

*WinWrap Basic Language*

# *AccuTerm***2000**

## **Scripting Language Reference Manual**

**AccuTerm 2000 release 4.0  
Manual revision Jan 10, 2001**

**AccuSoft Enterprises  
8041 Foothill Blvd.  
Sunland, California 91040**

**[www.asent.com](http://www.asent.com)  
Phone (818) 951-1891  
FAX (818) 951-3606**

---

# Forward

---

**AccuTerm 2000 uses the Sax Basic Engine (Copyright 1998 Sax Software Corp) to provide the core of the scripting environment in AccuTerm. This engine implements the WinWarp Basic Language (Copyright 1993-1997 Polar Engineering and Consulting).**

**This reference manual is the core WinWrap Basic Language reference, and does not include the language extensions and object reference information which is specific to AccuTerm. See the “Scripting” and “AccuTerm Object Reference” sections of the AccuTerm 2000 Programmers Guide for details on creating, executing and debugging scripts, language extensions and the AccuTerm object model.**

# WinWrap Basic

---

The WinWrap Basic Language provides the core language definition. It is Visual Basic for Applications(TM) compatible.

**WinWrap Basic Language**  
**Copyright 1993-1997 Polar Engineering and Consulting**  
**All rights reserved.**

**Printed Documentation**  
**Copyright 1993-1997 Polar Engineering and Consulting**  
**All rights reserved.**

**WinWrap** is a trademark of **Polar Engineering and Consulting**.  
**Visual Basic for Applications** is a trademark of **Microsoft Corporation**.

Polar Engineering and Consulting  
Phone: 907-776-5509  
FAX: 907-776-5508  
CompuServe: 75240,331  
Internet: 75240.331@compuserve.com

## Groups

---

<b>Declaration</b>	<b>#Reference, #Uses, Attribute, Class Module, Code Module, Const, Declare, Deftype, Dim, Enum...End Enum, Function...End Function, Object Module, Option, Private, Property...End Property, Public, ReDim, Static, Sub...End Sub, Type...End Type. WithEvents</b>
<b>Data Type</b>	<b>Any, Boolean, Byte, Currency, Date, Double, Integer, Long, Object, PortInt, Single, String, String*n, Variant, obj type, user enum, user type.</b>
<b>Assignment</b>	<b>Erase, Let, LSet, RSet, Set.</b>
<b>Flow Control</b>	<b>Call, Do...Loop, End, Exit, For...Next, For Each...Next, GoTo, If...ElseIf...Else...End If, MacroRun, MacroRunThis, Select Case...End Case, Stop, While...Wend.</b>
<b>Error Handling</b>	<b>Err, Error, On Error, Resume.</b>
<b>Conversion</b>	<b>Array, CBool, CByte, CCur, CDate, CDbI, CInt, CLng, CSng, CStr, CVar, CVDate, CVer, Val.</b>
<b>Variable Info</b>	<b>IsArray, IsDate, IsEmpty, IsError, IsMissing, IsNull, IsNumeric, IsObject, LBound, TypeName, UBound, VarType.</b>
<b>Constant</b>	<b>Empty, False, Nothing, Null, True, Win16, Win32.</b>
<b>Math</b>	<b>Abs, Atn, Cos, Exp, Fix, Int, Log, Randomize, Rnd, Sgn, Sin, Sqr, Tan.</b>
<b>String</b>	<b>Asc, AscB, AscW, Chr, ChrB, ChrW, Format, Hex, InStr, InStrB, InStrRev, LCase, Left, LeftB, Len, LenB, LTrim, Mid, MidB, Oct, Replace, Right, RightB, RTrim, Space, String, Str, StrComp, StrConv, Trim, UCase.</b>
<b>Object</b>	<b>CreateObject, GetObject, With...End With.</b>
<b>Time/Date</b>	<b>Date, DateAdd, DateDiff, DatePart, DateSerial, DateValue, Day, Hour, Minute, Month, Now, Second, Time, Timer, TimeSerial, TimeValue, Weekday, Year.</b>
<b>File</b>	<b>ChDir, ChDrive, Close, CurDir, Dir, EOF, FileAttr, FileCopy, FileDateTime, FileLen, FreeFile, Get, GetAttr, Input, Input, Kill, Line Input, Loc, Lock, LOF, Mkdir, Name, Open, Print, Put, Reset, Rmdir, Seek, Seek, SetAttr, Unlock, Write.</b>
<b>User Input</b>	<b>Dialog, GetFilePath, InputBox, MsgBox.</b>

<b>User Dialog</b>	<b>Begin Dialog...End Dialog, CancelButton, CheckBox, ComboBox, DropListBox, GroupBox, ListBox, OKButton, OptionButton, OptionGroup, Picture, PushButton, Text, TextBox.</b>
<b>Dialog Function</b>	<b>Dialog Func, DlgControlId, DlgCount, DlgEnable, DlgEnd, DlgFocus, DlgListBoxArray, DlgName, DlgNumber, DlgSetPicture, DlgText, DlgType, DlgValue, DlgVisible.</b>
<b>DDE</b>	<b>DDEExecute, DDEInitiate, DDEPoke, DDERequest, DDETerminate, DDETerminateAll.</b>
<b>Settings</b>	<b>DeleteSetting, GetAllSettings, GetSetting, SaveSetting</b>
<b>Miscellaneous</b>	<b>AboutWinWrapBasic, AppActivate, Attribute, Beep, CallersLine, Choose, Clipboard, Command, Debug.Print, DoEvents, Environ, IIf, MacroDir, QBColor, Rem, RGB, SendKeys, Shell, Wait.</b>
<b>Operator</b>	<b>Operators: +, -, ^, *, /, \, Mod, +, -, &amp;, =, &lt;&gt;, &lt;, &gt;, &lt;=, &gt;=, Like. Not, And, Or, Xor, Eqv, Imp, Is.</b>

## AboutWinWrapBasic Instruction

---

**Syntax**            `AboutWinWrapBasic [Timeout]`

**Group**             `Miscellaneous`

**Description**       `Show the WinWrap Basic about box.`

Parameter	Description
<i>Timeout</i>	This numeric value is the maximum number of seconds to show the about box. A value less than or equal to zero displays the about box until the user closes it. If this value is omitted then a three second timeout is used.

**Example**

```
Sub Main
    AboutWinWrapBasic
End Sub
```

## Abs Function

---

**Syntax**            `Abs (Num)`

**Group**             `Math`

**Description**       `Return the absolute value.`

Parameter	Description
<i>Num</i>	Return the absolute value of this number value.

**Example**

```
Sub Main
    Debug.Print Abs(9) ' 9
    Debug.Print Abs(0) ' 0
    Debug.Print Abs(-9) ' 9
End Sub
```

## Any Data Type

---

**Group**             `Data Type`

**Description**       `Any variable expression (Declare only).`

## AppActivate Instruction

---

<b>Syntax</b>	<code>AppActivate Title\$</code> -or- <code>AppActivate TaskID</code>						
<b>Group</b>	Miscellaneous						
<b>Description</b>	<p>Form 1: Activate the application top-level window titled <i>Title\$</i>. If no window by that title exists then the first window with at title that starts with <i>Title\$</i> is activated. If no window matches then an error occurs.</p> <p>Form 2: Activate the application top-level window for task <i>TaskID</i>. If no window for that task exists then an error occurs.</p>						
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>Title\$</i></td> <td>The name shown in the title bar of the window.</td> </tr> <tr> <td><i>TaskID</i></td> <td>This numeric value is the task identifier.</td> </tr> </tbody> </table>	Parameter	Description	<i>Title\$</i>	The name shown in the title bar of the window.	<i>TaskID</i>	This numeric value is the task identifier.
Parameter	Description						
<i>Title\$</i>	The name shown in the title bar of the window.						
<i>TaskID</i>	This numeric value is the task identifier.						
<b>See Also</b>	<b>SendKeys, Shell()</b> .						
<b>Example</b>	<pre>Sub Main     ' make ProgMan the active application     AppActivate "Program Manager" End Sub</pre>						

## Array Function

---

<b>Syntax</b>	<code>Array([expr[, ...]])</code>
<b>Group</b>	Conversion
<b>Description</b>	Return a variant value array containing the <i>exprs</i> .
<b>Example</b>	<pre>Sub Main     X = Array(0,1,4,9)     Debug.Print X(2) ' 4 End Sub</pre>

## Asc Function

---

<b>Syntax</b>	<code>Asc(S\$)</code>
<b>Group</b>	String
<b>Description</b>	Return the ASCII value.

Note: A similar function, AscB, returns the first byte in S\$. Another similar function, AscW, returns the Unicode number.

Parameter	Description
S\$	Return the ASCII value of the first char in this string value.

**See Also**

**Chr\$()**.

**Example**

```
Sub Main
    Debug.Print Asc("A") ' 65
End Sub
```

## Atn Function

---

**Syntax**            `Atn(Num)`

**Group**             `Math`

**Description**       `Return the arc tangent.`

Parameter	Description
<i>Num</i>	Return the arc tangent of this number value. This is the number of radians. There are 2*Pi radians in a full circle.

**Example**

```
Sub Main
    Debug.Print Atn(1)*4 ' 3.1415926535898
End Sub
```

## Attribute Definition/Statement

---

**Syntax**            `Attribute name = value`

**Group**             `Declaration`

**Description**       `All attribute definitions and statements are ignored except for:`

- ```
Public varname As Type
Attribute varname.VB_VarUserMemId = 0
```

Declares **Public** varname as the default property for a **class module** or **object module**.
- ```
Property [Get|Let|Set] propname ( ... )
Attribute propname.VB_UserMemId = 0
...
End Property
```

Declares **Property** propname as the default property for a **class module** or **object module**.

## Beep Instruction

---

<b>Syntax</b>	Beep
<b>Group</b>	Miscellaneous
<b>Description</b>	Sound the bell.
<b>Example</b>	<pre>Sub Main     Beep ' beep the bell End Sub</pre>

## Begin Dialog Definition

---

**Syntax**      `Begin Dialog UserDialog [X, Y,] DX, DY[, Title$] _  
                  [, .dialogfunc]  
                  User Dialog Item  
                  [User Dialog Item]...  
End Dialog`

**Group**        User Dialog

**Description** Define a **UserDialog** type to be used later in a **Dim As UserDialog** statement.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the screen to the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font. If this is omitted then the dialog will be centered.
<i>Y</i>	This number value is the distance from the top edge of the screen to the top edge of the dialog box. It is measured in 1/12 ths of the average character width for the dialog's font. If this is omitted then the dialog will be centered.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Title\$</i>	This string value is the title of the user dialog. If this is omitted then there is no title.
<i>dialogfunc</i>	This is the function name that implements the <b>DialogFunc</b> for this <b>UserDialog</b> . If this is omitted then the <b>UserDialog</b> doesn't have a dialogfunc.

User Dialog Item

One of: **CancelButton**, **CheckBox**, **ComboBox**,  
**DropListBox**, **GroupBox**, **ListBox**, **OKButton**,  
**OptionButton**, **OptionGroup**, **PushButton**, **Text**, **TextBox**.

---

**See Also**

**Dim As UserDialog.**

**Example**

```
Sub Main
  Begin Dialog UserDialog 200,120
    Text 10,10,180,15,"Please push the OK button"
    OKButton 80,90,40,20
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg ' show dialog (wait for ok)
End Sub
```

## Boolean Data Type

---

**Group** Data Type

**Description** A **True** or **False** value.

## Byte Data Type

---

**Group** Data Type

**Description** An 8 bit unsigned integer value.

## Call Instruction

---

**Syntax** `Call name[(arglist)]`  
-or-  
`name [arglist]`

**Group** Flow Control

**Description** Evaluate the *arglist* and call subroutine (or function) *name* with those values. Sub (or function) *name* must be previously defined by either a **Sub**, **Function** or **Property** definition. If *name* is a function then the result is discarded. If Call is omitted then *name* must be a subroutine and the *arglist* is not enclosed in parens.

**See Also** **Declare**, **Sub**.

**Example**

```

Sub Show(Title$,Value)
  Debug.Print Title$;"=";Value
End Sub

Sub Main
  Call Show("2000/9",2000/9) ' 222.2222222222
  Show "1<2",1<2           ' True
End Sub

```

## CallersLine Function

---

**Syntax** CallersLine[(Depth)]**Group** Miscellaneous**Description** Return the caller's line as a text string.

The text format is: "[macroname|subname#linenum] linetext".

Parameter	Description
<i>Depth</i>	This integer value indicates how deep into the stack to get the caller's line. If Depth = 0 then return the current line. If Depth = 1 then return the calling subroutine's current line, etc.. If Depth is greater than the call stack then a null string is returned. If this value is omitted then the depth is 1.

**Example**

```

Sub Main
  A
End Sub
Sub A
  Debug.Print CallersLine ' "[untitled 1]|Main# 2]
  A"
End Sub

```

## CancelButton Dialog Item Definition

---

**Syntax** CancelButton X, Y, DX, DY[, .Field]**Group** User Dialog**Description** Define a cancel button item. Pressing the Cancel button from a **Dialog** instruction causes a run-time error. (**Dialog**( ) function call returns 0.)

Parameter	Description
-----------	-------------

<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Field</i>	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this is omitted then the field name is "Cancel".

---

**See Also**

**Begin Dialog, Dim As UserDialog.**

**Example**

```
Sub Main
  Begin Dialog UserDialog 200,120
    Text 10,10,180,30,"Please push the Cancel
button"
    OKButton 40,90,40,20
    CancelButton 110,90,60,20
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg ' show dialog (wait for cancel)
  Debug.Print "Cancel was not pressed"
End Sub
```

## CBool Function

---

**Syntax**

CBool (Num| \$)

**Group**

Conversion

**Description**

Convert to a **boolean** value. Zero converts to **False**, while all other values convert to **True**.

Parameter	Description
<i>Num  \$</i>	Convert a number or string value to a boolean value.

**Example**

```
Sub Main
  Debug.Print CBool(-1) 'True
  Debug.Print CBool(0) 'False
  Debug.Print CBool(1) 'True
End Sub
```

## CByte Function

---

**Syntax**            CByte ( Num | \$ )

**Group**             Conversion

**Description**      Convert to a **byte** value.

Parameter	Description
Num \$	Convert a number or string value to a byte value.

**Example**

```
Sub Main
    Debug.Print CByte(1.6) ' 2
End Sub
```

## CCur Function

---

**Syntax**            CCur ( Num | \$ )

**Group**             Conversion

**Description**      Convert to a **currency** value.

Parameter	Description
Num \$	Convert a number or string value to a currency value.

**Example**

```
Sub Main
    Debug.Print CCur("1E6") ' 1000000
End Sub
```

## CDate Function

---

**Syntax**            CDate ( Num | \$ )  
-or-  
CVDate ( Num | \$ )

**Group**             Conversion

**Description**      Convert to a **date** value.

Parameter	Description
Num \$	Convert a number or string value to a date value.

**Example**

```
Sub Main
    Debug.Print CDate(2) ' 1/1/00
End Sub
```

## Cdbl Function

---

**Syntax** Cdbl(*Num* | \$)

**Group** Conversion

**Description** Convert to a **double** precision real.

Parameter	Description
<i>Num</i>   \$	Convert a number or string value to a double precision real.

**Example**

```
Sub Main
    Debug.Print Cdbl("1E6") ' 1000000
End Sub
```

## ChDir Instruction

---

**Syntax** ChDir *Name*\$

**Group** File

**Description** Change the current directory to *Name*\$.

Parameter	Description
<i>Name</i> \$	This string value is the path and name of the directory.

**See Also** ChDrive, CurDir\$().

**Example**

```
Sub Main
    ChDir "C:\\"
    Debug.Print CurDir$() ' "C:\\"
End Sub
```

## ChDrive Instruction

---

**Syntax** ChDrive *Drive*\$

**Group** File

**Description** Change the current drive to *Drive*\$.

Parameter	Description
<i>Drive</i> \$	This string value is the drive letter.

**See Also** ChDir, CurDir\$().

**Example**

```

Sub Main
  ChDrive "B"
  Debug.Print CurDir$() "B:\"
End Sub

```

## CheckBox Dialog Item Definition

---

**Syntax**            `CheckBox X, Y, DX, DY, Title$, .Field`

**Group**             User Dialog

**Description**      Define a checkbox item.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Field</i>	The value of the check box is accessed via this field. Checked is 1, and unchecked is 0.

**See Also**            **Begin Dialog, Dim As UserDialog.**

**Example**

```

Sub Main
  Begin Dialog UserDialog 200,120
    Text 10,10,180,15,"Please push the OK button"
    CheckBox 10,25,180,15,"&Check box",.Check
    OKButton 80,90,40,20
  End Dialog
  Dim dlg As UserDialog
  dlg.Check = 1
  Dialog dlg ' show dialog (wait for ok)
  Debug.Print dlg.Check
End Sub

```

## Choose Function

---

**Syntax**            `Choose(Index, expr[, ...])`

**Group**             Flow Control

**Description** Return the value of the *expr* indicated by *Index*.

Parameter	Description
<i>Index</i>	The numeric value indicates which <i>expr</i> to return. If this value is less than one or greater than the number of <i>exprs</i> then <b>Null</b> is returned.
<i>expr</i>	All expressions are evaluated.

**See Also** **If, Select Case, IIf( ).**

**Example**

```
Sub Main
    Debug.Print Choose(2, "Hi", "there") ' "there"
End Sub
```

## Chr\$ Function

---

**Syntax** Chr[ \$ ] ( *Num* )

**Group** String

**Description** Return a one char string for the ASCII value.

Note: A similar function, ChrB, returns a single byte ASCII string. Another similar function, ChrW, returns a single char Unicode string.

Parameter	Description
<i>Num</i>	Return one char string for this ASCII number value.

**See Also** **Asc( ).**

**Example**

```
Sub Main
    Debug.Print Chr$(48) ' "0"
End Sub
```

## CInt Function

---

**Syntax** CInt ( *Num* | \$ )

**Group** Conversion

**Description** Convert to a 16 bit **integer**. If *Num*|\$ is too big (or too small) to fit then an overflow error occurs.

Parameter	Description
<i>Num</i>  \$	Convert a number or string value to a 16 bit integer.

**Example**

```
Sub Main
  Debug.Print CInt(1.6) ' 2
End Sub
```

## Class Module

---

<b>Group</b>	Declaration
<b>Description</b>	<p>A class <i>module</i> implements an OLE Automation object.</p> <ul style="list-style-type: none"><li>• Has a set of <b>Public</b> <i>procedures</i> accessible from other <i>macros</i> and <i>modules</i>.</li><li>• These public symbols are accessed via an object variable.</li><li>• Public <b>Consts</b>, <b>Types</b>, arrays, fixed length strings are not allowed.</li><li>• A class module is similar to a <b>object module</b> except that no instance is automatically created.</li><li>• To create an instance use: <code>Dim Obj As classname</code> <code>Set Obj = New classname</code></li></ul>
<b>See Also</b>	<b>Code Module, Object Module, Uses.</b>

## Example

```
'A.WWB
'#Uses "File.CLS"
Sub Main
    Dim File As New File
    File.Attach "C:\AUTOEXEC.BAT"
    Debug.Print File.ReadLine
End Sub

'File.CLS
'File|New Module|Class Module
'Edit|Properties|Name=File
Option Explicit
Dim FN As Integer
Public Sub Attach(FileName As String)
    FN = FreeFile
    Open FileName For Input As #FN
End Sub
Public Sub Detach()
    If FN <> 0 Then Close #FN
    FN = 0
End Sub
Public Function ReadLine() As String
    Line Input #FN,ReadLine
End Function

Private Sub Class_Initialize()
    Debug.Print "Class_Initialize"
End Sub

Private Sub Class_Terminate()
    Debug.Print "Class_Terminate"
    Detach
End Sub
```

## Class\_Initialize Sub

---

<b>Syntax</b>	<pre>Private Sub Class_Initialize()     ... End Sub</pre>
<b>Group</b>	Declaration
<b>Description</b>	<b>Class module</b> initialization subroutine. Each time a new instance is created for a class module the <code>Class_Initialize</code> sub is called. If <code>Class_Initialize</code> is not defined then no special initialization occurs.
<b>See Also</b>	<b>Code Module</b> , <b>Class_Terminate</b> .

## Class\_Terminate Sub

---

<b>Syntax</b>	<pre>Private Sub Class_Terminate()     ... End Sub</pre>
<b>Group</b>	Declaration
<b>Description</b>	<b>Class module</b> termination subroutine. Each time an instance is destroyed for a class module the <code>Class_Terminate</code> sub is called. If <code>Class_Terminate</code> is not defined then no special termination occurs.
<b>See Also</b>	<b>Code Module</b> , <b>Class_Initialize</b> .

## Clipboard Instruction/Function

---

<b>Syntax</b>	<pre>Clipboard <i>Text</i>\$ -or- Clipboard[\$][ ( ) ]</pre>
<b>Group</b>	Miscellaneous
<b>Description</b>	Form 1: Set the clipboard to <i>Text</i> \$. This is like the Edit Copy menu command.

Form 2: Return the text in the clipboard.

Parameter	Description
<i>Text</i> \$	Put this string value into the clipboard.

<b>Example</b>	<pre>Sub Main     Debug.Print Clipboard\$()     Clipboard "Hello"     Debug.Print Clipboard\$() ' "Hello" End Sub</pre>
----------------	---

## CLng Function

---

<b>Syntax</b>	<code>CLng (Num  \$)</code>
<b>Group</b>	Conversion
<b>Description</b>	Convert to a 32 bit <b>long</b> integer. If <i>Num</i>  \$ is too big (or too small) to fit then an overflow error occurs.

Parameter	Description
-----------	-------------

*Num* | \$ Convert a number or string value to a 32 bit integer.

---

**Example**

```
Sub Main
    Debug.Print CLng(1.6) ' 2
End Sub
```

## Close Instruction

---

**Syntax** Close [[#]*StreamNum*][, ...]

**Group** File

**Description** Close *StreamNums*.

---

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros. If this is omitted then all open streams for the current <i>macro/module</i> are closed.

---

**See Also** **Open, Reset.**

**Example**

```
Sub Main
    ' read the first line of XXX and print it
    Open "XXX" For Input As #1
    Line Input #1,L$
    Debug.Print L$
    Close #1
End Sub
```

## Code Module

---

**Group** Declaration

**Description** A Code *module* implements a code library.

- Has a set of **Public procedures** accessible from other *macros* and *modules*.
- The public symbols are accessed directly.

**See Also** **Class Module, Object Module, Uses.**

## ComboBox Dialog Item Definition

---

**Syntax** ComboBox *X, Y, DX, DY, StrArray\$( ), .Field\$*

**Group** User Dialog

**Description** Define a combobox item. Combo boxes combine the functionality of an edit box and a list box.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>StrArray\$( )</i>	This one-dimensional array of strings establishes the list of choices. All the non-null elements of the array are used.
<i>Field\$</i>	The value of the combo box is accessed via this field. This is the text in the edit box.

**See Also** **Begin Dialog, Dim As UserDialog.**

**Example**

```

Sub Main
  Dim combos$(3)
  combos$(0) = "Combo 0"
  combos$(1) = "Combo 1"
  combos$(2) = "Combo 2"
  combos$(3) = "Combo 3"
  Begin Dialog UserDialog 200,120
    Text 10,10,180,15,"Please push the OK button"
    ComboBox 10,25,180,60,combos(),.combo$
    OKButton 80,90,40,20
  End Dialog
  Dim dlg As UserDialog
  dlg.combo$ = "none"
  Dialog dlg ' show dialog (wait for ok)
  Debug.Print dlg.combo$
End Sub

```

## Command\$ Function

---

**Syntax** Command[ \$ ]

**Group** Miscellaneous

**Description** Contains the value of the **MacroRun** parameters.

**See Also** **MacroRun.**

**Example**

```

Sub Main
  Debug.Print "Command line parameter is: ";
  Debug.Print Command$;
  Debug.Print " "
End Sub

```

## Const Definition

---

**Syntax**

```

[ | Private | Public ] _
Const name[type] [As Type] = expr[, ...]

```

**Group**

Declaration

**Description**

Define *name* as the value of *expr*. The *expr* may refer other constants or built-in functions. If the type of the constants is not specified, the type of *expr* is used. Constants defined outside a **Sub**, **Function** or **Property** block are available in the entire *macro/module*.

**Private** is assumed if neither **Private** or **Public** is specified.

Note: Const statement in a **Sub**, **Function** or **Property** block may not use **Private** or **Public**.

**Example**

```

Sub Main
  Const Pi = 4*Atn(1), e = Exp(1)
  Debug.Print Pi ' 3.14159265358979
  Debug.Print e ' 2.71828182845905
End Sub

```

## Cos Function

---

**Syntax**Cos(*Num*)**Group**

Math

**Description**

Return the cosine.

Parameter	Description
<i>Num</i>	Return the cosine of this number value. This is the number of radians. There are 2*Pi radians in a full circle.

**Example**

```

Sub Main
  Debug.Print Cos(1) ' 0.54030230586814
End Sub

```

## CreateObject Function

---

<b>Syntax</b>	CreateObject( <i>Class</i> \$)
<b>Group</b>	Object
<b>Description</b>	Create a new object of type <i>Class</i> \$. Use <b>Set</b> to assign the returned object to an object variable.

Parameter	Description
<i>Class</i> \$	This string value is the application's registered class name. If this application is not currently active it will be started.

**See Also**      **Objects.**

**Example**

```
Sub Main
    Dim App As Object
    Set App =
CreateObject("WinWrap.CppDemoApplication")
    App.Move 20,30 ' move icon to 20,30
    Set App = Nothing
    App.Quit      ' run-time error (no object)
End Sub
```

## CSng Function

---

<b>Syntax</b>	CSng( <i>Num</i>  \$)
<b>Group</b>	Conversion
<b>Description</b>	Convert to a <b>single</b> precision real. If <i>Num</i>  \$ is too big (or too small) to fit then an overflow error occurs.

Parameter	Description
<i>Num</i>  \$	Convert a number or string value to a single precision real.

**Example**

```
Sub Main
    Debug.Print CSng(Sqr(2)) ' 1.4142135381699
End Sub
```

## CStr Function

---

<b>Syntax</b>	CStr( <i>Num</i>  \$)
<b>Group</b>	Conversion

**Description** Convert to a **string**.

Parameter	Description
<i>Num</i>  \$	Convert a number or string value to a string value.

**Example**

```
Sub Main
    Debug.Print CStr(Sqr(2)) ' "1.4142135623731"
End Sub
```

## CurDir\$ Function

---

**Syntax** CurDir[\$]([*Drive*\$])

**Group** File

**Description** Return the current directory for *Drive*\$.

Parameter	Description
<i>Drive</i> \$	This string value is the drive letter. If this is omitted or null then return the current directory for the current drive.

**See Also** ChDir, ChDrive.

**Example**

```
Sub Main
    Debug.Print CurDir$()
End Sub
```

## Currency Data Type

---

**Group** Data Type

**Description** A 64 bit fixed point real. (A twos complement binary value scaled by 10000.)

## CVar Function

---

**Syntax** CVar(*Num*|\$)

**Group** Conversion

**Description** Convert to a **variant** value.

Parameter	Description
-----------	-------------

*Num*|\$ Convert a number or string value (or object reference) to a variant value.

---

**Example**

```
Sub Main
    Debug.Print CVar(Sqr(2)) ' 1.4142135623731
End Sub
```

## CVErr Function

---

**Syntax** CVErr(*Num*|\$)

**Group** Conversion

**Description** Convert to a **variant** that contains an error code. An error code can't be used in expressions.

---

Parameter	Description
<i>Num</i>  \$	Convert a number or string value to an error code.

---

**See Also** **IsError**.

**Example**

```
Sub Main
    Debug.Print CVErr(1) ' Error 1
End Sub
```

## Date Data Type

---

**Group** Data Type

**Description** A 64 bit real value. The whole part represents the date, while the fractional part is the time of day. (December 30, 1899 = 0.) Use #date# as a literal date value in an expression.

## Date Function

---

**Syntax** Date[\$]

**Group** Time/Date

**Description** Return today's date as a **date** value.

**See Also** **Now**, **Time**, **Timer**.

**Example**

```

Sub Main
  Debug.Print Date ' example: 1/1/1995
End Sub

```

## DateAdd Function

---

**Syntax** DateAdd(*interval*, *number*, *dateexpr*)

**Group** Time/Date

**Description** Return a **date** value a number of intervals from another date.

Parameter	Description
<i>interval</i>	This string value indicates which kind of interval to add.
<i>number</i>	Add this many intervals. Use a negative value to get an earlier date.
<i>dateexpr</i>	Calculate the new date relative to this date value. If this value is <b>Null</b> then <b>Null</b> is returned.

Interval	Description
yyy	Year
q	Quarter
m	Month
d	Day
w	Weekday
ww	Week
h	Hour
n	Minute
s	Second

**See Also** DateDiff, DatePart.

**Example**

```

Sub Main
  Debug.Print DateAdd("yyyy",1,#1/1/2000#) '1/1/2001
End Sub

```

## DateDiff Function

---

**Syntax** DateDiff(*interval*, *dateexpr1*, *dateexpr2*)

**Group** Time/Date

**Description** Return the number of intervals between two dates.

Parameter	Description
-----------	-------------

<i>interval</i>	This string value indicates which kind of interval to subtract.
<i>dateexpr1</i>	Calculate the from this date value to <i>dateexpr2</i> . If this value is <b>Null</b> then <b>Null</b> is returned.
<i>dateexpr2</i>	Calculate the from <i>dateexpr1</i> to this date value. If this value is <b>Null</b> then <b>Null</b> is returned.

Interval	Description
yyyy	Year
q	Quarter
m	Month
d	Day
w	Weekday
ww	Week
h	Hour
n	Minute
s	Second

**See Also**

**DateAdd, DatePart.**

**Example**

```
Sub Main
    Debug.Print DateDiff("yyyy", #1/1/1990#, #1/1/2000#)
' 10
End Sub
```

## DatePart Function

**Syntax**

`DatePart(interval, dateexpr)`

**Group**

Time/Date

**Description**

Return the number from the date corresponding to the interval.

Parameter	Description
<i>interval</i>	This string value indicates which kind of interval to extract.
<i>dateexpr</i>	Get the interval from this date value. If this value is <b>Null</b> then <b>Null</b> is returned.

Interval	Description (return value range)
yyyy	Year (100-9999)
q	Quarter (1-4)
m	Month (1-12)
d	Day (1-366)
w	Weekday (1-7)
ww	Week (1-53)
h	Hour (0-23)
n	Minute (0-59)

**See Also** [DateAdd](#), [DateDiff](#).

**Example**

```
Sub Main
    Debug.Print DatePart("yyyy", #1/1/2000#) ' 2000
End Sub
```

## DateSerial Function

---

**Syntax** `DateSerial(Year, Month, Day)`

**Group** Time/Date

**Description** Return a **date** value.

Parameter	Description
<i>Year</i>	This numeric value is the year (0 to 9999). (0 to 99 are interpreted as 1900 to 1999.)
<i>Month</i>	This numeric value is the month (1 to 12).
<i>Day</i>	This numeric value is the day (1 to 31).

**See Also** [DateValue](#), [TimeSerial](#), [TimeValue](#).

**Example**

```
Sub Main
    Debug.Print DateSerial(2000,7,4) '7/4/2000
End Sub
```

## DateValue Function

---

**Syntax** `DateValue(Date$)`

**Group** Math

**Description** Return the day part of the date encoded as a string.

Parameter	Description
<i>Date\$</i>	Convert this string value to the day part of date it represents.

**See Also** [DateSerial](#), [TimeSerial](#), [TimeValue](#).

**Example**

```
Sub Main
    Debug.Print DateValue("1/1/2000 12:00:01 AM")
    '1/1/2000
End Sub
```

## Day Function

---

**Syntax** `Day(dateexpr)`

**Group** Time/Date

**Description** Return the day of the month (1 to 31).

Parameter	Description
<i>dateexpr</i>	Return the day of the month for this date value. If this value is <b>Null</b> then <b>Null</b> is returned.

**See Also** `Date()`, `Month()`, `Weekday()`, `Year()`.

**Example**

```
Sub Main
    Debug.Print Day(#1/1/1900#) ' 1
End Sub
```

## DDEExecute Instruction

---

**Syntax** `DDEExecute ChanNum, Command$[, Timeout]`

**Group** DDE

**Description** Send the DDE Execute *Command\$* string via DDE *ChanNum*.

Parameter	Description
<i>ChanNum</i>	This is the channel number returned by the <b>DDEInitiate</b> function. Up to 10 channels may be used at one time.
<i>Command\$</i>	Send this command value to the server application. The interpretation of this value is defined by the server application.
<i>Timeout</i>	The command will generate an error if the number of seconds specified by the timeout is exceeded before the command has completed. The default is five seconds.

**Example**

```
Sub Main
    ChanNum = DDEInitiate("PROGMAN", "PROGMAN")
    DDEExecute ChanNum, "[CreateGroup(XXX)]"
    DDETerminate ChanNum
End Sub
```

## DDEInitiate Function

---

**Syntax** `DDEInitiate(App$, Topic$)`

**Group** DDE

**Description** Initiate a DDE conversation with *App\$* using *Topic\$*. If the conversation is successfully started then the return value is a channel number that can be used with other DDE instructions and functions.

Parameter	Description
<i>App\$</i>	Locate this server application.
<i>Topic\$</i>	This is the server application's topic. The interpretation of this value is defined by the server application.

**Example**

```

Sub Main
    ChanNum = DDEInitiate("PROGMAN", "PROGMAN")
    DDEExecute ChanNum, "[CreateGroup(XXX)]"
    DDETerminate ChanNum
End Sub

```

## DDEPoke Instruction

**Syntax** DDEPoke *ChanNum*, *Item\$*, *Data\$*[, *Timeout*]

**Group** DDE

**Description** Poke *Data\$* to the *Item\$* via DDE *ChanNum*.

Parameter	Description
<i>ChanNum</i>	This is the channel number returned by the <b>DDEInitiate</b> function. Up to 10 channels may be used at one time.
<i>Item\$</i>	This is the server application's item. The interpretation of this value is defined by the server application.
<i>Data\$</i>	Send this data value to the server application. The interpretation of this value is defined by the server application.
<i>Timeout</i>	The command will generate an error if the number of seconds specified by the timeout is exceeded before the command has completed. The default is five seconds.

**Example**

```

Sub Main
    ChanNum = DDEInitiate("PROGMAN", "PROGMAN")
    DDEPoke ChanNum, "Group", "XXX"
    DDETerminate ChanNum
End Sub

```

## DDERequest\$ Function

**Syntax** DDERequest[\$](*ChanNum*, *Item\$*[, *Timeout*])

**Group** DDE

**Description** Request information for *Item\$*. If the request is not satisfied then the return value will be a null string.

Parameter	Description
<i>ChanNum</i>	This is the channel number returned by the <b>DDEInitiate</b> function. Up to 10 channels may be used at one time.
<i>Item\$</i>	This is the server application's item. The interpretation of this value is defined by the server application.
<i>Timeout</i>	The command will generate an error if the number of seconds specified by the timeout is exceeded before the command has completed. The default is five seconds.

**Example**

```
Sub Main
  ChanNum = DDEInitiate("PROGMAN", "PROGMAN")
  Debug.Print DDERequest$(ChanNum, "Groups")
  DDETerminate ChanNum
End Sub
```

## DDETerminate Instruction

---

**Syntax** DDETerminate *ChanNum*

**Group** DDE

**Description** Terminate DDE *ChanNum*.

Parameter	Description
<i>ChanNum</i>	This is the channel number returned by the <b>DDEInitiate</b> function. Up to 10 channels may be used at one time.

**Example**

```
Sub Main
  ChanNum = DDEInitiate("PROGMAN", "PROGMAN")
  DDEExecute ChanNum, "[CreateGroup(XXX)]"
  DDETerminate ChanNum
End Sub
```

## DDETerminateAll Instruction

---

**Syntax** DDETerminateAll

**Group** DDE

**Description** Terminate all open DDE channels.

**Example**

```

Sub Main
  ChanNum = DDEInitiate("PROGMAN", "PROGMAN")
  DDEExecute ChanNum, "[CreateGroup(XXX)]"
  DDETerminateAll
End Sub

```

## Debug Object

---

**Syntax**            `Debug.Print [expr[; ...]][;]`

**Group**             Miscellaneous

**Description**      Print the *expr(s)* to the output window. Use ; to separate expressions. A *num* is it automatically converted to a string before printing (just like `Str$( )`). If the instruction does not end with a ; then a newline is printed at the end.

**Example**

```

Sub Main
  X = 4
  Debug.Print "X/2=";X/2 ' 2
  Debug.Print "Start..."; ' don't print a newline
  Debug.Print "Finish" ' print a newline
End Sub

```

## Declare Definition

---

**Syntax**

```

[ | Private | Public ] _
Declare Sub name Lib "dll name" _
  [Alias "module name"] [(param[, ...])]
-or-
[ | Private | Public ] _
Declare Function name[type] Lib "dll name" _
  [Alias "module name"] [(param[, ...])] [As type]

```

**Group**             Declaration

**Description**      Interface to a DLL defined subroutine or function. The values of the calling *arglist* are assigned to the *params*.

Declare defaults to **Public** if neither **Private** or **Public** is specified.

**WARNING!** Be very careful when declaring DLL subroutines or functions. If you make a mistake and declare the parementers or result incorrectly then Windows might halt. Save any open

documents before testing new DLL declarations.

**Err.LastDLLError** returns the error code for that last DLL call (Windows 32 bit versions only).

Parameter	Description
<i>name</i>	This is the name of the subroutine or function being defined. If Alias "module name" is omitted then this is the module name, too.
"dll name"	This is the DLL file where the module's code is.
"module name"	This is the name of the module in the DLL file. If this is #number then it is the ordinal number of the module. If it is omitted then <i>name</i> is the module name. The DLL is searched for the specified module name. For Win16, this is the only module name checked. For Win32, if this module exists, it is used. All As String parameters are converted from Unicode to ASCII prior to calling the DLL and from ASCII to Unicode afterwards. (Use "Unicode:module name" to prevent ASCII to Unicode conversion.) If the module does not exist, one or two other module names are tried: 1) For Windows NT only: The module name with a "W" appended is tried. All As String parameters are passed as Unicode to calling the DLL. 2) For Windows NT and Windows 95: The module name with an "A" appended is tried. All As String parameters are converted from Unicode to ASCII prior to calling the DLL and from ASCII to Unicode afterwards. If none of these module names is found a run-time error occurs.
<i>params</i>	A list of zero or more <i>params</i> that are used by the DLL subroutine or function. (Note: A ByVal string's value may be modified by the DLL.)

**See Also**

**Function, Sub, Call.**

## Example

```
'Win16 example
Declare Function GetActiveWindow% Lib "user" ()
Declare Function GetWindowTextLength% Lib "user" _
    (ByVal hwnd%)
Declare Sub GetWindowText Lib "user" _
    (ByVal hwnd%, ByVal lpsz$, ByVal cbMax%)

Function ActiveWindowTitle$()
    ActiveWindow = GetActiveWindow()
    TitleLen = GetWindowTextLength(ActiveWindow)
    Title$ = Space$(TitleLen)
    GetWindowText ActiveWindow, Title$, TitleLen+1
    ActiveWindowTitle$ = Title$
End Function

Sub Main
    Debug.Print ActiveWindowTitle$()
End Sub

'Win32 example
Declare Function GetActiveWindow& Lib "user32" ()
Declare Function GetWindowTextLengthA& Lib "user32" _
    (ByVal hwnd&)
Declare Sub GetWindowTextA Lib "user32" _
    (ByVal hwnd&, ByVal lpsz$, ByVal cbMax%)

Function ActiveWindowTitle$()
    ActiveWindow = GetActiveWindow()
    TitleLen = GetWindowTextLengthA(ActiveWindow)
    Title$ = Space$(TitleLen)
    GetWindowTextA ActiveWindow, Title$, TitleLen+1
    ActiveWindowTitle$ = Title$
End Function

Sub Main
    Debug.Print ActiveWindowTitle$()
End Sub
```

## Def Definition

---

<b>Syntax</b>	Def{Bool Cur Date Dbl Int Lng Obj Sng Str Var} _ letterrange[, ...]
<b>Group</b>	Declaration
<b>Description</b>	Define untyped variables as: <ul style="list-style-type: none"><li>• DefBool - <b>Boolean</b></li><li>• DefByte - <b>Byte</b></li></ul>

- DefCur - **Currency**
- DefDate - **Date**
- DefDbl - **Double**
- DefInt - **Integer**
- DefLng - **Long**
- DefObj - **Object**
- DefSng - **Single**
- DefStr - **String**
- DefVar - **Variant**

Parameter	Description
letterrange	<p>letter, or letter-letter: A letter is one of A to Z. When letter-letter is used, the first letter must be alphabetically before the second letter. Variable names that begin with a letter in this range default to declared type.</p> <p>If a variable name begins with a letter not specific in any letterrange then the variable is a <b>Variant</b>. The letterranges are not allowed to overlap.</p>

#### See Also

#### Option.

#### Example

```

DefInt A,C-W,Y' integer
DefBool B      ' boolean
DefStr X       ' string
              ' all others are variant

Sub Main
  B = 1          ' B is an boolean
  Debug.Print B ' True
  X = "A"       ' X is a string
  Debug.Print X ' "A"
  Z = 1         ' Z is a variant (anything)
  Debug.Print Z ' 1
  Z = "Z"
  Debug.Print Z ' "Z"
End Sub

```

## DeleteSetting Instruction

**Syntax** DeleteSetting *AppName\$, Section\$[, Key\$]*

**Group** Settings

**Description** Delete the settings for *Key* in *Section* in project *AppName*. Win16 and Win32s store settings in a .ini file named *AppName*. Win32 stores settings in the registration database.

Parameter	Description
<i>AppName</i> \$	This string value is the name of the project which has this <i>Section</i> and <i>Key</i> .
<i>Section</i> \$	This string value is the name of the section of the project settings.
<i>Key</i> \$	This string value is the name of the key in the section of the project settings. If this is omitted then delete the entire section.

**Example**

```
Sub Main
    SaveSetting "MyApp", "Font", "Size", 10
    DeleteSetting "MyApp", "Font", "Size"
End Sub
```

## Dialog Instruction/Function

**Syntax** `Dialog dialogvar[, default]`  
 -or-  
`Dialog(dialogvar[, default])`

**Group** User Input

**Description** Display the dialog associated with *dialogvar*. The initial values of the dialog fields are provided by *dialogvar*. If the **OK button** or any **push button** is pressed then the fields in dialog are copied to the *dialogvar*. The Dialog() function returns a value indicating which button was pressed. (See the result table below.)

Parameter	Description
<i>dlgvar</i>	This variable that holds the values of the fields in a dialog. Use <i>.field</i> to access individual fields in a dialog variable.
<i>default</i>	This numeric value indicates which button is the default button. (Pressing the Enter key on a non-button pushes the default button.) Use -2 to indicate that there is no default button. Other possible values are shown the result table below. If this value is omitted then the first <b>PushButton</b> , <b>OKButton</b> or <b>CancelButton</b> is the default button.

Result	Description
-1	<b>OK button</b> was pressed.
0	<b>Cancel button</b> was pressed.
>0	Nth <b>push button</b> was pressed.

**See Also** **Begin Dialog**, **Dim As UserDialog**.

## Example

```
Sub Main
Begin Dialog UserDialog 200,120
Text 10,10,180,15,"Please push the OK button"
OKButton 80,90,40,20
End Dialog
Dim dlg As UserDialog
Dialog dlg ' show dialog (wait for ok)
End Sub
```

## DialogFunc Prototype

---

### Syntax

```
Function dialogfunc(DlgItem$, Action%, SuppValue%) _
    As Boolean
Select Case Action%
Case 1 ' Dialog box initialization
...
Case 2 ' Value changing or button pressed
...
Case 3 ' TextBox or ComboBox text changed
...
Case 4 ' Focus changed
...
Case 5 ' Idle
...
Case 6 ' Function key
...
End Select
End Function
```

### Group

Dialog Function

### Description

A *dialogfunc* implements the dynamic dialog capabilities.

Parameter	Description
<i>DlgItem</i>	This string value is the name of the user dialog item's <i>field</i> .
<i>Action</i>	This numeric value indicates what action the dialog function is being asked to do.
<i>SuppValue</i>	This numeric value provides additional information for some actions.

---

Action	Description
1	Dialog box initialization. <i>DlgItem</i> is a null string. <i>SuppValue</i> is the dialog's window handle. Set <i>dialogfunc</i> = <b>True</b> to terminate the dialog.
2	<b>CheckBox</b> , <b>DropListBox</b> , <b>ListBox</b> or <b>OptionGroup</b> : <i>DlgItem</i> 's value has changed. <i>SuppValue</i> is the new value. <b>CancelButton</b> , <b>OKButton</b> or <b>PushButton</b> : <i>DlgItem</i> 's button was pushed. <i>SuppValue</i> is meaningless. Set <i>dialogfunc</i> = <b>True</b> to prevent the dialog from closing.

- 3           **ComboBox** or **TextBox**: *DlgItem*'s text changed and losing focus. *SuppValue* is the number of characters.
  - 4           Item *DlgItem* is gaining focus. *SuppValue* is the item that is losing focus. (The first item is 0, second is 1, etc.) Note: When the first item receives the focus there is no item that is losing focus. In that case *SuppValue* is -1.
  - 5           Idle processing. *DlgItem* is a null string. *SuppValue* is zero. Set *dialogfunc* = **True** to continue receiving idle actions.
  - 6           Function key (F1-F24) was pressed. *DlgItem* has the focus. *SuppValue* is the function key number and the shift/control/alt key state.  
             Regular function keys range from 1 to 24.  
             Shift function keys have &H100 added.  
             Control function keys have &H200 added.  
             Alt function keys have &H400 added.  
             (Alt-F4 closes the dialog and is never passed to the Dialog Function.)
- 

**See Also**  
**Example**

**Begin Dialog.**

```

Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
  Text 10,10,180,15,"Please push the OK button"
  TextBox 10,40,180,15,.Text
  OKButton 30,90,60,20
  PushButton 110,90,60,20,"&Hello"
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Debug.Print "Action=";Action%
  Debug.Print DlgItem$;"="";DlgText$(DlgItem$);""
  Debug.Print "SuppValue=";SuppValue%
  Select Case Action%
  Case 1 ' Dialog box initialization
  Beep
  Case 2 ' Value changing or button pressed
  If DlgItem$ = "Hello" Then
    MsgBox "Hello"
    DialogFunc% = True 'do not exit the dialog
  End If
  Case 4 ' Focus changed
  Debug.Print "DlgFocus="";DlgFocus();""
  Case 6 ' Function key
  If SuppValue And &H100 Then Debug.Print "Shift-";
  If SuppValue And &H200 Then Debug.Print "Ctrl-";
  If SuppValue And &H400 Then Debug.Print "Alt-";
  Debug.Print "F" & (SuppValue And &HFF)
  End Select
End Function

```

## Dim Definition

---

<b>Syntax</b>	<code>Dim [WithEvents] name[type]([dim[, ...]])[As [New] type][, ...]</code>
<b>Group</b>	Declaration
<b>Description</b>	Dimension var array(s) using the <i>dims</i> to establish the minimum and maximum index value for each dimension. If the <i>dims</i> are omitted then a scalar (single value) variable is defined. A dynamic array is declared using ( ) without any <i>dims</i> . It must be <b>ReDimensioned</b> before it can be used.
<b>See Also</b>	<b>Begin Dialog, Dialog, Option Base, Private, Public, ReDim, Static, WithEvents.</b>
<b>Example</b>	<pre>Sub DoIt(Size)     Dim C0,C1(),C2(2,3)     ReDim C1(Size) ' dynamic array     C0 = 1     C1(0) = 2     C2(0,0) = 3     Debug.Print C0;C1(0);C2(0,0) ' 1 2 3 End Sub  Sub Main     DoIt 1 End Sub</pre>

## Dir\$ Function

---

<b>Syntax</b>	<code>Dir[\$]([Pattern\$][, AttribMask])</code>
<b>Group</b>	File
<b>Description</b>	Scan a directory for the first file matching <i>Pattern\$</i> .
<hr/>	
<b>Parameter</b>	<b>Description</b>
<i>Pattern\$</i>	This string value is the path and name of the file search pattern. If this is omitted then continue scanning with the previous pattern. Each <i>macro</i> has its own independent search. A path relative to the current directory can be used.
<i>AttribMask</i>	This numeric value controls which files are found. A file with an <i>attribute</i> that matches will be found.
<hr/>	
<b>See Also</b>	<b>GetAttr( ).</b>

**Example**

```
Sub Main
  F$ = Dir$("*.*)"
  While F$ <> ""
    Debug.Print F$
    F$ = Dir$()
  Wend
End Sub
```

## DlgControlId Function

---

**Syntax** DlgControlId(*DlgItem*|\$)

**Group** Dialog Function

**Description** Return the *field's* window id.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

---

Parameter	Description
<i>DlgItem</i>  \$	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>field</i> name.

---

## Example

```
Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
  Text 10,10,180,15,"Please push the OK button"
  TextBox 10,40,180,15,.Text
  OKButton 30,90,60,20
  PushButton 110,90,60,20,"&Hello"
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Debug.Print "Action=";Action%
  Select Case Action%
  Case 1 ' Dialog box initialization
  Beep
  Case 2 ' Value changing or button pressed
  If DlgItem$ = "Hello" Then
    DialogFunc% = True 'do not exit the dialog
  End If
  Case 4 ' Focus changed
  Debug.Print "DlgFocus="";DlgFocus();""
  Debug.Print "DlgControlId(";DlgItem$;")=";
  Debug.Print DlgControlId(DlgItem$)
  End Select
End Function
```

## DlgCount Function

---

<b>Syntax</b>	DlgCount ( )
<b>Group</b>	Dialog Function
<b>Description</b>	Return the number of dialog items in the dialog.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

## Example

```
Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,.Text
    OKButton 30,90,60,20
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Debug.Print "Action=";Action%
  Select Case Action%
  Case 1 ' Dialog box initialization
    Beep
  Debug.Print "DlgCount=";DlgCount() ' 3
  End Select
End Function
```

## DlgEnable Instruction/Function

---

**Syntax**            DlgEnable *DlgItem*|\$[, *Enable*]  
                     -or-  
                     DlgEnable(*DlgItem*|\$)

**Group**             Dialog Function

**Description**      Instruction: Enable or disable *DlgItem*|\$.

                     Function: Return **True** if *DlgItem*|\$ is enabled.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

---

Parameter	Description
<i>DlgItem</i>  \$	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>field</i> name.
<i>Enable</i>	If this numeric value is <b>True</b> then enable <i>DlgItem</i>  \$. Otherwise, disable it. If this omitted then toggle it.

---

## Example

```
Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,.Text
    OKButton 30,90,60,20
    PushButton 110,90,60,20,"&Disable"
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Debug.Print "Action=";Action%
  Select Case Action%
  Case 1 ' Dialog box initialization
    Beep
  Case 2 ' Value changing or button pressed
    Select Case DlgItem$
    Case "Disable"
      DlgText DlgItem$,"&Enable"
      DlgEnable "Text",False
      DialogFunc% = True 'do not exit the dialog
    Case "Enable"
      DlgText DlgItem$,"&Disable"
      DlgEnable "Text",True
      DialogFunc% = True 'do not exit the dialog
    End Select
  End Select
End Function
```

## DlgEnd Instruction

---

<b>Syntax</b>	DlgEnd <i>ReturnCode</i>
<b>Group</b>	Dialog Function
<b>Description</b>	Set the return code for the <b>Dialog</b> Function and close the user dialog.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

---

Parameter	Description
<i>ReturnCode</i>	Return this numeric value.

---

## Example

```
Sub Main
  Begin Dialog UserDialog 210,120,.DialogFunc
    Text 10,10,190,15,"Please push the Close
button"
    OKButton 30,90,60,20
    CheckBox 120,90,60,20,"&Close",.CheckBox1
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Debug.Print "Action=";Action%
  Select Case Action%
  Case 1 ' Dialog box initialization
    Beep
  Case 2 ' Value changing or button pressed
    Select Case DlgItem$
    Case "CheckBox1"
      DlgEnd 1000
    End Select
  End Select
End Function
```

## DlgFocus Instruction/Function

---

**Syntax** DlgFocus *DlgItem*|\$  
-or-  
DlgFocus[\$]()

**Group** Dialog Function

**Description** Instruction: Move the focus to this *DlgItem*|\$.

Function: Return the *field* name which has the focus as a string.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
<i>DlgItem</i>  \$	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>field</i> name.

---

## Example

```
Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,.Text
    OKButton 30,90,60,20
    PushButton 110,90,60,20,"&Hello"
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Debug.Print "Action=";Action%
  Select Case Action%
  Case 1 ' Dialog box initialization
    Beep
  Case 2 ' Value changing or button pressed
  If DlgItem$ = "Hello" Then
    MsgBox "Hello"
    DialogFunc% = True 'do not exit the dialog
  End If
  Case 4 ' Focus changed
  Debug.Print "DlgFocus="";DlgFocus();""
  End Select
End Function
```

## DlgListBoxArray Instruction/Function

---

**Syntax** DlgListBoxArray *DlgItem*|\$, *StrArray*\$( )  
-or-  
DlgListBoxArray(*DlgItem*|\$[, *StrArray*\$( )])

**Group** Dialog Function

**Description** Instruction: Set the list entries for *DlgItem*|\$.

Function: Return the number entries in *DlgItem*|\$'s list.

This instruction/function must be called directly or indirectly from a *dialogfunc*. The *DlgItem*|\$ should refer to a **ComboBox**, **DropListBox** or **ListBox**.

Parameter	Description
<i>DlgItem</i>  \$	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>field</i> name.
<i>StrArray</i> \$( )	Set the list entries of <i>DlgItem</i>  \$. This one-dimensional array of strings establishes the list of choices. All the non-null elements of the array are used.

## Example

```
Dim lists$()

Sub Main
  ReDim lists$(0)
  lists$(0) = "List 0"
  Begin Dialog UserDialog 200,119,.DialogFunc
    Text 10,7,180,14,"Please push the OK button"
    ListBox 10,21,180,63,lists(),.list
    OKButton 30,91,40,21
    PushButton 110,91,60,21,"&Change"
  End Dialog
  Dim dlg As UserDialog
  dlg.list = 2
  Dialog dlg ' show dialog (wait for ok)
  Debug.Print dlg.list
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Select Case Action%
  Case 2 ' Value changing or button pressed
  If DlgItem$ = "Change" Then
    Dim N As Integer
    N = UBound(lists$)+1
    ReDim Preserve lists$(N)
    lists$(N) = "List " & N
    DlgListBoxArray "list",lists$()
    DialogFunc% = True 'do not exit the dialog
  End If
  End Select
End Function
```

## DlgName Function

---

**Syntax** DlgName[\$](DlgItem)

**Group** Dialog Function

**Description** Return the *field* name of the *DlgItem* number.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
<i>DlgItem</i>	This numeric value is the dialog item number. The first item is 0, second is 1, etc.

## Example

```
Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,.Text
    OKButton 30,90,60,20
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Debug.Print "Action=";Action%
  Select Case Action%
  Case 1 ' Dialog box initialization
    Beep
    For I = 0 To DlgCount()-1
      Debug.Print I;DlgName(I)
    Next I
  End Select
End Function
```

## DlgNumber Function

---

<b>Syntax</b>	DlgNumber( <i>DlgItem\$</i> )
<b>Group</b>	Dialog Function
<b>Description</b>	Return the number of the <i>DlgItem\$</i> . The first item is 0, second is 1, etc.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

---

Parameter	Description
<i>DlgItem\$</i>	This string value is the dialog item's <i>field</i> name.

---

## Example

```
Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,.Text
    OKButton 30,90,60,20
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Debug.Print "Action=";Action%
  Select Case Action%
    Case 1 ' Dialog box initialization
      Beep
    Case 4 ' Focus changed
      Debug.Print DlgItem$;"=";DlgNumber(DlgItem$)
  End Select
End Function
```

## DlgSetPicture Instruction

---

**Syntax** `DlgSetPicture DlgItem|$, FileName, Type`

**Group** Dialog Function

**Description** Instruction: Set the file name for *DlgItem|*\$.

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
<i>DlgItem </i> \$	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>field</i> name.
<i>FileName</i>	Set the file name of <i>DlgItem </i> \$ to this string value.
<i>Type</i>	This numeric value indicates the type of bitmap used. See below.

  

Type	Effect
0	<i>FileName</i> is the name of the bitmap file. If the file does not exist then "(missing picture)" is displayed.
3	The clipboard's bitmap is displayed. Not supported.
+16	Instead of displaying "(missing picture)" a run-time error occurs.

## Example

```
Sub Main
Begin Dialog UserDialog 200,120,.DialogFunc
Picture 10,10,180,75,"",0,.Picture
OKButton 30,90,60,20
PushButton 110,90,60,20,"&View"
End Dialog
Dim dlg As UserDialog
Debug.Print Dialog(dlg)
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
Debug.Print "Action=";Action%
Select Case Action%
Case 1 ' Dialog box initialization
Beep
Case 2 ' Value changing or button pressed
Select Case DlgItem$
Case "View"
FileName = GetFilePath("Bitmap","BMP")
DlgSetPicture "Picture",FileName,0
DialogFunc% = True 'do not exit the dialog
End Select
End Select
End Function
```

## DlgText Instruction/Function

---

**Syntax** DlgText *DlgItem*[\$, *Text*  
-or-  
DlgText[\$](*DlgItem*[\$])

**Group** Dialog Function

**Description** Instruction: Set the text for *DlgItem*[\$].

Function: Return the text from *DlgItem*[\$].

This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
<i>DlgItem</i> [\$]	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>field</i> name.
<i>Text</i>	Set the text of <i>DlgItem</i> [\$] to this string value.

## Example

```
Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,.Text
    OKButton 30,90,60,20
    PushButton 110,90,60,20,"&Now"
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Debug.Print "Action=";Action%
  Select Case Action%
  Case 1 ' Dialog box initialization
    Beep
  Case 2 ' Value changing or button pressed
    Select Case DlgItem$
    Case "Now"
      DlgText "Text",CStr(Now)
      DialogFunc% = True 'do not exit the dialog
    End Select
  End Select
End Function
```

## DlgType Function

---

<b>Syntax</b>	DlgType[\$](DlgItem \$)
<b>Group</b>	Dialog Function
<b>Description</b>	Return a string value indicating the type of the <i>DlgItem</i> \$. One of: "CancelButton", "CheckBox", "ComboBox", "DropListBox", "GroupBox", "ListBox", "OKButton", "OptionButton", "OptionGroup", "PushButton", "Text", "TextBox".

This instruction/function must be called directly or indirectly from a *dialogfunc*.

---

Parameter	Description
<i>DlgItem</i> \$_	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>field</i> name.

---

## Example

```
Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,.Text
    OKButton 30,90,60,20
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Debug.Print "Action=";Action%
  Select Case Action%
  Case 1 ' Dialog box initialization
    Beep
    For I = 0 To DlgCount()-1
      Debug.Print I;DlgType(I)
    Next I
  End Select
End Function
```

## DlgValue Instruction/Function

---

**Syntax**            DlgValue *DlgItem*[\$, *Value*  
                      -or-  
                      DlgValue(*DlgItem*[\$)

**Group**             Dialog Function

**Description**        Instruction: Set the numeric value *DlgItem*[\$.

                      Function: Return the numeric value for *DlgItem*[\$.

This instruction/function must be called directly or indirectly from a *dialogfunc*. The *DlgItem*[\$ should refer to a **CheckBox**, **DropListBox**, **ListBox** or **OptionGroup**.

---

Parameter	Description
<i>DlgItem</i> [\$	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>field</i> name.
<i>Text</i>	Set the text of <i>DlgItem</i> [\$ to this string value.

---

## Example

```
Sub Main
Begin Dialog UserDialog 150,147,.DialogFunc
GroupBox 10,7,130,77,"Direction",.Field1
PushButton 100,28,30,21,"&Up"
PushButton 100,56,30,21,"&Dn"
OptionGroup .Direction
    OptionButton 20,21,80,14,"&North",.North
    OptionButton 20,35,80,14,"&South",.South
    OptionButton 20,49,80,14,"&East",.East
    OptionButton 20,63,80,14,"&West",.West
OKButton 10,91,130,21
CancelButton 10,119,130,21
End Dialog
Dim dlg As UserDialog
Dialog dlg
MsgBox "Direction=" & dlg.Direction
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
Select Case Action%
Case 1 ' Dialog box initialization
Beep
Case 2 ' Value changing or button pressed
Select Case DlgItem$
Case "Up"
    DlgValue "Direction",0
    DialogFunc% = True 'do not exit the dialog
Case "Dn"
    DlgValue "Direction",1
    DialogFunc% = True 'do not exit the dialog
End Select
End Select
End Function
```

## DlgVisible Instruction/Function

---

**Syntax**            DlgVisible *DlgItem*|\$[, *Visible*]  
                      -or-  
                      DlgVisible(*DlgItem*|\$)

**Group**             Dialog Function

**Description**        Instruction: Show or hide *DlgItem*|\$.

                      Function: Return **True** if *DlgItem*|\$ is visible.

                      This instruction/function must be called directly or indirectly from a *dialogfunc*.

Parameter	Description
<i>DlgItem</i> §	If this is a numeric value then it is the dialog item number. The first item is 0, second is 1, etc. If this is a string value then it is the dialog item's <i>field</i> name.
<i>Enable</i>	If this numeric value is <b>True</b> then show <i>DlgItem</i> §. Otherwise, hide it. If this omitted then toggle it.

## Example

```

Sub Main
  Begin Dialog UserDialog 200,120,.DialogFunc
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,40,180,15,.Text
    OKButton 30,90,60,20
    PushButton 110,90,60,20,"&Hide"
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub

Function DialogFunc%(DlgItem$, Action%, SuppValue%)
  Debug.Print "Action=";Action%
  Select Case Action%
  Case 1 ' Dialog box initialization
    Beep
  Case 2 ' Value changing or button pressed
    Select Case DlgItem$
    Case "Hide"
      DlgText DlgItem$,"&Show"
      DlgVisible "Text",False
      DialogFunc% = True 'do not exit the dialog
    Case "Show"
      DlgText DlgItem$,"&Hide"
      DlgVisible "Text",True
      DialogFunc% = True 'do not exit the dialog
    End Select
  End Select
End Function

```

## Do Statement

```

Syntax
  Do
    statements
  Loop
  -or-
  Do {Until|While} condexpr
    statements
  Loop
  -or-
  Do

```

```
statements
Loop {Until|While} condexpr
```

<b>Group</b>	Flow Control
<b>Description</b>	<p>Form 1: Do <i>statements</i> forever. The loop can be exited by using <b>Exit</b> or <b>Goto</b>.</p> <p>Form 2: Check for loop termination before executing the loop the first time.</p> <p>Form 3: Execute the loop once and then check for loop termination.</p> <p><b>Loop Termination:</b></p> <ul style="list-style-type: none"><li>• Until <i>condexpr</i>: Do <i>statements</i> until <i>condexpr</i> is <b>True</b>.</li><li>• While <i>condexpr</i>: Do <i>statements</i> while <i>condexpr</i> is <b>True</b>.</li></ul>
<b>See Also</b>	<b>For, For Each, Exit Do, While.</b>
<b>Example</b>	<pre>Sub Main   I = 2   Do     I = I*2   Loop Until I &gt; 10   Debug.Print I ' 16 End Sub</pre>

## DoEvents Instruction

---

<b>Syntax</b>	DoEvents
<b>Group</b>	Miscellaneous
<b>Description</b>	This instruction allows other applications to process events.
<b>Example</b>	<pre>Sub Main   DoEvents ' let other apps work End Sub</pre>

## Double Data Type

---

<b>Group</b>	Data Type
<b>Description</b>	A 64 bit real value.

## DropListBox Dialog Item Definition

---

**Syntax** DropListBox *X, Y, DX, DY, StrArray\$( ), .Field*

**Group** User Dialog

**Description** Define a drop-down listbox item.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>StrArray\$( )</i>	This one-dimensional array of strings establishes the list of choices. All the non-null elements of the array are used.
<i>Field</i>	The value of the drop-down list box is accessed via this field. It is the index of the <i>StrArray\$( )</i> var.

**See Also** [Begin Dialog, Dim As UserDialog.](#)

**Example**

```
Sub Main
  Dim lists$(3)
  lists$(0) = "List 0"
  lists$(1) = "List 1"
  lists$(2) = "List 2"
  lists$(3) = "List 3"
  Begin Dialog UserDialog 200,120
    Text 10,10,180,15,"Please push the OK button"
    DropListBox 10,25,180,60,lists$(),.list
    OKButton 80,90,40,20
  End Dialog
  Dim dlg As UserDialog
  dlg.list = 2
  Dialog dlg ' show dialog (wait for ok)
  Debug.Print dlg.list
End Sub
```

## Empty Keyword

---

**Group** Constant

**Description** A *variantvar* that does not have any value.

## End Instruction

---

<b>Syntax</b>	End
<b>Group</b>	Flow Control
<b>Description</b>	The end instruction causes the <i>macro</i> to terminate immediately. If the macro was run by another macro using the <b>MacroRun</b> instruction then that macro continues on the instruction following the <b>MacroRun</b> .
<b>Example</b>	<pre>Sub DoSub   L\$ = UCase\$(InputBox\$("Enter End:"))   If L\$ = "END" Then End   Debug.Print "End was not entered." End Sub  Sub Main   Debug.Print "Before DoSub"   DoSub   Debug.Print "After DoSub" End Sub</pre>

## Enum Definition

---

<b>Syntax</b>	<pre>[   Private   Public ] _ Enum name   elem [ = value ]   [...] End Enum</pre>
<b>Group</b>	Declaration
<b>Description</b>	Define a new <i>userenum</i> . Each <i>elem</i> defines an element of the enum. If <i>value</i> is given then that is the element's value. The value can be any constant integer expression. If <i>value</i> is omitted then the element's value is one more than the previous element's value. If there is no previous element then zero is used.

Enum defaults to **Public** if neither **Private** or **Public** is specified.

**Example**

```

Enum Days
    Monday
    Tuesday
    Wednesday
    Thursday
    Friday
    Saturday
    Sunday
End Enum

Sub Main
    Dim D As Days
    For D = Monday To Friday
        Debug.Print D ' 0 through 4
    Next D
End Sub

```

## Environ Instruction/Function

---

**Syntax** Environ[\$](*Index*)  
 -or-  
 Environ[\$](*Name*)

**Group** Miscellaneous

**Description** Return an environment string.

Parameter	Description
<i>Index</i>	Return this environment string's value. If there is no environment string at this index a null string is returned. Indexes start at one.
<i>Name</i>	Return this environment string's value. If the environment string can't be found a null string is returned.

**Example**

```

Sub Main
    Debug.Print Environ("Path")
End Sub

```

## EOF Function

---

**Syntax** EOF(*StreamNum*)

**Group** File

**Description** Return **True** if *StreamNum* is at the end of the file.

Parameter	Description
-----------	-------------

*StreamNum* Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.

---

**Example**

```
Sub Main
  Open "XXX" For Input As #1
  While Not EOF(1)
    Line Input #1,L$
    Debug.Print L$
  Wend
  Close #1
End Sub
```

## Erase Instruction

---

**Syntax**

```
Erase arrayvar[, ...]
-or-
Erase usertypevar.elem[, ...]
```

**Group**

Assignment

**Description**

Reset *arrayvar* or *user defined type* array element to zero. (Dynamic arrays are reset to undimensioned arrays.) String arrays values are set to a null string. *arrayvar* must be declared as an array.

- Declare with **Dim**, **Private**, **Public** or **Static**.
- Declare as a parameter of **Sub**, **Function** or **Property** definition.

**Example**

```
Sub Main
  Dim X%(2)
  X%(1) = 1
  Erase X%
  Debug.Print X%(1) ' 0
End Sub
```

## Err Object

---

**Syntax**

Err

**Group**

Error Handling

**Description**

Set Err to zero to clear the last error event. Err in an expression returns the last error code. Add vbObjectError to your error number in OLE automation objects. Use Err.Raise or **Error** to

trigger an error event.

`Err[.Number]`

This is the error code for the last error event. Set it to zero (or use `Err.Clear`) to clear the last error condition. Use **Error** or `Err.Raise` to trigger an error event. This is the default property.

`Err.Description`

This string is the description of the last error event.

`Err.Source`

This string is the error source file name of the last error event.

`Err.HelpFile`

This string is the help file name of the last error event.

`Err.HelpContext`

This number is the help context id of the last error event.

`Err.Clear`

Clear the last error event.

```
Err.Raise [Number:=]errorcode _  
          [, [Source:=]source] _  
          [, [Description:=]errordesc] _  
          [, [HelpFile:=]helpfile] _  
          [, [HelpContext:=]context]
```

Raise an error event.

`Err.LastDLLError`

For 32 bit windows this returns the error code for the last DLL call (see **Declare**). For 16 bit windows this always returns 0.

### Example

```
Sub Main  
  On Error GoTo Problem  
  Err = 1 ' set to error #1 (handler not triggered)  
  Exit Sub  
  
  Problem: ' error handler  
  Error Err ' halt macro with message  
End Sub
```

## Error Instruction/Function

---

<b>Syntax</b>	<code>Error ErrorCode</code> -or- <code>Error[\$]([ErrorCode])</code>
<b>Group</b>	Error Handling
<b>Description</b>	Instruction: Signal error <i>ErrorCode</i> . This triggers error handling just like a real error. The current <i>procedure</i> 's error handler is activated, unless it is already active or there isn't one. In that case the calling <i>procedure</i> 's error handler is tried. (Use <b>Err.Raise</b> to provide complete error information.)

Function: The `Error()` function returns the error text string.

Parameter	Description
<i>ErrorCode</i>	This is the error number.

<b>Example</b>	<pre>Sub Main   On Error GoTo Problem   Err.Raise 1 ' simulate error #1 Exit Sub  Problem: ' error handler Debug.Print "Error\$=";Error\$ Resume Next End Sub</pre>
----------------	---

## Exit Instruction

---

<b>Syntax</b>	<code>Exit {All Do For Function Property Sub While}</code>
<b>Group</b>	Flow Control
<b>Description</b>	The exit instruction causes the <i>macro</i> to continue with out doing some or all of the remaining instructions.

Exit	Description
All	Exit all <i>macros</i> .
Do	Exit the <b>Do</b> loop.
For	Exit the <b>For</b> of <b>For Each</b> loop.
Function	Exit the <b>Function</b> block. Note: This instruction clears the <b>Err</b> and sets <b>Error\$</b> to null.
Property	Exit the <b>Property</b> block. Note: This instruction clears the <b>Err</b> and sets <b>Error\$</b> to null.

Sub Exit the **Sub** block. Note: This instruction clears the **Err** and sets **Error\$** to null.

While Exit the **While** loop.

---

## Example

```
Sub Main
  L$ = InputBox$("Enter Do, For, While, Sub or
All:")
  Debug.Print "Before DoSub"
  DoSub UCase$(L$)
  Debug.Print "After DoSub"
End Sub

Sub DoSub(L$)
  Do
    If L$ = "DO" Then Exit Do
    I = I+1
    Loop While I < 10
    If I = 0 Then Debug.Print "Do was entered"

    For I = 1 To 10
      If L$ = "FOR" Then Exit For
    Next I
    If I = 1 Then Debug.Print "For was entered"

    I = 10
    While I > 0
      If L$ = "WHILE" Then Exit While
      I = I-1
    Wend
    If I = 10 Then Debug.Print "While was entered"

    If L$ = "SUB" Then Exit Sub
    Debug.Print "Sub was not entered."
    If L$ = "ALL" Then Exit All
    Debug.Print "All was not entered."
  End Sub
```

## Exp Function

---

**Syntax** `Exp(Num)`

**Group** Math

**Description** Return the exponential.

---

Parameter	Description
<i>Num</i>	Return e raised to the power of this number value. The value e is approximately 2.718282.

---

**Example**

```

Sub Main
  Debug.Print Exp(1) ' 2.718281828459
End Sub

```

## False Keyword

---

**Group** Constant

**Description** A *condexpr* is false when its value is zero. A function that returns False returns the value 0.

## FileAttr Function

---

**Syntax** FileAttr(*StreamNum*, *ReturnValue*)

**Group** File

**Description** Return *StreamNum*'s open mode or file handle.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>ReturnValue</i>	1 - return the mode used to open the file: 1=Input, 2=Output, 4=Random, 8=Append, 32=Binary 2 - return the file handle

**See Also** **Open.**

**Example**

```

Sub Main
  Open "XXX" For Output As #1
  Debug.Print FileAttr(1,1) ' 2
  Close #1
End Sub

```

## FileCopy Instruction

---

**Syntax** FileCopy *FromName\$*, *ToName\$*

**Group** File

**Description** Copy a file.

Parameter	Description
<i>FromName\$</i>	This string value is the path and name of the source file. A path relative to the current directory can be used.

*ToName\$* This string value is the path and name of the destination file. A path relative to the current directory can be used.

---

**Example**

```
Sub Main
  FileCopy "C:\AUTOEXEC.BAT", "C:\AUTOEXEC.BAK"
End Sub
```

## FileDateTime Function

---

**Syntax** FileDateTime(*Name\$*)

**Group** File

**Description** Return the date and time file *Name\$* was last changed as a **date** value. If the file does not exist then a run-time error occurs.

---

Parameter	Description
<i>Name\$</i>	This string value is the path and name of the file. A path relative to the current directory can be used.

---

**Example**

```
Sub Main
  F$ = Dir$("*. *")
  While F$ <> ""
    Debug.Print F$;" ";FileDateTime(F$)
  F$ = Dir$()
  Wend
End Sub
```

## FileLen Function

---

**Syntax** FileLen(*Name\$*)

**Group** File

**Description** Return the length of file *Name\$*. If the file does not exist then a run-time error occurs.

---

Parameter	Description
<i>Name\$</i>	This string value is the path and name of the file. A path relative to the current directory can be used.

---

**Example**

```

Sub Main
  F$ = Dir$ ("*. *")
  While F$ <> ""
    Debug.Print F$;" ";FileLen(F$)
    F$ = Dir$()
  Wend
End Sub

```

## Fix Function

---

**Syntax**`Fix(Num)`**Group**

Math

**Description**

Return the integer value.

Parameter	Description
<i>Num</i>	Return the integer portion of this number value. The number is truncated. Positive numbers return the next lower integer. Negative numbers return the next higher integer.

**Example**

```

Sub Main
  Debug.Print Fix(9.9) ' 9
  Debug.Print Fix(0) ' 0
  Debug.Print Fix(-9.9) '-9
End Sub

```

## For Statement

---

**Syntax**

```

For Num = First To Last [Step Inc]
  statements
Next [Num]

```

**Group**

Flow Control

**Description**Execute *statements* while *Num* is in the range *First* to *Last*.

Parameter	Description
<i>Num</i>	This is the iteration variable.
<i>First</i>	Set <i>Num</i> to this value initially.
<i>Last</i>	Continue looping while <i>Num</i> is in the range. See <i>Step</i> below.
<i>Step</i>	If this number value is greater than zero then the for loop continues as long as <i>Num</i> is less than or equal to <i>Last</i> . If this number value is less than zero then the for loop continues as long as <i>Num</i> is greater than or equal to <i>Last</i> . If this is omitted then one is used.

**See Also**      **Do, For Each, Exit For, While.**

**Example**

```
Sub Main
  For I = 1 To 2000 Step 100
    Debug.Print I;I+I;I*I
  Next I
End Sub
```

## For Each Statement

---

**Syntax**      **For** Each *var* In *items*  
                  *statements*  
                  **Next** [*var*]

**Group**        Flow Control

**Description**      Execute *statements* for each item in *items*.

---

Parameter	Description
<i>var</i>	This is the iteration variable.
<i>items</i>	This is the collection of items to be done.

---

**See Also**      **Do, For, Exit For, While.**

**Example**

```
Sub Main
  Dim Document As Object
  For Each Document In App.Documents
    Debug.Print Document.Title
  Next Document
End Sub
```

## Format\$ Function

---

**Syntax**        Format[\$](*expr*[, *form*\$])

**Group**        String

**Description**      Return the formatted string representation of *expr*.

---

Parameter	Description
<i>expr</i>	Return the formatted string representation of this number value.
<i>form</i>	Format <i>expr</i> using to this string value. If this is omitted then return the <i>expr</i> as a string.

---

**See Also**      **Predefined Date Format, Predefined Number Format, User defined Date Format, User defined Number Format, User defined Text Format.**

## Format Predefined Date

---

**Description**      The following predefined date formats may be used with the **Format** function. Predefined formats may not be combined with user defined formats or other predefined formats.

Form	Description
General Date	Same as <b>user defined date format "c"</b>
Long Date	Same as <b>user defined date format "dddddd"</b>
Medium Date	<b>Not supported at this time.</b>
Short Date	Same as <b>user defined date format "dddd"</b>
Long Time	Same as <b>user defined date format "tttt"</b>
Medium Time	Same as <b>user defined date format "hh:mm AMPM"</b>
Short Time	Same as <b>user defined date format "hh:mm"</b>

## Format Predefined Number

---

**Description**      The following predefined number formats may be used with the **Format** function. Predefined formats may not be combined with user defined formats or other predefined formats.

Form	Description
General Number	Return number as is.
Currency	Same as <b>user defined number format "\$#,##0.00;(\$#,##0.00)"</b> <b>Not locale dependent at this time.</b>
Fixed	Same as <b>user defined number format "0.00"</b> .
Standard	Same as <b>user defined number format "#,##0.00"</b> .
Percent	Same as <b>user defined number format "0.00%"</b> .
Scientific	Same as <b>user defined number format "0.00E+00"</b> .
Yes/No	Return "No" if zero, else return "Yes".
True/False	Return "True" if zero, else return "False".
On/Off	Return "On" if zero, else return "Off".

**Example**

```
Sub Main
    Debug.Print Format$(2.145, "Standard") ' 2.15
End Sub
```

## Format User Defined Date

---

**Description** The following date formats may be used with the **Format** function. Date formats may be combined to create the user defined date format. User defined date formats may not be combined with other user defined formats or predefined formats.

Parameter	Description
:	insert localized time separator
/	insert localized date separator
c	insert dddd tttt, insert date only if t=0, insert time only if d=0
d	insert day number without leading zero
dd	insert day number with leading zero
ddd	insert abbreviated day name
dddd	insert full day name
dddd	insert date according to Short Date format
dddddd	insert date according to Long Date format
w	insert day of week number
ww	insert week of year number
m	insert month number without leading zero insert minute number without leading zero (if follows h or hh)
mm	insert month number with leading zero insert minute number with leading zero (if follows h or hh)
mmm	insert abbreviated month name
mmmm	insert full month name
q	insert quarter number
y	insert day of year number
yy	insert year number (two digits)
yyyy	insert year number (four digits, no leading zeros)
h	insert hour number without leading zero
hh	insert hour number with leading zero
n	insert minute number without leading zero
nn	insert minute number with leading zero
s	insert second number without leading zero
ss	insert second number with leading zero
tttt	insert time according to time format
AM/PM	use 12 hour clock and insert AM (hours 0 to 11) and PM (12 to 23)
am/pm	use 12 hour clock and insert am (hours 0 to 11) and pm (12 to 23)
A/P	use 12 hour clock and insert A (hours 0 to 11) and P (12 to 23)
a/p	use 12 hour clock and insert a (hours 0 to 11) and p (12 to 23)
AMPM	use 12 hour clock and insert localized AM/PM strings
\c	insert character c
"text"	insert literal text

## Example

# Format User Defined Number

---

**Description** The following number formats may be used with the **Format** function. Number formats may be combined to create the user defined number format. User defined number formats may not be combined with other user defined formats or predefined formats.

User defined number formats can contain up to four sections separated by ';':

- form - format for non-negative expr, '-'format for negative expr, empty and null expr return ""
- form;negform - negform: format for negative expr
- form;negform;zeroform - zeroform: format for zero expr
- form;negform;zeroform>nullform - nullform: format for empty or null expr

Parameter	Description
#	digit, don't include leading/trailing zero digits (all the digits left of decimal point are returned) eg. Format(19,"###") returns "19" eg. Format(19,"#") returns "19"
0	digit, include leading/trailing zero digits eg. Format(19,"000") returns "019" eg. Format(19,"0") returns "19"
.	decimal, insert localized decimal point eg. Format(19.9,"###.00") returns "19.90" eg. Format(19.9,"###.##") returns "19.9"
,	thousands, insert localized thousand separator every 3 digits "xxx," or "xxx," mean divide expr by 1000 prior to formatting two adjacent commas ",," means divide expr by 1000 again eg. Format(1900000,"0,,") returns "2" eg. Format(1900000,"0,,0") returns "1.9"
%	percent, insert %, multiply expr by 100 prior to formatting
:	insert localized time separator
/	insert localized date separator
E+ e+ E- e-	use exponential notation, insert E (or e) and the signed exponent eg. Format(1000,"0.00E+00") returns "1.00E+03" eg. Format(.001,"0.00E+00") returns "1.00E-03"
- + \$ ( ) space	insert literal char eg. Format(10,"\$#") returns "\$10"
\c	insert character c eg. Format(19,"\####\#") returns "#19#"

"text"                    insert literal text  
                          eg. Format(19,"###"###"###") returns "##19##"

---

**Example**

```
Sub Main
  Debug.Print Format$(2.145,"#.00") ' 2.15
End Sub
```

---

## Format User Defined Text

---

**Description**

The following text formats may be used with the **Format** function. Text formats may be combined to create the user defined text format. User defined text formats may not be combined with other user defined formats or predefined formats.

User defined text formats can contain one or two sections separated by ';':

- form - format for all strings
- form;nullform - nullform: format for null strings

---

Parameter	Description
@	char placeholder, insert char or space
&	char placeholder, insert char or nothing
<	all chars lowercase
>	all chars uppercase
!	fill placeholder from left-to-right (default is right-to-left)
\c	insert character c
"text"	insert literal text

---

**Example**

```
Sub Main
  Debug.Print Format("123","ab@c") ' 12ab3c
  Debug.Print Format("123","!ab@c") ' ab1c23
End Sub
```

---

## FreeFile Function

---

**Syntax**

FreeFile[( )]

**Group**

File

**Description**

Return the next unused shared stream number (greater than or equal to 256). Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.

**Example**

```

Sub Main
  Debug.Print FreeFile ' 256
  FN = FreeFile
  Open "XXX" For Output As #FN
  Debug.Print FreeFile ' 257
  Close #FN
  Debug.Print FreeFile ' 256
End Sub

```

## Friend Keyword

---

**Group** Declaration

**Description** Friend **Functions**, **Property**s and **Sub**s in a *module* are available in all other *macros*/modules that access it. Friends are not accessible via **Object** variables.

## Function Definition

---

**Syntax** [ | **Private** | **Public** | **Friend** ] \_  
Function *name*[*type*][([*param*[, ...]])] [*As type*]  
*statements*  
**End** Function

**Group** Declaration

**Description** User defined function. The function defines a set of *statements* to be executed when it is called. The values of the calling *arglist* are assigned to the *params*. Assigning to *name*[*type*] sets the value of the function result.

Function defaults to **Public** if **Private**, **Public** or **Friend** are not is specified.

**See Also** **Declare**, **Property**, **Sub**.

## Example

```
Function Power(X,Y)
    P = 1
    For I = 1 To Y
        P = P*X
    Next I
    Power = P
End Function

Sub Main
    Debug.Print Power(2,8) ' 256
End Sub
```

## Get Instruction

---

**Syntax**            `Get StreamNum, [RecordNum], var`

**Group**             `File`

**Description**      `Get a variable's value from StreamNum.`

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>RecordNum</i>	For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1. If this is omitted then the current position (or record number) is used.
<i>var</i>	This variable value is read from the file. For a fixed length variable (like <b>Long</b> ) the number of bytes required to restore the variable are read. For a <b>Variant</b> variable two bytes are read which describe its type and then the variable value is read accordingly. For a <i>usertype</i> variable each field is read in sequence. For an array variable each element is read in sequence. For a dynamic array variable the number of dimensions and range of each dimension is read prior to reading the array values. All binary data values are read from the file in <i>little-endian</i> format.  Note: When reading a string (or a dynamic array) from a Binary mode file the length (or array dimension) information is not read. The current string length determines how much string data is read. The current array dimension determines how many array elements are read.

**See Also**            `Open, Put.`

**Example**

```

Sub Main
  Dim V As Variant
  Open "SAVE_V.DAT" For Binary Access Read As #1
  Get #1, , V
  Close #1
End Sub

```

## GetAllSettings Function

---

**Syntax** GetAllSettings(*AppName\$, Section\$, Key\$*)

**Group** Settings

**Description** Get all of *Section's* settings in project *AppName*. Settings are returned in a **Variant**. **Empty** is returned if there are no keys in the section. Otherwise, the Variant contains a two dimension array: (I,0) is the key and (I,1) is the setting. Win16 and Win32s store settings in a .ini file named *AppName*. Win32 stores settings in the registration database.

Parameter	Description
<i>AppName\$</i>	This string value is the name of the project which has this <i>Section</i> and <i>Key</i> .
<i>Section\$</i>	This string value is the name of the section of the project settings.

**Example**

```

Sub Main
  SaveSetting "MyApp", "Font", "Size", 10
  SaveSetting "MyApp", "Font", "Name", "Courier"
  Settings = GetAllSettings("MyApp", "Font")
  For I = LBound(Settings) To UBound(Settings)
    Debug.Print Settings(I,0); "="; Settings(I,1)
  Next I
  DeleteSetting "MyApp", "Font"
End Sub

```

## GetAttr Function

---

**Syntax** GetAttr(*Name\$*)

**Group** File

**Description** Return the *attributes* for file *Name\$*. If the file does not exist then a run-time error occurs.

Parameter	Description
-----------	-------------

*Name\$* This string value is the path and name of the file. A path relative to the current directory can be used.

---

### Example

```
Sub Main
  F$ = Dir$ ("*. *")
  While F$ <> ""
    Debug.Print F$; " "; GetAttr(F$)
    F$ = Dir$()
  Wend
End Sub
```

## GetFilePath\$ Function

---

**Syntax** GetFilePath[\$]([DefName\$], [DefExt\$], [DefDir\$], \_  
[Title\$], [Option])

**Group** User Input

**Description** Put up a dialog box and get a file path from the user. The returned string is a complete path and file name. If the cancel button is pressed then a null string is returned.

---

Parameter	Description
<i>DefName\$</i>	Set the initial File Name in the to this string value. If this is omitted then *. <i>DefExt\$</i> is used.
<i>DefExt\$</i>	Initially show files whose extension matches this string value. (Multiple extensions can be specified by using ";" as the separator.) If this is omitted then * is used.
<i>DefDir\$</i>	This string value is the initial directory. If this is omitted then the current directory is used.
<i>Title\$</i>	This string value is the title of the dialog. If this is omitted then "Open" is used.
<i>Option</i>	This numeric value determines the file selection options. If this is omitted then zero is used. See table below.

---

Option	Effect
0	Only allow the user to select a file that exists.
1	Confirm creation when the user selects a file that does not exist.
2	Allow the user to select any file whether it exists or not.
3	Confirm overwrite when the user selects a file that exists.
+4	Selecting a different directory changes the application's current directory.

---

### Example

```
Sub Main
  Debug.Print GetFilePath$()
End Sub
```

## GetObject Function

---

**Syntax**            `GetObject([File$][, Class$])`

**Group**             Object

**Description**      Get an existing object of type *Class\$* from *File\$*. Use **Set** to assign the returned object to an object variable.

Parameter	Description
<i>File\$</i>	This is the file where the object resides. If this is omitted then the currently active object for <i>Class\$</i> is returned.
<i>Class\$</i>	This string value is the application's registered class name. If this application is not currently active it will be started. If this is omitted then the application associated with the file's extension will be started.

**Example**

```
Sub Main
  Dim App As Object
  Set App = GetObject("WinWrap.CppDemoApplication")
  App.Move 20,30 ' move icon to 20,30
  Set App = Nothing
  App.Quit      ' run-time error (no object)
End Sub
```

## GetSetting Function

---

**Syntax**            `GetSetting[$](AppName$, Section$, Key$[, Default$])`

**Group**             Settings

**Description**      Get the setting for *Key* in *Section* in project *AppName*. Win16 and Win32s store settings in a .ini file named *AppName*. Win32 stores settings in the registration database.

Parameter	Description
<i>AppName\$</i>	This string value is the name of the project which has this <i>Section</i> and <i>Key</i> .
<i>Section\$</i>	This string value is the name of the section of the project settings.
<i>Key\$</i>	This string value is the name of the key in the section of the project settings.
<i>Default\$</i>	Return this string value if no setting has been saved. If this is omitted then a null string is used.

**Example**

```

Sub Main
  SaveSetting "MyApp", "Font", "Size", 10
  Debug.Print GetSetting("MyApp", "Font", "Size") ' 10
End Sub

```

## Goto Instruction

---

**Syntax**            GoTo *label*

**Group**             Flow Control

**Description**      Go to the *label* and continue execution from there. Only *labels* in the current user defined *procedure* are accessible.

**Example**

```

Sub Main
  X = 2
Loop:
  X = X*X
  If X < 100 Then GoTo Loop
  Debug.Print X ' 256
End Sub

```

## GroupBox Dialog Item Definition

---

**Syntax**            GroupBox *X, Y, DX, DY, Title\$[, .Field]*

**Group**             User Dialog

**Description**      Define a groupbox item.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Title\$</i>	This string value is the title of the group box.
<i>Field</i>	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this identifier is omitted then the first two words of the title are used.

**See Also**            **Begin Dialog, Dim As UserDialog.**

**Example**

```

Sub Main
  Begin Dialog UserDialog 200,120
    Text 10,10,180,15,"Please push the OK button"
    GroupBox 10,25,180,60,"Group box"
    OKButton 80,90,40,20
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg ' show dialog (wait for ok)
End Sub

```

## Hex\$ Function

---

**Syntax** Hex[\$](Num)**Group** String**Description** Return a hex string.

Parameter	Description
Num	Return a hex encoded string for this number value.

**See Also** Oct\$( ), Str\$( ), Val( ).**Example**

```

Sub Main
  Debug.Print Hex$(15) 'F
End Sub

```

## Hour Function

---

**Syntax** Hour(dateexpr)**Group** Time/Date**Description** Return the hour of the day (0 to 23).

Parameter	Description
dateexpr	Return the hour of the day for this date value. If this value is <b>Null</b> then <b>Null</b> is returned.

**See Also** Minute( ), Second( ), Time( ).**Example**

```

Sub Main
  Debug.Print Hour(#12:00:01 AM#) ' 0
End Sub

```

## If Statement

---

<b>Syntax</b>	<pre>If <i>condexpr</i> Then [<i>instruction</i>] [Else <i>instruction</i>] -or- If <i>condexpr</i> Then     <i>statements</i> [ElseIf <i>condexpr</i> Then     <i>statements</i>]... [Else     <i>statements</i>] <b>End</b> If -or- If TypeOf <i>objexpr</i> <b>Is</b> <i>objtype</i> Then ...</pre>
<b>Group</b>	Flow Control
<b>Description</b>	<p>Form 1: Single line if statement. Execute the <i>instruction</i> following the Then if <i>condexpr</i> is <b>True</b>. Otherwise, execute the <i>instruction</i> following the Else. The Else portion is optional.</p> <p>Form 2: The multiple line if is useful for complex ifs. Each if <i>condexpr</i> is checked in turn. The first <b>True</b> one causes the following <i>statements</i> to be executed. If all are <b>False</b> then the Else's <i>statements</i> are executed. The ElseIf and Else portions are optional.</p> <p>Form 3: If <i>objexpr</i>'s type is the same type or a type descended from <i>objtype</i> the Then portion is executed.</p>
<b>See Also</b>	<b>Select Case, Choose(), IIf().</b>
<b>Example</b>	<pre>Sub Main     S = InputBox("Enter hello, goodbye, dinner or sleep:")     S = UCase(S)     If S = "HELLO" Then Debug.Print "come in"     If S = "GOODBYE" Then Debug.Print "see you later"     If S = "DINNER" Then         Debug.Print "Please come in."         Debug.Print "Dinner will be ready soon."     ElseIf S = "SLEEP" Then         Debug.Print "Sorry."         Debug.Print "We are full for the night"     End If End Sub</pre>

## IIf Function

---

**Syntax** `IIf(condexpr, TruePart, FalsePart)`

**Group** Miscellaneous

**Description** Return the value of the parameter indicated by *condexpr*. Both *TruePart* and *FalsePart* are evaluated.

Parameter	Description
<i>condexpr</i>	If this value is <b>True</b> then return <i>TruePart</i> . Otherwise, return <i>FalsePart</i> .
<i>TruePart</i>	Return this value if <i>condexpr</i> is <b>True</b> .
<i>FalsePart</i>	Return this value if <i>condexpr</i> is <b>False</b> .

**See Also** **If, Select Case, Choose()**.

**Example**

```
Sub Main
    Debug.Print IIf(1 > 0, "True", "False") ' "True"
End Sub
```

## Input Instruction

---

**Syntax** `Input [#]StreamNum, var[, ...]`

**Group** File

**Description** Get input from *StreamNum* and assign it to *vars*. Input values are comma delimited. Leading and trailing spaces are ignored. If the first char (following the leading spaces) is a quote (") then the string is terminated by an ending quote. Special values #NULL#, #FALSE#, #TRUE#, #date# and #ERROR number# are converted to their appropriate value and data type.

**See Also** **Line Input, Print, Write.**

**Example**

```
Sub Main
    Open "XXX" For Input As #1
    Input #1, A, B, C$
    Debug.Print A; B; C$
    Close #1
End Sub
```

## Input\$ Function

---

**Syntax** `Input[$](N, StreamNum)`

**Group** File

**Description** Return *N* chars from *StreamNum*.

Parameter	Description
<i>N</i>	Read this many chars. If fewer than that many chars are left before the end of file then a run-time error occurs.
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.

**Example**

```
Sub Main
  Open "XXX" For Input As #1
  L = LOF(1)
  T$ = Input$(L,1)
  Close #1
  Debug.Print T$;
End Sub
```

## InputBox\$ Function

---

**Syntax** InputBox[\$](*Prompt\$*[, *Title\$*][, *Default\$*][, *XPos*, *YPos*])

**Group** User Input

**Description** Display an input box where the user can enter a line of text. Pressing the OK button returns the string entered. Pressing the Cancel button returns a null string.

Parameter	Description
<i>Prompt\$</i>	Use this string value as the prompt in the input box.
<i>Title\$</i>	Use this string value as the title of the input box. If this is omitted then the input box does not have a title.
<i>Default\$</i>	Use this string value as the initial value in the input box. If this is omitted then the initial value is blank.
<i>XPos</i>	When the dialog is put up the left edge will be at this screen position. If this is omitted then the dialog will be centered.
<i>YPos</i>	When the dialog is put up the top edge will be at this screen position. If this is omitted then the dialog will be centered.

**Example**

```
Sub Main
  L$ = InputBox$("Enter some text:", _
    "Input Box Example", "asdf")
  Debug.Print L$
End Sub
```

## InStr Function

---

**Syntax** `InStr([Index, ]S1$, S2$)`

**Group** String

**Description** Return the index where S2\$ first matches S1\$. If no match is found return 0.

Note: A similar function, InStrB, returns the byte index instead.

Parameter	Description
<i>Index</i>	Start searching for S2\$ at this index in S1\$. If this is omitted then start searching from the beginning of S1\$.
S1\$	Search for S2\$ in this string value. If this value is <b>Null</b> then <b>Null</b> is returned.
S2\$	Search S1\$ for this string value. If this value is <b>Null</b> then <b>Null</b> is returned.

**See Also** **InStrRev()**, **Left\$()**, **Len()**, **Mid\$()**, **Replace\$()**, **Right\$()**.

**Example**

```
Sub Main
    Debug.Print InStr("Hello","l") ' 3
End Sub
```

## InStrRev Function

**Syntax** `InStrRev(S1$, S2$[, Index])`

**Group** String

**Description** Return the index where S2\$ last matches S1\$. If no match is found return 0.

Parameter	Description
S1\$	Search for S2\$ in this string value. If this value is <b>Null</b> then <b>Null</b> is returned.
S2\$	Search S1\$ for this string value. If this value is <b>Null</b> then <b>Null</b> is returned.
<i>Index</i>	Start searching for S2\$ ending at this index in S1\$. If this is omitted then start searching from the end of S1\$.

**See Also** **Left\$()**, **Len()**, **Mid\$()**, **Replace\$()**, **Right\$()**.

**Example**

```
Sub Main
    Debug.Print InStrRev("Hello","l") ' 4
End Sub
```

## Int Function

---

**Syntax**            `Int (Num)`

**Group**             `Math`

**Description**      `Return the integer value.`

Parameter	Description
<code>Num</code>	Return the largest integer which is less than or equal to this number value.

**Example**

```
Sub Main
  Debug.Print Int(9.9) ' 9
  Debug.Print Int(0)   ' 0
  Debug.Print Int(-9.9) '-10
End Sub
```

## Integer Data Type

---

**Group**             `Data Type`

**Description**      `A 16 bit integer value.`

## Is Operator

---

**Syntax**            `expr Is expr`

**Group**             `Operator`

**Description**      `Return the True if both exprs refer to the same object.`

**See Also**          `Objects.`

**Example**

```
Sub Main
  Dim X As Object
  Dim Y As Object
  Debug.Print X Is Y ' True
End Sub
```

## IsArray Function

---

**Syntax**            `IsArray(var)`

**Group** Variable Info  
**Description** Return the **True** if *var* is an array of values.

Parameter	Description
<i>var</i>	A array variable or a variant var can contain multiple of values.

**See Also** **TypeName, VarType.**

**Example**

```
Sub Main
    Dim X As Variant, Y(2) As Integer
    Debug.Print IsArray(X) 'False
    X = Array(1,4,9)
    Debug.Print IsArray(X) 'True
    X = Y
    Debug.Print IsArray(X) 'True
End Sub
```

## IsDate Function

---

**Syntax** IsDate(*expr*)  
**Group** Variable Info  
**Description** Return the **True** if *expr* is a valid date.

Parameter	Description
<i>expr</i>	A variant expression to test for a valid date.

**See Also** **TypeName, VarType.**

**Example**

```
Sub Main
    Dim X As Variant
    X = 1
    Debug.Print IsDate(X) 'False
    X = Now
    Debug.Print IsDate(X) 'True
End Sub
```

## IsEmpty Function

---

**Syntax** IsEmpty(*variantvar*)  
**Group** Variable Info  
**Description** Return the **True** if *variantvar* is **Empty**.

Parameter	Description
-----------	-------------

*variantvar* A variant var is **Empty** if it has never been assign a value.

---

**See Also**

**TypeName, VarType.**

**Example**

```
Sub Main
  Dim X As Variant
  Debug.Print IsEmpty(X) 'True
  X = 0
  Debug.Print IsEmpty(X) 'False
  X = Empty
  Debug.Print IsEmpty(X) 'True
End Sub
```

## IsError Function

---

**Syntax**

IsError(*expr*)

**Group**

Variable Info

**Description**

Return the **True** if *expr* is an error code.

---

Parameter	Description
<i>expr</i>	A variant expression to test for an error code value.

---

**See Also**

**TypeName, VarType.**

**Example**

```
Sub Main
  Dim X As Variant
  Debug.Print IsError(X) 'False
  X = CVErr(1)
  Debug.Print IsError(X) 'True
End Sub
```

## IsMissing Function

---

**Syntax**

IsMissing(*variantvar*)

**Group**

Variable Info

**Description**

Return the **True** if Optional parameter *variantvar* does not have a defaultvalue and it did not get a value. An Optional parameter may be omitted in the **Sub, Function** or **Property** call.

---

Parameter	Description
<i>variantvar</i>	Return <b>True</b> if this variant parameter's argument expression was not specified in the <b>Sub, Function</b> or <b>Property</b> call.

---

## Example

```
Sub Main
  Opt                               ' IsMissing(A)=True
  Opt "Hi"                          ' IsMissing(A)=False
  Many                             ' No args
  Many 1,"Hello"                   ' A(0)=1 A(1)=Hello
  OptBye                             ' "Bye"
  OptBye "No"                       ' "No"
End Sub

Sub Opt(Optional A)
  Debug.Print "IsMissing(A)=";IsMissing(A)
End Sub

Sub Many(ParamArray A())
  If LBound(A) > UBound(A) Then
    Debug.Print "No args"
  Else
    For I = LBound(A) To UBound(A)
      Debug.Print "A(" & I & ")=" & A(I) & " ";
    Next I
    Debug.Print
  End If
End Sub

Sub OptBye(Optional A As String = "Bye")
  Debug.Print A
End Sub
```

## IsNull Function

---

**Syntax**            `IsNull(expr)`

**Group**             Variable Info

**Description**      Return the **True** if *expr* is **Null**.

---

Parameter	Description
<i>expr</i>	A variant expression to test for <b>Null</b> .

---

**See Also**         **TypeName, VarType.**

**Example**

```

Sub Main
  Dim X As Variant
  Debug.Print IsEmpty(X) 'True
  Debug.Print IsNull(X) 'False
  X = 1
  Debug.Print IsNull(X) 'False
  X = "1"
  Debug.Print IsNull(X) 'False
  X = Null
  Debug.Print IsNull(X) 'True
  X = X*2
  Debug.Print IsNull(X) 'True
End Sub

```

## IsNumeric Function

---

**Syntax** `IsNumeric(expr)`**Group** Variable Info**Description** Return the **True** if *expr* is a numeric value.

Parameter	Description
<i>expr</i>	A variant expression is a numeric value if it is <i>numeric</i> or string value that represents a number.

**See Also** [TypeName](#), [VarType](#).**Example**

```

Sub Main
  Dim X As Variant
  X = 1
  Debug.Print IsNumeric(X) 'True
  X = "1"
  Debug.Print IsNumeric(X) 'True
  X = "A"
  Debug.Print IsNumeric(X) 'False
End Sub

```

## IsObject Function

---

**Syntax** `IsObject(var)`**Group** Variable Info**Description** Return the **True** if *var* contains an object reference.

Parameter	Description
-----------	-------------

*var* A var contains an object reference if it is *objexpr* reference.

---

**See Also**

**TypeName, VarType.**

**Example**

```
Sub Main
  Dim X As Variant
  X = 1
  Debug.Print IsObject(X) 'False
  X = "1"
  Debug.Print IsObject(X) 'False
  Set X = Nothing
  Debug.Print IsObject(X) 'True
End Sub
```

## Kill Instruction

---

**Syntax**

Kill *Name\$*

**Group**

File

**Description**

Delete the file named by *Name\$*.

---

Parameter	Description
<i>Name\$</i>	This string value is the path and name of the file. A path relative to the current directory can be used.

---

**Example**

```
Sub Main
  Kill "XXX"
End Sub
```

## LBound Function

---

**Syntax**

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

**Group**

Variable Info

**Description**

Return the lowest index.

---

Parameter	Description
<i>arrayvar</i>	Return the lowest index for this array variable.
<i>dimension</i>	Return the lowest index for this dimension of <i>arrayvar</i> . If this is omitted then return the lowest index for the first dimension.

---

**See Also**

**UBound( ).**

**Example**

```

Sub Main
  Dim A(-1 To 3, 2 To 6)
  Debug.Print LBound(A)    '-1
  Debug.Print LBound(A, 1) '-1
  Debug.Print LBound(A, 2) ' 2
End Sub

```

## LCase\$ Function

---

**Syntax** LCase[\$](S\$)**Group** String**Description** Return a string from S\$ where all the uppercase letters have been lowercased.

Parameter	Description
S\$	Return the string value of this after all chars have been converted to lowercase. If this value is <b>Null</b> then <b>Null</b> is returned.

**See Also** StrComp(), StrConv\$(), UCase\$().**Example**

```

Sub Main
  Debug.Print LCase$("Hello") ' "hello"
End Sub

```

## Left\$ Function

---

**Syntax** Left[\$](S\$, Len)**Group** String**Description** Return a string from S\$ with only the Len chars.

Note: A similar function, LeftB, returns the first Len bytes.

Parameter	Description
S\$	Return the left portion of this string value. If this value is <b>Null</b> then <b>Null</b> is returned.
Len	Return this many chars. If S\$ is shorter than that then just return S\$.

**See Also** InStr(), InStrRev(), Len(), Mid\$(), Replace\$(), Right\$().

**Example**

```
Sub Main
    Debug.Print Left$("Hello",2) ' "He"
End Sub
```

## Len Function

---

**Syntax**      Len(*S\$*)  
                  -or-  
                  Len(*usertypevar*)

**Group**        String

**Description**    Return the number of characters in *S\$*.

Note: A similar function, LenB, returns the number of bytes in the string. For a *usertypevar*, LenB returns the number of bytes of memory occupied by the variable's data.

Parameter	Description
<i>S\$</i>	Return the number of chars in this string value. If this value is <b>Null</b> then <b>Null</b> is returned.
<i>usertypevar</i>	Return the number of bytes required to store this user type variable. If the user type has any dynamic <b>String</b> and <b>Variant</b> elements the length returned may not be as big as the actual number of bytes required.

**See Also**      **InStr()**, **InStrRev()**, **Left\$()**, **Mid\$()**, **Replace\$()**, **Right\$()**.

**Example**

```
Sub Main
    Debug.Print Len("Hello") ' 5
End Sub
```

## Let Instruction

---

**Syntax**        [Let] *var* = *expr*

**Group**        Assignment

**Description**    Assign the value of *expr* to *var*. The keyword Let is optional.

**Example**

```
Sub Main
    Let X = 1
    X = X*2
    Debug.Print X ' 2
End Sub
```

## Like Operator

---

**Syntax** `str1 Like str2`

**Group** Operator

**Description** Return the **True** if *str1* matches pattern *str2*. The pattern in *str2* is one or more of the special character sequences shown in the following table.

Char(s)	Description
?	Match any single character.
*	Match zero or more characters.
#	Match a single digit (0-9).
[ <i>charlist</i> ]	Match any char in the list.
[! <i>charlist</i> ]	Match any char not in the list.

**Example**

```
Sub Main
    Dim X As Object
    Dim Y As Object
    Debug.Print X Is Y ' True
End Sub
```

## Line Input Instruction

---

**Syntax** `Line Input [#]StreamNum, S$`

**Group** File

**Description** Get a line of input from *StreamNum* and assign it to *S\$*.

**See Also** [Input](#), [Print](#), [Write](#).

**Example**

```
Sub Main
    Open "XXX" For Input As #1
    Line Input #1, S$
    Debug.Print S$
    Close #1
End Sub
```

## ListBox Dialog Item Definition

---

**Syntax** `Listbox X, Y, DX, DY, StrArray$( ), .Field`

**Group** User Dialog

**Description**

Define a listbox item.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>StrArray\$()</i>	This one-dimensional array of strings establishes the list of choices. All the non-null elements of the array are used.
<i>Field</i>	The value of the list box is accessed via this field. It is the index of the <i>StrArray\$()</i> var.

**See Also****Begin Dialog, Dim As UserDialog.****Example**

```

Sub Main
  Dim lists$(3)
  lists$(0) = "List 0"
  lists$(1) = "List 1"
  lists$(2) = "List 2"
  lists$(3) = "List 3"
  Begin Dialog UserDialog 200,120
    Text 10,10,180,15,"Please push the OK button"
    ListBox 10,25,180,60,lists$(),.list
    OKButton 80,90,40,20
  End Dialog
  Dim dlg As UserDialog
  dlg.list = 2
  Dialog dlg ' show dialog (wait for ok)
  Debug.Print dlg.list
End Sub

```

## Loc Function

**Syntax**`Loc(StreamNum)`**Group**

File

**Description**

Return *StreamNum* file position. For Random mode files this is the current record number minus one. For Binary mode files it is the current byte position minus one. Otherwise, it is the current byte position minus one divided by 128. The first position in the file is 0.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.

### Example

```
Sub Main
  Open "XXX" For Input As #1
  L = Loc(1)
  Close #1
  Debug.Print L ' 0
End Sub
```

## Lock Instruction

**Syntax**

```
Lock StreamNum
-or-
Lock StreamNum, RecordNum
-or-
Lock StreamNum, [start] To end
```

**Group** File

**Description** Form 1: Lock all of *StreamNum*.

Form 2: Lock a record (or byte) of *StreamNum*.

Form 3: Lock a range of records (or bytes) of *StreamNum*. If *start* is omitted then lock starting at the first record (or byte).

Note: Be sure to **Unlock** for each Lock instruction.

Note: For sequential files (Input, Output and Append) lock always affects the entire file.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>RecordNum</i>	For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1.
<i>start</i>	First record (or byte) in the range.
<i>end</i>	Last record (or byte) in the range.

**See Also** **Open, Unlock.**

**Example**

```

Sub Main
  Dim V As Variant
  Open "SAVE_V.DAT" For Binary As #1
  Lock #1
  Get #1, 1, V
  V = "Hello"
  Put #1, 1, V
  Unlock #1
  Close #1
End Sub

```

## LOF Function

---

**Syntax**      `LOF(StreamNum)`**Group**        File**Description**    Return *StreamNum* file length (in bytes).

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.

**Example**

```

Sub Main
  Open "XXX" For Input As #1
  L = LOF(1)
  Close #1
  Debug.Print L
End Sub

```

## Log Function

---

**Syntax**        `Log(Num)`**Group**        Math**Description**    Return the natural logarithm.

Parameter	Description
<i>Num</i>	Return the natural logarithm of this number value. The value e is approximately 2.718282.

**Example**

```

Sub Main
  Debug.Print Log(1) ' 0
End Sub

```

## Long Data Type

---

<b>Group</b>	Data Type
<b>Description</b>	A 32 bit integer value.

## LSet Instruction

---

**Syntax**            `LSet strvar = str`  
                      -or-  
                      `LSet usertypevar1 = usertypevar2`

**Group**             Assignment

**Description**       Form 1: Assign the value of *str* to *strvar*. Shorten *str* by removing trailing chars (or extend with blanks). The previous length *strvar* is maintained.

Form 2: Assign the value of *usertypevar2* to *usertypevar1*. If *usertypevar2* is longer than *usertypevar1* then only copy as much as *usertypevar1* can handle.

**See Also**          **RSet.**

**Example**           `Sub Main`  
                      `S$ = "123"`  
                      `LSet S$ = "A"`  
                      `Debug.Print ". ";S$;"." ' ".A ."`  
                      `End Sub`

## LTrim\$ Function

---

**Syntax**            `LTrim[$](S$)`

**Group**             String

**Description**       Return the string with *S\$*'s leading spaces removed.

Parameter	Description
<i>S\$</i>	Copy this string without the leading spaces. If this value is <b>Null</b> then <b>Null</b> is returned.

**See Also**          **RTrim\$(), Trim\$().**

**Example**

```

Sub Main
  Debug.Print ".";LTrim$(" x ");"."'".x ."
End Sub

```

## MacroDir\$ Function

---

**Syntax** MacroDir[\$]

**Group** Flow Control

**Description** Return the directory of the current macro. A run-time error occurs if the current macro has never been saved.

**See Also** **MacroRun.**

**Example**

```

Sub Main
  ' open the file called Data that is in the
  ' same directory as the macro
  Open MacroDir & "\Data" For Input As #1
  Line Input #1, S$
  Debug.Print S$
  Close #1
End Sub

```

## MacroRun Instruction

---

**Syntax** MacroRun MacroName\$[ , Command\$]

**Group** Flow Control

**Description** Play a *macro*. Execution will continue at the following statement after the macro has completed.

Parameter	Description
MacroName\$	Run the macro named by this string value.
Command\$	Pass this string value as the macro's <b>Command\$</b> value.

**See Also** **Command\$, MacroDir\$, MacroRunThis.**

**Example**

```

Sub Main
  Debug.Print "Before Demo"
  MacroRun "Demo"
  Debug.Print "After Demo"
End Sub

```

## MacroRunThis Instruction

---

**Syntax** MacroRunThis *MacroCode*\$

**Group** Flow Control

**Description** Play the *macro* code. Execution will continue at the following statement after the macro code has completed. The macro code can be either a single line or a complete macro.

Parameter	Description
<i>MacroName</i> \$	Run the macro named by this string value.

**See Also** **Command**\$, **MacroDir**\$, **MacroRun**.

**Example**

```
Sub Main
    Debug.Print "Before Demo"
    MacroRunThis "MsgBox "Hello""
    Debug.Print "After Demo"
End Sub
```

## Main Sub

---

**Syntax**

```
Sub Main()
    ...
End Sub
-or-
Private Sub Main()
    ...
End Sub
```

**Group** Declaration

**Description** Form 1: Each *macro* must define Sub Main. A macro is a "program". Running a macro starts the Sub Main and continues to execute until the subroutine finishes.

Form 2: A code *module* may define a Private Sub Main. This Sub Main is the **code module** initialization subroutine. If Main is not defined then no special initialization occurs.

**See Also** **Code Module**.

## Mid\$ Function/Assignment

---

**Syntax** Mid[\$](S\$, Index[, Len])  
 -or-  
 Mid[\$](strvar, Index[, Len]) = S\$

**Group** String

**Description** Function: Return the substring of S\$ starting at Index for Len chars.

Instruction: Assign S\$ to the substring in strvar starting at Index for Len chars.

Note: A similar function, MidB, returns the Len bytes starting a byte Index.

Parameter	Description (Mid Function)
S\$	Copy chars from this string value. If this value is <b>Null</b> then <b>Null</b> is returned.
Index	Start copying chars starting at this index value. If the string is not that long then return a null string.
Len	Copy this many chars. If the S\$ does not have that many chars starting at Index then copy the remainder of S\$.

Parameter	Description (Mid Assignment)
strvar	Change part of this string.
Index	Change strvar starting at this index value. If the string is not that long then it is not changed.
Len	The number of chars copied is smallest of: the value of Len, the length of S\$ and the remaining length of strvar. (If this value is omitted then the number of chars copied is the smallest of: the length of S\$ and the remaining length of strvar.)
S\$	Copy chars from this string value.

**See Also** InStr(), Left\$( ), Len( ), Replace\$( ), Right\$( ).

**Example**

```
Sub Main
  S$ = "Hello There"
  Mid$(S$,7) = "?????????"
  Debug.Print S$ "Hello ??????"
  Debug.Print Mid$("Hello",2,1) "e"
End Sub
```

## Minute Function

**Syntax** Minute(dateexpr)

**Group** Time/Date

**Description** Return the minute of the hour (0 to 59).

---

Parameter	Description
<i>dateexpr</i>	Return the minute of the hour for this date value. If this value is <b>Null</b> then <b>Null</b> is returned.

---

**See Also** **Hour()**, **Second()**, **Time()**.

**Example**

```
Sub Main
    Debug.Print Minute(#12:00:01 AM#) ' 0
End Sub
```

## Mkdir Instruction

---

**Syntax** Mkdir *Name\$*

**Group** File

**Description** Make directory *Name\$*.

---

Parameter	Description
<i>Name\$</i>	This string value is the path and name of the directory. A path relative to the current directory can be used.

---

**See Also** **Rmdir**.

**Example**

```
Sub Main
    Mkdir "C:\WWTEMP"
End Sub
```

## Month Function

---

**Syntax** Month(*dateexpr*)

**Group** Time/Date

**Description** Return the month of the year (1 to 12).

---

Parameter	Description
<i>dateexpr</i>	Return the month of the year for this date value. If this value is <b>Null</b> then <b>Null</b> is returned.

---

**See Also** **Date()**, **Day()**, **Weekday()**, **Year()**.

**Example**

```
Sub Main
    Debug.Print Month(#1/1/1900#) ' 1
End Sub
```

# MsgBox Instruction/Function

---

**Syntax**            `MsgBox Message$[, Type][, Title$]`  
                       -or-  
                       `MsgBox(Message$[, Type][, Title$])`

**Group**             User Input

**Description**       Show a message box titled *Title\$*. *Type* controls what the message box looks like (choose one value from each category). Use `MsgBox()` if you need to know what button was pressed. The result indicates which button was pressed.

Result	Value	Button Pressed
vbOK	1	OK button
vbCancel	2	Cancel button
vbAbort	3	Abort button
vbRetry	4	Retry button
vbIgnore	5	Ignore button
vbYes	6	Yes button
vbNo	7	No button

Parameter	Description
<i>Message\$</i>	This string value is the text that is shown in the message box.
<i>Type</i>	This number value controls the type of message box. Choose one value from each of the following tables.
<i>Title\$</i>	This string value is the title of the message box.

Button	Value	Effect
vbOkOnly	0	OK button
vbOkCancel	1	OK and Cancel buttons
vbAbortRetryIgnore	2	Abort, Retry, Ignore buttons
vbYesNoCancel	3	Yes, No, Cancel buttons
vbYesNo	4	Yes and No buttons
vbRetryCancel	5	Retry and Cancel buttons

Icon	Value	Effect
	0	No icon
vbCritical	16	Stop icon
vbQuestion	32	Question icon
vbExclamation	48	Attention icon
vbInformation	64	Information icon

Default	Value	Effect
---------	-------	--------

vbDefaultButton1	0	First button
vbDefaultButton2	256	Second button
vbDefaultButton3	512	Third button
Mode	Value	Effect
vbApplicationModal	0	Application modal
vbSystemModal	4096	System modal

### Example

```

Sub Main
  MsgBox "Please press OK button"
  If MsgBox("Please press OK button", vbOkCancel) =
vbOK Then
    Debug.Print "OK was pressed"
  Else
    Debug.Print "Cancel was pressed"
  End If
End Sub

```

## Name Instruction

**Syntax** Name *OldName\$* As *NewName\$*

**Group** File

**Description** Rename file *OldName\$* as *NewName\$*.

Parameter	Description
<i>OldName\$</i>	This string value is the path and name of the file. A path relative to the current directory can be used.
<i>NewName\$</i>	This is the new file name (and path). A path relative to the current directory can be used.

### Example

```

Sub Main
  Name "AUTOEXEC.BAK" As "AUTOEXEC.SAV"
End Sub

```

## Nothing Keyword

**Group** Constant

**Description** An *objexpr* that does not refer to any object.

## Now Function

---

**Syntax** Now

**Group** Time/Date

**Description** Return the current date and time as a **date** value.

**See Also** **Date, Time, Timer.**

**Example**

```
Sub Main
    Debug.Print Now ' example: 1/1/1995 10:05:32 AM
End Sub
```

## Null Keyword

---

**Group** Constant

**Description** A *variant expression* that is null. A null value propagates through an expression causing the entire expression to be Null. Attempting to use a Null value as a string or numeric argument causes a run-time error. A Null value prints as "#NULL#".

**Example**

```
Sub Main
    X = Null
    Debug.Print X = Null ' #NULL#
    Debug.Print IsNull(X) ' True
End Sub
```

## Object Data Type

---

**Group** Data Type

**Description** An object reference value. (see **Objects**)

## Object Module

---

**Group** Declaration

**Description** An object *module* implements an OLE Automation object.

- It has a set of **Public procedures** accessible from other *macros* and *modules*.
- These public symbols are accessed via the name of the object module or an object variable.
- Public **Consts**, **Types**, arrays, fixed length strings are not allowed.
- An object module is similar to a **class module** except that one instance is automatically created. That instance has the same name as the object module's name.
- To create additional instances use:

```
Dim Obj As objectname
Set Obj = New objectname
```

#### See Also

**Class Module, Code Module, Uses.**

#### Example

```
'A.WWB
'#Uses "System.OBM"
Sub Main
    Debug.Print Hex(System.Version)
End Sub

'System.OBM
Option Explicit
Declare Function GetVersion16 Lib "Kernel" _
    Alias "GetVersion" () As Long
Declare Function GetVersion32 Lib "Kernel32" _
    Alias "GetVersion" () As Long

Public Function Version() As Long
    If Win16 Then
        Version = GetVersion16
    Else
        Version = GetVersion32
    End If
End Function
```

## Object\_Initialize Sub

---

<b>Syntax</b>	Private Sub Object_Initialize() ... End Sub
<b>Group</b>	Declaration
<b>Description</b>	Object module initialization subroutine. Each time a new instance is created for a Object module the Object_Initialize sub is called.

If `Object_Initialize` is not defined then no special initialization occurs.

Note: `Object_Initialize` is also called for the instance that is automatically created.

**See Also**      **Object Module, Object\_Terminate.**

## Object\_Terminate Sub

---

**Syntax**      `Private Sub Object_Terminate()  
                  ...  
End Sub`

**Group**        Declaration

**Description**    Object module termination subroutine. Each time an instance is destroyed for a Object module the `Object_Terminate` sub is called. If `Object_Terminate` is not defined then no special termination occurs.

**See Also**      **Object Module, Object\_Initialize.**

## Oct\$ Function

---

**Syntax**        `Oct[$](Num)`

**Group**        String

**Description**    Return a octal string.

Parameter	Description
<i>Num</i>	Return an octal encoded string for this number value.

**See Also**      **Hex\$(), Str\$(), Val().**

**Example**        `Sub Main  
                  Debug.Print Oct$(15) '17  
End Sub`

## OKButton Dialog Item Definition

---

**Syntax**        `OKButton X, Y, DX, DY[, .Field]`

**Group** User Dialog

**Description** Define an OK button item. Pressing the OK button updates the *dlgvar* field values and closes the dialog. (**Dialog**( ) function call returns -1.)

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Field</i>	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this is omitted then the field name is "OK".

**See Also** **Begin Dialog, Dim As UserDialog.**

**Example**

```

Sub Main
  Begin Dialog UserDialog 200,120
    Text 10,10,180,30,"Please push the OK button"
    OKButton 80,90,40,20
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg ' show dialog (wait for ok)
End Sub

```

## On Error Instruction

**Syntax**

```

On Error GoTo 0
-or-
On Error GoTo label
-or-
On Error Resume Next

```

**Group** Error Handling

**Description** Form 1: Disable the error handler (default).

Form 2: Send error conditions to an error handler.

Form 3: Error conditions continue execution at the next statement.



## Example

```
Sub Main
  Open "XXX" For Output As #1
  Print #1, "1,2, "Hello""
  Close #1
End Sub
```

# Operators

---

## Syntax

^ Not \* / \ Mod + - & < <= > >= = <> **Is** And Or Xor Eqv  
Imp

## Description

These operators are available for numbers *n1* and *n2* or strings *s1* and *s2*. If any value in an expression is **Null** then the expression's value is **Null**. The order of operator evaluation is controlled by operator *precedence*.

Operator	Description
- <i>n1</i>	Negate <i>n1</i> .
<i>n1</i> ^ <i>n2</i>	Raise <i>n1</i> to the power of <i>n2</i> .
<i>n1</i> * <i>n2</i>	Multiply <i>n1</i> by <i>n2</i> .
<i>n1</i> / <i>n2</i>	Divide <i>n1</i> by <i>n2</i> .
<i>n1</i> \ <i>n2</i>	Divide the integer value of <i>n1</i> by the integer value of <i>n2</i> .
<i>n1</i> Mod <i>n2</i>	Remainder of the integer value of <i>n1</i> after dividing by the integer value of <i>n2</i> .
<i>n1</i> + <i>n2</i>	Add <i>n1</i> to <i>n2</i> .
<i>s1</i> + <i>s2</i>	Concatenate <i>s1</i> with <i>s2</i> .
<i>n1</i> - <i>n2</i>	Difference of <i>n1</i> and <i>n2</i> .
<i>s1</i> & <i>s2</i>	Concatenate <i>s1</i> with <i>s2</i> .
<i>n1</i> < <i>n2</i>	Return <b>True</b> if <i>n1</i> is less than <i>n2</i> .
<i>n1</i> <= <i>n2</i>	Return <b>True</b> if <i>n1</i> is less than or equal to <i>n2</i> .
<i>n1</i> > <i>n2</i>	Return <b>True</b> if <i>n1</i> is greater than <i>n2</i> .
<i>n1</i> >= <i>n2</i>	Return <b>True</b> if <i>n1</i> is greater than or equal to <i>n2</i> .
<i>n1</i> = <i>n2</i>	Return <b>True</b> if <i>n1</i> is equal to <i>n2</i> .
<i>n1</i> <> <i>n2</i>	Return <b>True</b> if <i>n1</i> is not equal to <i>n2</i> .
<i>s1</i> < <i>s2</i>	Return <b>True</b> if <i>s1</i> is less than <i>s2</i> .
<i>s1</i> <= <i>s2</i>	Return <b>True</b> if <i>s1</i> is less than or equal to <i>s2</i> .
<i>s1</i> > <i>s2</i>	Return <b>True</b> if <i>s1</i> is greater than <i>s2</i> .
<i>s1</i> >= <i>s2</i>	Return <b>True</b> if <i>s1</i> is greater than or equal to <i>s2</i> .
<i>s1</i> = <i>s2</i>	Return <b>True</b> if <i>s1</i> is equal to <i>s2</i> .
<i>s1</i> <> <i>s2</i>	Return <b>True</b> if <i>s1</i> is not equal to <i>s2</i> .
Not <i>n1</i>	Bitwise invert the integer value of <i>n1</i> . Only Not <b>True</b> is <b>False</b> .
<i>n1</i> And <i>n2</i>	Bitwise and the integer value of <i>n1</i> with the integer value <i>n2</i> .
<i>n1</i> Or <i>n2</i>	Bitwise or the integer value of <i>n1</i> with the integer value <i>n2</i> .

<i>n1</i> Xor <i>n2</i>	Bitwise exclusive-or the integer value of <i>n1</i> with the integer value <i>n2</i> .
<i>n1</i> Eqv <i>n2</i>	Bitwise equivalence the integer value of <i>n1</i> with the integer value <i>n2</i> (same as Not ( <i>n1</i> Xor <i>n2</i> )).
<i>n1</i> Imp <i>n2</i>	Bitwise implicate the integer value of <i>n1</i> with the integer value <i>n2</i> (same as (Not <i>n1</i> ) Or <i>n2</i> ).

---

## Example

```

Sub Main
  N1 = 10
  N2 = 3
  S1$ = "asdfg"
  S2$ = "hjkl"
  Debug.Print -N1           '-10
  Debug.Print N1 ^ N2      ' 1000
  Debug.Print Not N1       '-11
  Debug.Print N1 * N2      ' 30
  Debug.Print N1 / N2      ' 3.33333333333333
  Debug.Print N1 \ N2      ' 3
  Debug.Print N1 Mod N2    ' 1
  Debug.Print N1 + N2      ' 13
  Debug.Print S1$ + S2$    '"asdfghjkl"
  Debug.Print N1 - N2      ' 7
  Debug.Print N1 & N2      '"103"
  Debug.Print N1 < N2     'False
  Debug.Print N1 <= N2    'False
  Debug.Print N1 > N2     'True
  Debug.Print N1 >= N2    'True
  Debug.Print N1 = N2     'False
  Debug.Print N1 <> N2    'True
  Debug.Print S1$ < S2$   'True
  Debug.Print S1$ <= S2$  'True
  Debug.Print S1$ > S2$   'False
  Debug.Print S1$ >= S2$  'False
  Debug.Print S1$ = S2$   'False
  Debug.Print S1$ <> S2$  'True
  Debug.Print N1 And N2   ' 2
  Debug.Print N1 Or N2    ' 11
  Debug.Print N1 Xor N2   ' 9
  Debug.Print N1 Eqv N2   ' -10
  Debug.Print N1 Imp N2   ' -9
End Sub

```

## Option Definition

---

**Syntax**

```

Option Base [0|1]
-or-
Option Explicit
-or-
Option Private Module

```

<b>Group</b>	Declaration
<b>Description</b>	<p>Form 1: Set the default base index for array declarations. Affects <b>Dim</b>, <b>Static</b>, <b>Private</b>, <b>Public</b> and <b>ReDim</b>. Does not affect <b>Array</b>, <b>ParamArray</b> or arrays declare in a <b>Type</b>. Option Base 0 is the default.</p> <p>Form 2: Require all variables to be declared prior to use. Variables are declared using <b>Dim</b>, <b>Private</b>, <b>Public</b>, <b>Static</b> or as a parameter of <b>Sub</b>, <b>Function</b> or <b>Property</b> blocks.</p> <p>Form 3: Public symbols defined by the module are only accessible from the same project.</p>
<b>See Also</b>	<b>Dim</b> , <b>Private</b> , <b>Public</b> , <b>ReDim</b> , <b>Static</b> .
<b>Example</b>	<pre>Option Base 1 Option Explicit  Sub Main     Dim A     Dim C(2) ' same as Dim C(1 To 2)     Dim D(0 To 2)     A = 1     B = 2 ' B has not been declared End Sub</pre>

## OptionButton Dialog Item Definition

<b>Syntax</b>	<code>OptionButton X, Y, DX, DY, Title\$[, .Field]</code>
<b>Group</b>	User Dialog
<b>Description</b>	Define an option button item.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Title\$</i>	The value of this string is the title of the option button.

**See Also****Begin Dialog, Dim As UserDialog, OptionGroup.****Example**

```

Sub Main
  Begin Dialog UserDialog 200,120
    Text 10,10,180,15,"Please push the OK button"
    OptionGroup .options
      OptionButton 10,30,180,15,"Option &0"
      OptionButton 10,45,180,15,"Option &1"
      OptionButton 10,60,180,15,"Option &2"
    OKButton 80,90,40,20
  End Dialog
  Dim dlg As UserDialog
  dlg.options = 2
  Dialog dlg ' show dialog (wait for ok)
  Debug.Print dlg.options
End Sub

```

## OptionGroup Dialog Item Definition

---

**Syntax**

```

OptionGroup .Field
OptionButton X, Y, DX, DY, Title$, .Field]
OptionButton X, Y, DX, DY, Title$, .Field]
...

```

**Group**

User Dialog

**Description**

Define a optiongroup and option button items.

Parameter	Description
<i>Field</i>	The value of the option group is accessed via this field. This first option button is 0, the second is 1, etc.
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Title\$</i>	The value of this string is the title of the option button.

**See Also****Begin Dialog, Dim As UserDialog, OptionButton.**

## Example

```
Sub Main
Begin Dialog UserDialog 200,120
Text 10,10,180,15,"Please push the OK button"
OptionGroup .options
    OptionButton 10,30,180,15,"Option &0"
    OptionButton 10,45,180,15,"Option &1"
    OptionButton 10,60,180,15,"Option &2"
OKButton 80,90,40,20
End Dialog
Dim dlg As UserDialog
dlg.options = 2
Dialog dlg ' show dialog (wait for ok)
Debug.Print dlg.options
End Sub
```

## Picture Dialog Item Definition

---

**Syntax**            Picture *X, Y, DX, DY, FileName\$, Type[, .Field]*

**Group**             User Dialog

**Description**      Define a picture item. The bitmap is automatically sized to fit the item's entire area.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>FileName\$</i>	The value of this string is the .BMP file shown in the picture control.
<i>Type</i>	This numeric value indicates the type of bitmap used. See below.
<i>Field</i>	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this identifier is omitted then the first two words of the title are used.

---

Type	Effect
0	<i>FileName</i> is the name of the bitmap file. If the file does not exist then "(missing picture)" is displayed.
3	The clipboard's bitmap is displayed. Not supported.
+16	Instead of displaying "(missing picture)" a run-time error occurs.

---

**See Also**            **Begin Dialog, Dim As UserDialog.**

**Example**

```
Sub Main
  Begin Dialog UserDialog 200,120
    Picture 10,10,180,75,"SAMPLE.BMP",0
    OKButton 80,90,40,20
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg ' show dialog (wait for ok)
End Sub
```

## PortInt Data Type

---

**Group**            Data Type

**Description**    A portable integer value.

- For Win16: A 16 bit integer value.
- For Win32: A 32 bit integer value.

## Print Instruction

---

**Syntax**            Print #*StreamNum*, [*expr*[; ...][;]]

**Group**            File

**Description**    Print the *expr*(s) to *StreamNum*. Use ; to separate expressions. A *num* is it automatically converted to a string before printing (just like **Str\$( )**). If the instruction does not end with a ; then a newline is printed at the end.

**See Also**            **Input, Line Input, Write.**

**Example**

```
Sub Main
  A = 1
  B = 2
  C$ = "Hello"
  Open "XXX" For Output As #1
  Print #1,A;" ";B;" ";C$;" "
  Close #1
End Sub
```

## Private Definition

---

<b>Syntax</b>	<code>Private [WithEvents] name[type]([dim[, ...]]) [As [New] type][, ...]</code>
<b>Group</b>	Declaration
<b>Description</b>	Create arrays (or simple variables) which are available to the entire <i>macro/module</i> , but not other macros/modules. Dimension var array(s) using the <i>dims</i> to establish the minimum and maximum index value for each dimension. If the <i>dims</i> are omitted then a scalar (single value) variable is defined. A dynamic array is declared using ( ) without any <i>dims</i> . It must be <b>ReDimensioned</b> before it can be used. The Private statement must be placed outside of <b>Sub</b> , <b>Function</b> or <b>Property</b> blocks.
<b>See Also</b>	<b>Dim</b> , <b>Option Base</b> , <b>Public</b> , <b>ReDim</b> , <b>Static</b> , <b>WithEvents</b> .
<b>Example</b>	<pre>Private A0,A1(1),A2(1,1)  Sub Init     A0 = 1     A1(0) = 2     A2(0,0) = 3 End Sub  Sub Main     Init     Debug.Print A0;A1(0);A2(0,0) ' 1 2 3 End Sub</pre>

## Private Keyword

---

<b>Group</b>	Declaration
<b>Description</b>	<b>Private Consts</b> , <b>Declares</b> , <b>Functions</b> , <b>Property</b> s, <b>Subs</b> and <b>Types</b> are only available in the current <i>macro/module</i> .

## Property Definition

---

<b>Syntax</b>	<pre>[   Private   Public   Friend ] _ Property Get name[type]([param[, ...]]) [As type]     statements End Property -or- [   Private   Public   Friend ] _ Property [Let Set] name([param[, ...]])</pre>
---------------	---

```
statements
End Property
```

**Group**

Declaration

**Description**

User defined property. The property defines a set of *statements* to be executed when its value is used or changed. A property acts like a variable, except that getting its value calls Property Get and changing its value calls Property Let (or Property Set). Property Get and Property Let with the same *name* define a property that holds a value. Property Get and Property Set with the same *name* define a property that holds an object reference. The values of the calling *arglist* are assigned to the *params*. (For Property Let and Property Set the last parameter is the value on the right hand side of the assignment operator.)

Property defaults to **Public** if **Private**, **Public** or **Friend** are not is specified.

**See Also**

**Function, Sub.**

**Example**

```
Dim X_Value

Property Get X()
    X = X_Value
End Property

Property Let X(NewValue)
    If Not IsNull(NewValue) Then X_Value = NewValue
End Property

Sub Main
    X = "Hello"
    Debug.Print X
    X = Null
    Debug.Print X
End Sub
```

## Public Definition

---

**Syntax**

```
Public [ WithEvents ] name [ type ] [ ( [ dim [ , ... ] ] ) ] [ As  
[ New ] type ] [ , ... ]
```

**Group**

Declaration

**Description**

Create arrays (or simple variables) which are available to the entire *macro/module* and other macros/modules. Dimension var

array(s) using the *dims* to establish the minimum and maximum index value for each dimension. If the *dims* are omitted then a scalar (single value) variable is defined. A dynamic array is declared using ( ) without any *dims*. It must be **ReDimensioned** before it can be used. The Public statement must be placed outside of **Sub**, **Function** or **Property** blocks.

**See Also**

**Dim**, **Option Base**, **Private**, **ReDim**, **Static**, **WithEvents**.

**Example**

```
Public A0,A1(1),A2(1,1)

Sub Init
    A0 = 1
    A1(0) = 2
    A2(0,0) = 3
End Sub

Sub Main
    Init
    Debug.Print A0;A1(0);A2(0,0) ' 1 2 3
End Sub
```

## Public Keyword

---

<b>Group</b>	Declaration
<b>Description</b>	<b>Public Consts, Declares, Functions, Property, Subs and Types</b> in a <i>module</i> are available in all other <i>macros/modules</i> that access it.

## PushButton Dialog Item Definition

---

<b>Syntax</b>	PushButton <i>X, Y, DX, DY, Title\$[, .Field]</i>
<b>Group</b>	User Dialog
<b>Description</b>	Define a push button item. Pressing the push button updates the <i>dlgvar</i> field values and closes the dialog. ( <b>Dialog</b> ( ) function call returns the push button's ordinal number in the dialog. The first push button returns 1.)

Parameter	Description
X	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.

<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Title\$</i>	The value of this string is the title of the push button control.
<i>Field</i>	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this identifier is omitted then the first two words of the title are used.

---

### See Also

**Begin Dialog, Dim As UserDialog.**

### Example

```
Sub Main
  Begin Dialog UserDialog 200,120
    Text 10,10,180,30,"Please push the DoIt button"
    OKButton 40,90,40,20
    PushButton 110,90,60,20,"&Do It"
  End Dialog
  Dim dlg As UserDialog
  Debug.Print Dialog(dlg)
End Sub
```

## Put Instruction

---

**Syntax**            `Put StreamNum, [RecordNum], var`

**Group**             File

**Description**      Write a variable's value to *StreamNum*.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>RecordNum</i>	For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1. If this is omitted then the current position (or record number) is used.
<i>var</i>	This variable value is written to the file. For a fixed length variable (like <b>Long</b> ) the number of bytes required to store the variable are written. For a <b>Variant</b> variable two bytes which describe its type are written and then the variable value is written accordingly. For a <i>usertype</i> variable each field is written in sequence. For an array variable each element is written in sequence. For a dynamic array variable the number of dimensions and range of each dimension is written prior to writing the array values. All binary data values are written to the file in <i>little-endian</i> format.

Note: When a writing string (or a dynamic array) to a Binary

mode file the string length (or array dimension) information is not written. Only the string data or array elements are written.

---

**See Also**

**Get, Open.**

**Example**

```
Sub Main
  Dim V As Variant
  Open "SAVE_V.DAT" For Binary Access Write As #1
  Put #1, , V
  Close #1
End Sub
```

## QBColor Function

---

**Syntax**

QBColor(*num*)

**Group**

Miscellaneous

**Description**

Return the appropriate color defined by Quick Basic.

---

<b>num</b>	<b>color</b>
0	black
1	blue
2	green
3	cyan
4	red
5	magenta
6	yellow
7	white
8	gray
9	light blue
10	light green
11	light cyan
12	light red
13	light magenta
14	light yellow
15	bright white

---

**See Also**

**RGB().**

**Example**

```

Sub Main
  Debug.Print Hex(QBColor(1))  "' 800000"
  Debug.Print Hex(QBColor(7))  "'C0C0C0"
  Debug.Print Hex(QBColor(8))  "'808080"
  Debug.Print Hex(QBColor(9))  "'FF0000"
  Debug.Print Hex(QBColor(10)) "'FF00"
  Debug.Print Hex(QBColor(12)) "'FF"
  Debug.Print Hex(QBColor(15)) "'FFFFFF"
End Sub

```

## Randomize Instruction

---

**Syntax** Randomize

**Group** Math

**Description** Randomize the random number generator.

**See Also** **Rnd()**.

**Example**

```

Sub Main
  Randomize
  Debug.Print Rnd ' 0.????????????????
End Sub

```

## ReDim Instruction

---

**Syntax** ReDim [Preserve] *name*[*type*][([*dim*[, ...]])] [*As type*][, ...]  
-or-  
ReDim [Preserve] *usertypevar.elem*[*type*][([*dim*[, ...]])] [*As type*][, ...]

**Group** Declaration

**Description** Redimension a dynamic *arrayvar* or *user defined type* array element. Use Preserve to keep the array values. Otherwise, the array values will all be reset. When using preserve only the last index of the array may change, but the number of indexes may not. (A one-dimensional array can't be redimensioned as a two-dimensional array.)

**See Also** **Dim**, **Option Base**, **Private**, **Public**, **Static**.

**Example**

```

Sub Main
  Dim X()
  ReDim X(3)
  Debug.Print UBound(X) ' 3
  ReDim X(200)
  Debug.Print UBound(X) ' 200
End Sub

```

## Reference Comment

---

**Syntax**

```
'#Reference
{uuid}#vermajor.verminor#lcid#[path[#name]]
```

**Description**

The Reference comment indicates that the current *macro/module* references the type library identified. Reference comment lines must be the first lines in the macro/module (following the global **Attributes**). Reference comments are in reverse priority (from lowest to highest). The IDE does not display the reference comments.

Parameter	Description
uuid	Type library's universally unique identifier.
vermajor	Type library's major version number.
verminor	Type library's minor version number.
lcid	Type library's locale identifier.
path	Type library's path.
name	Type library's name.

**Example**

```
'#Reference {00025E01-0000-0000-C000-
000000000046}#4.0#0#C:\PROGRAM FILES\COMMON
FILES\MICROSOFT SHARED\DAO\DAO350.DLL#Microsoft DAO
3.5 Object Library
```

## Rem Instruction

---

**Syntax**

```
Rem ...
-or-
'...
```

**Group**

Miscellaneous

**Description**

Both forms are comments. The Rem form is an instruction. The ' form can be used at the end of any line. All text from either ' or

Rem to the end of the line is part of the comment. That text is not executed.

**Example**

```
Sub Main
    Debug.Print "Hello" ' prints to the output window
    Rem the macro terminates at Main's End Sub
End Sub
```

## Replace\$ Function

---

**Syntax** Replace[\$](*S*, *Pat*, *Rep*, [*Index*], [*Count*])

**Group** String

**Description** Replace *Pat* with *Rep* in *S*.

Parameter	Description
<i>S</i>	This string value is searched. Replacements are made in the string returned by Replace.
<i>Pat</i>	This string value is the pattern to look for.
<i>Rep</i>	This string value is the replacement.
<i>Index</i>	This numeric value is the starting index in <i>S</i> . Replace( <i>S</i> , <i>Pat</i> , <i>Rep</i> , <i>N</i> ) is equivalent to Replace(Mid( <i>S</i> , <i>N</i> ), <i>Pat</i> , <i>Rep</i> ). If this is omitted use 1.
<i>Count</i>	This numeric value is the maximum number of replacements that will be done. If this is omitted use -1 (which means replace all occurrences).

---

**See Also** InStr(), InStrRev(), Left\$, Len(), Mid\$, Right\$().

**Example**

```
Sub Main
    Debug.Print Replace$("abcabc", "b", "B")
' "aBcaBc"
    Debug.Print Replace$("abcabc", "b", "B", , 1)
' "aBcabc"
    Debug.Print Replace$("abcabc", "b", "B", 3) ' "caBc"
    Debug.Print Replace$("abcabc", "b", "B", 9) ' ""
End Sub
```

## Reset Instruction

---

**Syntax** Reset

**Group** File

**Description** Close all open streams for the current *macro/module*.

**See Also****Close, Open.****Example**

```

Sub Main
  ' read the first line of XXX and print it
  Open "XXX" For Input As #1
  Line Input #1,L$
  Debug.Print L$
  Reset
End Sub

```

## Resume Instruction

---

**Syntax**

```

Resume label
-or-
Resume Next

```

**Group**

Error Handling

**Description**Form 1: Resume execution at *label*.

Form 2: Resume execution at the next statement.

Once an error has occurred, the error handler can use `Resume` to continue execution. The error handler must use `Resume` or **Exit** at the end.

Note: This instruction clears the **Err** and sets **Error\$** to null.

**Example**

```

Sub Main
  On Error GoTo X
  Err.Raise 1
  Debug.Print "RESUMING"
  Exit Sub

X:  Debug.Print "Err=";Err
  Resume Next
End Sub

```

## RGB Function

---

**Syntax**RGB(*red, green, blue*)**Group**

Miscellaneous

**Description**

Return a color.

**See Also** [QBColor\(\)](#).

**Example**

```
Sub Main
    Debug.Print Hex( RGB(255,0,0) ) ' "FF0000"
End Sub
```

## Right\$ Function

---

**Syntax** `Right[$](S$, Len)`

**Group** String

**Description** Return the last *Len* chars of *S\$*.

Note: A similar function, [RightB](#), returns the last *Len* bytes.

Parameter	Description
<i>S\$</i>	Return the right portion of this string value. If this value is <b>Null</b> then <b>Null</b> is returned.
<i>Len</i>	Return this many chars. If <i>S\$</i> is shorter than that then just return <i>S\$</i> .

**See Also** [InStr\(\)](#), [InStrRev\(\)](#), [Left\\$\(\)](#), [Len\(\)](#), [Mid\\$\(\)](#), [Replace\\$\(\)](#).

**Example**

```
Sub Main
    Debug.Print Right$("Hello",3) ' "llo"
End Sub
```

## Rmdir Instruction

---

**Syntax** `Rmdir Name$`

**Group** File

**Description** Remove directory *Name\$*.

Parameter	Description
<i>Name\$</i>	This string value is the path and name of the directory. A path relative to the current directory can be used.

**See Also** [Mkdir](#).

**Example**

```
Sub Main
    Rmdir "C:\WWTEMP"
End Sub
```

## Rnd Function

---

**Syntax** `Rnd( [Num] )`

**Group** Math

**Description** Return a random number greater than or equal to zero and less than one.

Parameter	Description
<i>Num</i>	This number value is ignored.

**See Also** **Randomize.**

**Example**

```
Sub Main
    Debug.Print Rnd() ' 0.????????????????
End Sub
```

## RSet Instruction

---

**Syntax** `RSet strvar = str`

**Group** Assignment

**Description** Assign the value of *str* to *strvar*. Shorten *str* by removing trailing chars (or extend with leading blanks). The previous length *strvar* is maintained.

**See Also** **LSet.**

**Example**

```
Sub Main
    S$ = "123"
    RSet S$ = "A"
    Debug.Print ".";S$;"." ' ". A."
End Sub
```

## RTrim\$ Function

---

**Syntax** `RTrim[$](S$)`

**Group** String

**Description** Return the string with *S\$*'s trailing spaces removed.

Parameter	Description
<i>S\$</i>	Copy this string without the trailing spaces. If this value is <b>Null</b> then <b>Null</b> is returned.

**See Also** `LTrim$()`, `Trim$()`.

**Example**

```
Sub Main
    Debug.Print ".";RTrim$(" x ");"." ' ". x."
End Sub
```

## SaveSetting Instruction

---

**Syntax** `SaveSetting AppName$, Section$, Key$, Setting`

**Group** Settings

**Description** Save the *Setting* for *Key* in *Section* in project *AppName*. Win16 and Win32s store settings in a .ini file named *AppName*. Win32 stores settings in the registration database.

Parameter	Description
<i>AppName\$</i>	This string value is the name of the project which has this <i>Section</i> and <i>Key</i> .
<i>Section\$</i>	This string value is the name of the section of the project settings.
<i>Key\$</i>	This string value is the name of the key in the section of the project settings.
<i>Setting</i>	Set the key to this value. (The value is stored as a string.)

**Example**

```
Sub Main
    SaveSetting "MyApp", "Font", "Size", 10
End Sub
```

## Second Function

---

**Syntax** `Second(dateexpr)`

**Group** Time/Date

**Description** Return the second of the minute (0 to 59).

Parameter	Description
<i>dateexpr</i>	Return the second of the minute for this date value. If this value is <b>Null</b> then <b>Null</b> is returned.

**See Also** `Hour()`, `Minute()`, `Time()`.

**Example**

```
Sub Main
    Debug.Print Second(#12:00:01 AM#) ' 1
End Sub
```

## Seek Instruction

---

**Syntax**            `Seek [#]StreamNum, Count`

**Group**            File

**Description**     Position *StreamNum* for input *Count*.

---

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>Count</i>	For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1.

---

**See Also**        **Seek()**.

**Example**

```
Sub Main
  Open "XXX" For Input As #1
  Line Input #1,L$
  Seek #1,0 ' rewind to start of file
  Input #1,A
  Close #1
  Debug.Print A
End Sub
```

## Seek Function

---

**Syntax**            `Seek(StreamNum)`

**Group**            File

**Description**     Return *StreamNum* current position. For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1.

---

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.

---

**See Also**        **Seek.**

**Example**

```
Sub Main
  Open "XXX" For Input As #1
  Line Input #1,L$
  Debug.Print Seek(1)
  Close #1
End Sub
```

# Select Case Statement

---

**Syntax**

```
Select Case expr
  [Case caseexpr[, ...]
   statements]...
  [Case Else
   statements]
End Select
```

**Group** Flow Control

**Description** Select the appropriate case by comparing the *expr* with each of the *caseexprs*. Select the Case Else part if no *caseexpr* matches. (If the Case Else is omitted then skip the entire Select...End Select block.)

---

<b>caseexpr</b>	<b>Description</b>
<i>expr</i>	Execute if equal.
Is < <i>expr</i>	Execute if less than.
Is <= <i>expr</i>	Execute if less than or equal to.
Is > <i>expr</i>	Execute if greater than.
Is >= <i>expr</i>	Execute if greater than or equal to.
Is <> <i>expr</i>	Execute if not equal to.
<i>expr1</i> To <i>expr2</i>	Execute if greater than or equal to <i>expr1</i> and less than or equal to <i>expr2</i> .

---

**See Also** If, Choose(), IIf().

**Example**

```
Sub Main
  S = InputBox("Enter hello, goodbye, dinner or
sleep:")
  Select Case UCase(S)
  Case "HELLO"
  Debug.Print "come in"
  Case "GOODBYE"
  Debug.Print "see you later"
  Case "DINNER"
  Debug.Print "Please come in."
  Debug.Print "Dinner will be ready soon."
  Case "SLEEP"
  Debug.Print "Sorry."
  Debug.Print "We are full for the night"
  Case Else
  Debug.Print "What?"
  End Select
End Sub
```

## SendKeys Instruction

---

**Syntax**            `SendKeys Keys$[, Wait]`

**Group**            Miscellaneous

**Description**     Send *Keys\$* to Windows.

Parameter	Description
<i>Keys\$</i>	Send the keys in this string value to Windows.
<i>Wait</i>	If this is not zero then the keys are sent before executing the next instruction. If this is omitted or zero then the keys are sent during the following instructions.

  

Keys\$	Description
+	Shift modifier key: the following key is a shifted key
^	Ctrl modifier key: the following key is a control key
%	Alt modifier key: the following key is an alt key
~	Enter key
(keys)	Modifiers apply to all keys
{special n}	special key (n is an optional repeat count)
k	k Key (k is any single char)
K	Shift k Key (K is any capital letter)

<b>Special Keys</b>	<b>Key</b>	<b>Description</b>	<b>Key Description</b>
	k	k Key (any single char)	K
		shift k Key	
			Left Left Arrow
	Key		
	Cancel	Break Key	Right
		Right Arrow Key	
	Esc or Escape	Up	Up Arrow Key
		Escape Key	Down
		Down Arrow Key	
	Enter	Enter Key	PgUp
		Page Up Key	
	Menu	Menu Key (Alt)	PgDn
		Page Down Key	
	Help	Help Key (?)	Home
		Home Key	
	Prtsc	Print Screen Key	End End
	Key		
	Print	?	Select ?
	Execute	?	Clear
		Num Pad 5 Key	
	Tab	Tab Key	Pad0 to Pad9
	Pause	Pause Key	Num Pad 0-
	9 Keys		
	BS, BkSp or BackSpace	Pad*	Num Pad * Key
		Back Space Key	Pad+
		Num Pad + Key	
	Del or Delete	PadEnter	Num Pad Enter
		Delete Key	Pad- Num Pad -
	Key		
	Ins or Insert	Pad.	Num Pad . Key
		Insert Key	Pad/ Num Pad /
	Key		
			F1 to F24 F1
	to F24 Keys		

**See Also** **AppActivate, Shell()**.

**Example**

```

Sub Main
  SendKeys "%S"      ' send Alt-S (Search)
  SendKeys "GoTo~~" ' send G o T o {Enter} {Enter}
End Sub

```

## Set Instruction

---

<b>Syntax</b>	<code>Set objvar = objexpr</code> -or- <code>Set objvar = <b>New</b> objtype</code>
<b>Group</b>	Assignment
<b>Description</b>	Form 1: Set <i>objvar</i> 's object reference to the object reference of <i>objexpr</i> .

Form 2: Set *objvar*'s object reference to the a new instance of *objtype*.

The Set instruction is how object references are assigned.

<b>Example</b>	<pre>Sub Main   Dim App As Object   Set App =   CreateObject("WinWrap.CppDemoApplication")   App.Move 20,30 ' move icon to 20,30   Set App = <b>Nothing</b>   App.Quit      ' run-time error (no object) End Sub</pre>
----------------	--

## SetAttr Instruction

---

<b>Syntax</b>	<code>SetAttr Name\$, Attrib</code>
<b>Group</b>	File
<b>Description</b>	Set the <i>attributes</i> for file <i>Name\$</i> . If the file does not exist then a run-time error occurs.

---

Parameter	Description
<i>Name\$</i>	This string value is the path and name of the file. A path relative to the current directory can be used.
<i>Attrib</i>	Set the file's <i>attributes</i> to this numeric value.

---

<b>Example</b>	<pre>Sub Main   Attrib = GetAttr("XXX")   SetAttr "XXX",1 ' readonly   Debug.Print GetAttr("XXX") ' 1   SetAttr "XXX",Attrib End Sub</pre>
----------------	--

## Sgn Function

---

**Syntax** `Sgn (Num)`

**Group** Math

**Description** Return the sign.

Parameter	Description
<i>Num</i>	Return the sign of this number value. Return -1 for negative. Return 0 for zero. Return 1 for positive.

**Example**

```
Sub Main
    Debug.Print Sgn(9) ' 1
    Debug.Print Sgn(0) ' 0
    Debug.Print Sgn(-9) '-1
End Sub
```

## Shell Function

---

**Syntax** `Shell (Name$[, WindowType])`

**Group** Miscellaneous

**Description** Execute program *Name\$*. This is the same as using File|Run from the Program Manager. This instruction can run .COM, .EXE, .BAT and .PIF files. If successful, return the task ID.

Parameter	Description
<i>Name\$</i>	This string value is the path and name of the program to run. Command line arguments follow the program name. (A long file name containing a space must be surrounded by literal double quotes.)
<i>WindowType</i>	This controls how the application's main window is shown. See the table below.

WindowType	Value	Effect
vbHide	0	Hide Window
vbNormalFocus	1, 5, 9	Normal Window
vbMinimizedFocus	2	Minimized Window (default)
vbMaximizedFocus	3	Maximized Window
vbNormalNoFocus	4, 8	Normal Deactivated Window
vbMinimizedNoFocus	6, 7	Minimized Deactivated Window

**See Also**      **AppActivate, SendKeys.**

**Example**

```
Sub Main
  X = Shell("Calc") ' run the calc program
  AppActivate X
  SendKeys "% R" ' restore calc's main window
  SendKeys "30*2{+}10=",1 '70
End Sub
```

## Sin Function

---

**Syntax**            `Sin(Num)`

**Group**             Math

**Description**      Return the sine.

Parameter	Description
<i>Num</i>	Return the sine of this number value. This is the number of radians. There are 2*Pi radians in a full circle.

**Example**

```
Sub Main
  Debug.Print Sin(1) ' 0.8414709848079
End Sub
```

## Single Data Type

---

**Group**             Data Type

**Description**      A 32 bit real value.

## Space\$ Function

---

**Syntax**            `Space[$] (Len)`

**Group**             String

**Description**      Return the string *Len* spaces long.

Parameter	Description
<i>Len</i>	Create a string this many spaces long.

**See Also**          **String\$( ).**

**Example**

```

Sub Main
    Debug.Print ". ";Space$(3);"." ' ". . ."
End Sub

```

## Sqr Function

---

**Syntax**            `Sqr(Num)`

**Group**             `Math`

**Description**      Return the square root.

Parameter	Description
<i>Num</i>	Return the square root of this number value.

**Example**

```

Sub Main
    Debug.Print Sqr(9) ' 3
End Sub

```

## Static Definition

---

**Syntax**            `Static name[type][([dim[, ...]])][As [New] type][, ...]`

**Group**             `Declaration`

**Description**      A static variable retains its value between *procedure* calls. Dimension var array(s) using the *dims* to establish the minimum and maximum index value for each dimension. If the *dims* are omitted then a scalar (single value) variable is defined. A dynamic array is declared using ( ) without any *dims*. It must be **ReDimensioned** before it can be used.

**See Also**           **Dim, Option Base, Private, Public, ReDim.**

**Example**

```

Sub A
    Static X
    Debug.Print X
    X = "Hello"
End Sub

Sub Main
    A
    A ' prints "Hello"
End Sub

```

## Stop Instruction

---

<b>Syntax</b>	Stop
<b>Group</b>	Flow Control
<b>Description</b>	Pause execution. If execution is resumed then it starts at the next instruction. Use <b>End</b> to terminate the <i>macro</i> completely.
<b>Example</b>	<pre>Sub Main   For I = 1 To 10     Debug.Print I     If I = 3 Then Stop   Next I End Sub</pre>

## Str\$ Function

---

<b>Syntax</b>	Str[\$](Num)
<b>Group</b>	String
<b>Description</b>	Return the string representation of <i>Num</i> .

Parameter	Description
<i>Len</i>	Return the string representation of this number value. Positive values begin with a blank. Negative values begin with a dash '-':

**See Also** CStr(), Hex\$(), Oct\$(), Val().

<b>Example</b>	<pre>Sub Main   Debug.Print Str\$(9*9) ' 81 End Sub</pre>
----------------	---

## StrComp\$ Function

---

<b>Syntax</b>	StrComp(Str1, Str2, Comp)
<b>Group</b>	String
<b>Description</b>	Compare two strings.

Parameter	Description
<i>Str1</i>	Compare this string with <i>Str2</i> . If this value is <b>Null</b> then <b>Null</b> is returned.

<i>Str2</i>	Compare this string with <i>Str1</i> . If this value is <b>Null</b> then <b>Null</b> is returned.
<i>Comp</i>	This numeric value indicates the type of comparison. If this is omitted or zero then binary comparison is used. Otherwise, text comparison is used. (Text comparison is not case sensitive.)

Result	Description
-1	<i>Str1</i> is less than <i>Str2</i> .
0	<i>Str1</i> is equal to <i>Str2</i> .
1	<i>Str1</i> is greater than <i>Str2</i> .
<b>Null</b>	<i>Str1</i> or <i>Str2</i> is <b>Null</b> .

**See Also**

**LCase\$()**, **StrConv\$()**, **UCase\$()**.

**Example**

```
Sub Main
    Debug.Print StrComp("F","e") ' -1
    Debug.Print StrComp("F","e",1) ' 1
    Debug.Print StrComp("F","f",1) ' 0
End Sub
```

## StrConv\$ Function

**Syntax** `StrConv[$](Str,Conv)`

**Group** String

**Description** Convert the string.

Parameter	Description
<i>Str</i>	Convert this string value. If this value is <b>Null</b> then <b>Null</b> is returned.
<i>Conv</i>	This numeric value indicates the type of conversion. See conversion table below.

Conv	Value	Effect
vbUpperCase	1	Convert to upper case.
vbLowerCase	2	Convert to lower case.
vbProperCase	3	Convert to proper case. (Not supported.)
vbWide	4	Convert to wide. (Only supported for Win32 in eastern locales.)
vbNarrow	8	Convert to narrow. (Only supported for Win32 in eastern locales.)
vbKatakana	16	Convert to Katakana. (Only supported for Win32 in Japanese locales.)
vbHiragana	32	Convert to Hiragana. (Only supported for Win32 in Japanese locales.)

vbUnicode	64	Convert to Unicode. (Only supported for Win32.)
vbFromUnicode	128	Convert from Unicode. (Only supported for Win32.)

---

**See Also** [LCase\\$\(\)](#), [StrComp\(\)](#), [UCase\\$\(\)](#).

**Example**

```

Sub Main
  Dim B(1 To 3) As Byte
  B(1) = 65
  B(2) = 66
  B(3) = 67
  Debug.Print StrConv$(B,vbUnicode) ' "ABC"
End Sub

```

## String Data Type

---

<b>Group</b>	Data Type
<b>Description</b>	An arbitrary length string value. Some useful string constants are predefined: <ul style="list-style-type: none"> <li>• vbNullChar - same as Chr(0)</li> <li>• vbCrLf - same as Chr(13) &amp; Chr(10)</li> <li>• vbCr - same as Chr(13)</li> <li>• vbLf - same as Chr(10)</li> <li>• vbBack - same as Chr(8)</li> <li>• vbFormFeed - same as Chr(12)</li> <li>• vbTab - same as Chr(9)</li> <li>• vbVerticalTab - same as Chr(11)</li> </ul>

## String\*n Data Type

---

<b>Group</b>	Data Type
<b>Description</b>	A fixed length (n) string value.

## String\$ Function

---

<b>Syntax</b>	String[\$](Len, Char \$)
---------------	--------------------------

**Group** String

**Description** Return the string *Len* long filled with *Char* or the first char of *Char\$*.

Parameter	Description
<i>Len</i>	Create a string this many chars long.
<i>Char </i> \$	Fill the string with this char value. If this is a number value then use the ASCII char equivalent. If this is a string value use the first char of that string. If this value is <b>Null</b> then <b>Null</b> is returned.

**See Also** **Space\$()**.

**Example**

```
Sub Main
    Debug.Print String$(4,65)      ' "AAAA"
    Debug.Print String$(4,"ABC")  ' "AAAA"
End Sub
```

## Sub Definition

---

**Syntax**

```
[ | Private | Public | Friend ] _
Sub name([[param[, ...]])]
    statements
End Sub
```

**Group** Declaration

**Description** User defined subroutine. The subroutine defines a set of *statements* to be executed when it is called. The values of the calling *arglist* are assigned to the *params*. A subroutine does not return a result.

Sub defaults to **Public** if **Private**, **Public** or **Friend** are not is specified.

**See Also** **Declare**, **Function**, **Property**.

## Example

```
Sub IdentityArray(A()) ' A() is an array of numbers
  For I = LBound(A) To UBound(A)
    A(I) = I
  Next I
End Sub

Sub CalcArray(A(),B,C) ' A() is an array of numbers
  For I = LBound(A) To UBound(A)
    A(I) = A(I)*B+C
  Next I
End Sub

Sub ShowArray(A()) ' A() is an array of numbers
  For I = LBound(A) To UBound(A)
    Debug.Print "(";I;")=";A(I)
  Next I
End Sub

Sub Main
  Dim X(1 To 4)
  IdentityArray X() ' X(1)=1, X(2)=2, X(3)=3, X(4)=4
  CalcArray X(),2,3 ' X(1)=5, X(2)=7, X(3)=9,
  X(4)=11
  ShowArray X()    ' print X(1), X(2), X(3), X(4)
End Sub
```

## Tan Function

---

**Syntax**            Tan(*Num*)

**Group**             Math

**Description**      Return the tangent.

Parameter	Description
<i>Num</i>	Return the tangent of this number value.

**Example**            Sub Main  
                      Debug.Print Tan(1) ' 1.5574077246549  
                      End Sub

## Text Dialog Item Definition

---

**Syntax**            Text *X, Y, DX, DY, Title\$[, .Field]*

**Group**             User Dialog

**Description**

Define a text item.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Title\$</i>	The value of this string is the title of the text control.
<i>Field</i>	This identifier is the name of the field. The <i>dialogfunc</i> receives this name as <i>string</i> . If this identifier is omitted then the first two words of the title are used.

**See Also**

**Begin Dialog, Dim As UserDialog.**

**Example**

```
Sub Main
  Begin Dialog UserDialog 200,120
    Text 10,10,180,15,"Please push the OK button"
    OKButton 80,90,40,20
  End Dialog
  Dim dlg As UserDialog
  Dialog dlg ' show dialog (wait for ok)
End Sub
```

## TextBox Dialog Item Definition

---

**Syntax**

TextBox *X, Y, DX, DY, .Field\$[, Options]*

**Group**

User Dialog

**Description**

Define a textbox item.

Parameter	Description
<i>X</i>	This number value is the distance from the left edge of the dialog box. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>Y</i>	This number value is the distance from the top edge of the dialog box. It is measured in 1/12 ths of the character height for the dialog's font.
<i>DX</i>	This number value is the width. It is measured in 1/8 ths of the average character width for the dialog's font.
<i>DY</i>	This number value is the height. It is measured in 1/12 ths of the character height for the dialog's font.
<i>Field</i>	The value of the text box is accessed via this field.

*Options* If this numeric value is zero or omitted then a single line of text can be entered. If it is less than zero then a hidden password can be entered. If it is greater than zero then multiple lines of text can be entered.

---

**See Also** **Begin Dialog, Dim As UserDialog.**

**Example**

```
Sub Main
  Begin Dialog UserDialog 200,120
    Text 10,10,180,15,"Please push the OK button"
    TextBox 10,25,180,20,.Text$
    OKButton 80,90,40,20
  End Dialog
  Dim dlg As UserDialog
  dlg.Text$ = "none"
  Dialog dlg ' show dialog (wait for ok)
  Debug.Print dlg.Text$
End Sub
```

## Time Function

---

**Syntax** Time[\$]

**Group** Time/Date

**Description** Return the current time as a **date** value.

**See Also** **Date, Now, Timer.**

**Example**

```
Sub Main
  Debug.Print Time ' example: 09:45:00 am
End Sub
```

## Timer Function

---

**Syntax** Timer

**Group** Time/Date

**Description** Return the number of seconds past midnight. (This is a real number, accurate to about 1/18th of a second.)

**See Also** **Date, Now, Time.**

**Example**

```
Sub Main
  Debug.Print Timer ' example: 45188.13
End Sub
```

## TimeSerial Function

---

**Syntax** `TimeSerial(Hour, Minute, Second)`

**Group** Time/Date

**Description** Return a **date** value.

Parameter	Description
<i>Hour</i>	This numeric value is the hour (0 to 23).
<i>Minute</i>	This numeric value is the minute (0 to 59).
<i>Second</i>	This numeric value is the second (0 to 59).

**See Also** [DateSerial](#), [DateValue](#), [TimeValue](#).

**Example**

```
Sub Main
    Debug.Print TimeSerial(13,30,0) '1:30:00 PM
End Sub
```

## TimeValue Function

---

**Syntax** `TimeValue(Date$)`

**Group** Math

**Description** Return the time part of date encoded as a string value.

Parameter	Description
<i>Date\$</i>	Convert this string value to the time part of date it represents.

**See Also** [DateSerial](#), [DateValue](#), [TimeSerial](#).

**Example**

```
Sub Main
    Debug.Print TimeValue("1/1/2000 12:00:01 AM")
    '12:00:01 AM
End Sub
```

## Trim\$ Function

---

**Syntax** `Trim[$](S$)`

**Group** String

**Description** Return the string with *S\$*'s leading and trailing spaces removed.

Parameter	Description
-----------	-------------

S\$ Copy this string without the leading or trailing spaces. If this value is **Null** then **Null** is returned.

---

**See Also** LTrim\$( ), RTrim\$( ).

**Example**

```
Sub Main
    Debug.Print ".";Trim$(" x ");"."'".x."
End Sub
```

## True Keyword

---

**Group** Constant

**Description** A *conditional expression* is True when its value is non-zero. A function that returns True returns the value -1.

## Type Definition

---

**Syntax**

```
[ | Private | Public ] _
Type name
    elem [[dim[, ...]]] As type
    [...]
End Type
```

**Group** Declaration

**Description** Define a new *usertype*. Each *elem* defines an element of the type for storing data. *As type* defines the type of data that can be stored. A *user defined type variable* has a value for each *elem*. Use *.elem* to access individual element values.

Type defaults to **Public** if neither **Private** or **Public** is specified.

## Example

```
Type Employee
    FirstName As String
    LastName As String
    Title As String
    Salary As Double
End Type

Sub Main
    Dim e As Employee
    e.FirstName = "John"
    e.LastName = "Doe"
    e.Title = "President"
    e.Salary = 100000
    Debug.Print e.FirstName ' "John"
    Debug.Print e.LastName ' "Doe"
    Debug.Print e.Title ' "President"
    Debug.Print e.Salary ' 100000
End Sub
```

## TypeName Function

---

**Syntax**            TypeName[\$](var)

**Group**            Variable Info

**Description**     Return a string indicating the type of value stored in *var*.

Parameter	Description
<i>var</i>	Return a string indicating the type of value stored in this variable.

  

Result	Description
Empty	<i>Variant</i> variable is empty. It has never been assigned a value.
Null	<i>Variant</i> variable is null.
Integer	Variable contains an <b>integer</b> value.
Long	Variable contains a <b>long</b> value.
Single	Variable contains a <b>single</b> value.
Double	Variable contains a <b>double</b> value.
Currency	Variable contains a <b>currency</b> value.
Date	Variable contains a <b>date</b> value.
String	Variable contains a <b>string</b> value.
Object	Variable contains an <b>object</b> reference that is not Nothing. (An object may return a type name specific to that type of object.)
Nothing	Variable contains an <b>object</b> reference that is Nothing.
Error	Variable contains a error code value.
Boolean	Variable contains a <b>boolean</b> value.

Variant	Variable contains a variant value. (Only used for arrays of variants.)
Unknown	Variable contains a non-OLE Automation object reference.
Byte	Variable contains a <b>byte</b> value.
()	Variable contains an array value. The TypeName of the element followed by ().

---

**See Also**

**VarType.**

**Example**

```
Sub Main
  Dim X As Variant
  Debug.Print TypeName(X) ' "Empty"
  X = 1
  Debug.Print TypeName(X) ' "Integer"
  X = 100000
  Debug.Print TypeName(X) ' "Long"
  X = 1.1
  Debug.Print TypeName(X) ' "Double"
  X = "A"
  Debug.Print TypeName(X) ' "String"
  Set X = CreateObject("Word.Basic")
  Debug.Print TypeName(X) ' "Object"
  X = Array(0,1,2)
  Debug.Print TypeName(X) ' "Variant()"
End Sub
```

## UBound Function

---

**Syntax**            UBound(*arrayvar*[, *dimension*])

**Group**             Variable Info

**Description**      Return the highest index.

Parameter	Description
<i>arrayvar</i>	Return the highest index for this array variable.
<i>dimension</i>	Return the highest index for this dimension of <i>arrayvar</i> . If this is omitted then return the highest index for the first dimension.

**See Also**

**LBound().**

**Example**

```
Sub Main
  Dim A(3,6)
  Debug.Print UBound(A) ' 3
  Debug.Print UBound(A,1) ' 3
  Debug.Print UBound(A,2) ' 6
End Sub
```

## UCase\$ Function

---

**Syntax**            UCase[\$](S\$)  
**Group**             String  
**Description**        Return a string from S\$ where all the lowercase letters have been uppercased.

Parameter	Description
S\$	Return the string value of this after all chars have been converted to lowercase. If this value is <b>Null</b> then <b>Null</b> is returned.

**See Also**            LCase\$( ), StrComp( ), StrConv\$( ).

**Example**

```
Sub Main
    Debug.Print UCase$( "Hello" ) ' "HELLO"
End Sub
```

## Unlock Instruction

---

**Syntax**            Unlock StreamNum  
                     -or-  
                     Unlock StreamNum, RecordNum  
                     -or-  
                     Unlock StreamNum, [start] To end

**Group**             File

**Description**        Form 1: Unlock all of *StreamNum*.

Form 2: Unlock a record (or byte) of *StreamNum*.

Form 3: Unlock a range of records (or bytes) of *StreamNum*. If *start* is omitted then unlock starting at the first record (or byte).

Note: For sequential files (Input, Output and Append) unlock always affects the entire file.

Parameter	Description
<i>StreamNum</i>	Streams 1 through 255 are private to each macro. Streams 256 through 511 are shared by all macros.
<i>RecordNum</i>	For Random mode files this is the record number. The first record is 1. Otherwise, it is the byte position. The first byte is 1.
<i>start</i>	First record (or byte) in the range.
<i>end</i>	Last record (or byte) in the range.

**See Also**            **Lock, Open.**

**Example**

```
Sub Main
  Dim V As Variant
  Open "SAVE_V.DAT" For Binary As #1
  Lock #1
  Get #1, 1, V
  V = "Hello"
  Put #1, 1, V
  Unlock #1
  Close #1
End Sub
```

## UserDialog Data Type

---

**Group**            Data Type

**Description**    A *usertype* defined by **Begin Dialog** UserDialog.

## Uses Comment

---

**Syntax**            '#Uses "module" [Only:[Win16|Win32]] ...  
                      -or-  
                      '\$Include: "module"

**Description**      The Uses comment indicates that the current *macro/module* uses public and friend symbols from the *module*. The Only option indicates that the module is only loaded for that Windows platform.

Parameter	Description
<i>module</i>	Public and Friend symbols from this module are accessible. If the module name is a relative path then the path is relative to the macro/module containing the Uses comment. For example, if module "A:\B\C\D.BAS" has this uses comment: '#Uses "E.BAS" then it uses "A:\B\C\E.BAS".

**See Also**            **Class Module, Code Module, Object Module.**

**Example**

```
'Macro A.WWB
'#Uses "B.WWB"
Sub Main
    Debug.Print BFunc$("Hello") ' "HELLO"
End Sub

'Module B.WWB
Public Function BFunc$(S$)
    BFunc$ = UCase(S$)
End Sub
```

## Val Function

---

**Syntax** Val(*S\$*)

**Group** String

**Description** Return the value of the *S\$*.

Parameter	Description
<i>S\$</i>	Return the number value for this string value. A string value begins with &O is an octal number. A string value begins with &H is a hex number. Otherwise it is decimal number.

**Example**

```
Sub Main
    Debug.Print Val("-1000") '-1000
End Sub
```

## Variant Data Type

---

**Group** Data Type

**Description** An empty, numeric, currency, date, string, object, error code, null or array value.

## VarType Function

---

**Syntax** VarType(*var*)

**Group** Variable Info

**Description** Return a number indicating the type of value stored in *var*.

Parameter	Description
-----------	-------------

*var* Return a number indicating the type of value stored in this variable.

---

<b>Result</b>	<b>Value</b>	<b>Description</b>
vbEmpty	0	<i>Variant</i> variable is empty. It has never been assigned a value.
vbNull	1	<i>Variant</i> variable is null.
vbInteger	2	Variable contains an <b>integer</b> value.
vbLong	3	Variable contains a <b>long</b> value.
vbSingle	4	Variable contains a <b>single</b> value.
vbDouble	5	Variable contains a <b>double</b> value.
vbCurrency	6	Variable contains a <b>currency</b> value.
vbDate	7	Variable contains a <b>date</b> value.
vbString	8	Variable contains a <b>string</b> value.
vbObject	9	Variable contains an <b>object</b> reference.
vbError	10	Variable contains a error code value.
vbBoolean	11	Variable contains a <b>boolean</b> value.
vbVariant	12	Variable contains a variant value. (Only used for arrays of variants.)
vbDataObject	13	Variable contains a non-OLE Automation object reference.
vbDecimal	14	Variable contains a 12 byte fixed precision real.
vbByte	17	Variable contains a <b>byte</b> value.
+vbArray	8192	Variable contains an array value. Use VarType( ) And 255 to get the type of element stored in the array.

---

## See Also

## Example

### TypeName.

```
Sub Main
  Dim X As Variant
  Debug.Print VarType(X) ' 0
  X = 1
  Debug.Print VarType(X) ' 2
  X = 100000
  Debug.Print VarType(X) ' 3
  X = 1.1
  Debug.Print VarType(X) ' 5
  X = "A"
  Debug.Print VarType(X) ' 8
  Set X = CreateObject("Word.Basic")
  Debug.Print VarType(X) ' 9
  X = Array(0,1,2)
  Debug.Print VarType(X) ' 8204 (8192+12)
End Sub
```

## Wait Instruction

---

**Syntax**            `Wait Delay`

**Group**            `Miscellaneous`

**Description**      `Wait` for *Delay* seconds.

**Example**           `Sub Main`  
                      `Wait 5 ' wait for 5 seconds`  
                      `End Sub`

## Weekday Function

---

**Syntax**            `Weekday(dateexpr)`

**Group**            `Time/Date`

**Description**      Return the weekday.

- `vbSunday` (1) - Sunday
- `vbMonday` (2) - Monday
- `vbTuesday` (3) - Tuesday
- `vbWednesday` (4) - Wednesday
- `vbThursday` (5) - Thursday
- `vbFriday` (6) - Friday
- `vbSaturday` (7) - Saturday

---

Parameter	Description
<i>dateexpr</i>	Return the weekday for this date value. If this value is <b>Null</b> then <b>Null</b> is returned.

---

**See Also**            `Date()`, `Day()`, `Month()`, `Year()`.

**Example**            `Sub Main`  
                      `Debug.Print Weekday(#1/1/1900#) ' 2`  
                      `End Sub`

## While Statement

---

**Syntax**            `While condexpr`  
                      `statements`  
                      `Wend`

<b>Group</b>	Flow Control
<b>Description</b>	Execute <i>statements</i> while <i>condexpr</i> is <b>True</b> .
<b>See Also</b>	<b>Do, For, For Each, Exit While.</b>
<b>Example</b>	<pre> Sub Main   I = 2   While I &lt; 10     I = I*2   Wend   Debug.Print I ' 16 End Sub </pre>

## Win16 Keyword

---

<b>Group</b>	Constant
<b>Description</b>	<b>True</b> if running in 16 bits. <b>False</b> if running in 32 bits.

## Win32 Keyword

---

<b>Group</b>	Constant
<b>Description</b>	<b>True</b> if running in 32 bits. <b>False</b> if running in 16 bits.

## With Statement

---

<b>Syntax</b>	<pre> With <i>objexpr</i>   <i>statements</i> End With </pre>
<b>Group</b>	Object
<b>Description</b>	<i>Method</i> and <i>property</i> references may be abbreviated inside a With block. Use <i>.method</i> or <i>.property</i> to access the object specified by the With <i>objexpr</i> .

**Example**

```

Sub Main
  Dim App As Object
  Set App =
  CreateObject("WinWrap.CppDemoApplication")
  With App
    .Move 20,30 ' move icon to 20,30
  End With
End Sub

```

## WithEvents Definition

---

**Syntax** `[Dim | Private | Public] _  
WithEvents name As objtype[, ...]`

**Group** Declaration

**Description** Dimensioning a module level variable `WithEvents` allows the macro to implement event handling **Subs**. The variable's `As` type must be a type from a referenced type library (or language extension) which implements events.

**Remarks** This keyword is supported by the single DLL IDE/interpreter (aka the Enterprise edition). It is not supported by the interpreter implemented in `WW_CU516.DLL` or `WW_CU532.DLL`.

**See Also** **Dim, Private, Public.**

**Example**

```

Dim WithEvents X As Thing

Sub Main
  Set X = New Thing
  X.DoIt ' DoIt method raises DoingIt event
End Sub

Private Sub X_DoingIt
  Debug.Print "X.DoingIt event"
End Sub

```

## Write Instruction

---

**Syntax** `Write #StreamNum, expr[, ...]`

**Group** File

**Description** Write's *expr*(s) to *StreamNum*. String values are quoted. Null values are written as `#NULL#`. Boolean values are written as

#FALSE# or #TRUE#. Date values are written as #date#. Error codes are written as #ERROR number#.

**See Also**

**Input, Line Input, Print.**

**Example**

```
Sub Main
  A = 1
  B = 2
  C$ = "Hello"
  Open "XXX" For Output As #1
  Write #1,A,B,C$
  Close #1
End Sub
```

## Year Function

---

**Syntax**            *Year*(*dateexpr*)

**Group**            Time/Date

**Description**     Return the year.

---

Parameter	Description
<i>dateexpr</i>	Return the year for this date value. If this value is <b>Null</b> then <b>Null</b> is returned.

---

**See Also**

**Date(), Day(), Month(), Weekday().**

**Example**

```
Sub Main
  Debug.Print Year(#1/1/1900#) ' 1900
End Sub
```

## Objects Overview

---

OLE Automation provides access to objects in other applications. Each object supports a particular set of *methods* and *properties*. Each method/property has zero or more parameters. Parameters may be optional, in which case the parameter can be specified by using `name := value`.

- `objexpr.method [expr][, ...] [param := expr][,...]`  
Call *method* for *objexpr*.
- `objexpr.method([expr][, ...] [param := expr][,...])`  
Return the value of *method* for *objexpr*.
- `objexpr.property([expr][, ...] [param := expr][,...])`  
Return the value of *property* for *objexpr*.
- `objexpr([expr][, ...] [param := expr][,...])`  
Return the default value for the *objexpr*.
- `objexpr.property([expr][, ...]) = expr`  
Assign the value of *property* for *objexpr*.
- `objexpr([expr][, ...]) = expr`  
Assign the default value for the *objexpr*.
- `Set objexpr.property([expr][, ...]) = objexpr`  
Set the object reference of *property* for *objexpr*.

Note: `objexpr!name` is short hand for `objexpr.defaultproperty("name")`. Use `objexpr![name]` if name contains any characters that are not allowed in an identifier.



## Error List

---

The following table lists all error codes with the associated error text.

<b>Error</b>	<b>Description</b>
10000	Execution interrupted.
10001	Out of memory.
10008	Invalid '#Uses "module" comment.
10009	Invalid '#Uses module dependency.
10010	Macro is already running.
10011	Can't allocate memory to macro/module.
10012	Macro/module has syntax errors.
10013	Macro/module does not exist.
10014	Another macro is paused and can't continue at this time.
10017	No macro is currently active.
10018	Sub/Function does not exist.
10019	Wrong number of parameters.
10021	Can't allocate large array.
10022	Array is not dimensioned.
10023	Array index out of range.
10024	Array lower bound is larger than upper bound.
10025	Array has a different number of indexes.
10030	User dialog has not been defined.
10031	User pressed cancel.
10032	User dialog item id is out of range.
10033	No UserDialog is currently displayed.
10034	Current UserDialog is inaccessible.
10035	Wrong with, don't GoTo into or out of With blocks.
10040	Module could not be loaded.
10041	Function not found in module.
10048	File not opened with read access.
10049	File not opened with write access.
10050	Record length exceeded.
10051	Could not open file.
10052	File is not open.
10053	Attempt to read past end-of-file.
10054	Expecting a stream number in the range 1 to 511.
10055	Input does not match var type.
10056	Expecting a length in the range 1 to 32767.
10057	Stream number is already open.
10058	File opened in the wrong mode for this operation.
10059	Error occurred during file operation.
10060	Expression has an invalid floating point operation.

10061	Divide by zero.
10062	Overflow.
10063	Expression underflowed minimum representation.
10064	Expression loss of precision in representation.
10069	String value is not a valid number.
10071	Resume can only be used in an On Error handler.
10075	Null value can't be used here.
10080	Type mismatch.
10081	Type mismatch for parameter #1.
10082	Type mismatch for parameter #2.
10083	Type mismatch for parameter #3.
10084	Type mismatch for parameter #4.
10085	Type mismatch for parameter #5.
10086	Type mismatch for parameter #6.
10087	Type mismatch for parameter #7.
10088	Type mismatch for parameter #8.
10089	Type mismatch for parameter #9.
10090	OLE Automation error.
10091	OLE Automation: no such property or method.
10092	OLE Automation: server cannot create object.
10093	OLE Automation: server cannot load file.
10094	OLE Automation: Object var is 'Nothing'.
10095	OLE Automation: server could not be found.
10096	OLE Automation: no object currently active.
10097	OLE Automation: wrong number of parameters.
10098	OLE Automation: bad index.
10099	OLE Automation: no such named parameter.
10100	Directory could not be found.
10101	File could not be killed.
10102	Directory could not be created.
10103	File could not be renamed.
10104	Directory could not be removed.
10105	Drive not found.
10106	Source file could not be opened.
10107	Destination file could not be created.
10108	Source file could not be completely read.
10109	Destination file could not be completely written.
10110	Missing close brace '}'.
10111	Invalid key name.
10112	Missing close paren ')'.
10113	Missing close bracket ']'.
10114	Missing comma ','.
10115	Missing semi-colon ';'.
10116	SendKeys couldn't install the Windows journal playback hook.

10119	String too long (too many keys).
10120	Window could not be found.
10130	DDE is not available.
10131	Too many simultaneous DDE conversations.
10132	Invalid channel number.
10133	DDE operation did not complete in time.
10134	DDE server died.
10135	DDE operation failed.
10140	Can't access the clipboard.
10150	Window style must be in the range from 1 to 9.
10151	Shell failed.
10160	Declare is not implemented.
10200	Basic is halted due to an unrecoverable error condition.
10201	Basic is busy and can't provide the requested service.
10202	Basic call failed.
10203	Handler property: prototype specification is invalid.
10204	Handler is already in use.

---



## Terms

---

**arglist** [ | *expr* | *param:=expr* ][, ...]

A list of zero or more *exprs* that are assigned to the parameters of the *procedure*.

- A positional parameter may be skipped by omitting the expression. Only optional parameters may be skipped.
- Positional parameter assignment is done with *expr*. Each parameter is assigned in turn. By name parameter assignment may follow.
- By name parameter assignment is done with *param:=expr*. All following parameters must be assigned by name.

**arrayvar** A variable that holds an array of values. A *Variant* variable can hold an array. Dynamic arrays can be **ReDimensioned**.

**As [New] type** **Dim**, **Private**, **Public** and **Static** statements may declare variable types using *As type* or *As New objtype*. A variable declared using *As New objtype* is automatically created prior to use, if the variable is **Nothing**.

**As type** Variable and parameter types, as well as, function and property results may be specified using *As type*: **Boolean**, **Byte**, **Currency**, **Date**, **Double**, **Integer**, **Long**, **Object**, **PortInt**, **Single**, **String**, **String\*n**, **UserDialog**, **Variant**, *objtype*, *userenum*, *usertype*.

**attribute** A file attribute is zero or more of the following values added together.

Attribute	Value	Description
vbNormal	0	Normal file.
vbReadOnly	1	Read-only file.
vbHidden	2	Hidden file.
vbSystem	4	System file.
vbVolume	8	Volume label.
vbDirectory	16	MS-DOS directory.
vbArchive	32	File has changes since last backup.

**big-endian** Multiple byte data values (not strings) are stored with the highest order byte first. For example, the long integer &H01020304 is stored as this sequence of four bytes: &H01, &H02, &H03 and &H04. A Binary or Random file written using **Put** uses *little-endian* format so that it can be read using **Get** on any machine.

(Big-endian machines, like the Power-PC, reverse the bytes as they are read by **Get** or written by **Put**.)

- charlist** A group of one or more characters enclosed by [ ] as part of **Like** operator's right string expression.
- This list contains single characters and/or character ranges which describe the characters in the list.
  - A range of characters is indicated with a hyphen (-) between two characters. The first character must be ordinally less than or equal to the second character.
  - Special pattern characters like ?, \*, # and [ can be matched as literal characters.
  - The ] character can not be part of charlist, but it can be part of the pattern outside the charlist.
- condexpr** An expression that returns a numeric result. If the result is zero then the conditional is **False**. If the result is non-zero then the conditional is **True**.
- ```
0 'false
-1 'true
X > 20 'true if X is greater than 20
S$ = "hello" 'true if S$ equals "hello"
```
- dateexpr** An expression that returns a **date** result. Use #literal-date# to express a date value.
- ```
#1/1/2000# ' Jan 1, 2000
Now+7 ' seven days from now
DateSerial(Year(Now)+1,Month(Now),Day(Now))
' one year from now
```
- dialogfunc** A dialog function executes while a **UserDialog** is visible.
- dim** [*lower* To] *upper*
- Array dimension. If *lower* is omitted then the lower bound is zero or one depending on the **Option** Base setting. (The lower bound of an array element in a **Type** definition is not affected by the Option Base setting.) *upper* must be at least as big as *lower*.
- ```
Dim A(100 To 200) '101 values
```
- Note: For **ReDim** the *lower* and *upper* may be any valid expression. Otherwise, *lower* and *upper* must be constant expressions.
- dlgvar** A dialog variable holds values for fields in the dialog. Dialog variables are declared using **Dim** dlgvar As **UserDialog**.

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>expr</b>          | An expression that returns the appropriate result.                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>field</b>         | Use <code>.field</code> to access individual fields in a dialog variable.<br><pre>dlg.Name\$ dlg.ZipCode</pre>                                                                                                                                                                                                                                                                                                                                                      |
| <b>instruction</b>   | A single command.<br><pre>Beep Debug.Print "Hello" Today = Date</pre> <p>Multiple instructions may be used instead of a single instruction by separating the single instructions with colons.</p> <pre>X = 1:Debug.Print X If X = 1 Then Debug.Print "X=";X:Stop Beep ' must resume from Stop to get to here</pre>                                                                                                                                                  |
| <b>label</b>         | An identifier that <i>names</i> a statement. Identifiers start with a letter. Following chars may be a letter, an underscore or a digit.                                                                                                                                                                                                                                                                                                                            |
| <b>little-endian</b> | Multiple byte data values (not strings) are stored with the lowest order byte first. For example, the long integer &H01020304 is stored as this sequence of four bytes: &H04, &H03, &H02 and &H01. A Binary or Random file written using <b>Put</b> uses little-endian format so that it can be read using <b>Get</b> on any machine. ( <i>Big-endian</i> machines, like the Power-PC, reverse the bytes as they are read by <b>Get</b> or written by <b>Put</b> .) |
| <b>macro</b>         | A macro is like an application. Execution starts at the macro's <b>Sub Main</b> .                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>method</b>        | An object provides methods and <i>properties</i> . Methods can be called as subs (the return value is ignored), or used as functions (the return value is used).<br><br>If the method name contains characters that are not legal in a <i>name</i> , surround the method name with [].<br><pre>App.[Title\$]</pre>                                                                                                                                                  |
| <b>module</b>        | A file with <b>public</b> symbols that are accessible by other modules/ <i>macros</i> via the <b>#Uses</b> comment. <ul style="list-style-type: none"> <li>• A module is loaded on demand.</li> <li>• A <b>code module</b> is a code library.</li> <li>• An <b>object module</b> or <b>class module</b> implements an OLE automation object.</li> </ul>                                                                                                             |

- A module may also access other modules with its own #Uses comments.

|                |                                                                                                                                                                                                           |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>name</b>    | An identifier that names a variable or a user defined <i>procedure</i> . Identifiers start with a letter. Following chars may be a letter, an underscore or a digit.<br>Count<br>DaysTill2000<br>Get_Data |
| <b>num</b>     | An expression that returns a numeric result. Use &O to express an octal number. Use &H to express a hex number.                                                                                           |
| <b>numvar</b>  | A variable that holds one numeric value. The name of a numeric variable may be followed by the appropriate <i>type</i> char.                                                                              |
| <b>objexpr</b> | A expression that returns a reference to an object or <i>module</i> .<br>CreateObject("WinWrap.CDemoApplication")                                                                                         |
| <b>objtype</b> | A specific OLE type defined by your application, another application or by an <b>object module</b> or <b>class module</b> .                                                                               |
| <b>objvar</b>  | A variable that holds a <i>objexpr</i> which references an object. Object variables are declared using As <i>Object</i> in a <b>Dim</b> , <b>Private</b> or <b>Public</b> statement.                      |
| <b>param</b>   | [ [Optional] [   ByVal   ByRef ]   ParamArray ] <i>param</i> [ <i>type</i> ][ ( ) ]<br>[As <i>type</i> ] [ = <i>defaultvalue</i> ]                                                                        |

The *param* receives the value of the associated expression in the **Declare**, **Sub**, **Function** or **Property** call. (See *arglist*.)

- An Optional *param* may be omitted from the call. It may also have a *defaultvalue*. The parameter receives the defaultvalue if a value is not specified by the call. If the defaultvalue is omitted, the parameter is a **VARIANT** and no value is specified in the call then **IsMissing** will return **True**.
- All parameters following an Optional parameter must also be Optional.
- ParamArray may be used on the final *param*. It must be an array of **VARIANT** type. It must not follow any Optional parameters. The ParamArray receives all the expressions at the end of the call as an array. If **LBound(param) > UBound(param)** then the ParamArray didn't receive any expressions.

- If the *param* is not ByVal and the expression is merely a variable then the *param* is a reference to that variable (ByRef). (Changing *param* changes the variable.) Otherwise, the parameter variable is local to the *procedure*, so changing its value does not affect the caller.
- Use *param()* to specify an array parameter. An array parameter must be referenced and can not be passed by value. The bounds of the parameter array are available via **LBound()** and **UBound()**.

**precedence**

When several operators are used in an expression, each operator is evaluated in a predetermined order. Operators are evaluated in this order:

- ^ (power)
- - (negate)
- \* (multiply), / (divide)
- \ (integer divide)
- Mod (integer remainder)
- + (add), - (difference)
- & (string concatenate)
- = (equal), <> (not equal), < (less than) > (greater than), <= (less than or equal to), >= (greater than or equal to), **Like**, (string similarity) **Is** (object equivalence)
- Not (logical bitwise invert)
- And (logical bitwise and)
- Or (logical or bitwise or)
- Xor (logical or bitwise exclusive-or)
- Eqv (logical or bitwise equivalence)
- Imp (logical or bitwise implication)

Operators shown on the same line are evaluated from left to right.

**procedure**

A **subroutine**, **function** or **property**.

**property**

An object provides *methods* and properties. Properties may be used as values (like a function call) or changed (using assignment syntax).

If the property name contains characters that are not legal in a *name*, surround the property name with [].

App.[Title\$]

**statement** Zero or more *instructions*. A statement is at least one line long. **Begin Dialog**, **Do**, **For**, **If** (multiline), **Select Case**, **While** and **With** statements are always more than one line long. A single line statement continues on the next line if it ends a line with a space and an underscore ' \_'.

```
S$ = "This long string is easier to read, " + _
    "if it is broken across two lines."
Debug.Print S$
```

**str** An expression that returns a string result.

```
"Hello"
S$
S$ + " Goodbye"
S$ & " Goodbye"
Mid$(S$, 2)
```

**strarray** A variable that holds an array of string values. The name of a string variable may be followed by a \$.

**strvar** A variable that holds one string value. The name of a string variable may be followed by a \$.

```
FirstName$
```

**type** Variable and parameter types, as well as, function and property results may be specified using a type character as the last character in their name.

| Type char | As Type  |
|-----------|----------|
| %         | Integer  |
| ?         | PortInt  |
| &         | Long     |
| !         | Single   |
| #         | Double   |
| @         | Currency |
| \$        | String   |

**userenum** User defined enums are defined with **Enum**.

**usertype** User defined types are defined with **Type**.

**usertypevar** A user defined type variable holds values for elements of the user defined type. User defined types are defined using **Type**.

- Declare with **Dim**, **Private**, **Public** or **Static**.
- Declare as a parameter of **Sub**, **Function** or **Property** definition.

|                   |                                                                                                                         |
|-------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>var</b>        | A variable holds either a string, a numeric value or an array of values depending on its type.                          |
| <b>variantvar</b> | A variant variable can hold any type of value (except <b>String*n</b> or <i>usertypevar</i> ). or it can hold an array. |

