LANGUAGE REFERENCE

CONTENTS

Cypress Enable Scripting Language Elements

In this Section, the general elements of the Enable language are described. Enable scripts can include comments, statements, various representations of numbers, 11 variable data types including user defined types, and multiple flow of control structures. Enable is also extendable by calling external DLL's or calling functions back in the applications .exe file.

Comments

Comments are non-executed lines of code which are included for the benefit of the programmer. Comments can be included virtually anywhere in a script. Any text following an apostrophe or the word Rem is ignored by Enable. Rem and all other keywords and most names in Enable are not case sensitive

```
'             This whole line is a comment
rem           This whole line is a comment
REM           This whole line is a comment
Rem           This whole line is a comment
```

Comments can also be included on the same line as executed code:

```
MsgBox Msg     ' Display message.
```

Everything after the apostrophe is a comment.

Statements:

In Enable there is no statement terminator. More than one statement can be put on a line if they are separated by a colon.

```
X.AddPoint( 25, 100) :  X.AddPoint(  0,  75)
```

Which is equivalent to:

```
X.AddPoint( 25, 100)
X.AddPoint(  0,  75)
```

Line Continuation Character:

The underscore is the line continuation character in Enable. There must be a space before and after the line continuation character.

```
X.AddPoint _
( 25, 100)
```

Numbers

Cypress Enable supports three representations of numbers: Decimal, Octal and Hexadecimal. Most of the numbers used in this manual are decimal or base 10 numbers. However, if you need to use Octal (base 8) or hexadecimal (base 16) numbers simply prefix the number with &O or &H respectively.

Variable and Constant Names

Variable and Constant names must begin with a letter. They can contain the letters A to Z and a to z, the underscore "_", and the digits 0 to 9. Variable and constant names must begin with a letter, be no longer than 40 characters. and cannot be reserved words. For a table of reserved words, see the Language Overview section of this manual. One exception to this rule is that object member names and property names may be reserved words.

Variable Types

Variant

As is the case with Visual Basic, when a variable is introduced in Cypress Enable, it is not necessary to declare it first (see option explicit for an exception to this rule). When a variable is used but not declared then it is implicitly declared as a **variant** data type. Variants can also be declared explicitly using "As Variant" as in Dim x As Variant. The variant data type is capable of storing numbers, strings, dates, and times. When using a variant you do not have to explicitly convert a variable from one data type to another. This data type conversion is handled automatically.

```
Sub Main
        Dim x           'variant variable
        x = 10
        x = x + 8
        x = "F" & x
        print x         'prints F18
End Sub
```

A variant variable can readily change its type and its internal representation can be determined by using the function **VarType**. **VarType** returns a value that corresponds to the explicit data types. See VarType in A-Z Reference for return values.

When storing numbers in variant variables the data type used is always the most compact type possible. For example, if you first assign a small number to the variant it will be stored as an integer. If you then assign your variant to a number with a fractional component it will then be stored as a double.

For doing numeric operations on a variant variable it is sometimes necessary to determine if the value stored is a valid numeric, thus avoiding an error. This can be done with the **IsNumeric** function.

Variants and Concatenation

If a string and a number are concatenated the result is a string. To be sure your concatenation works regardless of the data type involved use the **&** operator. The **&** will not perform arithmetic on your numeric values it will simply concatenate them as if they were strings.

The **IsEmpty** function can be used to find out if a variant variable has been previously assigned.

Other Data Types

The twelve data types available in Cypress Enable are shown below:

Data Types

Variable		Type Declaration	Size
Byte		Dim BVar As Byte	0 to 255
Boolean		Dim BoolVar As Boolean	True or False
String	$	Dim Str_Var As String	0 to 65,500 char
Integer	%	Dim Int_Var As Integer	2 bytes
Long	&	Dim Long_Var As Long	4 bytes
Single	!	Dim Sing_Var As Single	4 bytes
Double	#	Dim Dbl_Var As Double	8 bytes
Variant		Dim X As Any	
Currency		Dim Cvar As Currency	8 bytes
Object		Dim X As Object	4 bytes
Date		Dim D As Date	8 bytes
User Defined Types			size of each element

Scope of Variables

Cypress Enable scripts can be composed of many files and each file can have many subroutines and functions in it. Variable names can be reused even if they are contained in separate files. Variables can be local or global.

Declaration of Variables

In Cypress Enable variables are declared with the **Dim** statement. To declare a variable other than a variant the variable must be followed by **As** or appended by a type declaration character such as a % for **Integer** type.

```
Sub Main
   Dim X As Integer
   Dim Y As Double
   Dim  Name$, Age%    ' multiple declaration on one line Dim v

End Sub
```

Control Structures

Cypress Enable has complete process control functionality. The control structures available are **Do** loops, **While** loops, **For** loops, **Select Case**, **If Then** , and **If Then Else.** In addition, Cypress Enable has one branching statement: **GoTo**. The **Goto** Statement branches to the label specified in the **Goto** Statement.

```
Goto label1
       .
       .
       .

label1:
```

The program execution jumps to the part of the program that begins with the label "Label1:".

Loop Structures

Do Loops

The **Do...Loop** allows you to execute a block of statements an indefinite number of times. The variations of the **Do...Loop** are **Do While**, **Do Until**, **Do Loop While**, and **Do Loop Until**.

```
Do While|Until condition
   Statement(s)...
   [Exit Do]
   Statement(s)...
Loop

Do Until condition
Statement(s)...
Loop

Do
Statements...
Loop While condition

Do
statements...
Loop Until condition
```

Do While and **Do Until** check the condition before entering the loop, thus the block of statements inside the loop are only executed when those conditions are met. **Do Loop While** and **Do Loop Until** check

the condition after having executed the block of statements thereby guaranteeing that the block of statements is executed at least once.

While Loop

The **While...Wend** loop is similar to the **Do While** loop. The condition is checked before executing the block of statements comprising the loop.

```
While condition
        statements...
Wend
```

For ... Next Loop

The **For...Next** loop has a counter variable and repeats a block of statements a set number of times. The counter variable increases or decreases with each repetition through the loop. The counter default is one if the **Step** variation is not used.

```
For counter = beginning value To ending value [Step increment]
statements...
Next
```

If and Select Statements

The **If...Then** block has a single line and multiple line syntax. The condition of an **If** statement can be a comparison or an expression, but it must evaluate to True or False.

```
If condition Then Statements...              'single line syntax

If condition Then
        'multiple line syntax
        statements...
End If
```

The other variation on the **If** statement is the **If...Then...Else** statement. This statement should be used when there is different statement blocks to be executed depending on the condition. There is also the **If...Then...ElseIf...** variation, these can get quite long and cumbersome, at which time you should consider using the **Select** statement.

```
If condition Then
        statements...
ElseIf condition Then
        statements...
Else

End If
```

The **Select Case** statement tests the same variable for many different values. This statement tends to be easier to read, understand and

follow and should be used in place of a complicated **If...Then...ElseIf** statement.

```
Select Case variable to test
        Case 1
                statements...
        Case 2
                statements...
        Case 3
                statements...
        Case Else
                statements...
End Select
```

See Language Reference A - Z for exact syntax and code examples.

Subroutines and Functions

Naming conventions

Subroutine and Function names can contain the letters A to Z and a to z, the underscore "_" and digits 0 to 9. The only limitation is that subroutine and function names must begin with a letter, be no longer than 40 characters, and not be reserved words. For a list of reserved words, see the table of reserved words in the Language Overview section of this manual.

Cypress Enable allows script developers to create their own functions or subroutines or to make DLL calls. Subroutines are created with the syntax "Sub <subname> .... End Sub". Functions are similar "Function <funcname> As <type> ... <funcname> = <value> ... End Function." DLL functions are declared via the **Declare** statement.

ByRef and ByVal

ByRef gives other subroutines and functions the permission to make changes to variables that are passed in as parameters. The keyword ByVal denies this permission and the parameters cannot be reassigned outside their local procedure. ByRef is the Enable default and does not need to be used explicitly. Because ByRef is the default all variables passed to other functions or subroutines can be changed, the only exception to this is if you use the ByVal keyword to protect the variable or use parentheses which indicate the variable is ByVal.

If the arguments or parameters are passed with parentheses around them, you will tell Enable that you are passing them ByVal

```
SubOne var1, var2, (var3)
```

The parameter var3 in this case is passed by value and cannot be changed by the subroutine SubOne.

```
Function R( X As String, ByVal n As Integer)
```

In this example the function R is receiving two parameters X and n. The second parameter n is passed by value and the contents cannot be changed from within the function R.

In the following code samples scalar variable and user defined types are passed by reference.

Scalar Variables

```
Sub Main
    Dim x(5) As Integer
    Dim i As Integer
    for i = 0 to 5
        x(i) = i
    next i
    Print i
    Joe (i), x  ‘ The parenthesis around it turn it into an expression which
passes by value
    print "should be 6: "; x(2), i
End Sub

Sub Joe( ByRef j As Integer, ByRef y() As Integer )
    print "Joe: "; j, y(2)
    j = 345
    for i = 0 to 5
        print "i: "; i; "y(i): "; y(i)
    next i
    y(2) = 3 * y(2)
End Sub
```

Passing User Defined Types by Ref to DLL’s and Enable functions

```
' OpenFile() Structure
Type OFSTRUCT
       cBytes As String * 1
       fFixedDisk As String * 1
       nErrCode As Integer
       reserved As String * 4
       szPathName As String * 128
End Type

' OpenFile() Flags
Global Const OF_READ = &H0
Global Const OF_WRITE = &H1
Global Const OF_READWRITE = &H2
Global Const OF_SHARE_COMPAT = &H0
Global Const OF_SHARE_EXCLUSIVE = &H10
Global Const OF_SHARE_DENY_WRITE = &H20
Global Const OF_SHARE_DENY_READ = &H30
Global Const OF_SHARE_DENY_NONE = &H40
Global Const OF_PARSE = &H100
Global Const OF_DELETE = &H200
Global Const OF_VERIFY = &H400
Global Const OF_CANCEL = &H800
```

```
Global Const OF_CREATE = &H1000
Global Const OF_PROMPT = &H2000
Global Const OF_EXIST = &H4000
Global Const OF_REOPEN = &H8000

Declare Function OpenFile Lib "Kernel" (ByVal lpFileName As String,
lpReOpenBuff As OFSTRUCT, ByVal wStyle As Integer) As Integer

Sub Main
    Dim ofs As OFSTRUCT
    ' Print OF_READWRITE
    ofs.szPathName = "c:\enable\openfile.bas"
    print ofs.szPathName
    ofs.nErrCode = 5
    print ofs.nErrCode
    OpenFile "t.bas", ofs
    print ofs.szPathName
    print ofs.nErrCode
End Sub
```

Calling Procedures in DLLs

DLLs or Dynamic-link libraries are used extensively by Engineers to funtions and subroutines located there. There are two main ways that Enable can be extended, one way is to call functions and subroutines in DLLs and the other way is to call functions and subroutines located in the calling application. The mechanisms used for calling procedures in either place are similar. (See the Declare Statement for more deatils)

To declare a DLL procedure or a procedure located in your calling application place a declare statement in your declares file or outside the code area. All declarations in Enable are Global to the run and accesible by all subroutines and functions. If the procedure does not return a value, declare it as a subroutine. If the procedure does have a return value declare it as a function.

```
Declare Function GetPrivateProfileString Lib "Kernel32" (ByVal
lpApplicationName As String, ByVal _ lpKeyName As String, ByVal lpDefault As
String, ByVal lpReturnedString As String, ByVal nSize As _ Integer, ByVal
lpFileName As String) As Integer

Declare Sub InvertRect Lib “User” (ByVal hDC AS Integer, aRect As Rectangle)
```

Notice the line extension character “-“ the underscore. If a piece of code is too long to fit on one line a line extension character can be used when needed.

Once a procedure is declared, you can call it just as you would another Enable Function.

It is important to note that Enable cannot verify that you are passing correct values to a DLL procedure. If you pass incorrect values, the procedure may fail.

Passing and Returning Strings

Cypress Enable maintains variable-length strings internally as BSTRs. BSTRs are defined in the OLE header files as OLECHAR FAR *. An OLECHAR is a UNICODE character in 32-bit OLE and an ANSI character in 16-bit OLE. A BSTR can contain NULL values because a length is also maintained with the BSTR. BSTRs are also NULL terminated so they can be treated as an LPSTR. Currently this length is stored immediately prior to the string. This may change in the future, however, so you should use the OLE APIs to access the string length.

You can pass a string from Cypress Enable to a DLL in one of two ways. You can pass it "by value" (ByVal) or "by reference". When you pass a string ByVal, Cypress Enable passes a pointer to the beginning of the string data (i.e. it passes a BSTR). When a string is passed by reference, Enable passes a pointer to a pointer to the string data (i.e. it passes a BSTR *).

OLE API
SysAllocString/SysAllocStringLen
SysAllocString/SysAllocStringLen
SysFreeString
SysStringLen
SysReAllocStringLen
SysReAllocString

NOTE: The BSTR is a pointer to the string, so you don't need to dereference it.

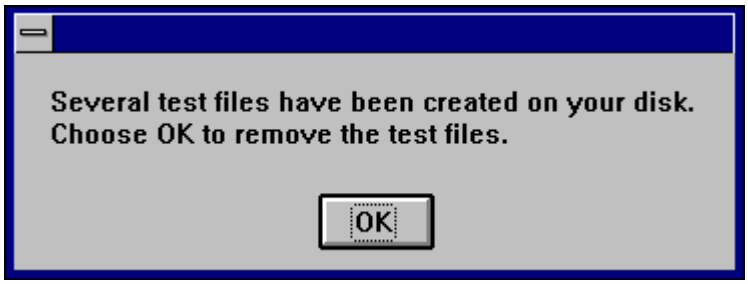
File Input/Output

Enable supports full sequential and binary file I/O.

Functions and Statements that apply to file access:
Dir, EOF, FileCopy, FileLen, Seek, Open, Close, Input, Line Input, Print and Write

```
' File I/O Examples

Sub Main
    Open "TESTFILE" For Input As #1   ' Open file.
    Do While Not EOF(1)               ' Loop until end of file.
      Line Input #1, TextLine ' Read line into variable.
        Print TextLine        ' Print to Debug window.
    Loop
    Close #1   ' Close file.

End Sub
```

```
Sub test

Open "MYFILE" For Input As #1 ' Open file for input.
Do While Not EOF(1)    ' Check for end of file.
       Line Input #1, InputData      ' Read line of data.
       MsgBox InputData
Loop
Close #1       ' Close file.

End Sub



Sub FileIO_Example()
       Dim Msg ' Declare variable.
       Call Make3Files()      ' Create data files.
       Msg = "Several test files have been created on your disk. "
       Msg = Msg & "Choose OK to remove the test files."
       MsgBox Msg
       For I = 1 To 3
              Kill "TEST" & I ' Remove data files from disk.
        Next I
End Sub

Sub Make3Files ()
       Dim I, FNum, FName     ' Declare variables.
       For I = 1 To 3
              FNum = FreeFile        ' Determine next file number.
              FName = "TEST" & FNum
              Open FName For Output As FNum ' Open file.
              Print #I, "This is test #" & I        ' Write string to file.
              Print #I, "Here is another ";  "line";  I
       Next I
       Close   ' Close all files.
End Sub
```



Arrays

Cypress Enable supports single and multi dimensional arrays. Using arrays you can refer to a series of variables by the same name each with a separate index. Arrays have upper and lower bounds. Enable allocates space for each index number in the array. Arrays should not be declared larger then necessary.

All the elements in an array have the same data type. Enable supports arrays of bytes, Booleans, longs, integers, singles, double, strings, variants and User Defined Types.

Ways to declare a fixed-size array:

- *Global array*, use the **Dim** statement outside the procedure section of a code module to declare the array.

- To create a *local* array, use the **Dim** statement inside a procedure.

Cypress Enable supports Dynamic arrays.

Declaring an array. The array name must be followed by the upper bound in parentheses. The upper bound must be an integer.

```
Dim  ArrayName (10) As Interger
Dim Sum (20) As Double
```

To create a global array, you simply use **Dim** outside the procedure:

```
Dim Counters (12) As Integer
Dim Sums (26) As Double

Sub Main () …
```

The same declarations within a procedure use **Static or Dim**:

```
Static Counters (12)  As Integer
Static Sums (22) As Double
```

The first declaration creates an array with 11 elements, with index numbers running from 0 to 10. The second creates an array with 21 elements. To change the default lower bound to 1 place an **Option Base** statement in the Declarations section of a module:

```
Option Base 1
```

Another way to specify the lower bound is to provide it explicitly (as an integer, in the range -32,768 to 32,767) using the **To** key word:

```
Dim Counters (1 To 13) As Integer
Dim Sums (100 To 126) As String
```

In the preceding declarations, the index numbers of Counters run from 1 to 13, and the index numbers of Sums run from 100 to 126.

Note: Many other versions of Basic allow you to use an array without first declaring it. Enable Basic does not allow this; you must declare an array before using it.

Loops often provide an efficient way to manipulate arrays. For example, the following **For** loop initializes all elements in the array to 5:

```
Static Counters (1 To 20) As Integer
Dim I As Integer
       For I = 1 To 20
              Counter ( I ) = 5
       Next I
…
```

MultiDimensional Arrays

Cypress Enable supports multidimensional arrays. For example the following example declares a two-dimensional array within a procedure.

Static Mat(20, 20) As Double

Either or both dimensions can be declared with explicit lower bounds.

Static Mat(1 to 10, 1 to 10) As Double

You can efficiently process a multidimensional array with the use of for loops. In the following statements the elemtents in a multidimensional array are set to a value.

```
Dim L As Integer, J As Integer
 Static TestArray(1 To 10, 1 to 10) As Double
   For L = 1 to 10
        For J = 1 to 10
              TestArray(L,J) = I * 10 + J
        Next J
   Next L
```

Arrays can be more than two dimensional. Enable does not have an arbitrary upper bound on array dimensions.

Dim ArrTest(5, 3, 2)

This declaration creates an arrray that has three dimensions with sizes 6 by 4, by 3 unless Option Base 1 is set previously in the code. The use of Option Base 1 sets the lower bound of all arrays to 1 instead of 0.

User Defined Types

Users can define their own types that are composites of other built-in or user defined types. Variables of these new composite types can be declared and then member variables of the new type can be accessed using dot notation. Only variables of user defined types that contain simple data types can be passed to DLL functions expecting ‘C’ structures.

User Defined types are created using the type statement, which must be placed outside the procedure in your Enable Code. User defined types are global. The variables that are declared as user defined types can be either global or local. User Defined Types in Enable cannot contain arrays at this time

```
Type type1
    a As Integer
    d As Double
    s As String
End Type

Type type2
    a As Integer
    o As type1
End Type

Dim type2a As type2
Dim type1a As type1

Sub TypeExample ()
    a = 5
    type1a.a = 7472
    type1a.d = 23.1415
    type1a.s = "YES"
    type2a.a = 43
    type2a.o.s = "Hello There"
    MsgBox type1a.a
    MsgBox type1a.d
    MsgBox type1a.s
    MsgBox type2a.a
    MsgBox type2a.o.s
    MsgBox a
End Sub
```

Dialog Support

Cypress Enable has support for custom dialogs. The syntax is similar to the syntax used in Microsoft Word Basic. The dialog syntax is not part of

Microsoft Visual Basic or Microsoft Visual Basic For Applications (VBA). Enable has complete support for dialogs. The type of dialogs supported are outlined below.

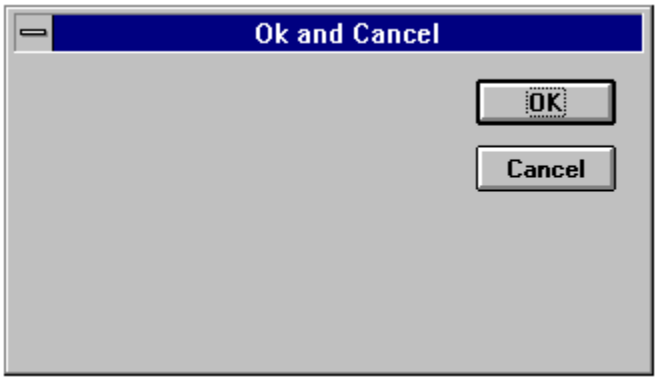
Dialog Box controls

Enable Basic supports the standard Windows dialog box controls. This section introduces the controls available for custom dialog boxes and provides guidelines for using them.

The Dialog Box syntax begins with the statement “Begin Dialog”. The first two parameters of this statement are optional. If they are left off the dialog will automatically be centered.

```
Begin Dialog DialogName1 240, 184, "Test Dialog"
Begin Dialog DialogName1 60, 60,240, 184, "Test Dialog"
```

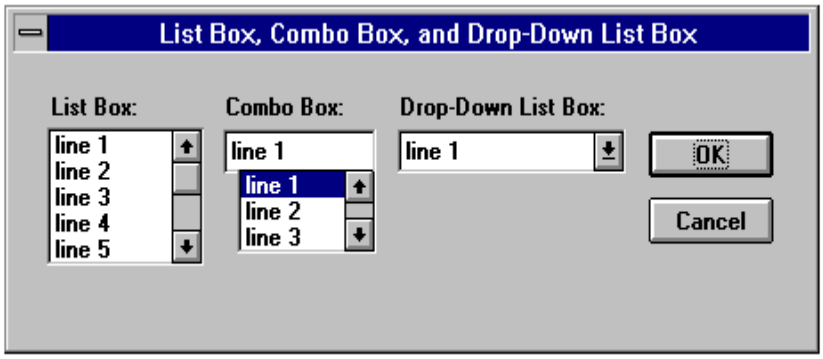
OK and Cancel Buttons




```
Sub Main
        Begin Dialog ButtonSample 16,32,180,96,"OK and Cancel"
        OKButton 132,8,40,14
        CancelButton 132,28,40,14
End Dialog
                Dim Dlg1 As ButtonSample
                Button = Dialog (Dlg1)
End Sub
```

Every custom dialog box must contain at least one “command” button - a OK button or a Cancel button. Enable includes separate dialog box definition statements for each of these two types of buttons.

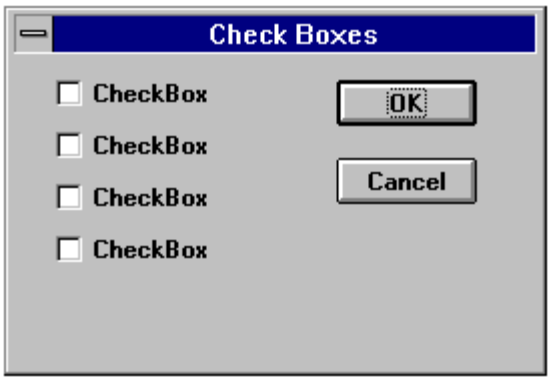
List Boxes, Combo Boxes and Drop-down List Boxes




```
Sub Main
            Dim MyList$ (5)
            MyList (0) = "line Item 1"
            MyList (1) = "line Item 2"
            MyList (2) = "line Item 3"
            MyList (3) = "line Item 4"
            MyList (4) = "line Item 5"
            MyList (5) = "line Item 6"

Begin Dialog BoxSample 16,35,256,89,"List Box, Combo Box, and Drop-Down List Box"
            OKButton 204,24,40,14
            CancelButton 204,44,40,14
            ListBox 12,24,48,40, MyList$( ),.Lstbox
            DropListBox 124,24,72,40, MyList$( ),.DrpList
            ComboBox 68,24,48,40, MyList$( ),.CmboBox
            Text 12,12,32,8,"List Box:"
            Text 124,12,68,8,"Drop-Down List Box:"
            Text 68,12,44,8,"Combo Box:"
End Dialog
            Dim Dlg1 As BoxSample
            Button = Dialog ( Dlg1 )

End Sub
```

You can use a list box, drop-down list box, or combo box to present a list of items from which the user can select. A drop-down list box saves space (it can drop down to cover other dialog box controls temporarily). A combo box allows the user either to select an item from the list or type in a new item. The items displayed in a list box, drop-down list box, or combo box are stored in an array that is defined before the instructions that define the dialog box.

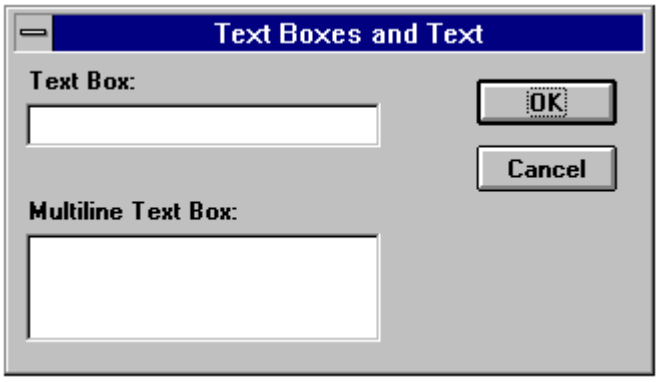
Check Boxes




```
Sub Main
        Begin Dialog CheckSample15,32,149,96,"Check Boxes"
        OKButton 92,8,40,14
        CancelButton 92,32,40,14
        CheckBox 12,8,45,8,"CheckBox",.CheckBox1
        CheckBox 12,24,45,8,"CheckBox",.CheckBox2
        CheckBox 12,40,45,8,"CheckBox",.CheckBox3
        CheckBox 12,56,45,8,"CheckBox",.CheckBox4
End Dialog
        Dim Dlg1 As CheckSample
        Button = Dialog ( Dlg1 )
End Sub
```

You use a check box to make a “yes or no” or “on or off” choice. for example, you could use a check box to display or hide a toolbar in your application.

Text Boxes and Text




```
Sub Main
        Begin Dialog TextBoxSample 16,30,180,96,"Text Boxes and Text"
        OKButton 132,20,40,14
        CancelButton 132,44,40,14
        Text 8,8,32,8,"Text Box:"
        TextBox 8,20,100,12,.TextBox1
        Text 8,44,84,8,"Multiline Text Box:"
        TextBox 8,56,100,32,.TextBox2
End Dialog
        Dim Dlg1 As TextBoxSample
        Button = Dialog ( Dlg1 )
```

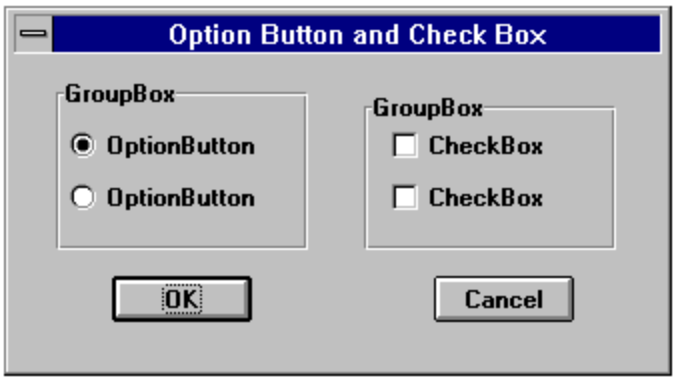
```
End Sub
```

a text box control is a box in which the user can enter text while the dialog box is displayed. By default, a text box holds a single line of text. Enable support single and multi-line text boxes. The last parameter of the textbox function contains a variable to set the textbox style.

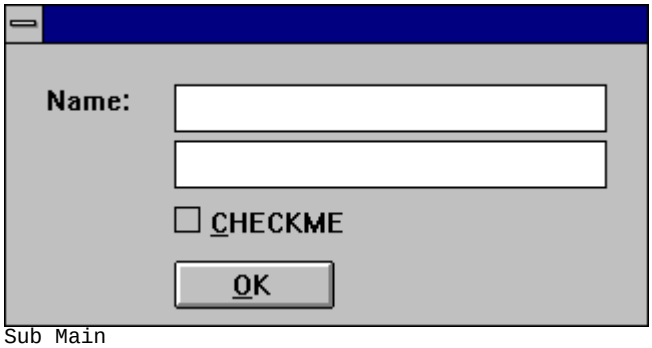
```
'=========================================================
' This sample shows how to implement a multiline textbox
'=========================================================
Const ES_LEFT             = &h0000&  'Try these different styles or-ed
together
Const ES_CENTER           = &h0001&  ' as the last parameter of Textbox the
change
Const ES_RIGHT            = &h0002&  ' the text box style.
Const ES_MULTILINE        = &h0004&  ' A 1 in the last parameter position
defaults to
Const ES_UPPERCASE        = &h0008&  ' A multiline, Wantreturn, AutoVScroll
testbox.
Const ES_LOWERCASE        = &h0010&
Const ES_PASSWORD         = &h0020&
Const ES_AUTOVSCROLL      = &h0040&
Const ES_AUTOHSCROLL      = &h0080&
Const ES_NOHIDESEL        = &h0100&
Const ES_OEMCONVERT       = &h0400&
Const ES_READONLY         = &h0800&
Const ES_WANTRETURN       = &h1000&
Const ES_NUMBER           = &h2000&

Sub Multiline
    Begin Dialog DialogType 60, 60, 140, 185, "Multiline text Dialog",
.DlgFunc
        TextBox 10, 10, 120, 150, .joe, ES_MULTILINE Or ES_AUTOVSCROLL Or
ES_WANTRETURN            ' Indicates multiline TextBox
         'TextBox 10, 10, 120, 150, .joe, 1 ' indicates multi-line textbox
         CancelButton 25, 168, 40, 12
        OKButton 75, 168, 40, 12
    End Dialog
    Dim Dlg1 As DialogType
    Dlg1.joe  = "The quick brown fox jumped over the lazy dog"
    ' Dialog returns -1 for OK, 0 for Cancel
    button = Dialog( Dlg1 )
    'MsgBox "button: " & button
    If button = 0 Then Exit Sub

    MsgBox "TextBox: "& Dlg1.joe
End Sub
```

Option Buttons and Group Boxes

You can have option buttons to allow the user to choose one option from several. Typically, you would use a group box to surround a group of option buttons, but you can also use a group box to set off a group of check boxes or any related group of controls.




```
Begin Dialog GroupSample 31,32,185,96,"Option Button and Check Box"
       OKButton 28,68,40,14
       CancelButton 120,68,40,14
       GroupBox 12,8,72,52,"GroupBox",.GroupBox1
       GroupBox 100,12,72,48,"GroupBox",.GroupBox2
       OptionGroup .OptionGroup1
       OptionButton 16,24,54,8,"OptionButton",.OptionButton1
       OptionButton 16,40,54,8,"OptionButton",.OptionButton2
       CheckBox 108,24,45,8,"CheckBox",.CheckBox1
       CheckBox 108,40,45,8,"CheckBox",.CheckBox2
End Dialog
       Dim Dlg1 As GroupSample
       Button = Dialog (Dlg1)
End Sub
```




```
Sub Main
    Begin Dialog DialogName1 60, 60, 160, 70
    TEXT 10, 10, 28, 12, "Name:"
    TEXTBOX 42, 10, 108, 12, .nameStr
    TEXTBOX 42, 24, 108, 12, .descStr
    CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
    OKBUTTON 42, 54, 40, 12
    End Dialog
    Dim Dlg1 As DialogName1
    Dialog Dlg1

    MsgBox Dlg1.nameStr
    MsgBox Dlg1.descStr
    MsgBox Dlg1.checkInt
End Sub
```

The Dialog Function

Cypress Enable supports the dialog function. This function is a user-defined function that can be called while a custom dialog box is displayed. The dialog

function makes nested dialog boxes possible and receives messages from the dialog box while it is still active.

When the function dialog() is called in Enable it displays the dialog box, and calls the dialog function for that dialog. Enable calls the dialog function to see if there are any commands to execute. Typical commands that might be used are disabling or hiding a control. By default all dialog box controls are enabled. If you want a control to be hidden you must explicitly make it disabled during initialization. After initialization Enable displays the dialog box. When an action is taken by the user Enable calls the dialog function and passes values to the function that indicate the kind of action to take and the control that was acted upon.

The dialog box and its function are connected in the dialog definition. A "function name" argument is added to the Begin Dialog instruction, and matches the name of the dialog function located in your Enable program.

```
Begin Dialog UserDialog1 60,60, 260, 188, "3", .Enable
```

The Dialog Box Controls

A dialog function needs an identifier for each dialog box control that it acts on. The dialog function uses string identifiers. String identifiers are the same as the identifiers used in the dialog record.

```
CheckBox 8, 56, 203, 16, "Check to display controls",. Chk1
```

The control's identifier and label are different. An identifier begins with a period and is the last parameter in a dialog box control instruction. In the sample code above "Check to display controls" is the label and .chk1 is the identifier.

The Dialog Function Syntax

The syntax for the dialog function is as follows:

```
Function FunctionName( ControlID$, Action%, SuppValue%)
       Statement Block
       FunctionName = ReturnValue
End Function
```

All parameters in the dialog function are required.

A dialog function returns a value when the user chooses a command button. Enable acts on the value returned. The default is to return 0 (zero) and close the dialog box. If a non zero is assigned the dialog box remains open. By keeping the dialog box open, the dialog function allows the user to do more than one command from the same dialog box. Dialog examples ship as part of the sample .bas programs and can be found in your install directory.

ControlID$ Receives the identifier of the dialog box control

Action Identifies the action that calls the dialog function. There are six possibilities, Enable supports the first 4.

Action 1 The value passed before the dialog becomes visible

Action 2 The value passed when an action is taken (i.e. a button is pushed, checkbox is checked etc...) The controlID$ is the same as the identifier for the control that was chosen

Action 3 Corresponds to a change in a text box or combo box. This value is passed when a control loses the focus (for example, when the user presses the TAB key to move to a different control) or after the user clicks an item in the list of a combo box (an *Action* value of 2 is passed first). Note that if the contents of the text box or combo box do not change, an *Action* value of 3 is not passed. When *Action* is 3, *ControlID$* corresponds to the identifier for the text box or combo box whose contents were changed.

Action 4 Corresponds to a change of focus. When *Action* is 4, *ControlID$* corresponds to the identifier of the control that is gaining the focus. *SuppValue* corresponds to the numeric identifier for the control that lost the focus. A Dialog function cannot display a message box or dialog box in response to an *Action* value of 4

SuppValue receives supplemental information about a change in a dialog box control. The information SuppValue receives depends on which control calls the dialog function. The following *SuppValue* values are passed when *Action* is 2 or 3.

Control	SuppValue passed
ListBox, DropListBox, or ComboBox	Number of the item selected where 0 (zero) is the first item in the list box, 1 is the second item, and so on.
CheckBox	1 if selected, 0 (zero) if cleared.
OptionButton	Number of the option button selected, where 0 (zero) is the first option button within a group, 1 is the second option button, and so on.
TextBox	Number of characters in the text box.
ComboBox	If Action is 3, number of characters in the combo box.
CommandButton	A value identifying the button chosen. This value is not often used, since the same information is available from the ControlID$ value.

Statements and Functions Used in Dialog Functions

Statement or Function	Action or Result
DlgControlId	Returns the numeric equivalent of Identifier$, the string identifier for a dialog box control.
DlgEnable, DlgEnable()	The **DlgEnable** statement is used to enable or disable a dialog box control. When a control is disabled, it is visible in the dialog box, but is dimmed and not functional. **DlgEnable()** is used to determine whether or not the control is enabled.
DlgFocus, DlgFocus()	The **DlgFocus** statement is used to set the focus on a dialog box control. (When a dialog box control has the focus, it is highlighted.) **DlgFocus()** returns the identifier of the control that has the focus.
DlgListBoxArray, DlgListBoxArray()	The **DlgListBoxArray** statement is used to fill a list box or combo box with the elements of an array. It can be used to change the contents of a list box or combo box while the dialog box is displayed. **DlgListBoxArray()** returns an item in an array and the number of items in the array.
DlgSetPicture	The **DlgSetPicture** statement is used in a dialog function to set the graphic displayed by a picture control.
DlgText, DlgText	The **DlgText** statement is used to set the text or text label for a dialog box control. **TheDlgText()** function returns the label of a control.

DlgValue, DlgValue()	The **DlgValue** statement is used to select or clear a dialog box control. Then **DlgValue()** function returns the setting of a control.
DlgVisible, DlgVisible()	The **DlgVisible** statement is used to hide or show a dialog box control. The **DlgVisible()** function is used to determine whether a control is visible or hidden.

DlgControlId Function

DlgControlId(*Identifier)*
Used within a dialog function to return the numeric identifier for the dialog box control specified by *Identifier*, the string identifier of the dialog box control. Numeric identifiers are numbers, starting at 0 (zero) , that correspond to the positions of the dialog box control instructions within a dialog box definition. For example, consider the following instruction in a dialog box definition:
CheckBox 90, 50, 30, 12, "&Update", .MyCheckBox
The instruction DlgControlId("MyCheckBox") returns 0 (zero) if the CheckBox instruction is the first instruction in the dialog box definition, 1 if it is the second, and so on.
In most cases, your dialog functions will perform actions based on the string identifier of the control that was selected.

DlgFocus Statement, DlgFocus() Function

DlgFocus Identifier
DlgFocus()
The DlgFocus statement is used within a dialog function to set the focus on the dialog box control identified by Identifier while the dialog box is displayed. When a dialog box control has the focus, it is active and responds to keyboard input. For example, if a text box has the focus, any text you type appears in that text box.
The DlgFocus() function returns the string identifier for the dialog box control that currently has the focus.
Example:
This example sets the focus on the control "MyControl1" when the dialog box is initially displayed. (The main subroutine that contains the dialog box definition is not shown.)

```
Function MyDlgFunction( identifier, action, suppvalue)
Select Case action
    Case 1            ' The dialog box is displayed
        DlgFocus "MyControl1"
    Case 2
        ' Statements that perform actions based on which control is selected
    End Select
End Function
```

DlgListBoxArray, DlgListBoxArray()

DlgListBoxArray Identifier, ArrayVariable()
DlgListBoxArray(Identifier, ArrayVariable())
The DlgListBoxArray statement is used within a dialog function to fill a ListBox, DropListBox, or ComboBox with the contents of ArrayVariable() while the dialog box is displayed.
The DlgListBoxArray() function fills ArrayVariable() with the contents of the ListBox, DropListBox, or ComboBox specified by Identifier and returns the number of entries in the ListBox, DropListBox, or ComboBox. The ArrayVariable() parameter is optional (and currently not implemented) with the DlgListBoxArray() function; if ArrayVariable() is omitted, DlgListBoxArray() returns the number of entries in the specified control.

DlgSetPicture

DlgSetPicture Identifier, PictureName

The DlgSetPicture function is used to set the graphic displayed by a picture control in a dialog.

The Identifier is a string or numeric representing the dialog box. The PictureName is a string that identifies the picture to be displayed.

DlgValue, DlgValue()

DlgValue Identifier, Value
DlgValue(Identifier)

The DlgValue statement is used in a dialog function to select or clear a dialog box control by setting the numeric value associated with the control specified by Identifier. For example, DlgValue “MyCheckBox”, 1 selects a check box, DlgValue “MyCHeckBox”, 0 clears a check box, and DlgValue “MyCheckBox”, -1 fills the check box with gray. An error occurs if Identifier specifies a dialog box control such as a text box or an option button that cannot be set with a numeric value.

The following dialog function uses a Select Case control structure to check the value of Action. The SuppValue is ignored in this function.

```
'This sample file outlines dialog capabilities, including nesting dialog
boxes.

Sub Main

   Begin Dialog UserDialog1 60,60, 260, 188, "3", .Enable
        Text 8,10,73,13, "Text Label:"
        TextBox 8, 26, 160, 18, .FText
        CheckBox 8, 56, 203, 16, "Check to display controls",. Chk1
```

```
       GroupBox 8, 79, 230, 70, "This is a group box:", .Group
       CheckBox 18,100,189,16, "Check to change button text", .Chk2
       PushButton 18, 118, 159, 16, "File History", .History
       OKButton 177, 8, 58, 21
       CancelButton 177, 32, 58, 21
 End Dialog

    Dim Dlg1 As UserDialog1
    x = Dialog( Dlg1 )
End Sub


Function Enable( ControlID$, Action%, SuppValue%)

Begin Dialog UserDialog2 160,160, 260, 188, "3", .Enable
       Text 8,10,73,13, "New dialog Label:"
       TextBox 8, 26, 160, 18, .FText
       CheckBox 8, 56, 203, 16, "New CheckBox",. ch1
       CheckBox 18,100,189,16, "Additional CheckBox", .ch2
       PushButton 18, 118, 159, 16, "Push Button", .but1
       OKButton 177, 8, 58, 21
       CancelButton 177, 32, 58, 21
 End Dialog
 Dim Dlg2 As UserDialog2
 Dlg2.FText = "Your default string goes here"

Select Case Action%

Case 1
       DlgEnable "Group", 0
       DlgVisible "Chk2", 0
       DlgVisible "History", 0
Case 2
       If ControlID$ = "Chk1" Then
                 DlgEnable "Group"
                 DlgVisible "Chk2"
                 DlgVisible "History"
       End If

       If ControlID$ = "Chk2" Then
                 DlgText "History", "Push to display nested dialog"
       End If

       If ControlID$ = "History" Then
                  Enable =1
                  x = Dialog( Dlg2 )
       End If


Case Else

End Select
Enable =1

End Function
```

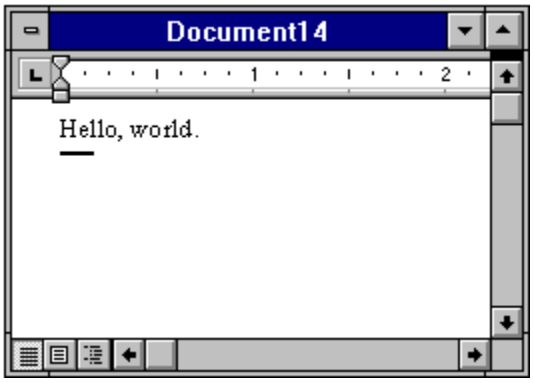
OLE Automation

What is OLE Automation?

OLE Automation is a standard, promoted by Microsoft, that applications use to expose their OLE objects to development tools, Enable Basic, and containers that support OLE Automation. A spreadsheet application may expose a worksheet, chart, cell, or range of cells all as different types of objects. A word processor might expose objects such as application, paragraph, sentence, bookmark, or selection.

When an application supports OLE Automation, the objects it exposes can be accessed by Enable Basic. You can use Enable Basic to manipulate these objects by invoking methods on the object, or by getting and setting the object's properties, just as you would with the objects in Enable Basic. For example, if you created an OLE Automation object named MyObj, you might write code such as this to manipulate the object:

```
Sub Main
Dim MyObj As Object
Set MyObj = CreateObject ("Word.Basic")
MyObj.FileNewDefault
MyObj.Insert "Hello, world."
MyObj.Bold 1
End Sub
```



The following syntax is supported for the **GetObject** function:

```
Set MyObj = GetObject ("", class)
```

Where class is the parameter representing the class of the object to retrieve. The first parameter at this time must be an empty string.

The properties and methods an object supports are defined by the application that created the object. See the application's documentation for details on the properties and methods it supports.

Accessing an object

The following functions and properties allow you to access an OLE Automation object:

Name	Description
CreateObject Function	Creates a new object of a specified type.
GetObject Function	Retrieves an object pointer to a running application.

What is an OLE Object?

An OLE *Automation Object* is an instance of a class within your application that you wish to manipulate programmatically, such as with Cypress Enable. These may be new classes whose sole purpose is to collect and expose data and functions in a way that makes sense to your customers.

The object becomes programmable when you expose those member functions. OLE Automation defines two types of members that you may expose for an object:

Methods are member functions that perform an action on an object. For example, a Document object might provide a Save method.

Properties are member function pairs that set or return information about the state of an object. For example, a Drawing object might have a style property.

For example, Microsoft suggests the following objects could be exposed by implementing the listed methods and properties for each object:

OLE Automation object	Methods	Properties
Application	Help	ActiveDocument
	Quit	Application
	Add Data	Caption
	Repeat	DefaultFilePath
	Undo	Documents
		Height
		Name
		Parent

		Path
		Printers
		StatusBar
		Top
		Value
		Visible
		Width

Document	Activate	Application
	Close	Author
	NewWindow	Comments
	Print	FullName
	PrintPreview	Keywords
	RevertToSaved	Name
	Save	Parent
	SaveAs	Path
		ReadOnly
		Saved
		Subject
		Title
		Value

To provide access to more than one instance of an object, expose a collection object. A collection object manages other objects. All collection objects support iteration over the objects they manage. For example, Microsoft suggests an application with a multiple document interface (MDI) might expose a Documents collection object with the following methods and properties:

Collection object	**Methods**	**Properties**
Documents	Add	Application

	Close	Count
	Item	Parent
	Open	

OLE Fundamentals

Object linking and embedding (OLE) is a technology that allows a programmer of Windows-based applications to create an application that can display data from many different applications, and allows the user to edit that data from within the application in which it was created. In some cases, the user can even edit the data from within their application.

The following terms and concepts are fundamental to understanding OLE.

OLE Object

An OLE object refers to a discrete unit of data supplied by an OLE application. An application can expose many types of objects. For example a spreadsheet application can expose a worksheet, macro sheet, chart, cell, or range of cells all as different types of objects. You use the OLE control to create linked and embedded objects. When a linked or embedded object is created, it contains the name of the application that supplied the object, its data (or, in the case of a linked object, a reference to the data), and an image of the data.

OLE Automation

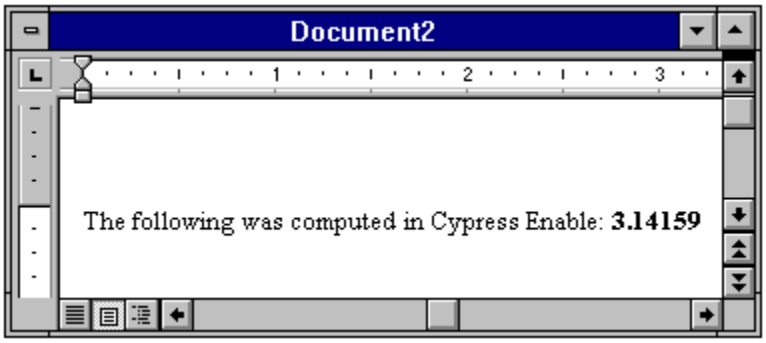
Some applications provide objects that support OLE Automation. You can use Enable Basic to programmatically manipulate the data in these objects. Some objects that support OLE Automation also support linking and embedding. You can create an OLE Automation object by using the CreateObject function.

Class

An objects class determines the application that provides the objects data and the type of data the object contains. The class names of some commonly used Microsoft applications include MSGraph, MSDraw, WordDocument, and ExcelWorksheet.

OLE Automation and Microsoft Word example:

```
Sub OLEexample()
    Dim word As Object
    Dim myData As String

    myData = 4 * Atn(1)       ' Demonstrates Automatic type conversion
    Set word = CreateObject("Word.Basic")
    Word.AppShow
    word.FileNewDefault
    word.Insert "The following was computed in Cypress Enable: "
    word.Bold 1               ' Show value in boldface
    word.Insert myData
    word.Bold 0

   MsgBox "Done"
End Sub
```



Making Applications Work Together

Operations like linking and object embedding need applications to work together in a coordinated fashion. However, there is no way that Windows can be set up, in advance, to accommodate all the applications and dynamic link libraries that can be installed. Even within an application, the user has the ability to select various components to install.

As part of the installation process, Windows requires that applications supporting DDE/OLE features register their support by storing information in several different locations. The most important of these to cypress enable is the registration database.

WIN.INI

The win.ini file contains a special section called [embedding] that contains information about each of three applications that operate as object servers.

The Registration Database.

Starting with Windows 3.1, Each Windows system maintains a *registration database* file that records details about the DDE and OLE functions supported by the installed applications. The database is stored in file called **REG.DAT** in the \ **WINDOWS** directory.

The Registration database

The registration database is a file called **REG.DAT**. The file is a database that contains information that controls a variety of activities relating to data integration using DDE and OLE. The information contained in the **REG.DAT** database can be divided into four basic categories.

Associations.

The table contains information that associates files with specific extensions to particular applications. This is essentially the same function performed by the [extensions] section of the **WIN.INI**.

Shell Operations.

Windows contains two programs that are refered to as *Shell* programs. The term *Shell* refers to a program that organizes basic operating system tasks, like running applications, opening files, and sending files to the printer. Shell programs use list, windows, menus, and dialog boxes to perform these operations. In contrast, command systems like DOS require the entry of explicit command lines to accomplish these tasks

OLE Object Servers.

The registration database maintains a highly structured database of the details needed by programs that operate as object servers. This is by far the most complex task performed by the database. There is no **WIN.INI** equivalent for this function.

DDE/OLE Automation.

The registration database contains the details and the applications that support various types of DDE/OLE Automation operations.

It is useful to appreciate the difference in structure between the **WIN.INI** file and the **REG.DAT** database. **WIN.INI** is simply a text document. There are no special structures other than headings (simply titles enclosed in brackets) that organize the information. If you want to locate an item in the **WIN.INI** file, you must search through the file for the specific item you want to locate. The registration database is a tree-like, structured database used for storing information relating to program and file operations, in particular, those that involve the use of DDE or OLE. The tree structure makes it easier to keep the complex set of instructions, needed to implement DDE and OLE operations, organized and accessible by the applications that need to use them. This is not possible when you are working with a text document like **WIN.INI.** The **WIN.INI** file records all sorts or information about the Windows system in a simple sequential listing.

Scripting Language Overview

Quick reference of the Functions and Statements available

Type/Functions/Statements

Flow of Control

Goto, End, OnError, Stop, Do...Loop, Exit Loop, For...Next, Exit For, If..Then..Else...End If, Stop, While...Wend, Select Case

Converting

Chr, Hex, Oct, Str, CDbl, CInt, Clng, CSng, CStr, CVar, CVDate, Asc, Val, Date, DateSerial, DateValue, Format, Fix, Int, Day, Weekday, Month, Year, Hour, Minute, Second, TimeSerial, TimeValue

Dialog

Text, TextBox, ListBox, DropList, ComboBox, CheckBox, OKButton, BeginDialog, EndDialog, OptionGroup, OKButton, CancelButton, PushButton, Picture, GroupBox, Multi-line TextBox,

File I/O

FileCopy, ChDir, ChDrive, CurDir, CurDir, MkDir,RmDir, Open, Close, Print #, Kill, FreeFile, LOF, FileLen, Seek, EOF, Write #, Input, Line Input, Dir, Name, GetAttr, SetAttr, Dir, Get, Put

Math

Exp, Log, Sqr, Rnd, Abs, Sgn, Atn, Cos, Sin, Tan, Int, Fix

Procedures

Call, Declare, Function, End Function, Sub, End Sub, Exit, Global

---

Strings

Let, Len, InStr, Left, Mid, Asc, Chr, Right, LCase, Ucase, InStr, LTrim, RTrim, Trim, Option Compare, Len, Space, String, StrComp Format,

---

Variables and Constants

Dim, IsNull, IsNumeric,VarType, Const, IsDate, IsEmpty, IsNull, Option Explicit, Global, Static,

---

Error Trapping

On Error, Resume

Date/Time

Date, Now, Time, Timer

DDE

DDEInitiate, DDEExecute, DDETerminate

Arrays

Option Base, Option Explicit, Static, Dim, Global, Lbound, Ubound, Erase, ReDim

---

Miscellaneous

SendKeys, AppActivate, Shell, Beep, Rem, CreateObject, GetObject Randomize

Data Types

Variable	Type Specifier	usage

String	$	Dim Str_Var As String
Integer	%	Dim Int_Var As Integer
Long	&	Dim Long_Var As Long
Single	!	Dim Sing_Var As Single
Double	#	Dim Dbl_Var As Double
Variant		Dim X As Any
Boolean		Dim X As Boolean
Byte		Dim X As Byte
Object		Dim X As Object
Currency		(Not currently supported)

Operators

Arithmetic Operators

Operator	Function	Usage
^	Exponentiation	x = y^2
-	Negation	x = -2
*	Multiplication	x% = 2 * 3
/	division	x = 10/2
Mod	Modulo	x = y Mod z
+	Addition	x = 2 + 3
-	Subtraction	x = 6 - 4

*Arithmetic operators follow mathematical rules of precedence
* '+' or '&' can be used for string concatenation.

Operator Precedence

Operator	Description	Order
()	parenthesis	highest
^	exponentiation	
-	unary minus	

/,*	division/multplication	
mod	modulo	
+, -, &	addition, subtraction, concatenation	
=, <>, <, >,<=,>=	relational	
not	logical negation	
and	logical conjunction	
or	logical disjunction	
Xor	logical exclusion	
Eqv	logical Equivalence	
Imp	logical Implication	lowest

Relational Operators

Operator	Function	Usage
<	Less than	x < Y
<=	Less than or equal to	x <= Y
=	Equals	x = Y
>=	Greater than or equal to	x >= Y
>	Greater than	x > Y
<>	Not equal to	x <> Y

Logical Operators

Operator	Function	Usage
Not	Logical Negation	If Not (x)
And	Logical And	If (x> y) And (x < Z)
Or	Logical Or	if (x = y) Or (x = z)

Functions, Statements, Reserved words - Quick Reference

Abs, Access, Alias, And, Any
App, AppActivate, Asc, Atn, As

Base, Beep, Begin, Binary, ByVal
Call, Case, ChDir, ChDrive, Choose, Chr, Const, Cos, CurDir, CDbl, CInt, CLng, CSng, CStr, CVar, CVDate,Close, CreateObject
Date, Day, Declare, Dim, Dir, Do...Loop,Dialog, DDEInitiate DDEExecute, DateSerial, DateValue, Double
Else, ElseIf, End, EndIf, EOF, Eqv, Erase, Err, Error
Exit, Exp, Explicit
False, FileCopy, FileLen, Fix, For,
For...Next, Format, Function
Get, GetAttr, GoTo, Global, Get Object
Hex, Hour
If...Then...Else...[End If], Imp, Input, InputBox, InStr, Int, Integer, Is, IsEmpty, IsNull, IsNumeric, IsDate
Kill
LBound, LCase, Left, Len, Let, LOF,Log, Long, Loop, LTrim
Line Input
Mid,Minute, MkDir, Mod, Month, MsgBox
Name, Next, Not, Now
Oct,On, Open, OKButton,Object, Option, Optional, Or, On Error
Print, Print #, Private, Put
Randomize, Rem, ReDim, RmDir, Rnd, Rtrim
Seek, SendKeys, Set, SetAttr, Second, Select, Shell, Sin, Sqr, Stop,Str, Sng, Single, Space, Static, Step, Stop, Str, String, Sub, StringComp
Tan,Text, TextBox, Time, Timer, TimeSerial, TimeVale, Then, Type, Trim, True, To, Type
UBound, UCase, Ucase, Until
Val, Variant, VarType
Write #, While, Weekday, Wend, With
Xor
Year

Language Reference A - Z

Abs Function

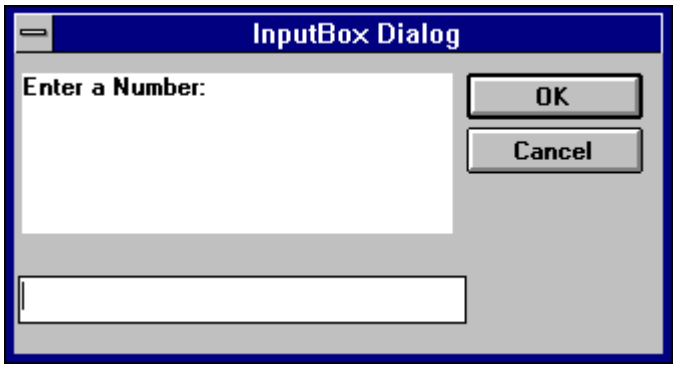
Abs (*number*)

Returns the absolute value of a number.

The data type of the return value is the same as that of the number argument. However, if the number argument is a Variant of VarType (String) and can be converted to a number, the return value will be a Variant of VarType (Double). If the numeric expression results in a Null, _Abs returns a Null.

Example:

```
Sub Main

        Dim Msg, X, Y

        X = InputBox("Enter a Number:")
        Y = Abs(X)

        Msg = "The number you entered is "  & X
        Msg = Msg + ". The Absolute value of " & X & " is " & Y
        MsgBox Msg 'Display Message.

End Sub
```



AppActivate Statement

AppActivate “*app*”

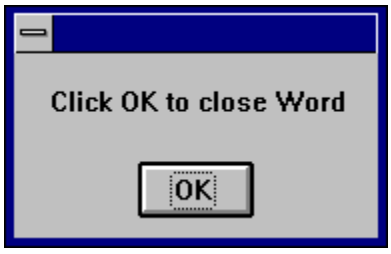
Activates an application.

The parameter *app* is a string expression and is the name that appears in the title bar of the application window to activate.

Related Topics: Shell, SendKeys

Example:

```
Sub Main ()
        AppActivate "Microsoft Word"
        SendKeys “%F,%N,Cypress Enable”,True
        Msg = “Click OK to close Word”
        MsgBox Msg
        AppActivate “Microsoft Word”
        SendKeys “%F,%C,N”, True
End Sub
```



Asc Function

Asc (*str*)

Returns a numeric value that is the ASCII code for the first character in a string.

Example:

```
Sub Main ()
    Dim I, Msg                          ' Declare variables.
    For I = Asc("A") To Asc("Z")        ' From A through Z.
        Msg = Msg & Chr(I)              ' Create a string.
    Next I
    MsgBox Msg                          ' Display results.
End Sub
```

Atn Function

Atn (*rad*)

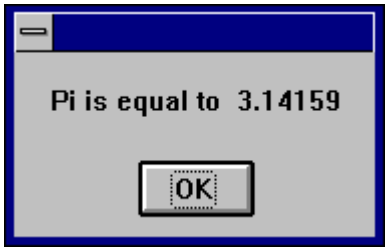
Returns the arc tangent of a number

The argument *rad* can be any numeric expression. The result is expressed in radians

Related Topics: Cos, Tan, Sin

Example:

```
Sub AtnExample ()
    Dim Msg, Pi         ' Declare variables.
    Pi = 4 * Atn(1)     ' Calculate Pi.
    Msg = "Pi is equal to " & Str(Pi)
    MsgBox Msg          ' Display results.
End Sub
```



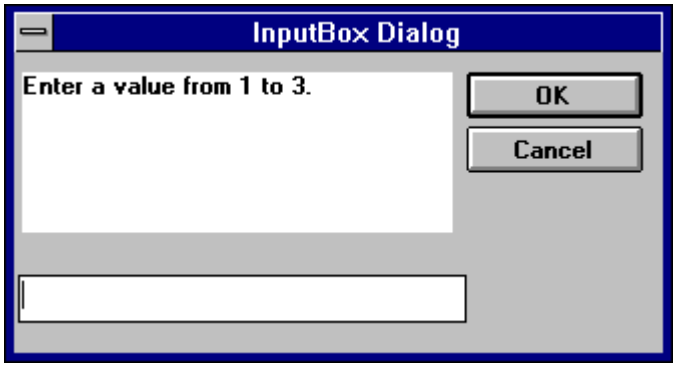
Beep Statement

Beep

Sounds a tone through the computer's speaker. The frequency and duration of the beep depends on hardware, which may vary among computers.

Example:

```
Sub BeepExample ()
    Dim Answer, Msg            ' Declare variables.
    Do
        Answer = InputBox("Enter a value from 1 to 3.")
        If Answer >= 1 And Answer <= 3 Then   ' Check range.
            Exit Do                           ' Exit Do...Loop.
        Else
            Beep                               ' Beep if not in range.
        End If
    Loop
    MsgBox "You entered a value in the proper range."
End Sub
```



Call Statement

Call funcname [(parameter(s)]
or
[parameter(s)]

Activates an Enable Subroutine called *name* or a DLL function with the name *name*. The first parameter is the name of the function or subroutine to call, and the second is the list of arguments to pass to the called function or subroutine.

You are never required to use the Call statement when calling an Enable subroutine or a DLL function. Parentheses must be used in the argument list if the Call statement is being used.

Example:

```
Sub Main ()
       Call Beep
       MsgBox "Returns a Beep"
```

```
End Sub
```

CBool Function

CBool (*expression*)

Converts expressions from one data type to a boolean. The parameter *expression* must be a valid string or numeric expression.

Example:

```
Sub Main

  Dim A, B, Check

  A = 5: B = 5
  Check = CBool(A = B)
  Print Check

  A = 0
  Check = CBool(A)
  Print Check

End Sub
```

CDate Function

CVDate (*expression*)

Converts any valid expression to a Date variable with a vartype of 7. The parameter *expression* must be a valid string or numeric date expression and can represent a date from January 1, 30 through December 31, 9999.

Example:

```
Sub Main

Dim MyDate, MDate, MTime, MSTime
MybDate = "May 29, 1959"      ' Define date.
MDate = CDate(MybDate) ' Convert to Date data type.

MTime = "10:32:27 PM"  ' Define time.
```

```
MSTime = CDate(MTime)  ' Convert to Date data type.

Print MDate
Print MSTime

End Sub
```

CDbl Function

CDbl (*expression*)

Converts expressions from one data type to a double. The parameter *expression* must be a valid string or numeric expression.

Example:

```
Sub Main ()
    Dim y As Integer

    y = 25555  'the integer expression only allows for 5 digits
    If  VarType(y) = 2 Then
    Print y

      x = CDbl(y)   'Converts the integer value of y to a double value in x
      x = x * 100000 'y is now 10 digits in the form of x   '
       Print x
    End If

End Sub
```

ChDir Statement

ChDir *pathname*

Changes the default directory

Pathname: [*drive*:] [\ ] *dir*[*dir*]...

The parameter *pathname* is a string limited to fewer then 128 characters. The *drive* parameter is optional. The *dir* parameter is a directory name. ChDir changes the default directory on the current drive, if the drive is omitted.

Related Topics: CurDir, CurDir$, ChDrive, Dir, Dir$, MkDir, RmDir

Example:

```
Sub Main ()
```

```
        Dim Answer, Msg, NL    ' Declare variables.
        NL = Chr(10)   ' Define newline.
        CurPath = CurDir()     ' Get current path.
        ChDir "\"
        Msg = "The current directory has been changed to "
        Msg = Msg & CurDir() & NL & NL & "Press OK to change back "
        Msg = Msg & "to your previous default directory."
        Answer = MsgBox(Msg)   ' Get user response.
        ChDir CurPath  ' Change back to user default.
        Msg = "Directory changed back to " & CurPath & "."
        MsgBox Msg     ' Display results.
End Sub
```

ChDrive Statement

ChDrive *drivename*

Changes the default drive

The parameter *drivename* is a string and must correspond to a an existing drive. If *drivename* contains more than one letter, only the first character is used.

Example:

```
Sub Main ()
        Dim Msg, NL    ' Declare variables.
        NL = Chr(10)   ' Define newline.
        CurPath = CurDir()     ' Get current path.
        ChDir "\"
        ChDrive "C:"
        Msg = "The current directory has been changed to "
        Msg = Msg & CurDir() & NL & NL & "Press OK to change back "
        Msg = Msg & "to your previous default directory."
        MsgBox Msg     ' Get user response.
        ChDir CurPath  ' Change back to user default.
        Msg = "Directory changed back to " & CurPath & "."
        MsgBox Msg     ' Display results.
End Sub
```

Related Topics: ChDir, CurDir, MkDir, RmDir

CheckBox

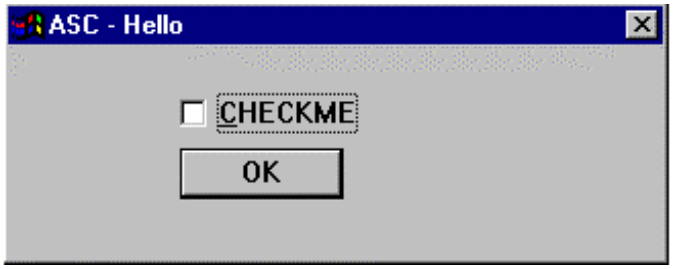
CheckBox *starting x position, starting y position, width, height*

For selecting one or more in a series of choices

Example:

```
    Sub Main ()
        Begin Dialog DialogName1 60, 70, 160, 50, "ASC - Hello"

            CHECKBOX 42, 10, 48, 12, "&CHECKME", .checkInt
            OKBUTTON 42, 24, 40, 12
```

```
 End Dialog
Dim Dlg1 As DialogName1
Dialog Dlg1
If Dlg1.checkInt = 0 Then
  Q = "didn't check the box."
 Else
  Q = "checked the box."
End If
MsgBox "You " & Q

End Sub
```



Choose Function

Choose(*number, choice1, [choice2,] [choice3,]…*)

Returns a value from a list of arguments

Choose will return a null value if number is less than one or greater than the number of choices in the list. If *number* is not an integer it will be rounded to the nearest integer.

Example:

```
Sub Main
    number = 2
    GetChoice = Choose(number, "Choice1", "Choice2", "Choice3")
    Print GetChoice
End Sub
```

Chr Function

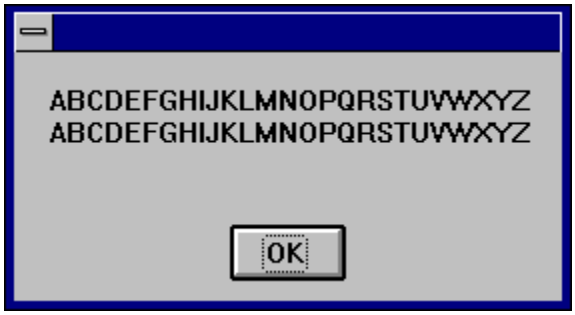
Chr(*int*)

Returns a one-character string whose ASCII number is the argument

Chr returns a String

Example:

```
Sub ChrExample ()
        Dim X, Y, Msg, NL
        NL = Chr(10)
        For X = 1 to 2
                For Y = Asc("A") To Asc("Z")
                        Msg = Msg & Chr(Y)
                Next Y
        Msg = Msg & NL
        Next X
        MsgBox Msg
End Sub
```



CInt Function

CInt (*expression*)

Converts any valid expression to an integer.

Example:

```
Sub Main ()
    Dim y As Long

    y = 25
    Print VarType(y)

    If  VarType(y) = 3 Then
        Print y
        x = CInt(y)   'Converts the long value of y to an integer value in x
         Print x
        Print VarType(x)
    End If

End Sub
```

CLng Function

CLng (*expression*)

Converts any valid expression into a long.

Example:

```
Sub Main ()
    Dim y As Integer

    y = 25000  'the integer expression can only hold five digits
    If  VarType(y) = 2 Then
        Print y
      x = CLng(y)  'Converts the integer value of x to a long value in x
      x = x * 10000 'y is now ten digits in the form of x
       Print x
    End If

End Sub
```

Close Statement

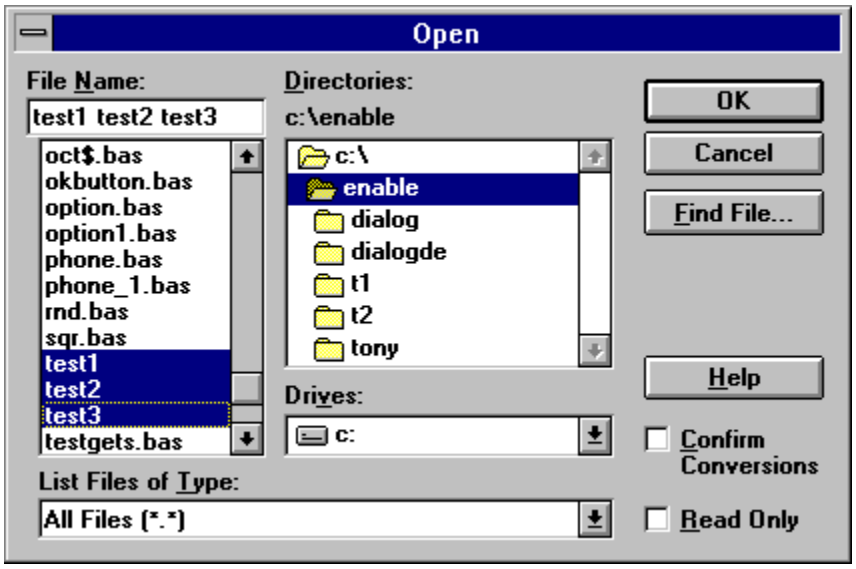
Close *[[#filenumber] [, [#]filenumber],,,*

The Close Statement takes one argument *filenumber*. *Filenumber* is the number used with the Open Statement to open the file. If the Close Statement is used without any arguments it closes all open files.

Example:

```
Sub Main
Open "c:\test.txt" For Input As #1
Do While Not EOF(1)
       MyStr = Input(10, #1)
       MsgBox MyStr
Loop
Close #1

End Sub
```

```
Sub Make3Files ()
       Dim I, FNum, FName     ' Declare variables.
       For I = 1 To 3
              FNum = FreeFile        ' Determine next file number.
              FName = "TEST" & FNum
              Open FName For Output As FNum ' Open file.
              Print #I, "This is test #" & I        ' Write string to file.
              Print #I, "Here is another ";  "line";  I
       Next I
       Close   ' Close all files.
End Sub
```



Const Statement

Const *name = expression*

Assigns a symbolic name to a constant value.

A constant must be defined before it is used.

The definition of a Const in Cypress Enable outside the procedure or at the module level is a global. The syntax Global Const and Const are used below outside the module level are identical.

A type declaration character may be used however if none is used Enable will automatically assign one of the following data types to the constant, long (if it is a long or integer), Double (if a decimal place is present), or a String (if it is a string).

Example:

```
Global Const Height = 14.4357     '
Const PI = 3.14159      'Global to all procedures in a module
Sub Main ()
        Begin Dialog DialogName1 60, 60, 160,70, "ASC - Hello"
        TEXT 10, 10, 100, 20, "Please fill in the radius of circle x"
        TEXT 10, 40, 28, 12, "Radius"
        TEXTBOX 42, 40, 28, 12, .Radius
        OKBUTTON 42, 54,40, 12
End Dialog
Dim Dlg1 As DialogName1
Dialog Dlg1
CylArea = Height * (Dlg1.Radius * Dlg1.Radius) * PI
MsgBox "The volume of Cylinder x is " & CylArea
End Sub
```

Cos Function

Cos (*rad*)

Returns the cosine of an angle

The argument *rad* must be expressed in radians and must be a valid numeric expression.Cos will by default return a double unless a single or integer is specified as the return value.

Example:

```
Sub Main()
    Dim J As Double
    Dim I As Single                    ' Declare variables.
    Dim K As Integer
    For I =1 To 10    '
        Msg = Msg & Cos(I) & ", "          'Cos function call
        J=Cos(I)
        Print J
        K=Cos(I)
        Print K
    Next I
    MsgBox Msg                       ' Display results.
    MsgBox Msg1
End Sub
```

CreateObject Function

CreateObject (*class*)

Creates an OLE automation object.

```
Sub Command1_Click ()
    Dim word6 As object
    Set word6 = CreateObject("Word.Basic")
    word6.FileNewDefault
    word6.InsertPara
    word6.Insert "Attn:"
    word6.InsertPara
    word6.InsertPara
    word6.InsertPara
    word6.Insert "          Vender Name: "
    word6.Bold 1
    name = "Some Body"
    word6.Insert name
    word6.Bold 0
    word6.InsertPara
    word6.Insert "    Vender Address:"
    word6.InsertPara
    word6.Insert "        Vender Product:"
```

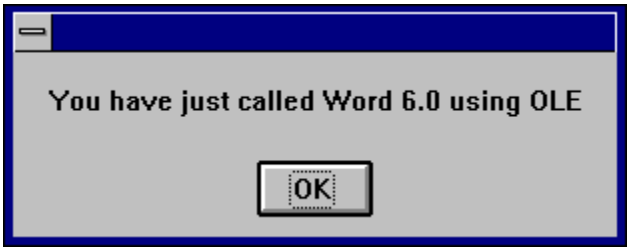
```
    word6.InsertPara
    word6.InsertPara
    word6.Insert "Dear Vender:"
    word6.InsertPara
    word6.InsertPara
    word6.Insert "The letter you are reading was created with Cypress Enable."
    word6.Insert " Using OLE Automation Cypress Enable can call any other OLE _ enable
    word6.Insert "application.  Enable is a Basic Scripting Language for _ applications
    word6.InsertPara
    word6.InsertPara
    word6.Insert "      Product Name: Cypress Enable"
    word6.InsertPara
    word6.Insert "      Company Name: Cypress Software Inc."
    word6.InsertPara

    word6.InsertPara
   MsgBox "You have just called Word 6.0 using OLE"
End Sub
```

Vender Name: **Client Name**
Vender Address:
Vender Product:

Dear Vender:

The letter you are reading was created with Cypress Enable.Using OLE Automation Cypress Enable can call any other OLE enabled application. Enable is a Basic Scripting Language for applications

Product Name: Cypress Enable
Company Name: Cypress Software Inc.



CSng Function

CSng (*expression*)

Converts any valid expression to a Single.

Example:

```
Sub Main ()
    Dim y As Integer

    y = 25
    If  VarType(y) = 2 Then
        Print y
        x = CSng(y)   'Converts the integer value of y to a single value in x
       Print x
    End If
```

CStr Function

CStr(*expression*)

Converts any valid expression to a String.

Example:

```
Sub Main
 Dim Y As Integer
  Y = 25
     Print Y
 If VarType(Y) = 2 Then
    X = CStr(Y) 'converts Y To a Str
    X = X + "hello"  'It is now possible to combine Y with strings
     Print X
 End If
End Sub
```

CurDir Function

CurDir (*drive*)

Returns the current path for the specified drive

CurDir returns a Variant; CurDir$ returns a String.

Example:

```
'Declare Function CurDir Lib "NewFuns.dll" ()  As String
Sub Form_Click ()
       Dim Msg, NL    ' Declare variables.
       NL = Chr(10)   ' Define newline.
       Msg = "The current directory is: "
       Msg = Msg & NL & CurDir()
       MsgBox Msg     ' Display message.
End Sub
```

CVar Function

CVar (*expression*)

Converts any valid expression to a Variant.

Example:

```
Sub Main

        Dim MyInt As Integer
        MyInt = 4534
        Print MyInt
        MyVar = CVar(MyInt & "0.23") 'makes MyInt a Variant + 0.32
        Print MyVar

End Sub
```

Date Function

Date, Date()

Returns the current system date

Date returns a Variant of VarType 8 (String) containing a date.

Example:

```
' Format Function Example
' This example shows various uses of the Format function to format values
' using both named and user-defined formats.  For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running.  When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used.  When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale.  For this example, English/United States is assumed.
```

```
' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main
x = Date()
Print Date
Print x
Print “VarType: “ & VarType(Date)
MyTime = "08:04:23 PM"
MyDate = "03/03/95"
MyDate = "January 27, 1993"

SysDate = Date
MsgBox Sysdate,0,"System Date"

MsgBox Now,0,"Now"
MsgBox MyTime,0,"MyTime"

MsgBox Second( MyTime ) & " Seconds"
MsgBox Minute( MyTime ) & " Minutes"
MsgBox Hour( MyTime ) & " Hours"

MsgBox Day( MyDate ) & " Days"
MsgBox Month( MyDate ) & " Months"
MsgBox Year( MyDate ) & " Years"

' Returns current system time in the system-defined long time format.
MsgBox Format(Time, "Short Time") & " Short Time"
MsgBox Format(Time, "Long Time") & "Long Time"

' Returns current system date in the system-defined long date format.
MsgBox Format(Date, "Short Date") & " Short Date"
MsgBox Format(Date, "Long Date") & " Long Date"

MyDate = "30 December 91" ' use of European date
print Mydate

MsgBox MyDate,0,"MyDate International..."
MsgBox Day(MyDate),0,"day"
MsgBox Month(MyDate),0,"month"
MsgBox Year(MyDate),0,"year"

MyDate = "30-Dec-91" ' another of European date usage
print Mydate


MsgBox MyDate,0,"MyDate International..."
MsgBox Day(MyDate),0,"day"
MsgBox Month(MyDate),0," month"
MsgBox Year(MyDate),0,"year"


MsgBox Format("This is it", ">")        ' Returns "THIS IS IT".

End Sub
```

DateSerial Function

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

Returns a variant (Date) corresponding to the year, month and day that were passed in. All three parameters for the DateSerial Function are required and must be valid.

Related Topics: DateValue. TimeSerial, TimeValue

Example:

```
Sub Main

        Dim MDate
        MDate = DateSerial(1959, 5, 29)
        Print MDate

End Sub
```

DateValue Function

DateValue(*dateexpression*)

Returns a variant (Date) corresponding to the string date expression that was passed in. *dateexpression* can be a string or any expression that can represent a date, time or both a date and a time.

Related Topics: DateSerial. TimeSerial, TimeValue

Example:

```
Sub Main()
Dim v As Variant
Dim d As Double
        d = Now
        Print d
        v = DateValue("1959/05/29")
        MsgBox (VarType(v))
        MsgBox (v)
End Sub
```

Day Function

Day(*dateexpression*)

Returns a variant date corresponding to the string date expression that was passed in. *dateexpression* can be a string or any expression that can represent a date.

Related Topics: Month, Weekday, Hour, Second

Example:

```
Sub Main

        Dim MDate, MDay
        MDate = #May 29, 1959#
        MDay = Day(MDate)
        Print "The Day listed is the " & MDay

End Sub
```

Declare Statement

Declare Sub *procedurename Lib Libname*$ [*Alias aliasname*$][(*argument list*)]

Declare Function *procedurename Lib Libname*$ [*Alias aliasname*$] [(*argument list*)][As *Type*]

The Declare statement makes a reference to an external procedure in a Dynamic Link Library (DLL).

The *procedurename* parameter is the name of the function or subroutine being called.

The *Libname* parameter is the name of the DLL that contains the procedure.

The optional Alias *aliasname* clause is used to supply the procedure name in the DLL if different from the name specified on the procedure parameter. When the optional *argument list* needs to be passed the format is as follows:

([ByVal] variable [As type] [,ByVal] variable [As type]]…])

The optional ByVal parameter specifies that the variable is [passed by value instead of by reference (see “ByRef and ByVal” in this manual). The optional As type parameter is used to specify the data type. Valid types are String, Integer, Double, Long, and Varaint (see “Variable Types” in this manual).

If a procedure has no arguments, use double parentheses () only to assure that no arguments are passed. For example:

Declare Sub OntTime Lib “Check” ()

Cypress Enable extentions to the declare statement. The following syntax is not supported by Microsoft Visual Basic.

Declare Function *procedurename App* [*Alias aliasname*$] [(*argument list*)][As *Type*]

This form of the Declare statement makes a reference to a function located in the executable file located in the application where Enable is embedded.

Related Topics: Call

Example:

```
Declare Function GetFocus Lib "User" () As Integer
Declare Function GetWindowText Lib "User" (ByVal hWnd%, ByVal Mess$, ByVal cbMax%) As
Integer

Sub Main
    Dim hWindow%
    Dim str1 As String *51
    Dim str2 As String * 25

    hWindow% = GetFocus()
    print "GetWindowText returned: ", GetWindowText( hWindow%, str1,51 )
    print "GetWindowText2 returned: ", GetWindowText( hWindow%, str2, 25)
    print str1
    print str2

End Sub
```

Dialog, Dialog Function

Dialog(*DialogRecord*)

Returns a value corresponding to the button the user chooses.

The Dialog() function is used to display the dialog box specified by *DialogRecord* . *DialogRecord* is the name of the dialog and must be defined in a preceeding Dim statement.

The return value or button:
 -1 = OK button
  0 = Cancel button
> 0 A command button where 1 is the first PushButton in the definition of the dialog and 2 is the second and so on.

Example:

```
' This sample shows all of the dialog controls on one dialog and how to
' vary the response based on which PushButton was pressed.

Sub Main ()
    Dim MyList$(2)
    MyList(0) = "Banana"
    MyList(1) = "Orange"
    MyList(2) = "Apple"
    Begin Dialog DialogName1 60, 60, 240, 184, "Test Dialog"
        Text 10, 10, 28, 12, "Name:"
        TextBox 40, 10,50, 12, .joe
        ListBox 102, 10, 108, 16, MyList$(), .MyList1
        ComboBox 42, 30, 108, 42, MyList$(), .Combo1
        DropListBox 42, 76, 108, 36, MyList$(), .DropList1$
        OptionGroup .grp1
            OptionButton 42, 100, 48, 12, "Option&1"
            OptionButton 42, 110, 48, 12, "Option&2"
        OptionGroup .grp2
            OptionButton 42, 136, 48, 12, "Option&3"
            OptionButton 42, 146, 48, 12, "Option&4"
        GroupBox 132, 125, 70, 36, "Group"
        CheckBox 142, 100, 48, 12, "Check&A", .Check1
        CheckBox 142, 110, 48, 12, "Check&B", .Check2
        CheckBox 142, 136, 48, 12, "Check&C", .Check3
        CheckBox 142, 146, 48, 12, "Check&D", .Check4
        CancelButton 42, 168, 40, 12
        OKButton 90, 168, 40, 12
        PushButton 140, 168, 40, 12, "&Push Me 1"
        PushButton 190, 168, 40, 12, "Push &Me 2"
    End Dialog
    Dim Dlg1 As DialogName1
    Dlg1.joe  = "Def String"
    Dlg1.MyList1 = 1
    Dlg1.Combo1 = "Kiwi"
    Dlg1.DropList1 = 2
    Dlg1.grp2 = 1
    ' Dialog returns -1 for OK, 0 for Cancel, button # for PushButtons
    button = Dialog( Dlg1 )
    'MsgBox "button: " & button 'uncomment for button return vale
    If button = 0 Then Exit Sub

    MsgBox "TextBox: "& Dlg1.joe
    MsgBox "ListBox: " & Dlg1.MyList1
    MsgBox Dlg1.Combo1
    MsgBox Dlg1.DropList1
    MsgBox "grp1: " & Dlg1.grp1
    MsgBox "grp2: " & Dlg1.grp2
    Begin Dialog DialogName2 60, 60, 160, 60, "Test Dialog 2"
        Text 10, 10, 28, 12, "Name:"
        TextBox 42, 10, 108, 12, .fred
        OkButton 42, 44, 40, 12
    End Dialog
    If button = 2 Then
        Dim Dlg2 As DialogName2
        Dialog Dlg2
        MsgBox Dlg2.fred
    ElseIf button = 1 Then
        Dialog Dlg1
        MsgBox Dlg1.Combo1
    End If
End Sub
```

Dim Statement

Dim variablename[(*subscripts*)][As *Type*][,name][As *Type*]]

Allocates storage for and declares the data type of variables and arrays in a module.

The types currently supported are integer, long, single, double and string and variant.

Example:

```
Sub Main
    Dim x As Long
    Dim y As Integer
    Dim z As single
    Dim a As double
    Dim s As String
    Dim v As Variant ' This is the same as Dim x or Dim x as any
End Sub
```

Dir Function

Dir[(*path,attributes*)]

Returns a file/directory name that matches the given *path* and *attributes*.

Example:

```
'========================================================
' Bitmap sample using the Dir Function
'========================================================

Sub DrawBitmapSample
    Dim MyList()
    Begin Dialog BitmapDlg 60, 60, 290, 220, "Enable bitmap sample", .DlgFunc
        ListBox 10, 10, 80, 180, MyList(), .List1, 2
        Picture 100, 10, 180, 180, "Forest.bmp", 0, .Picture1
        CancelButton 42, 198, 40, 12
        OKButton 90, 198, 40, 12
    End Dialog

    Dim frame As BitmapDlg

    ' Show the bitmap dialog
    Dialog frame
End Sub

Function DlgFunc( controlID As String, action As Integer, suppValue As Integer
)

    DlgFunc = 1       ' Keep dialog active

    Select Case action
    Case 1 ' Initialize
        temp = Dir( "c:\Windows\*.bmp" )
        count = 0
        While temp <> ""
            count = count + 1
            temp = Dir
        Wend
        Dim x() As String
```

```
        ReDim x(count)
        x(0) = Dir( "c:\Windows\*.bmp" )
        For i = 1 To count
            x(i) = dir
        Next i
        DlgListBoxArray "List1", x()
    Case 2 ' Click
        fileName = "c:\windows\" & DlgText("List1")
        DlgSetPicture "Picture1", fileName
    End Select
End Function
```

DlgEnable Statement

DlgEnable "*ControlName*", *Value*

This statement is used to enable or disable a particular control on a dialog box.

The parameter *ControlName* is the name of the control on the dialog box. The parameter *Value* is the value to set it to. 1 = Enable, 0 = Disable. On is equal to 1 in the example below. If the second parameter is omitted the status of the control toggles. The entire example below can be found in the dialog section of this manual and in the example .bas files that ship with Cypress Enable.

Related Topics: DlgVisible, DlgText

Example:

```
Function Enable( ControlID$, Action%, SuppValue%)
Begin Dialog UserDialog2 160,160, 260, 188, "3", .Enable
       Text 8,10,73,13, "New dialog Label:"
       TextBox 8, 26, 160, 18, .FText
       CheckBox 8, 56, 203, 16, "New CheckBox",. ch1
       CheckBox 18,100,189,16, "Additional CheckBox", .ch2
       PushButton 18, 118, 159, 16, "Push Button", .but1
       OKButton 177, 8, 58, 21
       CancelButton 177, 32, 58, 21
 End Dialog

 Dim Dlg2 As UserDialog2
 Dlg2.FText = "Your default string goes here"
Select Case Action%

Case 1
       DlgEnable "Group", 0
       DlgVisible "Chk2", 0
       DlgVisible "History", 0
Case 2
       If ControlID$ = "Chk1" Then
                 DlgEnable "Group", On
                 DlgVisible "Chk2"
                 DlgVisible "History"
       End If
```

```
        If ControlID$ = "Chk2" Then
                  DlgText "History", "Push to display nested dialog"
        End If

        If ControlID$ = "History" Then
                   Enable =1
                    Number = 4
                   MsgBox SQR(Number) & " The sqr of 4 is 2"
                   x = Dialog( Dlg2 )
        End If

        If ControlID$ = "but1" Then


        End If

Case Else

End Select
Enable =1

End Function
```

DlgText Statement

DlgText *"ControlName", String*

This statement is used to set or change the text of a dialog control.

The parameter *ControlName* is the name of the control on the dialog box. The parameter *String* is the value to set it to.

Related Topics: DlgEnable, DlgVisible

Example:

```
        If ControlID$ = "Chk2" Then
                  DlgText "History", "Push to display nested dialog"
        End If
```

DlgVisible Statement

DlgVisible *"ControlName", Value*

This statement is used to hide or make visible a particular control on a dialog box.

The parameter *ControlName* is the name of the control on the dialog box. The parameter *Value* is the value to set it to. 1 = Visible, 0 = Hidden. On is equal to 1. If the second parameter is omitted the status of the control toggles. The entire example below can be found in the dialog section of this manual and in the example .bas files that ship with Cypress Enable.

Related Topics: DlgEnable, DlgText

Example:

```
If ControlID$ = "Chk1" Then
          DlgEnable "Group", On
          DlgVisible "Chk2"
          DlgVisible "History"
End If
```

Do...Loop Statement

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

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

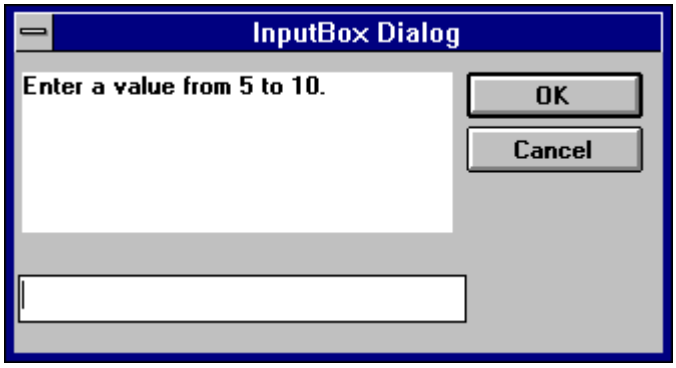
Repeats a group of statements while a condition is true or until a condition is met.

Related Topics: While, Wend

Example:

```
Sub Main ()
    Dim Value, Msg         ' Declare variables.
    Do
        Value = InputBox("Enter a value from 5 to 10.")
        If Value >= 5 And Value <= 10 Then
            Exit Do         ' Exit Do...Loop.
        Else
            Beep            ' Beep if not in range.
        End If
```

```
    Loop
End Sub
```



End Statement

End[{*Function* | *If* | *Sub*}]

Ends a program or a block of statements such as a Sub procedure or a function.

Related Topics: Exit, Function, If...Then...Else, Select Case, Stop

Example:

```
Sub Main()

        Dim Var1 as String

        Var1 = "hello"
        MsgBox " Calling Test"
        Test Var1
        MsgBox Var1

End Sub

Sub Test(wvar1 as string)

        wvar1 = "goodbye"
        MsgBox "Use of End Statement"
        End

End Sub
```

EOF Function

EOF(*Filenumber*)

Returns a value during file input that indicates whether the end of a file has been reached.

Related Topics: Open Statement

Example:

```
' Input Function Example

' This example uses the Input function to read 10 characters at a time from a
' file and display them in a MsgBox.  This example assumes that TESTFILE is a
'text file with a few lines of 'sample data.

Sub Main
    Open "TESTFILE" For Input As #1  ' Open file.
    Do While Not EOF(1)              ' Loop until end of file.
       MyStr = Input(10, #1)  ' Get ten characters.
       MsgBox MyStr
    Loop
    Close #1                 ' Close file.
End Sub
```

Erase Statement

Erase *arrayname*[,*arrayname*]

Reinitializes the elements of a fixed array.

Related Topics: Dim

Example:

```
' This example demonstrates some of the  features of arrays.  The lower bound
' for an array is 0 unless it is specified or option base has set it as is
' done in this example.

Option Base 1

Sub Main
' Declare array variables.
Dim Num(10) As Integer   ' Integer array.
Dim StrVarArray(10) As String   ' Variable-string array.
Dim StrFixArray(10) As String * 10   ' Fixed-string array.
Dim VarArray(10) As Variant   ' Variant array.
Dim DynamicArray() As Integer   ' Dynamic array.
ReDim DynamicArray(10)   ' Allocate storage space.
Erase Num   ' Each element set to 0.
Erase StrVarArray   ' Each element set to zero-length
   ' string ("").
Erase StrFixArray   ' Each element set to 0.
Erase VarArray   ' Each element set to Empty.
Erase DynamicArray   ' Free memory used by array.

End Sub
```

Exit Statement

Exit {*Do* | *For* | *Function* | *Sub* }

Exits a loop or procedure

Related Topics End Statement, Stop Statement

Example:

```
' This sample shows Do ... Loop with Exit Do to get out.

Sub Main ()
    Dim Value, Msg                                  ' Declare variables.
    Do
        Value = InputBox("Enter a value from 5 to 10.")
        If Value >= 5 And Value <= 10 Then        ' Check range.
            Exit Do                                  ' Exit Do...Loop.
        Else
            Beep                                     ' Beep if not in range.
        End If
    Loop

End Sub
```

Exp

Exp(*num*)

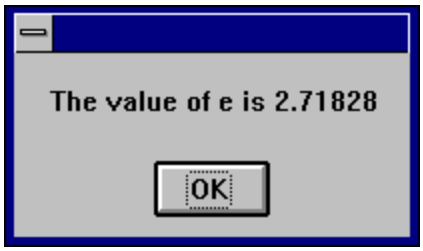
Returns the base of the natural log raised to a power (e ^ num).

The value of the constant e is approximately 2.71828.

Related Topics: Log

Example:

```
Sub ExpExample ()
       ' Exp(x) is e ^x so Exp(1) is e ^1 or e.
       Dim Msg, ValueOfE      ' Declare variables.
       ValueOfE = Exp(1)      ' Calculate value of e.
       Msg = "The value of e is " & ValueOfE
       MsgBox Msg     ' Display message.
End Sub
```



FileCopy Function

FileCopy(*sourcefile, destinationfile*)

Copies a file from source to destination.

The *sourcefile* and *destinationfile* parameters must be valid string expressions. *sourcefile* is the file name of the file to copy, *destinationfile* is the file name to be copied to.

Example:

```
Dim SourceFile, DestinationFile
SourceFile = "SRCFILE"    ' Define source file name.
DestinationFile = "DESTFILE"    ' Define target file name.
FileCopy SourceFile, DestinationFile    ' Copy source to target.
```

FileLen Function

FileLen(*filename*)

Returns a Long integer that is the length of the file in bytes

Related Topics: LOF Function

Example:

Sub Main

```
    Dim MySize
    MySize = FileLen("C:\TESTFILE")       ' Returns file length (bytes).
    Print MySize

End Sub
```

Fix Function

Fix(*number*)

Returns the integer portion of a number

Related Topics: Int

Example:

```
Sub Main

        Dim MySize
        MySize = Fix(4.345)
        Print MySize

End Sub
```

For each … Next Statement

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

Repeats the group of statments for each element in an array of a collection. For each … Next statements can be nested if each loop element is unique. The For Each…Next statement cannot be used with and array of user defined types.

Example:

```
Sub Main
  dim z(1 to 4) as double
  z(1) = 1.11
  z(2) = 2.22
  z(3) = 3.33
  For Each v In z
    Print v
  Next v
End Sub
```

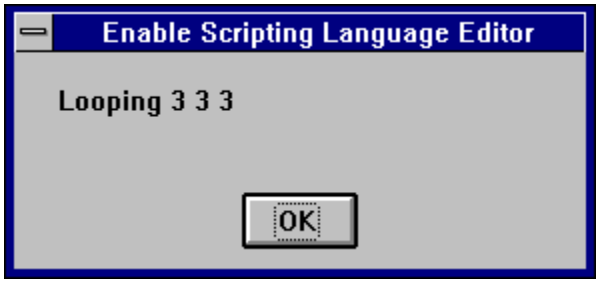
For...Next Statement

For *counter = expression1* to *expression2* [Step *increment]*
[*statements*]
Next [*counter*]

Repeats the execution of a block of statements for a specified number of times.

Example:

```
Sub main ()

Dim x,y,z

        For x = 1 to 5
                For y = 1 to 5
                        For z = 1 to 5
                        Print "Looping" ,z,y,x
                        Next z
                Next y
        Next x
End Sub
```



Format Function

Format (*expression* [,*fmt*])

Formats a string, number or variant datatype to a format expression.

Format returns returns a string

Part	**Description**
expression	Expression to be formatted.
fmt	A string of characters that specify how the expression is to displayed. or the name of a commonly-used format that has been predefined in Enable Basic. Do not mix different type format expressions in a single fmt parameter.

if the *fmt* parameter is omitted or is zero-length and the *expression* parameter is a numeric, **Format[$]** provides the same functionality as the **Str[$]** function by converting the numeric value to the appropriate return data type, Positive numbers convert to strings using **Format[$]** lack the leading space reserved for displaying the sign of the value, whereas those converted using **Str[$]** retain the leading space.

To format numbers, you can use the commonly-used formats that have been predefined in Enable Basic or you can create user-defined formats with standard characters that have special meaning when used in a format expression.

Predefined numeric format names:

Format Name	Description
General Number	Display the number as is, with no thousand Separators
Fixed	Display at least one digit to the left and two digits to the right of the decimal separator.
Standard	Display number with thousand separator, if appropriate; display two digits to the right of the decimal separator.
Percent	Display number multiplied by 100 with a percent sign (%) appended to the right' display two digits to the right of the decimal separator.

Format Name	Description
Scientific	Use standard scientific notation.
True/False	Display False if number is 0, otherwise display True.

The following shows the characters you can use to create user-defined number formats.

Character	Meaning
Null string	Display the number with no formatting.
0	Digit placeholder.

Display a digit or a zero

If the number being formatted has fewer digits than there are zeros (on either side of the decimal) in the format expression, leading or trailing

zeros are displayed. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, the number is rounded to as many decimal places as there are zeros. If the number has more digits to left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, the extra digits are displayed without modification.

Digit placeholder.

Displays a digit or nothing. If there is a digit in the expression being formatted in the position where the # appears in the format string, displays it; otherwise, nothing is displayed.

. Decimal placeholder.

The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator.

Character	Meaning

% Percentage placeholder.

The percent character (%) is inserted in the position where it appears in the format string. The expression is multiplied by 100.

, Thousand separator.

The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator.

Use of this separator as specified in the format statement contains a comma surrounded by digit placeholders(0 or #). Two adjacent commas or a comma immediately to the left of the decimal separator (whether or not a decimal is specified) means “scale the number by dividing it by 1000, rounding as needed.”

E-E+e-e+ Scientific format.

If the format expression contains at least one digit placeholder (0 or #) to the right of E-,E+,e- or e+, the number is displayed in scientific formatted E or e inserted between the number and its exponent. The number of digit placeholders to the right determines the number of digits in the exponent.

Use E- or e- to place a minus sign next to negative exponents. Use E+ or e+ to place a plus sign next to positive exponents.

: Time separator.

The actual character used as the time separator depends on the Time Format specified in the International section of the Control Panel.

/ Date separator.

The actual character used as the date separator in the formatted out depends on Date Format specified in the International section of the Control Panel.

Character	Meaning
- + $ ( ) space	Display a literal character. To display a character other than one of those listed, precede it with a backslash (\).
\	Display the next character in the format string. The backslash itself isn't displayed. To display a backslash, use two backslashes (\\). Examples of characters that can't be displayed as literal characters are the date- and time- formatting characters (a,c,d,h,m,n,p,q,s,t,w,y, and /:), the numeric -formatting characters(#,0,%,E,e,comma, and period), and the string-formatting characters (@,&,<,>, and !).
"String"	Display the string inside the double quotation marks. To include a string in *fmt* from within Enable, you must use the ANSI code for a double quotation mark Chr(34) to enclose the text.
*	Display the next character as the fill character. Any empty space in a field is filled with the character following the asterisk.

Unless the *fmt* argument contains one of the predefined formats, a format expression for numbers can have from one to four sections separated by semicolons.

If you use	The result is
One section only	The format expression applies to all values.
Two	The first section applies to positive values, the second to negative sections values.
Three	The first section applies to positive values, the second to negative sections values, and the third to zeros.
Four	The first section applies to positive values, the second to negative section values, the third to zeros, and the fourth to **Null** values.

The following example has two sections: the first defines the format for positive values and zeros; the second section defines the format for negative values.

"$#,##0; ($#,##0)"

If you include semicolons with nothing between them. the missing section is printed using the format of the positive value. For example, the following format displays positive and negative values using the format in the first section and displays "Zero" if the value is zero.

"$#,##0;;\Z\e\r\o"

Some sample format expressions for numbers are shown below. (These examples all assume the Country is set to United States in the International section of the Control Panel.) The first column contains the format strings. The other columns contain the output the results if the formatted data has the value given in the column headings

Format (*fmt*)	Positive 3	Negative 3	Decimal .3	Null
Null string	3	-3	0.3	
0	3	-3	1	
0.00	3.00	-3.00	0.30	

#,##0	3	-3	1	
#,##0.00;;;Nil	3.00	-3.00	0.30	Nil
$#,##0;($#,##0)	$3	($3)	$1	
$#,##0.00;($#,##0.00)	$3.00	($3.00)	$0.30	
0%	300%	-300%	30%	
0.00%	300.00%	-300.00%	30.00%	
0.00E+00	3.00E+00	-3.00E+00	3.00E-01	
0.00E-00	3.00E00	-3.00E00	3.00E-01	

Numbers can also be used to represent date and time information. You can format date and time serial numbers using date and time formats or number formats because date/time serial numbers are stored as floating-point values.

To format dates and times, you can use either the commonly used format that have been predefined or create user-defined time formats using standard meaning of each:

The following table shows the predefined data format names you can use and the meaning of each.

Format Name	Description
General	Display a date and/or time. for real numbers, display a date and time.(e.g. 4/3/93 03:34 PM); If there is no fractional part, display only a date (e.g. 4/3/93); if there is no integer part, display time only (e.g. 03:34 PM).
Long Date	Display a Long Date, as defined in the International section of the Control Panel.
Medium	Display a date in the same form as the Short Date, as defined in the international section of the Control Panel, except spell out the month abbreviation.

Short Date	Display a Short Date, as defined in the International section of the Control Panel.
Long Time	Display a Long Time, as defined in the International section of the Control panel. Long Time includes hours, minutes, seconds.
Medium Time	Display time in 12-hour format using hours and minuets and the Time AM/PM designator.
Short Time	Display a time using the 24-hour format (e.g. 17:45)

This table shows the characters you can use to create user-defined date/time formats.

Character	Meaning
c	Display the date as dddd and display the time as ttttt. in the order.
d	Display the day as a number without a leading zero (1-31).
dd	Display the day as a number with a leading zero (01-31).
ddd	Display the day as an abbreviation (Sun-Sat).
ddddd	Display a date serial number as a complete date (including day , month, and year).

Character	Meaning
w	Display the day of the week as a number (1- 7).
ww	Display the week of the year as a number (1-53).
m	Display the month as a number without a leading zero (1-12). If m immediately follows h or hh, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01-12). If mm immediately follows h or hh, the minute rather than the month is displayed.

mmm	Display the month as an abbreviation (Jan-Dec).
mmmm	Display the month as a full month name (January-December).
q	display the quarter of the year as a number (1-4).
y	Display the day of the year as a number (1-366).
yy	Display the day of the year as a two-digit number (00-99)
yyyy	Display the day of the year as a four-digit number (100-9999).
h	Display the hour as a number without leading zeros (0-23).
hh	Display the hour as a number with leading zeros (00-23).
n	Display the minute as a number without leading zeros (0-59).
nn	Display the minute as a number with leading zeros (00-59).
s	Display the second as a number without leading zeros (0-59).
ss	Display the second as a number with leading zeros (00-59).
ttttt	Display a time serial number as a complete time (including hour, minute, and second) formatted using the time separator defined by the Time Format in the International section of the Control Panel. A leading zero is displayed if the Leading Zero option is selected and the time is before 10:00 A.M. or P.M. The default time format is h:mm:ss.
AM/PM	Use the 12-hour clock and display an uppercase AM/PM
am/pm	Use the 12-hour clock display a lowercase am/pm
Character	**Meaning**
A/P	Use the 12-hour clock display a uppercase A/P
a/p	Use the 12-hour clock display a lowercase a/p

AMPM	Use the 12-hour clock and display the contents of the 11:59 string (s1159) in the WIN.INI file with any hour before noon; display the contents of the 2359 string (s2359) with any hour between noon and 11:59 PM. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as it exists in the WIN.INI file. The default format is AM/PM.

The Following are examples of user-defined date and time formats:

Format	Display
m/d/yy	2/26/65
d-mmmm-yy	26-February-65
d-mmmm	26 February
mmmm-yy	February 65
hh:nn	AM/PM 06:45 PM
h:nn:ss a/p	6:45:15 p
h:nn:ss	18:45:15
m/d/yy/h:nn	2/26/65 18:45

Strings can also be formatted with **Format[$].** A format expression for strings can have one section or two sections separated by a semicolon.

If you use	The result is
One section only	The format applies to all string data.
Two sections **Null**	The first section applies to string data, the second to values and zero-length strings.

The following characters can be used to create a format expression for strings:

@	Character placeholder.

Character	Meaning

@	Character placeholder. Displays a character or a space. Placeholders are filled from right to left unless there is an ! character in the format string.
&	Character placeholder. Display a character or nothing.
<	Force lowercase.
>	Force uppercase.
!	Force placeholders to fill from left to right instead of right to left.

Related Topic: **Str, Str$ Function.**.

Example:

```
' Format Function Example

' This example shows various uses of the Format function to format values
' using both named and user-defined formats.  For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running.  When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used.  When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale.  For this example, English/United States is assumed.

' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main

MyTime = "08:04:23 PM"
MyDate = "03/03/95"
MyDate = "January 27, 1993"

MsgBox Now
MsgBox MyTime

MsgBox Second( MyTime ) & " Seconds"
MsgBox Minute( MyTime ) & " Minutes"
MsgBox Hour( MyTime ) & " Hours"

MsgBox Day( MyDate ) & " Days"
MsgBox Month( MyDate ) & " Months"
MsgBox Year( MyDate ) & " Years"

' Returns current system time in the system-defined long time format.
MsgBox Format(Time, "Short Time")
```

```
MyStr = Format(Time, "Long Time")

' Returns current system date in the system-defined long date format.
MsgBox Format(Date, "Short Date")
MsgBox Format(Date, "Long Date")

MyStr Format(MyTime, "h:n:s")        ' Returns "17:4:23".
MyStr Format(MyTime, "hh:nn:ss")' Returns "20:04:22 ".
MyStr Format(MyDate, "dddd, mmm d yyyy")' Returns "Wednesday, Jan 27 1993".

' If format is not supplied, a string is returned.
MsgBox Format(23)                      ' Returns "23".

' User-defined formats.
MsgBox Format(5459.4, "##,##0.00")      ' Returns "5,459.40".
MsgBox Format(334.9, "###0.00")         ' Returns "334.90".
MsgBox Format(5, "0.00%")               ' Returns "500.00%".
MsgBox Format("HELLO", "<")             ' Returns "hello".
MsgBox Format("This is it", ">")        ' Returns "THIS IS IT".

End Sub
```

FreeFile Function

FreeFile

Returns an integer that is the next available file handle to be used by the Open Statement.

Related Topics: Open, Close, Write

Example:

```
Sub Main
Dim Mx, FileNumber
For Mx = 1 To 3
       FileNumber = FreeFile
       Open "c:\e1\TEST" & Mx For Output As #FileNumber
       Write #FileNumber, "This is a sample."
       Close #FileNumber
Next Mx

Open "c:\e1\test1" For Input As #1
Do While Not EOF(1)
       MyStr = Input(10, #1)
       MsgBox MyStr
Loop
Close #1


End Sub
```

Function Statement

Function *Fname* [(*Arguments*)] [As *type*]
[*statements*]
Functionname = expression
[*statements*]

Functionname = expression
End Function

Declares and defines a procedure that can receive arguments and return a value of a specified data type.

When the optional argument list needs to be passed the format is as follows:

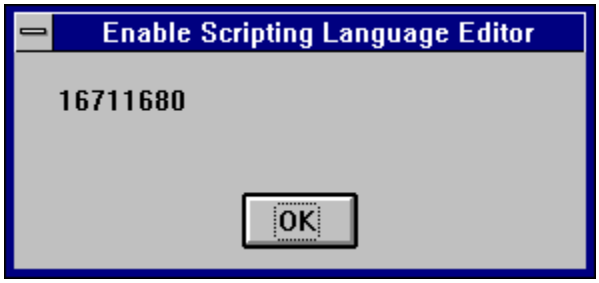
([ByVal] variable [As type] [,ByVal] variable [As type]]…])

The optional ByVal parameter specifies that the variable is [passed by value instead of by reference (see “ByRef and ByVal” in this manual). The optional As type parameter is used to specify the data type. Valid types are String, Integer, Double, Long, and Varaint (see “Variable Types” in this manual).

Related Topics: Dim, End, Exit, Sub

Example:

```
Sub Main
Dim I as integer
       For I = 1 to 10
       Print GetColor2(I)
       Next I
End Sub
Function GetColor2( c% ) As Long
   GetColor2 = c% * 25
   If c% > 2 Then
      GetColor2 = 255         ' 0x0000FF - Red
   End If
   If c% > 5 Then
      GetColor2 = 65280              ' 0x00FF00 - Green
   End If
   If c% > 8 Then
      GetColor2 = 16711680    ' 0xFF0000 - Blue
   End If
End Function
```



Get Statement

GetStatement [#] *filenmber*,[*recordnumber*], *variablename*

Reads from a disk file into a variable

The Get Statement has these parts:

Filenumber The number used to Open the file with.
Recordnumber For files opened in Binary mode *recordnumber* is the byte position where reading starts.
VariableName The name of the variable used to receive the data from the file.

Related Topics: Open, Put

Get Object Function

GetObject(*filename[,class])*

The GetObject Function has two parameters a filename and a class. The filename is the name of the file containing the object to retrieve. If filename is an empty string then class is required. Class is a string containing the class of the object to retrieve.

Related Topics: CreateObject

Global Statement

Global Const *constant*

The Global Statement must be outside the procedure section of the script. Global variables are available to all functions and subroutines in your program

Related Topics: Dim, Const and Type Statements

Example:

```
Global Const Height = 14.4357     '
Const PI = 3.14159      'Global to all procedures in a module
Sub Main ()
        Begin Dialog DialogName1 60, 60, 160,70, "ASC - Hello"
        TEXT 10, 10, 100, 20, "Please fill in the radius of circle x"
        TEXT 10, 40, 28, 12, "Radius"
        TEXTBOX 42, 40, 28, 12, .Radius
```

```
       OKBUTTON 42, 54,40, 12
End Dialog
Dim Dlg1 As DialogName1
Dialog Dlg1
CylArea = Height * (Dlg1.Radius * Dlg1.Radius) * PI
MsgBox "The volume of Cylinder x is " & CylArea
End Sub
```

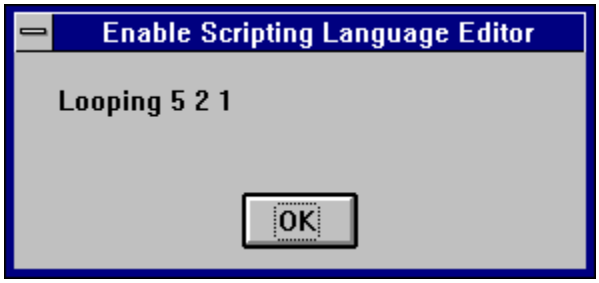
GoTo Statement

GoTo *label*

Branches unconditionally and without return to a specified label in a procedure.

Example:

```
Sub main ()

       Dim x,y,z

       For x = 1 to 5
              For y = 1 to 5
                     For z = 1 to 5
                     Print "Looping" ,z,y,x
                            If y > 3 Then
                                   GoTo Label1
                            End If
                     Next z
              Next y
       Next x
Label1:

End Sub
```



Hex

Hex (*num*)

Returns the hexadecimal value of a decimal parameter.

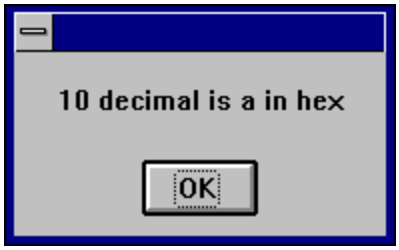
Hex returns a string

The parameter *num* can be any valid number. It is rounded to nearest whole number before evaluation.

Related Topics: Oct, Oct$

Example:

```
Sub Main ()

        Dim Msg As String, x%
        x% = 10
        Msg =Str( x%) &  " decimal is "
        Msg = Msg & Hex(x%) & " in hex "
        MsgBox Msg
End Sub
```



Hour Function

Hour(*string*)

The Hour Function returns an integer between 0 and 23 that is the hour of the day indicated in the parameter *number*.

The parameter string is any number expressed as a string that can represent a date and time from January 1, 1980 through December 31, 9999.

Example:

```
' This example shows various uses of the Format function to format values
' using both named and user-defined formats.  For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running.  When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used.  When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale.  For this example, English/United States is assumed.

' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main

MyTime = "08:04:23 PM"
MyDate = "03/03/95"
MyDate = "January 27, 1993"
```

```
MsgBox Now
MsgBox MyTime

MsgBox Second( MyTime ) & " Seconds"
MsgBox Minute( MyTime ) & " Minutes"
MsgBox Hour( MyTime ) & " Hours"

MsgBox Day( MyDate ) & " Days"
MsgBox Month( MyDate ) & " Months"
MsgBox Year( MyDate ) & " Years"

' Returns current system time in the system-defined long time format.
MsgBox Format(Time, "Short Time")
MyStr = Format(Time, "Long Time")

' Returns current system date in the system-defined long date format.
MsgBox Format(Date, "Short Date")
MsgBox Format(Date, "Long Date")

' This section not yet supported
'MyStr = Format(MyTime, "h:n:s")        ' Returns "17:4:23".
'MyStr = Format(MyTime, "hh:nn:ss AMPM")' Returns "05:04:23 PM".
'MyStr = Format(MyDate, "dddd, nnn d yyyy")' Returns "Wednesday, Jan 27 1993".

' If format is not supplied, a string is returned.
MsgBox Format(23)                       ' Returns "23".

' User-defined formats.
MsgBox Format(5459.4, "##,##0.00")      ' Returns "5,459.40".
MsgBox Format(334.9, "###0.00")         ' Returns "334.90".
MsgBox Format(5, "0.00%")               ' Returns "500.00%".
MsgBox Format("HELLO", "<")             ' Returns "hello".
MsgBox Format("This is it", ">")        ' Returns "THIS IS IT".

End Sub
```

HTMLDialog

HTMLDialog (*path*, *number*)

Runs a DHTML dialog that is specified in the path.

Example:

```
x =HtmlDialog( "c:\enable40\htmlt.htm", 57 )

‘See sample code on the samples disk htmldlg.bas
```

If...Then...Else Statement

Syntax 1 If *condition* Then *thenpart* [Else *elsepart*]

Syntax 2

If condition Then

.

[statement(s)]

.
ElseIf condition Then
.
  [statement(s)]
.
Else
.
  [statements(s)]
.
End If

Syntax 2
If conditional Then statement

Allows conditional statements to be executed in the code.

Related Topics: Select Case

Example:

```
Sub IfTest
        ' demo If...Then...Else
        Dim msg as String
        Dim nl as String
        Dim someInt as Integer

        nl = Chr(10)
        msg = "Less"
        someInt = 4

        If 5 > someInt Then msg = "Greater" : Beep
        MsgBox “” & msg

        If 3 > someInt Then
                msg = "Greater"
                Beep
        Else
                msg = "Less"
        End If
        MsgBox “” & msg

        If someInt = 1 Then
                msg = "Spring"
        ElseIf someInt = 2 Then
                msg = "Summer"
        ElseIf someInt = 3 Then
                msg = "Fall"
        ElseIf someInt = 4 Then
                msg = "Winter"
        Else
                msg = "Salt"
        End If
        MsgBox “” & msg

End Sub
```

Input # Statement

Input # *filenumber*, *variablelist*

Input # Statement reads data from a sequential file and assigns that data to variables.

The Input # Statement has two parameters *filenumber* and *variablelist*. *filenumber* is the number used in the open statement when the file was opened and *variablelist* is a Comma-delimited list of the variables that are assigned when read from the file..

Example:

```
Dim MyString, MyNumber
Open "c:\TESTFILE" For Input As #1   ' Open file for input.
Do While Not EOF(1)   ' Loop until end of file.
   Input #1, MyString, MyNumber   ' Read data into two variables.
Loop
Close #1   ' Close file.
```

Input Function

Input(*n* , [#] *filenumber*)

Input returns characters from a sequential file.

The input function has two parameters *n* and *filenumber*. *n* is the number of bytes to be read from a file and *filenumber* is the number used in the open statement when the file was opened.

Example:

```
Sub Main
    Open "TESTFILE" For Input As #1   ' Open file.
    Do While Not EOF(1)               ' Loop until end of file.
        MyStr = Input(10, #1)  ' Get ten characters.
        MsgBox MyStr
    Loop
    Close #1                  ' Close file.
End Sub
```

InputBox Function

InputBox(*prompt*[,[*title*][,[*default*][,*xpos*,*ypos*]]])

InputBox returns a String.

Prompt is string that is displayed usually to ask for input type or information.

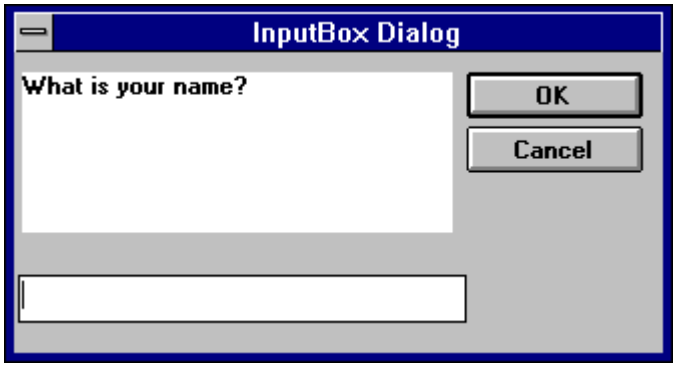
Title is a string that is displayed at the top of the input dialog box.

Default is a string that is displayed in the text box as the default entry.

Xpos and Ypos and the x and y coodinates of the relative location of the input dialog box.

Example:

```
Sub Main ()
        Title$ = "Greetings"
        Prompt$ = "What is your name?"
        Default$ = ""
        X% = 200
        Y% = 200
        N$ = InputBox$(Prompt$, Title$, Default$, X%, Y%)
End Sub
```



InStr

InStr(*numbegin*, *string1*, *string2*)

Returns the character position of the first occurrence of *string2* within *string1*.

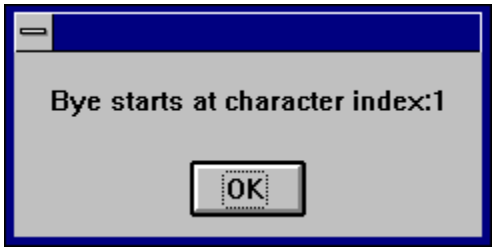
The *numbegin* parameter is not optional and sets the starting point of the search. *numbegin* must be a valid positive integer no greater than 65,535.

string1 is the string being searched and *string2* is the string we are looking for.

Related Topics: Mid Function

Example:

```
Sub Main ()
        B$ = "Good Bye"
        A% = InStr(2, B$, "Bye")
        C% = Instr(3, B$, "Bye")
End Sub
```



Int Function

Int(*number*)

Returns the integer portion of a number

Related Topics: Fix

IsArray Function

IsArray(*variablename*)

Returns a boolean value True or False indicating whether the parameter vaiablename is an array.

Related Topics: IsEmpty, IsNumeric, VarType, IsObject

Example:

```
Sub Main

        Dim MArray(1 To 5) As Integer, MCheck

        MCheck = IsArray(MArray)
        Print MCheck

End Sub
```

IsDate

IsDate(*variant*)

Returns a value that indicates if a variant parameter can be converted to a date.

Related Topics: IsEmpty, IsNumeric, VarType

Example:

```
Sub Main
Dim x As String

        Dim MArray As Integer, MCheck
        MArray = 345
        x = "January 1, 1987"
        MCheck = IsDate(MArray)
        MChekk = IsDate(x)
        MArray1 = CStr(MArray)
        MCheck1 = CStr(MCheck)
        Print MArray1 & " is a date " & Chr(10) & MCheck
        Print x & " is a date" & Chr(10) & MChekk
End Sub
```

IsEmpty

IsEmpty(*variant*)

Returns a value that indicates if a variant parameter has been initialized.

Related Topics: IsDate, IsNull, IsNumeric, VarType

Example:

```
' This sample explores the concept of an empty variant

Sub Main
    Dim x       ' Empty
    x = 5       ' Not Empty - Long
    x = Empty   ' Empty
    y = x       ' Both Empty
    MsgBox “x” & " IsEmpty: " & IsEmpty(x)
End Sub
```

IsNull

IsNull(*v*)

Returns a value that indicates if a variant contains the NULL value.

The parameter *v* can be any variant. IsNull returns a TRUE if *v* contains NULL. If isNull returns a FALSE the variant expression is not NULL.

The NULL value is special because it indicates that the *v* parameter contains no data. This is different from a null-string, which is a zero length string and an empty string which has not yet been initialized.

Related Topics: IsDate, IsEmpty, IsNumeric, VarType

IsNumeric

IsNumeric(*v*)

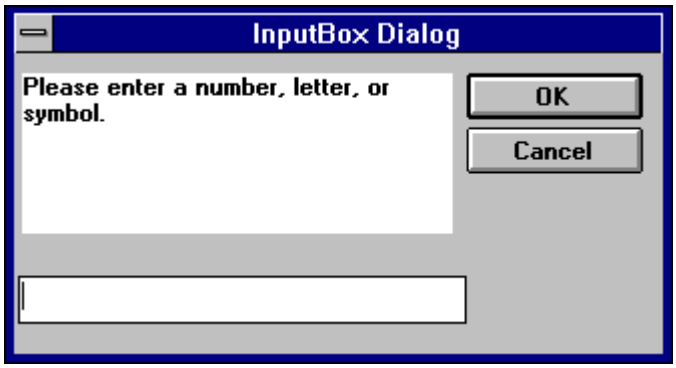
Returns a TRUE or FALSE indicating if the *v* parameter can be converted to a numeric data type.

The parameter *v* can be any variant, numeric value, Date or string (if the string can be interpreted as a numeric).

Related topics: IsDate, IsEmpty, IsNull, VarType

Example:

```
Sub Form_Click ()
        Dim TestVar    ' Declare variable.
        TestVar = InputBox("Please enter a number, letter, or symbol.")
        If IsNumeric(TestVar) Then    ' Evaluate variable.
                MsgBox "Entered data is numeric." ‘ Message if number.
        Else
                MsgBox "Entered data is not numeric." ' Message if not.
        End If
End Sub
```



IsObject Function

IsObject(*objectname*)

Returns a boolean value True or False indicating whether the parameter objectname is an object.

Related Topics: IsEmpty, IsNumeric, VarType, IsObject

Example:

```
Sub Main

        Dim MyInt As Integer, MyCheck
        Dim MyObject As Object
        Dim YourObject As Object
       Set MyObject = CreateObject("Word.Basic")

        Set YourObject = MyObject
        MyCheck = IsObject(YourObject)

      Print MyCheck

End Sub
```

Kill Statement

Kill *filename*

Kill will only delete files. To remove a directory use the RmDir Statement

Related Topics: RmDir

Example:

```
Const NumberOfFiles = 3

Sub Main ()
    Dim Msg                   ' Declare variable.
    Call MakeFiles()          ' Create data files.
    Msg = "Several test files have been created on your disk. You may see "
    Msg = Msg & "them by switching tasks. Choose OK to remove the test files."
    MsgBox Msg
    For I = 1 To NumberOfFiles
        Kill "TEST" & I               ' Remove data files from disk.
    Next I
End Sub

Sub MakeFiles ()
    Dim I, FNum, FName                ' Declare variables.
    For I = 1 To NumberOfFiles
        FNum = FreeFile               ' Determine next file number.
        FName = "TEST" & I
        Open FName For Output As FNum ' Open file.
        Print #FNum, "This is test #" & I    ' Write string to file.
        Print #FNum, "Here is another ";  "line";  I
    Next I
    Close                             ' Close all files.
End Sub
```

LBound Function

LBound(*array [,dimension]*)

Returns the smallest available subscript for the dimension of the indicated array.

Related Topics: UBound Function

Example:

```
' This example demonstrates some of the  features of arrays.  The lower bound
' for an array is 0 unless it is specified or option base has set as is
' done in this example.

Option Base 1

Sub Main
    Dim a(10) As Double
    MsgBox "LBound: " & LBound(a) & " UBound: " & UBound(a)
    Dim i As Integer
    For i = 0 to 3
        a(i) = 2 + i * 3.1
    Next i
    Print a(0),a(1),a(2), a(3)
End Sub
```

LCase, Function

Lcase[$](*string*)

Returns a string in which all letters of the string parameter have been converted to upper case.

Related Topics: Ucase Function

Example:

```
' This example uses the LTrim and RTrim functions to strip leading and
' trailing spaces, respectively, from a string variable.  It
' uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

Sub Main
    MyString = "  <-Trim->  " ' Initialize string.
    TrimString = LTrim(MyString)     ' TrimString = "<-Trim->  ".
    MsgBox "|" & TrimString & "|"
    TrimString = LCase(RTrim(MyString))      ' TrimString = "  <-trim->".
    MsgBox "|" & TrimString & "|"
    TrimString = LTrim(RTrim(MyString))      ' TrimString = "<-Trim->".
    MsgBox "|" & TrimString & "|"
    ' Using the Trim function alone achieves the same result.
    TrimString = UCase(Trim(MyString))       ' TrimString = "<-TRIM->".
    MsgBox "|" & TrimString & "|"
End Sub
```

Left

Left(*string*, *num*)

Returns the left most *num* characters of a string parameter.

Left returns a Variant, Left$ returns a String

Example:

```
Sub Main ()
        Dim LWord, Msg, RWord, SpcPos, UsrInp ' Declare variables.
        Msg = "Enter two words separated by a space."
        UsrInp = InputBox(Msg) ' Get user input.
        print UsrInp
        SpcPos = InStr(1, UsrInp, " ")        ' Find space.
        If SpcPos Then
                LWord = Left(UsrInp, SpcPos - 1)      ' Get left word.
                print "LWord: ";  LWord
                RWord = Right(UsrInp, Len(UsrInp) - SpcPos)  ' Get right word.
                Msg = "The first word you entered is " & LWord
                Msg = Msg & "." & " The second word is "
                Msg = "The first word you entered is <" & LWord & ">"
                Msg = Msg & RWord & "."
        Else
                Msg = "You didn't enter two words."
        End If
        MsgBox Msg     ' Display message.
        MidTest = Mid("Mid Word Test", 4, 5)
        Print MidTest
End Sub
```

Len

Len(*string*)

Returns the number of characters in a string.

Related Topics: InStr

Example:

```
Sub Main ()
        A$ = "Cypress Enable"
        StrLen% = Len(A$)      'the value of StrLen is 14
        MsgBox StrLen%
End Sub
```

Let Statement

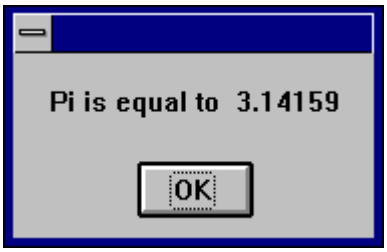
[Let] *variablename* = *expression*

Let assigns a value to a variable.

Let is an optional keyword that is rarely used. The Let statement is required in older versions of BASIC.

Example:

```
Sub Form_Click ()
    Dim Msg, Pi         ' Declare variables.
    Let Pi = 4 * Atn(1)    ' Calculate Pi.
    Msg = "Pi is equal to " & Str(Pi)
    MsgBox Msg          ' Display results.
End Sub
```



Line Input # Statement

Line Input # *filenumber* and *name*

Reads a line from a sequential file into a String or Variant variable.

The parameter *filenumber* is used in the open statement to open the file. The parameter name is the name of a variable used to hold the line of text from the file.

Related Topics: Open

Example:

```
' Line Input # Statement Example:
' This example uses the Line Input # statement to read a line from a
' sequential file and assign it to a variable. This example assumes that
' TESTFILE is a text file with a few lines of sample data.

Sub Main
    Open "TESTFILE" For Input As #1   ' Open file.
    Do While Not EOF(1)               ' Loop until end of file.
        Line Input #1, TextLine       ' Read line into variable.
        Print TextLine         ' Print to Debug window.
    Loop
    Close #1   ' Close file.

End Sub
```

LOF

LOF(*filenumber*)

Returns a long number for the number of bytes in the open file.
The parameter filenumber is required and must be an integer.

Related Topics: FileLen

Example:

```
Sub Main

       Dim FileLength
       Open "TESTFILE" For Input As #1
       FileLength = LOF(1)
      Print FileLength
       Close #1

End Sub
```

Log

Log(*num*)

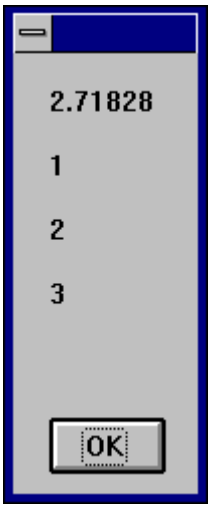
Returns the natural log of a number
The parameter *num* must be greater than zero and be a valid number.

Related Topics: Exp, Sin, Cos

Example:

```
Sub Form_Click ( )
       Dim I, Msg, NL
       NL = Chr(13) & Chr(10)
       Msg = Exp(1) & NL
       For I = 1 to 3
       Msg = Msg & Log(Exp(1) ^ I ) & NL
       Next I
       MsgBox Msg
```

```
End Sub
```



Mid Function

string = Mid(*strgvar*,*begin*,*length*)

Returns a substring within a string.

Example:

```
Sub Main ()
        Dim LWord, Msg, RWord, SpcPos, UsrInp ' Declare variables.
        Msg = "Enter two words separated by a space."
        UsrInp = InputBox(Msg) ' Get user input.
        print UsrInp
        SpcPos = InStr(1, UsrInp, " ")        ' Find space.
        If SpcPos Then
                LWord = Left(UsrInp, SpcPos - 1)      ' Get left word.
                print "LWord: ";  LWord
                RWord = Right(UsrInp, Len(UsrInp) - SpcPos)  ' Get right word.
                Msg = "The first word you entered is " & LWord
                Msg = Msg & "." & " The second word is "
                Msg = "The first word you entered is <" & LWord & ">"
                Msg = Msg & RWord & "."
        Else
                Msg = "You didn't enter two words."
        End If
        MsgBox Msg     ' Display message.
        MidTest = Mid("Mid Word Test", 4, 5)
        Print MidTest
End Sub
```

Minute Function

Minute(*string*)

Returns an integer between 0 and 59 representing the minute of the hour.

```
' Format Function Example
```

```
' This example shows various uses of the Format function to format values
' using both named and user-defined formats.  For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running.  When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used.  When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale.  For this example, English/United States is assumed.

' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main

MyTime = "08:04:23 PM"
MyDate = "03/03/95"
MyDate = "January 27, 1993"

MsgBox Now
MsgBox MyTime

MsgBox Second( MyTime ) & " Seconds"
MsgBox Minute( MyTime ) & " Minutes"
MsgBox Hour( MyTime ) & " Hours"

MsgBox Day( MyDate ) & " Days"
MsgBox Month( MyDate ) & " Months"
MsgBox Year( MyDate ) & " Years"

End Sub
```

MkDir

MkDir *path*

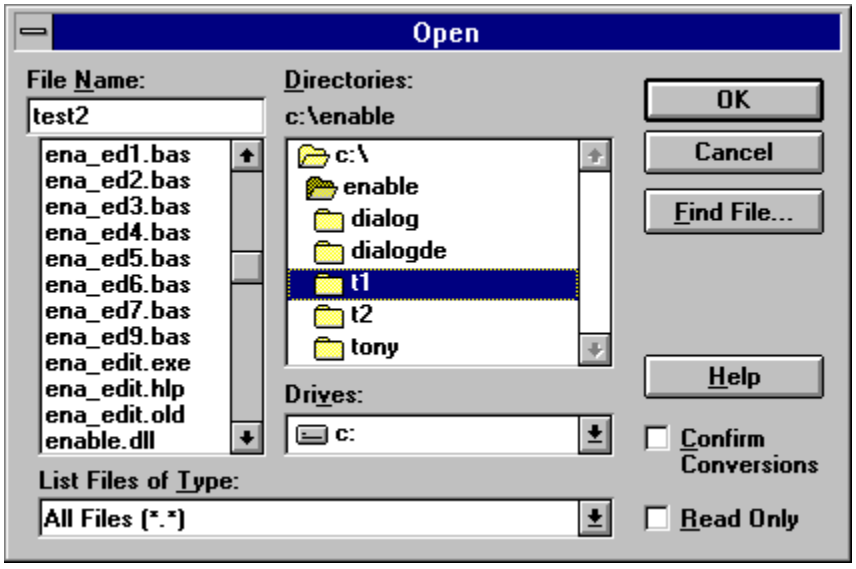
Creates a new directory.

The parameter *path* is a string expression that must contain fewer than 128 characters.

Example:

```
Sub Main
   Dim DST As String

   DST = "t1"
```

```
   mkdir DST
   mkdir "t2"
End Sub
```



Month Function

Month(*number*)

Returns an integer between 1 and 12, inclusive, that represents the month of the year.

Related Topics: Day, Hour, Weekday, Year

Example:

```
Sub Main
        MyDate = "03/03/96"
        print MyDate
        x = Month(MyDate)
        print x

End Sub
```

MsgBox Function MsgBox Statement

MsgBox (*msg, [type] [, title]*)

Displays a message in a dialog box and waits for the user to choose a button.

The first parameter *msg* is the string displayed in the dialog box as the message. The second and third parameters are optional and respectively designate the type of buttons and the title displayed in the dialog box.

MsgBox Function returns a value indicating which button the user has chosen; the MsgBox statement does not.

Value	Meaning
0	Display OK button only.
1	Display OK and Cancel buttons.
2	Display Abort, Retry, and Ignore buttons.
3	Display Yes, No, and Cancel buttons.
4	Display Yes and No buttons.
5	Display Retry and Cancel buttons.
16	Stop Icon
32	Question Icon
48	Exclamation Icon
64	Information Icon
0	First button is default.
256	Second button is default.
512	Third button is default.
768	Fourth button is default
0	Application modal.
4096	System modal

The first group of values (1-5) describes the number and type of buttons displayed in the dialog box; the second group (16, 32, 48, 64) describes the icon style; the third group (0, 256, 512) determines which button is the default; and the fourth group (0, 4096) determines the modality of the message box. When adding numbers to create a final value for the argument type, use only one number from each group. If omitted, the default value for type is 0.

title:
String expression displayed in the title bar of the dialog box. If you omit the argument title, MsgBox has no default title.

The value returned by the MsgBox function indicates which button has been selected, as shown below:

Value	Meaning
1	OK button selected.
2	Cancel button selected.
3	Abort button selected.
4	Retry button selected.
5	Ignore button selected.
6	Yes button selected.
7	No button selected.

If the dialog box displays a Cancel button, pressing the Esc key has the same effect as choosing Cancel.

MsgBox Function, MsgBox Statement Example

The example uses MsgBox to display a close without saving message in a dialog box with a Yes button a No button and a Cancel button. The Cancel button is the default response. The MsgBox function returns a value based on the button chosen by the user. The MsgBox statement uses that value to display a message that indicates which button was chosen.

Related Topics: InputBox, InputBox$ Function

Example:

```
Dim Msg, Style, Title, Help, Ctxt, Response, MyString
Msg = "Do you want to continue ?"   ' Define message.
'Style = vbYesNo + vbCritical + vbDefaultButton2   ' Define
buttons.
Style = 4 + 16 + 256   ' Define buttons.
Title = "MsgBox Demonstration"   ' Define title.
Help = "DEMO.HLP"   ' Define Help file.
Ctxt = 1000   ' Define topic
      ' context.
      ' Display message.
Response = MsgBox(Msg, Style, Title, Help, Ctxt)
If Response = vbYes Then   ' User chose Yes.
   MyString = "Yes"   ' Perform some action.
Else   ' User chose No.
   MyString = "No"   ' Perform some action.
End If
```

Name Statement

Name *oldname* As *newname*

Changes the name of a directory or a file.

The parameters *oldname* and *newname* are strings that can optionally contain a path.

Related Topics: Kill, ChDir

Now Function

Now

Returns a date that represents the current date and time according to the setting of the computer's system date and time

The Now function returns a Variant data type containing a date and time that are stored internally as a double. The number is a date and time from January 1, 100 through December 31, 9999, where January 1, 1900 is 2. Numbers to the left of the decimal point represent the date and numbers to the right represent the time.

Related Topics:

Example:

```
Sub Main ()
        Dim Today
        Today = Now
End Sub
```

Oct Function

Oct (*num*)

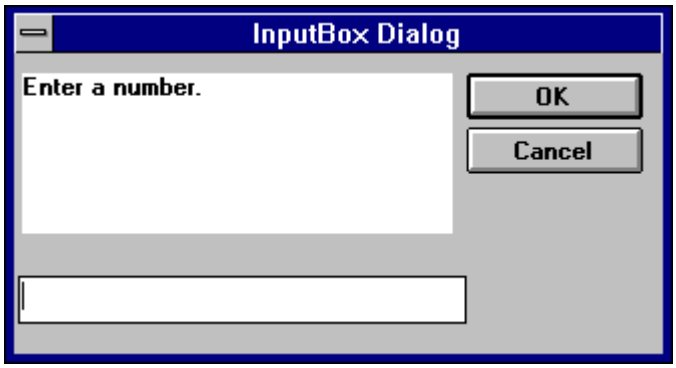
Returns the octal value of the decimal parameter

Oct returns a string

Related Topics: Hex

Example:

```
Sub Main ()
        Dim Msg, Num    ' Declare variables.
        Num = InputBox("Enter a number.")     ' Get user input.
        Msg = Num & " decimal is &O"
        Msg = Msg & Oct(Num) & " in octal notation."
        MsgBox Msg      ' Display results.
End Sub
```

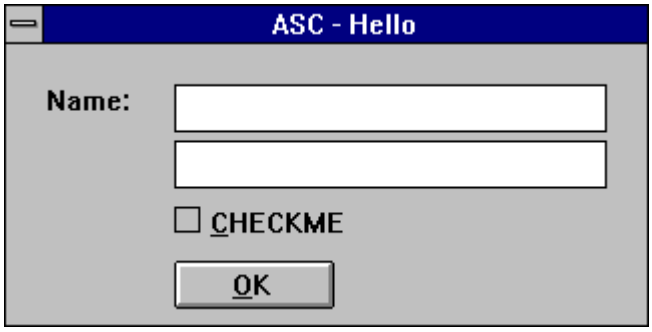


OKButton

OKBUTTON starting x position, starting y position, width, Height

For selecting options and closing dialog boxes

```
Sub Main ()
    Begin Dialog DialogName1 60, 60, 160, 70, "ASC - Hello"
        TEXT 10, 10, 28, 12, "Name:"
        TEXTBOX 42, 10, 108, 12, .nameStr
        TEXTBOX 42, 24, 108, 12, .descStr
        CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
        OKBUTTON 42, 54, 40, 12
    End Dialog
    Dim Dlg1 As DialogName1
    Dialog Dlg1

    MsgBox Dlg1.nameStr
    MsgBox Dlg1.descStr
    MsgBox Dlg1.checkInt
End Sub
```



On Error

On Error { *GoTo line* | *Resume Next* | *GoTo 0* }

Enables error-handling routine and specifies the line label of the error-handling routine.

Related Topics: Resume

The line parameter refers to a label. That label must be present in the code or an error is generated.

Example:

```
Sub Main
  On Error GoTo dude
  Dim x as object
  x.draw        ' Object not set
  jpe           ' Undefined function call
  print 1/0     ' Division by zero
  Err.Raise 6  ' Generate an "Overflow" error
  MsgBox "Back"
  MsgBox "Jack"
  Exit Sub
dude:
  MsgBox "HELLO"
  Print Err.Number, Err.Description
  Resume Next
  MsgBox "Should not get here!"
  MsgBox "What?"
End Sub
```

Errors can be raised with the syntax:

Err.Raise x

The list below shows the corresponding descriptions for the defined values of x.

5: "Invalid procedure call";
6: "Overflow";
7: "Out of memory";
9: "Subscript out of range";
10: "Array is fixed or temporarily locked";
11: "Division by zero";
13: "Type mismatch";
14: "Out of string space";
16: "Expression too complex";
17: "Can't perform requested operation";
18: "User interrupt occurred";

```
20: "Resume without error";
28: "Out of stack space";
35: "Sub, Function, or Property not defined";
47: "Too many DLL application clients";
48: "Error in loading DLL";
49: "Bad DLL calling convention";
51: "Internal error";
52: "Bad file name or number";
53: "File not found";
54: "Bad file mode";
55: "File already open";
57: "Device I/O error";
58: "File already exists";
59: "Bad record length";
60: "Disk full";
62: "Input past end of file";
63: "Bad record number";
67: "Too many files";
68: "Device unavailable";
70: "Permission denied";
71: "Disk not ready";
74: "Can't rename with different drive";
75: "Path/File access error";
76: "Path not found";
91: "Object variable or With block variable not set";
92: "For loop not initialized";
93: "Invalid pattern string";
94: "Invalid use of Null";
// OLE Automation Messages
429: "OLE Automation server cannot create object";
430: "Class doesn't support OLE Automation";
432: "File name or class name not found during OLE Automation operation";
438: "Object doesn't support this property or method";
440: "OLE Automation error";
443: "OLE Automation object does not have a default value";
445: "Object doesn't support this action";
446: "Object doesn't support named arguments";
447: "Object doesn't support current local setting";
448: "Named argument not found";
449: "Argument not optional";
450: "Wrong number of arguments";
451: "Object not a collection";
// Miscellaneous Messages
444: "Method not applicable in this context";
452: "Invalid ordinal";
```

453: "Specified DLL function not found";
457: "Duplicate Key";
460: "Invalid Clipboard format";
461: "Specified format doesn't match format of data";
480: "Can't create AutoRedraw image";
481: "Invalid picture";
482: "Printer error";
483: "Printer driver does not supported specified property";
484: "Problem getting printer information from from the system.";
// Make sure the printer is setp up correctly.
485: "invalid picture type";
520: "Can't empty Clipboard";
521: "Can't open Clipboard";

Open Statement

Open filename$ [For *mode*] [Access *access*] As [#]*filenumber*

Opens a file for input and output operations.

You must open a file before any I/O operation can be performed on it.The Open statement has these parts:

Part	Description
file	File name or path.
*mod*e	Reserved word that specifies the file mode: **Append, Binary Input, Output**
access	Reserved word that specifies which operations are permitted on the **open file: Read, Write.**
filenumber	Integer expression with a value between 1 and 255, inclusive. When an Open statement is executed, filenumber is associated with the file as long as it is open. Other I/O statements can use the number to refer to the file.

If file doesn't exist, it is created when a file is opened for Append, Binary or Output modes.

The argument mode is a reserved word that specifies one of the following file modes.

Mode	Description
Input	Sequential input mode.
Output	Sequential output mode.

Append Sequential output mode. Append sets the file pointer to the end of the file. A Print # or Write # statement then extends (appends to) the file.

The argument access is a reserved word that specifies the operations that can be performed on the opened file. If the file is already opened by another process and the specified type of access is not allowed, the Open operation fails and a Permission denied error occurs. The Access clause works only if you are using a version of MS-DOS that supports networking (MS-DOS version 3.1 or later). If you use the Access clause with a version of MS-DOS that doesn't support networking, a feature unavailable error occurs. The argument access can be one of the following reserved words.

Access type	Description
Read	Opens the file for reading only.
Write	Opens the file for writing only.
Read Write	Opens the file for both reading and riting. This mode is valid only for Random and Binary files and files opened for Append mode.

The following example writes data to a test file and reads it back.

Example :

```
Sub Main ()

        Open "TESTFILE" For Output As #1      ' Open to write file.
        userData1$ = InputBox("Enter your own text here")
        userData2$ = InputBox("Enter more of your own text here")
        Write #1, "This is a test of the Write # statement."
        Write #1,userData1$, userData2
        Close #1

        Open "TESTFILE" for Input As #2       ' Open to read file.
        Do While Not EOF(2)
```

```
                Line Input #2, FileData       ' Read a line of data.
                PRint FileData        ' Construct message.

        Loop
        Close #2              ' Close all open files.
        MsgBox "Testing Print Statement"      ' Display message.
        Kill "TESTFILE"       ' Remove file from disk.
End Sub
```

Option Base Statement

Option Base *number*

Declares the default lower bound for array subscripts.

The Option Base statement is never required. If used, it can appear only once in a module, it can occur only in the Declarations section, and must be used before you declare the dimensions of any arrays.

The value of number must be either 0 or 1. The default base is 0.

The To clause in the Dim, Global, and Static statements provides a more flexible way to control the range of an array's subscripts. However, if you don't explicitly set the lower bound with a To clause, you can use Option Base to change the default lower bound to 1.

The example uses the Option Base statement to override the default base array subscript value of 0.

Related Topics: Dim, Global and Lbound Statements

Example :

```
Option Base 1  ' Module level statement.
Sub Main
        Dim A(), Msg, NL      ' Declare variables.
        NL = Chr(10)   ' Define newline.
        ReDim A(20)    ' Create an array.
        Msg = "The lower bound of the A array is " & LBound(A) & "."
        Msg = Msg & NL & "The upper bound is " & UBound(A) & "."
        MsgBox Msg     ' Display message.
End Sub
```

Option Explicit Statement

Option Explicit

Forces explicit declaration of all variables.

The Option explicit statement is used outside of the script in the declarations section. This statement can be contained in a declare file or outside of any script in a file or buffer. If this statement is contained in the middle of a file the rest of the compile buffer will be affected.

Related Topics: Const and Global Statements

Example :

```
Option Explicit
Sub Main
       Print y    'because y is not explicitly dimmed an error will occur.

End Sub
```

Print Method

Print [*expr, expr...*] Print a string to an object.

Related Topics:

Example:

```
Sub PrintExample ()
    Dim Msg, Pi                        ' Declare variables.
    Let Pi = 4 * _Atn(1)               ' Calculate Pi.
    Msg = "Pi is equal to " & Str(Pi)
    MsgBox Msg                         ' Display results.
    Print Pi                  'Pints the results in the
                                       ' compiler messages window
End Sub
```

Print # Statement

Print # *filenumber*, [[{Spc(*n*) | Tab(*n*)}][*expressionlist*] [{; | ,}]]

Writes data to a sequential file.

Print statement Description:

filenumber:
Number used in an Open statement to open
a sequential file. It can be any
number of an open file. Note that the
number sign (#) preceding filenumber is not optional.

Spc(*n*):
Name of the Basic function optionally used to insert *n* spaces into the printed output. Multiple use is permitted.

Tab(*n*):
Name of the Basic function optionally used to tab to the *nth* column before printing
expressionlist. Multiple use is permitted.

expressionlist :
Numeric and/or string expressions to be written to the file.

{;|,}
Character that determines the position of the next character printed. A semicolon means the next character is printed immediately after the last character; a comma means the next character is printed at the start of the next print zone. Print zones begin every 14 columns. If neither character is specified, the next character is printed on the next line.

If you omit expressionlist, the Print # statement prints a blank line in the file, but you must include the comma. Because Print # writes an image of the data to the file, you must delimit the data so it is printed correctly. If you use commas as delimiters, Print # also writes the blanks between print fields to the file.

The Print # statement usually writes Variant data to a file the same way it writes any other data type. However, there are some exceptions:

If the data being written is a Variant of VarType 0 (Empty), Print # writes nothing to the file for that data item.

If the data being written is a Variant of VarType 1 (Null), Print # writes the literal #NULL# to the file.

If the data being written is a Variant of VarType 7 (Date), Print # writes the date to the file using the Short Date format defined in the WIN.INI file. When

either the date or the time component is missing or zero, Print # writes only the part provided to the file.

The following example writes data to a test file.

Example :

```
Sub Main
        Dim I, FNum, FName     ' Declare variables.
        For I = 1 To 3
                FNum = FreeFile        ' Determine next file number.
                FName = "TEST" & FNum
                Open FName For Output As FNum ' Open file.
                Print #I, "This is test #" & I        ' Write string to file.
                Print #I, "Here is another ";  "line";  I
        Next I
        Close   ' Close all files.
End Sub
```

The following example writes data to a test file and reads it back.

```
Sub Main ()
        Dim FileData, Msg, NL  ' Declare variables.
        NL = Chr(10)   ' Define newline.
        Open "TESTFILE" For Output As #1      ' Open to write file.
        Print #2, "This is a test of the Print # statement."
        Print #2,              ' Print blank line to file.
        Print #2, "Zone 1", "Zone 2"  ' Print in two print zones.
        Print #2, "With no space between"  ;  "." ' Print two strings together.
        Close
        Open "TESTFILE" for Input As #2       ' Open to read file.
        Do While Not EOF(2)
                Line Input #2, FileData       ' Read a line of data.
                Msg = Msg & FileData & NL     ' Construct message.
                MsgBox Msg
        Loop
        Close          ' Close all open files.
        MsgBox "Testing Print Statement"      ' Display message.
        Kill "TESTFILE"        ' Remove file from disk.
End Sub
```

Randomize Statement

Randomize[*number*]

Used to Initialize the random number generator.

The Randomize statement has one optional parameter *number*. This parameter can be any valid number and is used to initialize the random number generator. If you omit the parameter then the value returned by the Timer function is used as the default parameter to seed the random number generator.

Example:

```
Sub Main
```

```
        Dim MValue

        Randomize   ' Initialize random-number generator.
        MValue = Int((6 * Rnd) + 1)
        Print MValue

End Sub
```

ReDim Statement

ReDim *varname(subscripts)[As Type][,varname(subscripts*)]

Used to declare dynamic arrays and reallocate storage space.

The ReDim statement is used to size or resize a dynamic array that has already been declared using the Dim statement with empty parentheses. You can use the ReDim statement to repeatedly change the number of elements in and array but not to change the number of dimensions in an array or the type of the elements in the array.

Example:

```
Sub Main

Dim TestArray() As Integer
Dim I
ReDim TestArray(10)
For I = 1 To 10
      TestArray(I) = I + 10
      Print TestArray(I)
Next I

End Sub
```

Rem Statement

Rem *remark* ‘remark

Used to include explanatory remarks in a program.

The parameter *remark* is the text of any comment you wish to include in the code.

Example:

```
Rem This is a remark

Sub Main()

    Dim Answer, Msg                                 ' Declare variables.
    Do
```

```
        Answer = InputBox("Enter a value from 1 to 3.")
        Answer = 2
        If Answer >= 1 And Answer <= 3 Then      ' Check range.
            Exit Do                               ' Exit Do...Loop.
        Else
            Beep                                  ' Beep if not in range.
        End If
    Loop
    MsgBox "You entered a value in the proper range."
End Sub
```

Right Function

Right (*stringexpression, n*)

Returns the right most n characters of the string parameter.

The parameter *stringexpression* is the string from which the rightmost characters are returned.

The parameter *n* is the number of characters that will be returned and must be a long integer.

Related Topics: Len, Left, Mid Functions.

Example:

```
' The example uses the Right function to return the first of two words
' input by the user.

Sub Main ()
        Dim LWord, Msg, RWord, SpcPos, UsrInp ' Declare variables.
        Msg = "Enter two words separated by a space."
        UsrInp = InputBox(Msg) ' Get user input.
        print UsrInp
        SpcPos = InStr(1, UsrInp, " ")       ' Find space.
        If SpcPos Then
                LWord = Left(UsrInp, SpcPos - 1)     ' Get left word.
                print "LWord: ";  LWord
                RWord = Right(UsrInp, Len(UsrInp) - SpcPos)  ' Get right word.
                Msg = "The first word you entered is " & LWord
                Msg = Msg & "." & " The second word is "
                Msg = "The first word you entered is <" & LWord & ">"
                Msg = Msg & RWord & "."
        Else
                Msg = "You didn't enter two words."
        End If
        MsgBox Msg     ' Display message.
End Sub
```

RmDir Statement

RmDir *path*

Removes an existing directory.

The parameter *path* is a string that is the name of the directory to be removed.

Related Topics: ChDir, CurDir

Example:

```
' This sample shows the functions mkdir (Make Directory)
' and rmdir (Remove Directory)

Sub Main
   Dim dirName As String

   dirName = "t1"
   mkdir dirName
   mkdir "t2"
   MsgBox "Directories: t1 and t2 created. Press OK to remove them"
   rmdir "t1"
   rmdir "t2"
End Sub
```

Rnd Function

Rnd (*number*)

Returns a random number.

The parameter *number* must be a valid numeric expression.

Example:

'Rnd Function Example

```
'The example uses the Rnd function to simulate rolling a pair of dice by
'generating random values from 1 to 6.  Each time this program is run,


Sub Main ()
        Dim Dice1, Dice2, Msg  ' Declare variables.
        Dice1 = CInt(6 * Rnd() + 1)   ' Generate first die value.
        Dice2 = CInt(6 * Rnd() + 1)   ' Generate second die value.
        Msg = "You rolled a " & Dice1
        Msg = Msg & " and a " & Dice2
        Msg = Msg & " for a total of "
        Msg = Msg & Str(Dice1 + Dice2) & "."
        MsgBox Msg     ' Display message.
End Sub
```

Second Function

Second (*number*)

Returns an integer that is the second portion of the minute in the time parameter.

The parameter *number* must be a valid numeric expression.

Related Topics: Day, Hour, Minute, Now.

Example:

```
' Format Function Example

' This example shows various uses of the Format function to format values
' using both named and user-defined formats.  For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running.  When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used.  When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale.  For this example, English/United States is assumed.

' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main

MyTime = "08:04:23 PM"
MyDate = "03/03/95"
MyDate = "January 27, 1993"

MsgBox Now
MsgBox MyTime

MsgBox Second( MyTime ) & " Seconds"
MsgBox Minute( MyTime ) & " Minutes"
MsgBox Hour( MyTime ) & " Hours"

MsgBox Day( MyDate ) & " Days"
MsgBox Month( MyDate ) & " Months"
MsgBox Year( MyDate ) & " Years"

' Returns current system time in the system-defined long time format.
MsgBox Format(Time, "Short Time")
MyStr = Format(Time, "Long Time")

' Returns current system date in the system-defined long date format.
MsgBox Format(Date, "Short Date")
MsgBox Format(Date, "Long Date")

'This section not yet supported
MsgBox Format(MyTime, "h:n:s")        ' Returns "17:4:23".
MsgBox Format(MyTime, "hh:nn:ss")' Returns "05:04:23".
MsgBox Format(MyDate, "dddd, mmm d yyyy")' Returns "Wednesday, Jan 27 1993".

' If format is not supplied, a string is returned.
MsgBox Format(23)                       ' Returns "23".

' User-defined formats.
MsgBox Format(5459.4, "##,##0.00")      ' Returns "5,459.40".
MsgBox Format(334.9, "###0.00")         ' Returns "334.90".
MsgBox Format(5, "0.00%")               ' Returns "500.00%".
MsgBox Format("HELLO", "<")             ' Returns "hello".
MsgBox Format("This is it", ">")        ' Returns "THIS IS IT".

End Sub
```

Seek Function

Seek (*filenumber*)

The parameter *filenumber* is used in the open statement and must be a valid numeric expression.

Seek returns a number that represents the byte position where the next operation is to take place. The first byte in the file is at position 1.

Related Topics: Open

Example:

```
Sub Main
    Open "TESTFILE" For Input As #1  ' Open file for reading.
    Do While Not EOF(1)              ' Loop until end of file.
       MyChar = Input(1, #1)  ' Read next character of data.
       Print Seek(1)          ' Print byte position  .
    Loop
    Close #1                 ' Close file.
End Sub
```

Seek Statement

Seek *filenumber, position*

The parameter *filenumber* is used in the open statement and must be a valid numeric expression, the parameter *position* is the number that indicates where the next read or write is to occur. In Cypress Enable Basic position is the byte position relative to the beginning of the file.

Seek statement sets the position in a file for the next read or write

Related Topics: Open

Example:

```
Sub Main
    Open "TESTFILE" For Input As #1   ' Open file for reading.
    For i = 1 To 24 Step 3            ' Loop until end of file.

       Seek #1, i             ' Seek to byte position
       MyChar = Input(1, #1)  ' Read next character of data.
       Print MyChar           'Print character of data
    Next i
    Close #1                  ' Close file.
End Sub
```

Select Case Statement

Executes one of the statement blocks in the case based on the test variable

Select Case *testvar*
    Case *var1*
        Statement Block
    Case *var2*
        Statement Block
    Case Else
        Statement Block
End Select

The syntax supported by the Select statement includes the “To” keyword, a coma delimited list and a constant or variable.

Select Case Number ' Evaluate Number.
Case 1 To 5 ' Number between 1 and 5, inclusive.
…
' The following is the only Case clause that evaluates to True.
Case 6, 7, 8 ' Number between 6 and 8.
…
Case 9 To 10 ' Number is 9 or 10.
…
Case Else ' Other values.
…
End Select

Related Topics: If...Then...Else

Example:

```
' This rather tedious test shows nested select statements and if uncommented,
' the exit for statement

Sub Test ()
    For x = 1 to 5
        print x
        Select Case x
        Case 2
            Print "Outer Case Two"
        Case 3
            Print "Outer Case Three"
'            Exit For
            Select Case x
            Case 2
                Print "Inner Case Two"
            Case 3
                Print "Inner Case Three"
'                Exit For
            Case Else  ' Must be something else.
                Print "Inner Case Else:", x
            End Select

            Print "Done with Inner Select Case"
        Case Else      ' Must be something else.
            Print "Outer Case Else:",x
        End Select
    Next x
    Print "Done with For Loop"
End Sub
```

SendKeys Function

SendKeys (*Keys, [wait]*)

Sends one or more keystrokes to the active window as if they had been entered at the keyboard

The SendKeys statement has two parameters. The first parameter *keys* is a string and is sent to the active window. The second parameter *wait* is optional and if omitted is assumed to be false. If wait is true the keystrokes must be processed before control is returned to the calling procedure.

Example:

```
Sub Main ()
        Dim I, X, Msg  ' Declare variables.
        X = Shell("Calc.exe", 1)      ' Shell Calculator.
        For I = 1 To 5 ' Set up counting loop.
                SendKeys I & "{+}", True      ' Send keystrokes to Calculator
        Next I  ' to add each value of I.
        AppActivate "Calculator"      ' Return focus to Calculator.
        SendKeys "%{F4}", True ' Alt+F4 to close Calculator.
End Sub
```

Set Statement

Set *Object* = {[New] *objectexpression* | Nothing}

Assigns an object to an object variable.

Related Topics: Dim, Global, Static

Example:

```
Sub Main
        Dim visio As Object
        Set visio = CreateObject( "visio.application" )
        Dim draw As Object
        Set draw = visio.Documents
        draw.Open "c:\visio\drawings\Sample1.vsd"
        MsgBox "Open docs: " & draw.Count
        Dim page As Object
        Set page = visio.ActivePage
        Dim red As Object
        Set red = page.DrawRectangle (1, 9, 7.5, 4.5)
        red.FillStyle = "Red fill"

        Dim cyan As Object
        Set cyan = page.DrawOval (2.5, 8.5, 5.75, 5.25)
        cyan.FillStyle = "Cyan fill"

        Dim green As Object
        Set green = page.DrawOval (1.5, 6.25, 2.5, 5.25)
        green.FillStyle = "Green fill"
```

```
        Dim DarkBlue As Object
        set DarkBlue = page.DrawOval (6, 8.75, 7, 7.75)
        DarkBlue.FillStyle = "Blue dark fill"

        visio.Quit
End Sub
```

Shell Function

Shell (*app [, style]*)

Runs an executable program.

The shell function has two parameters. The first one, *app* is the name of the program to be executed. The name of the program in *app* must include a .PIF, .COM, .BAT, or .EXE file extension or an error will occur. The second argument, *style* is the number corresponding to the style of the window . It is also optional and if omitted the program is opened minimized with focus.

Window styles:
Normal with focus 1,5,9
Minimized with focus (default) 2
Maximized with focus 3
normal without focus 4,8
minimized without focus 6,7

Return value: ID, the task ID of the started program.

Example:

```
' This example uses Shell to leave the current application and run the
' Calculator program included with Microsoft Windows; it then
' uses the SendKeys statement to send keystrokes to add some numbers.

Sub Main ()
        Dim I, X, Msg  ' Declare variables.
        X = Shell("Calc.exe", 1)      ' Shell Calculator.
        For I = 1 To 5 ' Set up counting loop.
                SendKeys I & "{+}", True      ' Send keystrokes to Calculator
        Next I  ' to add each value of I.
        AppActivate "Calculator"      ' Return focus to Calculator.
        SendKeys "%{F4}", True ' Alt+F4 to close Calculator.
```

```
End Sub
```

Sin Function

Sin (*rad*)

Returns the sine of an angle that is expressed in radians

Example:

```
Sub Main ()
        pi = 4 * Atn(1)
        rad = 90 * (pi/180)
        x = Sin(rad)
        print x
End Sub
```

Space Function

Space[$] (*number*)

Skips a specified number of spaces in a print# statement.

The parameter number can be any valid integer and determines the number of blank spaces.

Example:

```
' This sample shows the space function

Sub Main

    MsgBox "Hello" & Space(20) & "There"

End Sub
```

Sqr Function

Sqr(*num*)

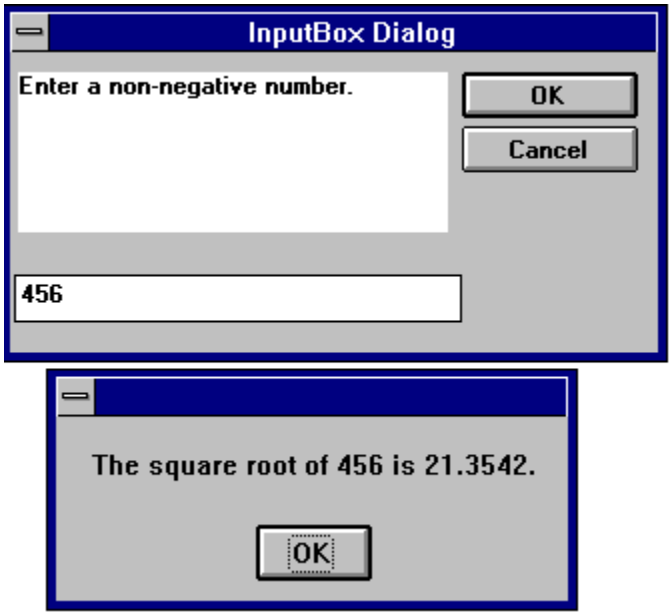
Returns the square root of a number.

The parameter *num* must be a valid number greater than or equal to zero.

Example:

```
Sub Form_Click ()
        Dim Msg, Number        ' Declare variables.
```

```
        Msg = "Enter a non-negative number."
        Number = InputBox(Msg) ' Get user input.
        If Number < 0 Then
                Msg = "Cannot determine the square root of a negative number."
        Else
                Msg = "The square root of " & Number & " is "
                Msg = Msg & Sqr(Number) & "."
        End If
        MsgBox Msg      ' Display results.

End Sub
```



Static Statement

Static *variable*

Used to declare variables and allocate storage space. These variables will retain their value through the program run

Related Topics: Dim, Function, Sub

Example:

```
' This example shows how to use the static keyword to retain the value of
' the variable i in sub Joe.  If Dim is used instead of Static then i
' is empty when printed on the second call as well as the first.

Sub Main
    For i = 1 to 2
        Joe 2
    Next i
End Sub

Sub Joe( j as integer )
    Static i
    print i
    i = i + 5
```

```
    print i
End Sub
```

Stop Statement

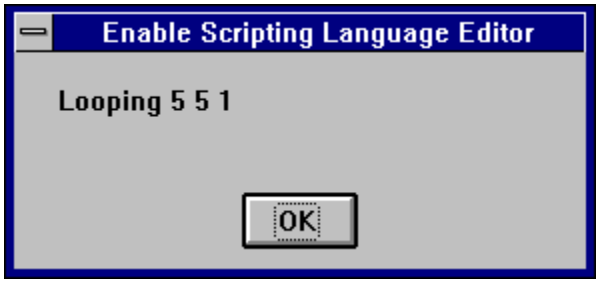
Stop

Ends execution of the program

The Stop statement can be placed anywhere in your code.

Example:

```
Sub main ()

        Dim x,y,z

        For x = 1 to 5
                For y = 1 to 5
                        For z = 1 to 5
                        Print "Looping" ,z,y,x
                        Next z
                Next y
               Stop
        Next x
End Sub
```



Str Function

Str(*numericexpr*)

Returns the value of a numeric expression.

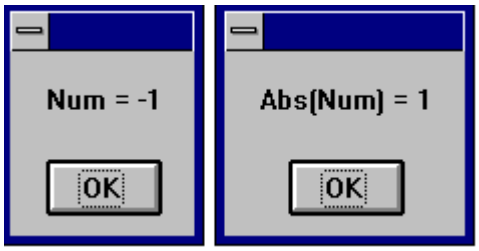
Str returns a String.

Related topics: Format, Val

Example:

```
Sub main ()
```

```
        Dim msg
        a = -1
        msgBox "Num = " & Str(a)
        MsgBox "Abs(Num) =" & Str(Abs(a))

End Sub
```



StrComp Function

StrComp(*nstring1,string2, [compare]*)

Returns a variant that is the result of the comparison of two strings

Example:

```
Sub Main

Dim MStr1, MStr2, MComp
        MStr1 = "ABCD": MStr2 = "today"       ' Define variables.
        print MStr1, MStr2
        MComp = StrComp(MStr1, MStr2) ' Returns -1.
        print MComp
        MComp = StrComp(MStr1, MStr2) ' Returns -1.
        print MComp
        MComp = StrComp(MStr2, MStr1) ' Returns 1.
        print MComp
End Sub
```

String Function

String (*numeric*, *charcode*)

String returns a string.

String is used to create a string that consists of one character repeated over and over.

Related topics: Space Function

Example:

```
Sub Main

    Dim MString
```

```
    MString = String(5, "*")   ' Returns "*****".
    MString = String(5, 42)    ' Returns "44444".
    MString = String(10, "Today")      ' Returns "TTTTTTTTTT".
    Print MString
End Sub
```

Sub Statement

Sub *SubName* [(*arguments*)]
    Dim [*variable*(s)]
    [*statementblock*]
    [Exit Function]
End Sub

Declares and defines a Sub procedures name, parameters and code.

When the optional argument list needs to be passed the format is as follows:

([ByVal] variable [As type] [,ByVal] variable [As type]]…])

The optional ByVal parameter specifies that the variable is [passed by value instead of by reference (see “ByRef and ByVal” in this manual). The optional As type parameter is used to specify the data type. Valid types are String, Integer, Double, Long, and Varaint (see “Variable Types” in this manual).

Related Topics: Call, Dim, Function

Example:

```
Sub Main
   Dim DST As String

   DST = "t1"
   mkdir DST
   mkdir "t2"
End Sub
```

Tan Function

Tan(*angle*)

Returns the tangent of an angle as a double.

The parameter *angle* must be a valid angle expressed in radians.

Related Topic: Atn, Cos, Sin

Example:

```
' This sample program show the use of the Tan function

Sub Main ()
    Dim Msg, Pi         ' Declare variables.
    Pi = 4 * Atn(1)     ' Calculate Pi.
    Msg = "Pi is equal to " & Pi
    MsgBox Msg          ' Display results.
    x = Tan(Pi/4)
    MsgBox  x & " is the tangent of Pi/4"
End Sub
```

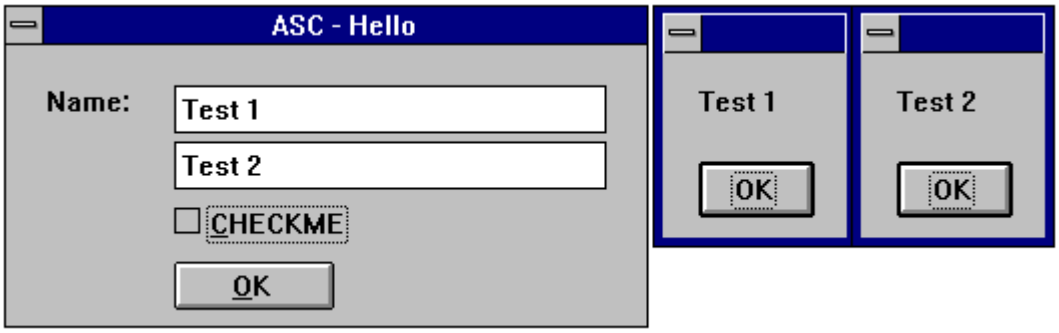
Text Statement

Text Starting X position, Starting Y position, Width, Height, Label

Creates a text field for titles and labels.

Example:

```
Sub Main()
    Begin Dialog DialogName1 60, 60, 160, 70, "ASC - Hello"
        TEXT 10, 10, 28, 12, "Name:"
        TEXTBOX 42, 10, 108, 12, .nameStr
        TEXTBOX 42, 24, 108, 12, .descStr
        CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
        OKBUTTON 42, 54, 40, 12
    End Dialog
    Dim Dlg1 As DialogName1
    Dialog Dlg1

    MsgBox Dlg1.nameStr
    MsgBox Dlg1.descStr
    MsgBox Dlg1.checkInt
End Sub
```



TextBox Statement

TextBox Starting X position, Starting Y position, Width, Height, Default String

Creates a Text Box for typing in numbers and text

Example:

```
Sub Main ()
    Begin Dialog DialogName1 60, 60, 160, 70, "ASC - Hello"
        TEXT 10, 10, 28, 12, "Name:"
        TEXTBOX 42, 10, 108, 12, .nameStr
        TEXTBOX 42, 24, 108, 12, .descStr
        CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
        OKBUTTON 42, 54, 40, 12
    End Dialog
    Dim Dlg1 As DialogName1
    Dialog Dlg1

    MsgBox Dlg1.nameStr
    MsgBox Dlg1.descStr
    MsgBox Dlg1.checkInt
End Sub
```

Time Function

Time[()]

Returns the current system time.

Related topics: To set the time use the TIME$ statement.

Example:

```
Sub Main
  x = Time(Now)
  Print x
End Sub
```

Timer Event

Timer

Timer Event is used to track elapsed time or can be display as a stopwatch in a dialog. The timers value is the number of seconds from midnight.

Related topics: DateSerial, DateValue, Hour Minute, Now, Second TimeValue.

Example:

```
Sub Main

       Dim TS As Single
       Dim TE As Single
       Dim TEL As Single
```

```
        TS = Timer

        MsgBox "Starting Timer"

        TE = Timer

        TT = TE - TS
        Print TT

End Sub
```

TimeSerial - Function

TimeSerial (*hour, minute, second*)

Returns the time serial for the supplied parameters *hour, minute, second.*

Related topics: DateSerial, DateValue, Hour Minute, Now, Second TimeValue.

Example:

```
Sub Main

        Dim MTime
        MTime = TimeSerial(12, 25, 27)
        Print MTime

End Sub
```

TimeValue - Function

TimeValue (*TimeString*)

Returns a double precision serial number based of the supplied string parameter.

Midnight = TimeValue(“23:59:59”)

Related topics: DateSerial, DateValue, Hour Minute, Now, Second TimeSerial.

Example:

```
Sub Main

Dim MTime
MTime = TimeValue("12:25:27 PM")
Print MTime

End Sub
```

Trim, LTrim, RTrim Functions

[L| R] Trim (*String*)

Ltrim, Rtrim and Trim all Return a copy of a string with leading, trailing or both leading and trailing spaces removed.

Ltrim, Rtrim and Trim all return a string

Ltrim removes leading spaces.
Rtrim removes trailing spaces.
Trim removes leading and trailing spaces.

Example:

```
' This example uses the LTrim and RTrim functions to strip leading and
' trailing spaces, respectively, from a string variable.  It
' uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

Sub Main
    MyString = "  <-Trim->  " ' Initialize string.
    TrimString = LTrim(MyString)     ' TrimString = "<-Trim->  ".
    MsgBox "|" & TrimString & "|"
    TrimString = LCase(RTrim(MyString))      ' TrimString = "  <-trim->".
    MsgBox "|" & TrimString & "|"
    TrimString = LTrim(RTrim(MyString))      ' TrimString = "<-Trim->".
    MsgBox "|" & TrimString & "|"
    ' Using the Trim function alone achieves the same result.
    TrimString = UCase(Trim(MyString))       ' TrimString = "<-TRIM->".
    MsgBox "|" & TrimString & "|"
End Sub
```

Type Statement

Type usertype elementname As typename
[elementname As typename]
. . .
End Type

Defines a user-defined data type containing one or more elements.

The **Type** statement has these parts:

Part	Description
Type	Marks the beginning of a user-defined type.
usertype	Name of a user-defined data type. It follows standard variable naming conventions.
elementname	Name of an element of the user-defined data type. It follows standard variable-naming conventions.
subscripts	Dimensions of an array element. You can declare multiple dimensions.
typename	One of these data types: Integer, Long, Single, Double, String (for variable-length strings), String * length (for fixed-length strings), Variant, or another user-defined type. The argument typename can't be an object type. End Type Marks the end of a user-defined type.

Once you have declared a user-defined type using the Type statement, you can declare a variable of that type anywhere in your script. Use Dim or Static to declare a variable of a user-defined type. Line numbers and line labels aren't allowed in Type...End Type blocks.

User-defined types are often used with data records because data records frequently consist of a number of related elements of different data types. Arrays cannot be an element of a user defined type in Enable.

Example:

```
' This sample shows some of the features of user defined types

Type type1
    a As Integer
    d As Double
    s As String
End Type

Type type2
    a As String
    o As type1
End Type

Type type3
    b As Integer
    c As type2
End Type
```

```
Dim type2a As type2
Dim type2b As type2
Dim type1a As type1
Dim type3a as type3

Sub Form_Click ()
    a = 5
    type1a.a = 7472
    type1a.d = 23.1415
    type1a.s = "YES"
    type2a.a = "43 - forty three"
    type2a.o.s = "Yaba Daba Doo"
    type3a.c.o.s = "COS"
    type2b.a = "943 - nine hundred and forty three"
    type2b.o.s = "Yogi"
    MsgBox type1a.a
    MsgBox type1a.d
    MsgBox type1a.s
    MsgBox type2a.a
    MsgBox type2a.o.s
    MsgBox type2b.a
    MsgBox type2b.o.s
    MsgBox type3a.c.o.s
    MsgBox a
End Sub
```

UBound Function

Ubound(*arrayname[,dimension*])

Returns the value of the largest usable subscript for the specified dimension of an array.

Related Topics: Dim, Global, Lbound, and Option Base

Example:

```
' This example demonstrates some of the  features of arrays.  The lower bound
' for an array is 0 unless it is specified or option base is set it as is
' done in this example.

Option Base 1

Sub Main
    Dim a(10) As Double
    MsgBox "LBound: " & LBound(a) & " UBound: " & UBound(a)
    Dim i As Integer
    For i = 1 to 3
        a(i) = 2 + i
    Next i
    Print a(1),a(1),a(2), a(3)
End Sub
```

UCase Function

Ucase (*String*)

Returns a copy of *String* in which all lowercase characters have been converted to uppercase.

Related Topics: Lcase, Lcase$ Function

Example:

```
' This example uses the LTrim and RTrim functions to strip leading and
' trailing spaces, respectively, from a string variable.  It
' uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

Sub Main
    MyString = "  <-Trim->  " ' Initialize string.
    TrimString = LTrim(MyString)     ' TrimString = "<-Trim->  ".
    MsgBox "|" & TrimString & "|"
    TrimString = LCase(RTrim(MyString))      ' TrimString = "  <-trim->".
    MsgBox "|" & TrimString & "|"
    TrimString = LTrim(RTrim(MyString))      ' TrimString = "<-Trim->".
    MsgBox "|" & TrimString & "|"
    ' Using the Trim function alone achieves the same result.
    TrimString = UCase(Trim(MyString))       ' TrimString = "<-TRIM->".
    MsgBox "|" & TrimString & "|"
End Sub
```

Val

Val(*string*)

Returns the numeric value of a string of characters.

Example:

```
Sub main
       Dim Msg
       Dim YourVal As Double
       YourVal = Val(InputBox$("Enter a number"))
       Msg = "The number you enered is: " & YourVal
       MsgBox Msg
End Sub
```

VarType

VarType(*varname*)

Returns a value that indicates how the parameter *varname* is stored internally.

The parameter *varname* is a variant data type.

VarType	return values:
Empty	0

Null	1
Integer	2
Long	3
Single	4
Double	5
Currency	6 (not available at this time)
Date/Time	7
String	8

Related Topics: IsNull, IsNumeric

Example:

```
If VarType(x) = 5 Then Print "Vartype is Double"   'Display variable
type
```

Weekday Function

Weekday(*date,firstdayof week*)

Returns a integer containing the whole number for the weekday it is representing.

Related Topics: Hour, Second, Minute, Day

Example:

```
Sub Main

x = Weekday(#5/29/1959#)
Print x

End Sub
```

While...Wend Statement

While condition

.
.
.
[StatementBlock]
.
.
.

Wend

While begins the while...Wend flow of control structure. Condition is any numeric or expression that evaluates to true or false. If the condition is true the statements are executed. The statements can be any number of valid Enable Basic statements. Wend ends the While...Wend flow of control structure.

Related Topics: Do...Loop Statement

Example:

```
Sub Main
       Const Max = 5
       Dim A(5) As String
       A(1) = "Programmer"
       A(2) = "Engineer"
       A(3) = "President"
       A(4) = "Tech Support"
       A(5) = "Sales"
       Exchange = True

       While Exchange
              Exchange = False
              For I = 1 To Max
                  MsgBox A(I)
              Next     I
       Wend
```

With Statement

With object
    [STATEMENTS]
End With

The With statement allows you to proeform a series of commands or statements on a particular object without again refering to the name of that object. With statements can be nested by putting one With block within another With block. You will need to fully specify any object in an inner With block to any memeber of an object in an outer With block.

Related Topics: While Statement and Do Loop

Example:

```
' This sample shows some of the features of user defined types and the with
' statement

Type type1
    a As Integer
    d As Double
    s As String
End Type
```

```
Type type2
    a As String
    o As type1
End Type

Dim type1a As type1
Dim type2a As type2

Sub Main ()

    With type1a
      .a = 65
      .d = 3.14
    End With
    With type2a
      .a = "Hello, world"
      With .o
        .s = "Goodbye"
      End With
   End With
    type1a.s = "YES"
    MsgBox type1a.a
    MsgBox type1a.d
    MsgBox type1a.s
    MsgBox type2a.a
    MsgBox type2a.o.s

End Sub
```

Write # - Statement

Write #*filenumber* [,*parameterlist*]

Writes and formats data to a sequential file that must be opened in output or append mode.

A comma delimited list of the supplied parameters is written to the indicated file. If no parameters are present, the newline character is all that will be written to the file.

Related Topics: Open and Print# Statements

Example:

```
Sub Main ()

       Open "TESTFILE" For Output As #1      ' Open to write file.
       userData1$ = InputBox("Enter your own text here")
       userData2$ = InputBox("Enter more of your own text here")
       Write #1, "This is a test of the Write # statement."
       Write #1,userData1$, userData2
       Close #1

       Open "TESTFILE" for Input As #2       ' Open to read file.
       Do While Not EOF(2)
              Line Input #2, FileData       ' Read a line of data.
              PRint FileData         ' Construct message.

       Loop
       Close #2              ' Close all open files.
       MsgBox "Testing Print Statement"      ' Display message.
```

```
       Kill "TESTFILE"        ' Remove file from disk.
End Sub
```

Year Function

Year(*serial#*)

Returns an integer representing a year between 1930 and 2029, inclusive. The returned integer represents the year of the serial parameter.

The parameter *serial#* is a string that represents a date.

If *serial* is a Null, this function returns a Null.

Related Topics: Date, Date$ Function/Statement, Day, Hour, Month, Minute, Now, Second.

Example:

```
Sub Main
       MyDate = "11/11/94"
       x = Year(MyDate)
      print x
End Sub
```

INDEX

K

L

M

N

O

P

R

S

T

U

V

W

Y