evaluated.

If *number* is	Oct returns
Null	**Null**
Empty	Zero (0)
Any other number	Up to 11 octal characters

You can represent octal numbers directly by preceding numbers in the proper range with `&O`. For example, `&O10` is the octal notation for decimal 8.

Right Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctRightC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctRightX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctRightS"}

Returns a **Variant** (**String**) containing a specified number of characters from the right side of a string.

Syntax

Right(***string***, ***length***)

The **Right** function syntax has these named arguments:

Part	Description
string	Required. String expression from which the rightmost characters are returned. If ***string*** contains **Null**, **Null** is returned.
length	Required; **Variant** (**Long**). Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in ***string***, the entire string is returned.

Remarks

To determine the number of characters in ***string***, use the **Len** function.

Note Use the **RightB** function with byte data contained in a string. Instead of specifying the number of characters to return, ***length*** specifies the number of bytes.

RSet Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmRSetC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmRSetX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmRSetS"}

Right aligns a string within a string variable.

Syntax

RSet *stringvar* **=** *string*

The **RSet** statement syntax has these parts:

Part	Description
stringvar	Required. Name of string variable.
string	Required. String expression to be right-aligned within *stringvar*.

Remarks

If *stringvar* is longer than *string*, **RSet** replaces any leftover characters in *stringvar* with spaces, back to its beginning.

Note **RSet** can't be used with user-defined types.

Space Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctSpaceC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctSpaceX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctSpaceS"}

Returns a **Variant** (**String**) consisting of the specified number of spaces.

Syntax

Space(*number*)

The required *number* argument is the number of spaces you want in the string.

Remarks

The **Space** function is useful for formatting output and clearing data in fixed-length strings.

Str Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctStrC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctStrX":1}
{ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctStrS"}

Returns a **Variant** (**String**) representation of a number.

Syntax

Str(*number*)

The required *number* argument is a **Long** containing any valid numeric expression.

Remarks

When numbers are converted to strings, a leading space is always reserved for the sign of *number*. If *number* is positive, the returned string contains a leading space and the plus sign is implied.

Use the **Format** function to convert numeric values you want formatted as dates, times, or currency or in other user-defined formats. Unlike **Str**, the **Format** function doesn't include a leading space for the sign of *number*.

Note The **Str** function recognizes only the period (**.**) as a valid decimal separator. When different decimal separators may be used (for example, in international applications), use **CStr** to convert a number to a string.

StrComp Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctStrCompC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctStrCompX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctStrCompS"}

Returns a **Variant** (**Integer**) indicating the result of a string comparison.

Syntax

StrComp(***string1***, ***string2***[, ***compare***])

The **StrComp** function syntax has these named arguments:

Part	Description
string1	Required. Any valid string expression.
string2	Required. Any valid string expression.
compare	Optional. Specifies the type of string comparison. The ***compare*** argument can be omitted, or it can be 0, 1 or 2. Specify **0** (default) to perform a binary comparison. Specify **1** to perform a textual comparison. For Microsoft Access only, specify **2** to perform a comparison based on information contained in your database. If ***compare*** is **Null**, an error occurs. If ***compare*** is omitted, the **Option Compare** setting determines the type of comparison.

Return Values

The **StrComp** function has the following return values:

If	StrComp returns
string1 is less than ***string2***	-1
string1 is equal to ***string2***	0
string1 is greater than ***string2***	1
string1 or ***string2*** is **Null**	**Null**

StrConv Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctStrConvC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctStrConvX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctStrConvS"}

Returns a **Variant** (**String**) converted as specified.

Syntax

StrConv(***string, conversion***)

The **StrConv** function syntax has these named arguments:

Part	Description
string	Required. String expression to be converted.
conversion	Required; **Integer**. The sum of values specifying the type of conversion to perform.

Settings

The ***conversion*** argument settings are:

Constant	Value	Description
vbUpperCase	1	Converts the string to uppercase characters.
vbLowerCase	2	Converts the string to lowercase characters.
vbProperCase	3	Converts the first letter of every word in string to uppercase.
vbWide*	4*	Converts narrow (single-byte) characters in string to wide (double-byte) characters.
vbNarrow*	8*	Converts wide (double-byte) characters in string to narrow (single-byte) characters.
vbKatakana**	16**	Converts Hiragana characters in string to Katakana characters.
vbHiragana**	32**	Converts Katakana characters in string to Hiragana characters.
vbUnicode	64	Converts the string to Unicode using the default code page of the system.
vbFromUnicode	128	Converts the string from Unicode to the default code page of the system.

* Applies to Far East locales.

** Applies to Japan only.

Note These constants are specified by Visual Basic for Applications. As a result, they may be used anywhere in your code in place of the actual values. Most can be combined, for example, **vbUpperCase + vbWide**, except when they are mutually exclusive, for example, **vbUnicode + vbFromUnicode**. The constants **vbWide**, **vbNarrow**, **vbKatakana**, and **vbHiragana** cause run-time errors when used in locales where they do not apply.

The following are valid word separators for proper casing: **Null** (**Chr$(**0**)**), horizontal tab (**Chr$(**9**)**), linefeed (**Chr$(**10**)**), vertical tab (**Chr$(**11**)**), form feed (**Chr$(**12**)**), carriage return (**Chr$(**13**)**), space (SBCS) (**Chr$(**32**)**). The actual value for a space varies by country for DBCS.

String Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctStringC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctStringX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctStringS"}

Returns a **Variant** (**String**) containing a repeating character string of the length specified.

Syntax

String(***number***, ***character***)

The **String** function syntax has these named arguments:

Part	Description
number	Required; **Long**. Length of the returned string. If ***number*** contains **Null**, **Null** is returned.
character	Required; **Variant**. Character code specifying the character or string expression whose first character is used to build the return string. If ***character*** contains **Null**, **Null** is returned.

Remarks

If you specify a number for ***character*** greater than 255, **String** converts the number to a valid character code using the formula:

character **Mod** 256

UCase Function

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vafctUCaseC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vafctUcaseX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vafctUcaseS"}

Returns a **Variant** (**String**) containing the specified string, converted to uppercase.

Syntax

UCase(*string*)

The required *string* argument is any valid string expression. If *string* contains **Null**, **Null** is returned.

Remarks

Only lowercase letters are converted to uppercase; all uppercase letters and nonletter characters remain unchanged.

Keywords by Task

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Category	Description
Arrays	Creating, defining, and using arrays.
Compiler Directives	Controlling compiler behavior.
Control Flow	Looping and controlling procedure flow.
Conversion	Converting numbers and data types.
Data Types	Data types and variant subtypes.
Dates and Times	Converting and using date and time expressions.
Directories and Files	Controlling the file system and processing files.
Errors	Trapping and returning error values.
Financial	Performing financial calculations.
Input and Output	Receiving input and displaying or printing output.
Math	Performing trigonometric and other mathematical calculations.
Miscellaneous	Starting other applications and processing events.
Operators	Comparing expressions and other operations.
String Manipulation	Manipulating strings and string type data.
Variables and Constants	Declaring and defining variables and constants.

Arrays Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Verify an array.	**IsArray**
Create an array.	**Array**
Change default lower limit.	**Option Base**
Declare and initialize an array.	**Dim**, **Private**, **Public**, **ReDim**, **Static**
Find the limits of an array.	**LBound**, **UBound**
Reinitialize an array.	**Erase**, **ReDim**

Collection Object Keyword Summary

Action	Keywords
Create a **Collection** object.	**Collection**
Add an object to a collection.	**Add**
Remove an object from a collection.	**Remove**
Reference an item in a collection.	**Item**

Compiler Directive Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Define compiler constant.	**#Const**
Compile selected blocks of code.	**#If...Then...#Else**

Control Flow Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Branch.	**GoSub...Return**, **GoTo**, **On Error**, **On...GoSub**, **On...GoTo**
Exit or pause the program.	**DoEvents**, **End**, **Exit**, **Stop**
Loop.	**Do...Loop**, **For...Next**, **For Each...Next**, **While...Wend**, **With**
Make decisions.	**Choose**, **If...Then...Else**, **Select Case**, **Switch**
Use procedures.	**Call**, **Function**, **Property Get**, **Property Let**, **Property Set**, **Sub**

Conversion Keyword Summary

Action	Keywords
ANSI value to string.	**Chr**
String to lowercase or uppercase.	**Format**, **LCase**, **UCase**
Date to serial number.	**DateSerial**, **DateValue**
Decimal number to other bases.	**Hex**, **Oct**
Number to string.	**Format**, **Str**
One data type to another.	**CBool**, **CByte**, **CCur**, **CDate**, **CDbl**, **CDec**, **CInt**, **CLng**, **CSng**, **CStr**, **CVar**, **CVErr**, **Fix**, **Int**
Date to day, month, weekday, or year.	**Day**, **Month**, **Weekday**, **Year**
Time to hour, minute, or second.	**Hour**, **Minute**, **Second**
String to ASCII value.	**Asc**
String to number.	**Val**
Time to serial number.	**TimeSerial**, **TimeValue**

Data Types Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Convert between data types.	**CBool**, **CByte**, **CCur**, **CDate**, **CDbl**, **CDec**, **CInt**, **CLng**, **CSng**, **CStr**, **CVar**, **CVErr**, **Fix**, **Int**
Set intrinsic data types.	**Boolean**, **Byte**, **Currency**, **Date**, **Double**, **Integer**, **Long**, **Object**, **Single**, **String**, **Variant** (default)
Verify data types.	**IsArray**, **IsDate**, **IsEmpty**, **IsError**, **IsMissing**, **IsNull**, **IsNumeric**, **IsObject**

Dates and Times Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Get the current date or time.	**Date**, **Now**, **Time**
Perform date calculations.	**DateAdd**, **DateDiff**, **DatePart**
Return a date.	**DateSerial**, **DateValue**
Return a time.	**TimeSerial**, **TimeValue**
Set the date or time.	**Date**, **Time**
Time a process.	**Timer**

Directories and Files Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Change directory or folder.	**ChDir**
Change the drive.	**ChDrive**
Copy a file.	**FileCopy**
Make directory or folder.	**MkDir**
Remove directory or folder.	**RmDir**
Rename a file, directory, or folder.	**Name**
Return current path.	**CurDir**
Return file date/time stamp.	**FileDateTime**
Return file, directory, label attributes.	**GetAttr**
Return file length.	**FileLen**
Return file name or volume label.	**Dir**
Set attribute information for a file.	**SetAttr**

Errors Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Generate run-time errors.	**Clear**, **Error**, **Raise**
Get error messages.	**Error**
Provide error information.	**Err**
Return **Error** variant.	**CVErr**
Trap errors during run time.	**On Error**, **Resume**
Type verification.	**IsError**

Financial Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Calculate depreciation.	**DDB**, **SLN**, **SYD**
Calculate future value.	**FV**
Calculate interest rate.	**Rate**
Calculate internal rate of return.	**IRR**, **MIRR**
Calculate number of periods.	**NPer**
Calculate payments.	**IPmt**, **Pmt**, **PPmt**
Calculate present value.	**NPV**, **PV**

Input and Output Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Access or create a file.	**Open**
Close files.	**Close**, **Reset**
Control output appearance.	**Format**, **Print**, **Print #**, **Spc**, **Tab**, **Width #**
Copy a file.	**FileCopy**
Get information about a file.	**EOF**, **FileAttr**, **FileDateTime**, **FileLen**, **FreeFile**, **GetAttr**, **Loc**, **LOF**, **Seek**
Manage files.	**Dir**, **Kill**, **Lock**, **Unlock**, **Name**
Read from a file.	**Get**, **Input**, **Input #**, **Line Input #**
Return length of a file.	**FileLen**
Set or get file attributes.	**FileAttr**, **GetAttr**, **SetAttr**
Set read-write position in a file.	**Seek**
Write to a file.	**Print #**, **Put**, **Write #**

Math Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Derive trigonometric functions.	**Atn**, **Cos**, **Sin**, **Tan**
General calculations.	**Exp**, **Log**, **Sqr**
Generate random numbers.	**Randomize**, **Rnd**
Get absolute value.	**Abs**
Get the sign of an expression.	**Sgn**
Perform numeric conversions.	**Fix**, **Int**

Miscellaneous Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Process pending events.	**DoEvents**
Run other programs.	**AppActivate**, **Shell**
Send keystrokes to an application.	**SendKeys**
Sound a beep from computer.	**Beep**
System.	**Environ**
Provide a command-line string.	**Command**
Automation.	**CreateObject**, **GetObject**
Color.	**QBColor**, **RGB**

Operators Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Arithmetic.	**^**, **– , *, /, \, Mod, +, &**
Comparison.	**=**, **<>**, **<**, **>**, **<=**, **>=**, **Like**, **Is**
Logical operations.	**Not**, **And**, **Or**, **Xor**, **Eqv**, **Imp**

Registry Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Delete program settings.	**DeleteSetting**
Read program settings.	**GetSetting**, **GetAllSettings**
Save program settings.	**SaveSetting**

String Manipulation Keyword Summary

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaidxkeywordsbytaskC"}

Action	Keywords
Compare two strings.	**StrComp**
Convert strings.	**StrConv**
Convert to lowercase or uppercase.	**Format**, **LCase**, **UCase**
Create string of repeating character.	**Space**, **String**
Find length of a string.	**Len**
Format a string.	**Format**
Justify a string.	**LSet**, **RSet**
Manipulate strings.	**InStr**, **Left**, **LTrim**, **Mid**, **Right**, **RTrim**, **Trim**
Set string comparison rules.	**Option Compare**
Work with ASCII and ANSI values.	**Asc**, **Chr**

Variables and Constants Keyword Summary

Action	Keywords
Assign value.	**Let**
Declare variables or constants.	**Const**, **Dim**, **Private**, **Public**, **New**, **Static**
Declare module as private.	**Option Private Module**
Get information about a variant.	**IsArray**, **IsDate**, **IsEmpty**, **IsError**, **IsMissing**, **IsNull**, **IsNumeric**, **IsObject**, **TypeName**, **VarType**
Refer to current object.	**Me**
Require explicit variable declarations.	**Option Explicit**
Set default data type.	**Def**type

Trappable Errors

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamsgTrappableErrorsC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamsgTrappableErrorsS"}

Trappable errors can occur while an application is running. Some trappable errors can also occur during development or compile time. You can test and respond to trappable errors using the **On Error** statement and the **Err** object. Unused error numbers in the range 1 – 1000 are reserved for future use by Visual Basic.

Code	Message
3	Return without GoSub
5	Invalid procedure call
6	Overflow
7	Out of memory
9	Subscript out of range
10	This array is fixed or temporarily locked
11	Division by zero
13	Type mismatch
14	Out of string space
16	Expression too complex
17	Can't perform requested operation
18	User interrupt occurred
20	Resume without error
28	Out of stack space
35	Sub, Function, or Property not defined
47	Too many DLL application clients
48	Error in loading DLL
49	Bad DLL calling convention
51	Internal error
52	Bad file name or number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
58	File already exists
59	Bad record length
61	Disk full
62	Input past end of file
63	Bad record number
67	Too many files
68	Device unavailable
70	Permission denied
71	Disk not ready
74	Can't rename with different drive
75	Path/File access error
76	Path not found
91	Object variable or With block variable not set

92	For loop not initialized
93	Invalid pattern string
94	Invalid use of **Null**
97	Can't call Friend procedure on an object that is not an instance of the defining class
98	A property or method call cannot include a reference to a private object, either as an argument or as a return value (Error 98)
298	System DLL could not be loaded
320	Can't use character device names in specified file names
321	Invalid file format
322	Can’t create necessary temporary file
325	Invalid format in resource file
327	Data value named not found
328	Illegal parameter; can't write arrays
335	Could not access system registry
336	ActiveX component not correctly registered
337	ActiveX component not found
338	ActiveX component did not run correctly
360	Object already loaded
361	Can't load or unload this object
363	ActiveX control specified not found
364	Object was unloaded
365	Unable to unload within this context
368	The specified file is out of date. This program requires a later version
371	The specified object can't be used as an owner form for Show
380	Invalid property value
381	Invalid property-array index
382	Property Set can't be executed at run time
383	Property Set can't be used with a read-only property
385	Need property-array index
387	Property Set not permitted
393	Property Get can't be executed at run time
394	Property Get can't be executed on write-only property
400	Form already displayed; can't show modally
402	Code must close topmost modal form first
419	Permission to use object denied
422	Property not found
423	Property or method not found
424	Object required
425	Invalid object use
429	ActiveX component can't create object or return reference to this object
430	Class doesn't support Automation
432	File name or class name not found during Automation operation
438	Object doesn't support this property or method

440	Automation error
442	Connection to type library or object library for remote process has been lost
443	Automation object doesn't have a default value
445	Object doesn't support this action
446	Object doesn't support named arguments
447	Object doesn't support current locale setting
448	Named argument not found
449	Argument not optional or invalid property assignment
450	Wrong number of arguments or invalid property assignment
451	Object not a collection
452	Invalid ordinal
453	Specified DLL function not found
454	Code resource not found
455	Code resource lock error
457	This key is already associated with an element of this collection
458	Variable uses a type not supported in Visual Basic
459	This component doesn't support the set pf events
460	Invalid Clipboard format
461	Specified format doesn't match format of data
480	Can't create AutoRedraw image
481	Invalid picture
482	Printer error
483	Printer driver does not support specified property
484	Problem getting printer information from the system. Make sure the printer is set up correctly
485	Invalid picture type
486	Can't print form image to this type of printer
520	Can't empty Clipboard
521	Can't open Clipboard
735	Can't save file to TEMP directory
744	Search text not found
746	Replacements too long
31001	Out of memory
31004	No object
31018	Class is not set
31027	Unable to activate object
31032	Unable to create embedded object
31036	Error saving to file
31037	Error loading from file

Character Set (128 – 255)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamscANSITable2C"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamscANSITable2S"}

128	•	160	[space]	192	À	224	à
129	•	161	¡	193	Á	225	á
130	•	162	¢	194	Â	226	â
131	•	163	£	195	Ã	227	ã
132	•	164	¤	196	Ä	228	ä
133	•	165	¥	197	Å	229	å
134	•	166	¦	198	Æ	230	æ
135	•	167	§	199	Ç	231	ç
136	•	168	¨	200	È	232	è
137	•	169	©	201	É	233	é
138	•	170	ª	202	Ê	234	ê
139	•	171	«	203	Ë	235	ë
140	•	172	¬	204	Ì	236	ì
141	•	173		205	Í	237	í
142	•	174	®	206	Î	238	î
143	•	175	¯	207	Ï	239	ï
144	•	176	°	208	Ð	240	ð
145	‘	177	±	209	Ñ	241	ñ
146	’	178	²	210	Ò	242	ò
147	•	179	³	211	Ó	243	ó
148	•	180	´	212	Ô	244	ô
149	•	181	µ	213	Õ	245	õ
150	•	182	¶	214	Ö	246	ö
151	•	183	·	215	×	247	÷
152	•	184	¸	216	Ø	248	ø
153	•	185	¹	217	Ù	249	ù
154	•	186	º	218	Ú	250	ú
155	•	187	»	219	Û	251	û
156	•	188	¼	220	Ü	252	ü
157	•	189	½	221	Ý	253	ý
158	•	190	¾	222	Þ	254	þ
159	•	191	¿	223	ß	255	ÿ

- These characters aren't supported by Microsoft Windows.

AddressOf Operator

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vaoprAddressOfC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vaoprAddressOfX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vaoprAddressOfS"}

A unary operator that causes the address of the procedure it precedes to be passed to an API procedure that expects a function pointer at that position in the argument list.

Syntax

AddressOf *procedurename*

The required *procedurename* specifies the procedure whose address is to be passed. It must represent a procedure in a standard module module in the project in which the call is made.

Remarks

When a procedure name appears in an argument list, usually the procedure is evaluated, and the address of the procedure's return value is passed. **AddressOf** permits the address of the procedure to be passed to a Windows API function in a dynamic-link library (DLL), rather passing the procedure's return value. The API function can then use the address to call the Basic procedure, a process known as a callback. The **AddressOf** operator appears only in the call to the API procedure. However, in the **Declare** statement that describes the API function to which the pointer is passed, the procedure address argument must be declared **As Any**.

Although you can use **AddressOf** to pass procedure pointers among Basic procedures, you can't call a function through such a pointer from within Basic. This means, for example, that a class written in Basic can't make a callback to its controller using such a pointer. When using **AddressOf** to pass a procedure pointer among procedures within Basic, the parameter of the called procedure must be typed **As Long**.

Warning Using **AddressOf** may cause unpredictable results if you don't completely understand the concept of function callbacks. You must understand how the Basic portion of the callback works, and also the code of the DLL into which you are passing your function address. Debugging such interactions is difficult since the program runs in the same process as the development environment. In some cases, systematic debugging may not be possible.

Note You can create your own call-back function prototypes in DLLs compiled with Microsoft Visual C++ (or similar tools). To work with **AddressOf**, your prototype must use the __stdcall calling convention. The default calling convention (__cdecl) will not work with **AddressOf**.

Since the caller of a callback is not within your program, it is important that an error in the callback procedure not be propagated back to the caller. You can accomplish this by placing the **On Error Resume Next** statement at the beginning of the callback procedure.

Assert Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vamthAssertC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vamthAssertX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vamthAssertA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vamthAssertS"}

Conditionally suspends execution at the line on which the method appears.

Syntax

object.**Assert** *booleanexpression*

The **Assert** method syntax has the following object qualifier and argument:

Part	Description
object	Required. Always the **Debug** object.
booleanexpression	Required. An expression that evaluates to either **True** or **False**.

Remarks

Assert invocations work only within the development environment. When the module is compiled into an executable, the method calls on the **Debug** object are omitted.

All of *booleanexpression* is always evaluated. For example, even if the first part of an **And** expression evaluates **False**, the entire expression is evaluated.

Friend

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vakeyFriendC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vakeyFriendX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vakeyFriendS"}

Modifies the definition of a procedure in a class module to make the procedure callable from modules that are outside the class, but part of the project within which the class is defined.

Syntax

[**Private** | **Friend** | **Public**] [**Static**] [**Sub** | **Function** | **Property**] *procedurename*

The required *procedurename* is the name of the procedure to be made visible throughout the project, but not visible to controllers of the class.

Remarks

Public procedures in a class can be called from anywhere, even by controllers of instances of the class. Declaring a procedure **Private** prevents controllers of the object from calling the procedure, but also prevents the procedure from being called from within the project in which the class itself is defined. **Friend** makes the procedure visible throughout the project, but not to a controller of an instance of the object. **Friend** can appear only in class modules, and can only modify procedure names, not variables or types. Procedures in a class can access the **Friend** procedures of all other classes in a project. **Friend** procedures don't appear in the type library of their class. A **Friend** procedure can't be late bound.

RaiseEvent Statement

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vastmRaiseEventC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vastmRaiseEventX":1} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vastmRaiseEventS"}

Fires an event declared at module level within a class, form, or document.

Syntax

RaiseEvent *eventname* [**(***argumentlis***t)**]

The required *eventname* is the name of an event declared within the module and follows Basic variable naming conventions.

The **RaiseEvent** statement syntax has these parts:

Part	Description
eventname	Required. Name of the event to fire.
argumentlist	Optional. Comma-delimited list of variables, arrays, or expressions The *argumentlist* must be enclosed by parentheses. If there are no arguments, the parentheses must be omitted.

Remarks

If the event has not been declared within the module in which it is raised, an error occurs. The following fragment illustrates an event declaration and a procedure in which the event is raised.

```
' Declare an event at module level of a class module
Event LogonCompleted (UserName as String)

Sub
    ' Raise the event.
   RaiseEvent LogonCompleted ("AntoineJan")
End Sub
```

If the event has no arguments, including empty parentheses, in the **RaiseEvent**, invocation of the event causes an error. You can't use **RaiseEvent** to fire events that are not explicitly declared in the module. For example, if a form has a Click event, you can't fire its Click event using **RaiseEvent**. If you declare a Click event in the form module, it shadows the form's own Click event. You can still invoke the form's Click event using normal syntax for calling the event, but not using the **RaiseEvent** statement.

Event firing is done in the order that the connections are established. Since events can have **ByRef** parameters, a process that connects late may receive parameters that have been changed by an earlier event handler.

AddressOf Operator Example

The following example creates a form with a list box containing an alphabetically sorted list of the fonts in your system.

To run this example, create a form with a list box on it. The code for the form is as follows:

```
Option Explicit

Private Sub Form_Load()
    Module1.FillListWithFonts List1
End Sub
```

Place the following code in a module. The third argument in the definition of the EnumFontFamilies function is a **Long** that represents a procedure. The argument must contain the address of the procedure, rather than the value that the procedure returns. In the call to EnumFontFamilies, the third argument requires the **AddressOf** operator to return the address of the EnumFontFamProc procedure, which is the name of the callback procedure you supply when calling the Windows API function, **EnumFontFamilies**. Windows calls EnumFontFamProc once for each of the font families on the system when you pass **AddressOf** EnumFontFamProc to **EnumFontFamilies**. The last argument passed to **EnumFontFamilies** specifies the list box in which the information is displayed.

```
'Font enumeration types
Public Const LF_FACESIZE = 32
Public Const LF_FULLFACESIZE = 64

Type LOGFONT
        lfHeight As Long
        lfWidth As Long
        lfEscapement As Long
        lfOrientation As Long
        lfWeight As Long
        lfItalic As Byte
        lfUnderline As Byte
        lfStrikeOut As Byte
        lfCharSet As Byte
        lfOutPrecision As Byte
        lfClipPrecision As Byte
        lfQuality As Byte
        lfPitchAndFamily As Byte
        lfFaceName(LF_FACESIZE) As Byte
End Type

Type NEWTEXTMETRIC
        tmHeight As Long
        tmAscent As Long
        tmDescent As Long
        tmInternalLeading As Long
        tmExternalLeading As Long
        tmAveCharWidth As Long
        tmMaxCharWidth As Long
        tmWeight As Long
        tmOverhang As Long
        tmDigitizedAspectX As Long
        tmDigitizedAspectY As Long
        tmFirstChar As Byte
        tmLastChar As Byte
```

```
        tmDefaultChar As Byte
        tmBreakChar As Byte
        tmItalic As Byte
        tmUnderlined As Byte
        tmStruckOut As Byte
        tmPitchAndFamily As Byte
        tmCharSet As Byte
        ntmFlags As Long
        ntmSizeEM As Long
        ntmCellHeight As Long
        ntmAveWidth As Long
End Type

' ntmFlags field flags
Public Const NTM_REGULAR = &H40&
Public Const NTM_BOLD = &H20&
Public Const NTM_ITALIC = &H1&

'  tmPitchAndFamily flags
Public Const TMPF_FIXED_PITCH = &H1
Public Const TMPF_VECTOR = &H2
Public Const TMPF_DEVICE = &H8
Public Const TMPF_TRUETYPE = &H4

Public Const ELF_VERSION = 0
Public Const ELF_CULTURE_LATIN = 0

'  EnumFonts Masks
Public Const RASTER_FONTTYPE = &H1
Public Const DEVICE_FONTTYPE = &H2
Public Const TRUETYPE_FONTTYPE = &H4

Declare Function EnumFontFamilies Lib "gdi32" Alias _
    "EnumFontFamiliesA" _
    (ByVal hDC As Long, ByVal lpszFamily As String, _
    ByVal lpEnumFontFamProc As Long, LParam As Any) As Long
Declare Function GetDC Lib "user32" (ByVal hWnd As Long) As Long
Declare Function ReleaseDC Lib "user32" (ByVal hWnd As Long, _
    ByVal hDC As Long) As Long

Function EnumFontFamProc(lpNLF As LOGFONT, lpNTM As NEWTEXTMETRIC, _
    ByVal FontType As Long, LParam As ListBox) As Long
Dim FaceName As String
Dim FullName As String
    FaceName = StrConv(lpNLF.lfFaceName, vbUnicode)
    LParam.AddItem Left$(FaceName, InStr(FaceName, vbNullChar) - 1)
    EnumFontFamProc = 1
End Function

Sub FillListWithFonts(LB As ListBox)
Dim hDC As Long
    LB.Clear
    hDC = GetDC(LB.hWnd)
    EnumFontFamilies hDC, vbNullString, AddressOf EnumFontFamProc, LB
    ReleaseDC LB.hWnd, hDC
End Sub
```

Assert Method Example

The following example shows how to use the **Assert** method. The example requires a form with two button controls on it. The default button names are Command1 and Command2.

When the example runs, clicking the Command1 button toggles the text on the button between 0 and 1. Clicking Command2 either does nothing or causes an assertion, depending on the value displayed on Command1. The assertion stops execution with the last statement executed, the Debug.Assert line, highlighted.

```
Option Explicit
Private blnAssert As Boolean
Private intNumber As Integer

Private Sub Command1_Click()
    blnAssert = Not blnAssert
    intNumber = IIf(intNumber <> 0, 0, 1)
    Command1.Caption = intNumber
End Sub

Private Sub Command2_Click()
    Debug.Assert blnAssert
End Sub

Private Sub Form_Load()
    Command1.Caption = intNumber
    Command2.Caption = "Assert Tester"
End Sub
```

Friend Example

When placed in a class module, the following code makes the member variable dblBalance accessible to all users of the class within the project. Any user of the class can get the value; only code within the project can assign a value to that variable.

```
Private dblBalance As Double

Public Property Get Balance() As Double
    Balance = dblBalance
End Property

Friend Property Let Balance(dblNewBalance As Double)
    dblBalance = dblNewBalance
End Property
```

RaiseEvent Statment Example

The following example uses events to count off seconds during a demonstration of the fastest 100 meter race. The code illustrates all of the event-related methods, properties, and statements, including the **RaiseEvent** statement.

The class that raises an event is the event source, and the classes that implement the event are the sinks. An event source can have multiple sinks for the events it generates. When the class raises the event, that event is fired on every class that has elected to sink events for that instance of the object.

The example also uses a form (`Form1`) with a button (`Command1`), a label (`Label1`), and two text boxes (`Text1` and `Text2`). When you click the button, the first text box displays "From Now" and the second starts to count seconds. When the full time (9.84 seconds) has elapsed, the first text box displays "Until Now" and the second displays "9.84"

The code for `Form1` specifies the initial and terminal states of the form. It also contains the code executed when events are raised.

```
Option Explicit

Private WithEvents mText As TimerState

Private Sub Command1_Click()
    Text1.Text = "From Now"
    Text1.Refresh
    Text2.Text = "0"
    Text2.Refresh
    Call mText.TimerTask(9.84)
End Sub

Private Sub Form_Load()
    Command1.Caption = "Click to Start Timer"
    Text1.Text = ""
    Text2.Text = ""
    Label1.Caption = "The fastest 100 meters ever run took this long:"
    Set mText = New TimerState
    End Sub

Private Sub mText_ChangeText()
    Text1.Text = "Until Now"
    Text2.Text = "9.84"
End Sub

Private Sub mText_UpdateTime(ByVal dblJump As Double)
    Text2.Text = Str(Format(dblJump, "0"))
    DoEvents
End Sub
```

The remaining code is in a class module named TimerState. Included among the commands in this module are the **Raise Event** statements.

```
Option Explicit
Public Event UpdateTime(ByVal dblJump As Double)
Public Event ChangeText()

Public Sub TimerTask(ByVal Duration As Double)
    Dim dblStart As Double
    Dim dblSecond As Double
```

```
    Dim dblSoFar As Double
    dblStart = Timer
    dblSoFar = dblStart

    Do While Timer < dblStart + Duration
        If Timer - dblSoFar >= 1 Then
            dblSoFar = dblSoFar + 1
            RaiseEvent UpdateTime(Timer - dblStart)
        End If
    Loop

    RaiseEvent ChangeText

End Sub
```