Interrupt Process command: A TNVTPlus command that terminates a running program on the network host.

break command: A Telnet command that emulates a break key or attention signal on the network host. Not all servers support this feature.

lock state: A setting that specifies which, if any, lock key will affect the key's modified shift state.

Valid lock states are Caps Lock, Num Lock, and None.

shift state: A setting that specifies a key's modified behavior when Caps Lock or Num Lock is on. Valid shift states are

- VT52
- Shift
- Ctrl
- Ctrl+Shift
- Alt
- Alt+Shift
- Alt+Ctrl
- Alt+Ctrl+Shift

dead key: A key that, when pressed, does not display a character, but is combined with the keystroke that follows it to create a composite character. Dead keys are usually used to create letters with accent marks. Sometimes also called non-escaping key.

compose key: The key on the PC keyboard that is mapped to the VT compose key function. On a VT terminal, a compose key lets you combine two or three keystrokes to create a character that is not available on the keyboard. For example, use the compose key to create a letter with an accent mark or a symbol (such as a currency sign), or a ligature. The default compose key for TNVTPlus is Alt+F7.

auto print: A VT220/VT320/VT420 feature that takes the output from a program on the host, displays it to the screen, and stores it in a spool file. With auto print on, everything displayed on the screen is captured in a spool file. Then, when turn auto print is turned off, the data in the spool file is sent to your printer.

spool file: A temporary file used to store the output of an auto print or print controller escape sequence until the output is sent to the printer.

connection: The path between two hosts on the network. When two network hosts are connected, they can exchange information. Compare with session.

session: All the interactions between your PC and a host beginning with the initial connection and ending when you or the host explicitly disconnect. The program configuration settings for that host connection are a part of the session.

session definition: The configuration settings for a particular session or host connection. A session definition might include such settings as the hostname of a computer on the network and your login name for that computer, as well as other values that you specify. The set of session parameters you can specify differs with each program.

hostname: The name of a host. The hostname is one form of the computer's TCP/IP network address; the other is its complete numeric network address (IP address). You can access a host by its hostname or its numeric network address.

IP address: A number that uniquely identifies a host that uses the TCP/IP communication protocol. The form of an IP address is four groups of numbers separated by periods, for example, **128.127.55.55**. The Internet Protocol is defined in RFC 791.

Telnet: The standard TCP/IP remote login protocol. With Telnet, you can work from your PC as if it were a terminal physically attached to another machine. Some programs that provide Telnet services are TNVTPlus and TN3270/TN5250.

server: A host that makes a service available to other computers on a network (its clients). Typical services include transferring files, printing files, and managing logins from network users.

username: The name, assigned by a network system administrator or an Internet service provider, that you use to log in to a computer on a network.

password: A word or string of characters that you supply to log in to another system, workgroup, or domain on a network. Systems that accept the username **anonymous** often require you to provide either your e-mail address or **guest** as the password.

filename conventions: The rules that users of an operating system must follow to name files. A TCP/IP network usually contains computers that run different operating systems. Each operating system has different conventions for naming files. For example, both the number and kinds of characters that can be used in a name are often subject to limits. When you use some TCP/IP supported services such as Telnet and FTP, use the filenaming conventions of the host system to work with files that are on the host.

case sensitivity: The ability of a program to evaluate the difference between the capitalized and non-capitalized versions of a character. Case-sensitive programs treat cat, CAT, and Cat, for example, as distinct items. On a case-sensitive operating system, such as the UNIX system, you must spell commands and filenames with the appropriate capitalization. Case sensitivity also affects the way that files are listed when sorted in alphabetical order.

wildcard: A character such as ***** or **?** that represents one or more characters in a filename. Each operating system supports its own wildcard characters and syntax.

port: A connecting point to a service offered by a host. There are some default ports. For example, hosts that offer FTP services use port 21; hosts that offer Telnet services use port 23; SMTP uses port 25; and Web servers use port 80.

ASCII: An acronym for American Standard Code for Information Interchange. A standard computer character set used in text files. ASCII files do not contain program or formatting instructions.

OLE (Object Linking and Embedding) Automation: A standard interface through which programs make their features available to scripting tools and other programs. You can automate repetitive tasks by writing scripts that use OLE Automation objects.

OLE Automation object: A combination of information and ways of processing that information in OLE Automation. An OLE Automation object has functions that make some of its information and processes available to other programs. These functions are the objects properties and methods.

OLE method: An OLE Automation object function that specifies an action the object can perform.

OLE property: An OLE Automation object function that gets or sets information about the state of the object. For example, the Visible property determines whether the object is visible on the computer screen.

system administrator (or network administrator): The person at your workplace who is responsible for configuring and maintaining your network.

' Abs Function Example

Close | Copy All | Copy | Print

'This example finds the difference between two variables, `oldacct` and `newacct`.

```
Sub main
Dim oldacct, newacct, count
    oldacct=InputBox("Enter the oldacct number")
    newacct=InputBox("Enter the newacct number")
    count=Abs(oldacct-newacct)
    MsgBox "The absolute value is: " &count
End Sub
```

See Also

- Exp
- Fix
- Int
- Log
- Rnd
- Sgn
- Sqr

' AppActivate Statement Example

Close | Copy All | Copy | Print

```
'This example opens the Windows bitmap file ARCADE.BMP in Paintbrush. (Paintbrush must
already be open before running this example. It must also not be minimized.)

Sub main
    MsgBox "Opening C:\WINDOWS\ARCADE.BMP in Paintbrush."
    AppActivate "Paintbrush - (Untitled)"
    SendKeys "%FOC:\WINDOWS\ARCADE.BMP{Enter}",1
    MsgBox "File opened."
End Sub
```

See Also

SendKeys
Shell

Close | Copy All | Copy | Print

' Asc Function Example

```
'This example asks the user for a letter and returns its ASCII value.

Sub main
    Dim userchar
    userchar=InputBox("Type a letter:")
    MsgBox "The ASC value for " & userchar & " is: " & Asc(userchar)
End Sub
```

See Also

Chr

' Assert Function Example

Close | Copy All | Copy | Print

(None)

See Also

(None)

' Atn Function Example

Close | Copy All | Copy | Print

'This example finds the roof angle necessary for a house with an attic ceiling of 8 feet (at the roof peak) and a 16 foot span from the outside wall to the center of the house. The Atn function returns the angle in radians; it is multiplied by 180/PI to convert it to degrees.

```
Sub main
    Dim height, span, angle, PI
    PI=3.14159
    height=8
    span=16
    angle=Atn(height/span)*(180/PI)
    MsgBox "The angle is " & Format(angle, "##.##") & " degrees"
End Sub
```

See Also

- Cos
- Sin
- Tan
- Derived Trigonometric Functions

' Beep Statement Example

Close | Copy All | Copy | Print

'This example beeps and displays a message in a box if the variable *balance* is less than 0. (If you have a set of speakers hooked up to your computer, you might need to turn them on to hear the beep.)

```
Sub main
    Dim expenses, balance, msgtext
    balance=InputBox("Enter your account balance")
    expenses=1000
    balance=balance-expenses
    If balance<0 then
        Beep
        Msgbox "I'm sorry, your account is overdrawn."
    Else
        Msgbox "Your balance minus expenses is: " &balance
    End If
End Sub
```

See Also

- InputBox
- MsgBox Statement
- Print

' Begin Dialog... End Dialog Statement Example

Close | Copy All | Copy | Print

```
'This example defines and displays a dialog box with each type of item in it: list box, combo box,
buttons, etc.

Sub main
    Dim ComboBox1() as String
    Dim ListBox1() as String
    Dim DropListBox1() as String
    ReDim ListBox1(0)
    ReDim ComboBox1(0)
    ReDim DropListBox1(3)
    ListBox1(0)="C:\"
    ComboBox1(0)=Dir("C:\*.*")
    For x=0 to 2
     DropListBox1(x)=Chr(65+x) & ":"
    Next x
    Begin Dialog UserDialog 274, 171, "OPEN Script Dialog Box"
        ButtonGroup .ButtonGroup1
        Text   9, 3, 69, 13, "Filename:", .Text1
        DropComboBox   9, 14, 81, 119, ComboBox1(), .ComboBox1
        Text   106, 2, 34, 9, "Directory:", .Text2
        ListBox   106, 12, 83, 39, ListBox1(), .ListBox2
        Text   106, 52, 42, 8, "Drive:", .Text3
        DropListBox   106, 64, 95, 44, DropListBox1(), .DropListBox1
        CheckBox   9, 142, 62, 14, "List .TXT files", .CheckBox1
        GroupBox   106, 111, 97, 57, "File Range"
        OptionGroup .OptionGroup2
            OptionButton   117, 119, 46, 12, "All pages", .OptionButton3
            OptionButton   117, 135, 67, 8, "Range of pages", .OptionButton4
        Text   123, 146, 20, 10, "From:", .Text6
        Text   161, 146, 14, 9, "To:", .Text7
        TextBox   177, 146, 13, 12, .TextBox4
        TextBox   145, 146, 12, 11, .TextBox5
        OKButton   213, 6, 54, 14
        CancelButton   214, 26, 54, 14
        PushButton 213, 52, 54, 14, "Help", .Push1
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- Dialog
- DropComboBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

' Button Statement Example

'This example defines a dialog box with a combination list box and three buttons.

```
Sub main
    Dim fchoices as String
    fchoices="File1" & Chr(9) & "File2" & Chr(9) & "File3"
    Begin Dialog UserDialog 185, 94, "OPEN Script Dialog Box"
        Text   9, 5, 69, 10, "Filename:", .Text1
        DropComboBox   9, 17, 88, 71, fchoices, .ComboBox1
        ButtonGroup .ButtonGroup1
        OKButton   113, 14, 54, 13
        CancelButton   113, 33, 54, 13
        Button 113, 57, 54, 13, "Help", .Push1
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin Dialog...End Dialog Statement
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- DropComboBox
- DropListBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

Close | Copy All | Copy | Print

' ButtonGroup Statement Example

'This example defines a dialog box with a group of three buttons.

```
Sub main
    Begin Dialog UserDialog 34,0,231,140, "OPEN Script Dialog Box"
        ButtonGroup .bg
        PushButton 71,17,88,17, "&Button 0"
        PushButton 71,50,88,17, "&Button 1"
        PushButton 71,83,88,17, "&Button 2"
    End Dialog
    Dim mydialog as UserDialog
    Dialog mydialog
    Msgbox "Button " & mydialog.bg & " was pressed."
End Sub
```

See Also

- Begin Dialog...End Dialog Statement
- Button
- CancelButton
- Caption
- CheckBox
- ComboBox
- DropComboBox
- DropListBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

' Call Statement Example

Close | Copy All | Copy | Print

'This example calls a subprogram named CREATEFILE to open a file, write the numbers 1 to 10 in it and leave it open. The calling procedure then checks the file's mode. If the mode is 1 (open for Input) or 2 (open for Output), the procedure closes the file.

```
Declare Sub createfile()
Sub main
    Dim filemode as Integer
    Dim attrib as Integer
    Call createfile
    attrib=1
    filemode=FileAttr(1,attrib)
    If filemode=1 or 2 then
        MsgBox "File was left open. Closing now."
        Close #1
    End If
    Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the numbers 1-10 into a file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=1 to 10
        Write #1, x
    Next x
End Sub
```

See Also

Declare

' CancelButton Statement Example

Close | Copy All | Copy | Print

'This example defines a dialog box with a combination list box and three buttons.

```
Sub main
    Dim fchoices as String
    fchoices="File1" & Chr(9) & "File2" & Chr(9) & "File3"
    Begin Dialog UserDialog 185, 94, "OPEN Script Dialog Box"
        Text   9, 5, 69, 10, "Filename:", .Text1
        DropComboBox   9, 17, 88, 71, fchoices, .ComboBox1
        ButtonGroup .ButtonGroup1
        OKButton   113, 14, 54, 13
        CancelButton   113, 33, 54, 13
        PushButton 113, 57, 54, 13, "Help", .Push1
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin Dialog...End Dialog Statement
- Button
- ButtonGroup
- Caption
- CheckBox
- ComboBox
- DropComboBox
- DropListBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

' Caption Statement Example

Close | Copy All | Copy | Print

'This example defines a dialog box with a combination list box and three buttons. The Caption statement changes the dialog box title to "Example -Caption Statement".

```
Sub main
    Dim fchoices as String
    fchoices="File1" & Chr(9) & "File2" & Chr(9) & "File3"
    Begin Dialog UserDialog 185, 94
        Caption "Example-Caption Statement"
        Text   9, 5, 69, 10, "Filename:", .Text1
        DropComboBox   9, 17, 88, 71, fchoices, .ComboBox1
        ButtonGroup .ButtonGroup1
        OKButton   113, 14, 54, 13
        CancelButton   113, 33, 54, 13
        PushButton 113, 57, 54, 13, "Help", .Push1
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin Dialog...End Dialog Statement
- Button
- CancelButton
- ButtonGroup
- CheckBox
- ComboBox
- DropComboBox
- DropListBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

' CCur Function Example

Close | Copy All | Copy | Print

'This example converts a yearly payment on a loan to a currency value with four decimal places. A subsequent Format statement formats the value to two decimal places before displaying it in a message box.

```
Sub main
Dim aprate, totalpay,loanpv
    Dim loanfv, due, monthlypay
    Dim yearlypay, msgtext
    loanpv=InputBox("Enter the loan amount: ")
    aprate=InputBox("Enter the annual percentage rate: ")
    If aprate >1 then
        aprate=aprate/100
    End If
    aprate=aprate/12
    totalpay=InputBox("Enter the total number of pay periods: ")
    loanfv=0
Rem Assume payments are made at end of month
    due=0
    monthlypay=Pmt(aprate,totalpay,-loanpv,loanfv,due)
    yearlypay=CCur(monthlypay*12)
    msgtext= "The yearly payment is: " & Format(yearlypay, "Currency")
    MsgBox msgtext
End Sub
```

See Also

- CDbl
- CInt
- CLng
- CSng
- CStr
- CVar
- CVDate

' CDbl Function Example

Close | Copy All | Copy | Print

```
'This example calculates the square root of 2 as a double-precision floating point value and
displays it in scientific notation.

Sub main
Dim value
    Dim msgtext
    value=CDbl(Sqr(2))
    msgtext= "The square root of 2 is: " & Value
    MsgBox msgtext
End Sub
```

See Also

- CCur
- CInt
- CLng
- CSng
- CStr
- CVar
- CVDate

' ChDir Statement Example

Close | Copy All | Copy | Print

'This example changes the current directory to C:\WINDOWS, if it is not already the default.

```
Sub main
    Dim newdir as String
    newdir="c:\windows"
    If CurDir <> newdir then
        ChDir newdir
    End If
    MsgBox "The default directory is now: " & newdir
End Sub
```

See Also

- ChDrive
- CurDir
- Dir
- MkDir
- RmDir

Close | Copy All | Copy | Print

' ChDrive Statement Example

```
'This example changes the default drive to A:.

Sub main
    Dim newdrive as String
    newdrive="A:"
    If Left(CurDir,2) <> newdrive then
        ChDrive newdrive
    End If
    MsgBox "The default drive is now " & newdrive
End Sub
```

See Also

- ChDir
- CurDir
- Dir
- MkDir
- RmDir

' CheckBox Statement Example

Close | Copy All | Copy | Print

'This example defines a dialog box with a combination list box, a check box, and three buttons.

```
Sub main
    Dim ComboBox1() as String
    ReDim ComboBox1(0)
    ComboBox1(0)=Dir("C:\*.*")
    Begin Dialog UserDialog 166, 76, "OPEN Script Dialog Box"
        Text   9, 3, 69, 13, "Filename:", .Text1
        DropComboBox   9, 14, 81, 119, ComboBox1(), .ComboBox1
        CheckBox   10, 39, 62, 14, "List .TXT files", .CheckBox1
        OKButton   101, 6, 54, 14
        CancelButton   101, 26, 54, 14
        PushButton 101, 52, 54, 14, "Help", .Push1
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin Dialog...End Dialog Statement
- Button
- ButtonGroup
- CancelButton
- Caption
- ComboBox
- DropComboBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

' Chr Function Example

Close | Copy All | Copy | Print

'This example displays the character equivalent for an ASCII code between 65 and 122 typed by the user.

```
Sub main
    Dim numb as Integer
    Dim msgtext
    Dim out
    out=0
    Do Until out
        numb=InputBox("Type a number between 65 and 122:")
        If Chr$(numb)>="A" AND Chr$(numb)<="Z" OR Chr$(numb)>="a" AND _
            Chr$(numb)<="z" then
            msgtext="The letter for the number " & numb   &" is: " & Chr$(numb)
            out=1
        ElseIf numb=0 then
            Exit Sub
        Else
            Beep
            msgtext="Does not convert to a character; try again."
        End If
        MsgBox msgtext
    Loop
End Sub
```

See Also

- Asc
- CCur
- CDbl
- CInt
- CLng
- CSng
- CStr
- CVar
- CVDate
- Format
- Val

Close | Copy All | Copy | Print

' CInt Function Example

'This example calculates the average of ten golf scores.

```
Sub main
    Dim score As Integer
    Dim x, sum
    Dim msgtext
    Let sum=0
    For x=1 to 10
        score=InputBox("Enter golf score #"&x &":")
        sum=sum+score
    Next x
    msgtext="Your average is: " & Format(CInt(sum/(x-1)),"General Number")
    MsgBox msgtext
End Sub
```

See Also

- CCur
- CDbl
- CLng
- CSng
- CStr
- CVar
- CVDate

Close | Copy All | Copy | Print

' Clipboard Example

'This example places the text string "Hello, world." on the Clipboard.

```
Sub main
    Dim mytext as String
    mytext="Hello, world."
    Clipboard.Settext mytext
    MsgBox "The text: '" & mytext & "' added to the Clipboard."
End Sub
```

See Also

(None)

' CLng Function Example

Close | Copy All | Copy | Print

'This example divides the US national debt by the number of people in the country to find the amount of money each person would have to pay to wipe it out. This figure is converted to a Long integer and formatted as Currency.

```
Sub main
    Dim debt As Single
    Dim msgtext
    Const Populace = 250000000
    debt=InputBox("Enter the current US national debt:")
    msgtext="The $/citizen is: " & Format(CLng(Debt/Populace), "Currency")
    MsgBox msgtext
End Sub
```

See Also

- CCur
- CDbl
- CInt
- CSng
- CStr
- CVar
- CVDate

' Close Statement Example

Close | Copy All | Copy | Print

'This example opens a file for Random access, gets the contents of one variable, and closes the file again. The subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram.

```
Declare Sub createfile()
Sub main
    Dim acctno as String*3
    Dim recno as Long
    Dim msgtext as String
    Call createfile
    recno=1
    newline=Chr(10)
    Open "C:\TEMP001" For Random As #1 Len=3
    msgtext="The account numbers are:" & newline & newline
    Do Until recno=11
            Get #1,recno,acctno
            msgtext=msgtext & acctno
            recno=recno+1
    Loop
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the numbers 1-10 into a file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=1 to 10
        Write #1, x
    Next x
    Close #1
End Sub
```

See Also

- Open
- Reset
- Stop

' ComboBox Statement Example

Close | Copy All | Copy | Print

'This example defines a dialog box with a combination list and text box and three buttons.

```
Sub main
    Dim ComboBox1() as String
    ReDim ComboBox1(0)
    ComboBox1(0)=Dir("C:\*.*")
    Begin Dialog UserDialog 166, 142, "OPEN Script Dialog Box"
        Text   9, 3, 69, 13, "Filename:", .Text1
        ComboBox   9, 14, 81, 119, ComboBox1(), .ComboBox1
        OKButton   101, 6, 54, 14
        CancelButton   101, 26, 54, 14
        PushButton 101, 52, 54, 14, "Help", .Push1
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin Dialog...End Dialog Statement
- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- DropComboBox
- DropListBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

' Command Function Example

Close | Copy All | Copy | Print

```
'This example opens the file entered by the user on the command line.

Sub main
    Dim filename as String
    Dim cmdline as String
    Dim cmdlength as Integer
    Dim position as Integer
    cmdline=Command
    If cmdline="" then
        MsgBox "No command line information."
        Exit Sub
    End If
    cmdlength=Len(cmdline)
    position=InStr(cmdline,Chr(32))
    filename=Mid(cmdline,position+1,cmdlength-position)
    On Error Resume Next
    Open filename for Input as #1
    If Err<>0 then
        MsgBox "Error loading file."
        Exit Sub
    End If
    MsgBox "File " & filename & " opened."
    Close #1
    MsgBox "File " & filename & " closed."
End Sub
```

See Also

- AppActivate
- DoEvents
- Environ
- SendKeys
- Shell

Close Copy All Copy Print

' Const Statement Example

'This example divides the US national debt by the number of people in the country to find the amount of money each person would have to pay to wipe it out. This figure is converted to a Long integer and formatted as Currency.

```
Sub main
    Dim debt As Single
    Dim msgtext
    Const Populace=250000000
    debt=InputBox("Enter the current US national debt:")
    msgtext="The $/citizen is: " & Format(CLng(Debt/Populace), "Currency")
    MsgBox msgtext
End Sub
```

See Also

- Declare
- Def*type*
- Dim
- Let
- Type

' Cos Function Example

Close | Copy All | Copy | Print

```
'This example finds the length of a roof, given its pitch and the distance of the house from its
center to the outside wall.

Sub main
    Dim bwidth, roof,pitch
    Dim msgtext
    Const PI=3.14159
    Const conversion=PI/180
    pitch=InputBox("Enter roof pitch in degrees")
    pitch=Cos(pitch*conversion)
    bwidth=InputBox("Enter 1/2 of house width in feet")
    roof=bwidth/pitch
    msgtext="The length of the roof is " & Format(roof, "##.##") & " feet."
    MsgBox msgtext
End Sub
```

See Also

- Atn
- Sin
- Tan
- Derived Trigonometric Functions

' CreateObject Function Example

Close | Copy All | Copy | Print

'This example uses the CreateObject function to open the software product VISIO (if it is not already open).

```
Sub main
    Dim visio as Object
    Dim doc as Object
    Dim i as Integer, doccount as Integer

'Initialize Visio
    on error resume next
    Set visio = GetObject(,"visio.application")          ' find Visio
    If (visio Is Nothing) then
        Set visio = CreateObject("visio.application") ' find Visio
        If (visio Is Nothing) then
            Msgbox "Couldn't find Visio!"
            Exit Sub
        End If
    End If
    MsgBox "Visio is open."
End Sub
```

See Also

- GetObject
- Is
- Me
- New
- Nothing
- Object Class
- Typeof

' CSng Function Example

Close | Copy All | Copy | Print

'This example calculates the factorial of a number. A factorial (notated with an exclamation mark, !) is the product of a number and each integer between it and the number 1. For example, 5 factorial, or 5!, is the product of 5*4*3*2*1, or the value 120.

```
Sub main
    Dim number as Integer
    Dim factorial as Double
    Dim msgtext
    number=InputBox("Enter an integer between 1 and 170:")
    If number<=0 then
        Exit Sub
    End If
    factorial=1
    For x=number to 2 step -1
        factorial=factorial*x
    Next x
Rem If number =<35, then its factorial is small enough to be stored
Rem as a single-precision number
    If number<35 then
        factorial=CSng(factorial)
    End If
    msgtext="The factorial of " & number & " is: " & factorial
    MsgBox msgtext
End Sub
```

See Also

- CCur
- CDbl
- CInt
- CLng
- CStr
- CVar
- CVDate

' CStr Function Example

Close | Copy All | Copy | Print

'This example converts a variable from a value to a string and displays the result. Variant type 5 is Double and type 8 is String.

```
Sub main
    Dim var1
    Dim msgtext as String
    var1=InputBox("Enter a number:")
    var1=var1+10
    msgtext="Your number + 10 is: " & var1 & Chr(10)
    msgtext=msgtext & "which makes its Variant type: " & Vartype(var1)
    MsgBox   msgtext
    var1=CStr(var1)
    msgtext="After conversion to a string," & Chr(10)
    msgtext=msgtext & "the Variant type is: " & Vartype(var1)
    MsgBox msgtext
End Sub
```

See Also

- Asc
- CCur
- CDbl
- Chr
- CInt
- CLng
- CSng
- CVar
- CVDate
- Format

' $CStrings Metacommand Example

Close | Copy All | Copy | Print

'This example displays two lines, the first time using the C-language characters "\n" for a carriage return and line feed.

```
Sub main
    '$CStrings
    MsgBox "This is line 1\n This is line 2 (using C Strings)"
    '$NoCStrings
    MsgBox "This is line 1" +Chr$(13)+Chr$(10)+"This is line 2 (using Chr)"
End Sub
```

See Also

- $Include
- $NoCStrings
- Rem

' CurDir Statement Example

Close | Copy All | Copy | Print

'This example changes the current directory to C:\WINDOWS, if it is not already the default.

```
Sub main
    Dim newdir as String
    newdir="c:\windows"
    If CurDir <> newdir then
        ChDir newdir
    End If
    MsgBox "The default directory is now: " & newdir
End Sub
```

See Also

- ChDir
- ChDrive
- Dir
- MkDir
- RmDir

Close | Copy All | Copy | Print

' CVar Function Example

'This example converts a string variable to a variant variable.

```
Sub main
    Dim answer as Single
    answer=100.5
    MsgBox "'Answer' is DIM'ed as Single with the value: " & answer
    answer=CVar(answer)
    answer=Fix(answer)
    MsgBox "'Answer' is now a variant with a type of: " & VarType(answer)
End Sub
```

See Also

- CCur
- CDbl
- CInt
- CLng
- CSng
- CStr
- CVDate

' CVDate Function Example

Close | Copy All | Copy | Print

'This example displays the date for one week from the date entered by the user.

```
Sub main
Dim str1 as String
    Dim nextweek
    Dim msgtext
i: str1=InputBox$("Enter a date:")
    answer=IsDate(str1)
    If answer=-1 then
        str1=CVDate(str1)
        nextweek=DateValue(str1)+7
        msgtext="One week from the date entered is:
        msgtext=msgtext & "Format(nextweek,"dddddd")
        MsgBox msgtext
    Else
        MsgBox "Invalid date or format. Try again."
        Goto i
    End If
End Sub
```

See Also

- Asc
- CCur
- CDbl
- Chr
- CInt
- CLng
- CSng
- CStr
- CVar
- Format
- Val

Close | Copy All | Copy | Print

' Date Function Example

'This example displays the date for one week from the today's date (the current date on the computer).

```
Sub main
    Dim nextweek
    nextweek=CVar(Date)+7
    MsgBox "One week from today is: " & Format(nextweek,"ddddd")
End Sub
```

See Also

- CVDate
- Date Statement
- Format
- Now
- Time Function
- Time Statement
- Timer
- TimeSerial

Close | Copy All | Copy | Print

' Date Statement Example

```
'This example changes the system date to a date entered by the user.

Sub main
    Dim userdate
    Dim answer
i: userdate=InputBox("Enter a date for the system clock:")
    If userdate="" then
        Exit Sub
    End If
    answer=IsDate(userdate)
    If answer=-1 then
        Date=userdate
    Else
        MsgBox "Invalid date or format. Try again."
        Goto i
    End If
End Sub
```

See Also

- Date Function
- Time Function
- Time Statement

' DateSerial Function Example

Close | Copy All | Copy | Print

'This example finds the day of the week New Year's day will be for the year 2000.

```
Sub main
    Dim newyearsday
    Dim daynumber
    Dim msgtext
    Dim newday as Variant
    Const newyear=2000
    Const newmonth=1
    Let newday=1
    newyearsday=DateSerial(newyear,newmonth,newday)
    daynumber=Weekday(newyearsday)
    msgtext="New Year's day 2000 falls on a " & Format(daynumber, "dddd")
    MsgBox msgtext
End Sub
```

See Also

- DateValue
- Day
- Month
- Now
- TimeSerial
- TimeValue
- Weekday
- Year

' DateValue Function Example

Close | Copy All | Copy | Print

```
'This example displays the date for one week from the date entered by the user

Sub main
    Dim str1 as String
    Dim nextweek
    Dim msgtext
i: str1=InputBox$("Enter a date:")
    answer=IsDate(str1)
    If answer=-1 then
        str1=CVDate(str1)
        nextweek=DateValue(str1)+7
        msgtext="One week from your date is: " & Format(nextweek,"dddddd")
        MsgBox msgtext
    Else
        MsgBox "Invalid date or format. Try again."
        Goto i
    End If
End Sub
```

See Also

- DateSerial
- Day
- Month
- Now
- TimeSerial
- TimeValue
- Weekday
- Year

Close | Copy All | Copy | Print

' Day Function Example

'This example finds the month (1-12) and day (1-31) values for this Thursday.

```
Sub main
    Dim x, today, msgtext
    Today=DateValue(Now)
    Let x=0
    Do While Weekday(Today+x)<> 5
        x=x+1
    Loop
    msgtext="This Thursday is: " & Month(Today+x) & "/" & Day(Today+x)
    MsgBox msgtext
End Sub
```

See Also

- Date Function
- Date Statement
- Hour
- Minute
- Month
- Now
- Second
- Weekday
- Year

' DDEAppReturnCode Function Example

Close | Copy All | Copy | Print

(None)

See Also

- DDEExecute
- DDEInitiate
- DDEPoke
- DDERequest()
- DDETerminate

' DDEExecute Statement Example

Close | Copy All | Copy | Print

'This example opens Microsoft Write, uses DDEPoke to write the text "Hello, world" to the open document (Untitled) and uses DDEExecute to save the text to the file TEMP001.

```
Sub main
    Dim channel as Integer
    Dim appname as String
    Dim topic as String
    Dim testtext as String
    Dim item as String
    Dim pcommand as String
    Dim msgtext as String
    Dim x as Integer
    appname="Write"
    topic="Untitled"
    item="Page1"
    testtext="Hello, world."
    On Error Goto Errhandler
    x=Shell(appname & ".EXE")
    channel = DDEInitiate(appname, topic)
    If channel=0 then
        MsgBox "Unable to open Write."
        Exit Sub
    End If
    DDEPoke channel, item, testtext
    pcommand="[%FS(" + Chr$(34) + "C:\TEMP001" + Chr$(34) + ")]"
    DDEExecute channel, pcommand
    msgtext="The text: " & testtext & " saved to C:\TEMP001." & Chr$(13)
    msgtext=msgtext & Chr$(13) & "Delete? (Y/N)"
    answer=InputBox(msgtext)
    If answer="Y" or answer="y" then
        Kill "C:\TEMP001"
    End If
    DDETerminate channel
    Exit Sub
Errhandler:
    If Err<>0 then
      MsgBox "DDE Access failed."
    End If
End Sub
```

See Also

- DDEAppReturnCode()
- DDEInitiate()
- DDEPoke
- DDERequest()
- DDETerminate

' DDEInitiate Function Example

Close | Copy All | Copy | Print

```
'This example uses DDEInitiate to open a channel to the Paintbrush application. It uses
DDERequest to obtain the list of available topics (using the System topic).

Sub main
    Dim channel as Integer
    Dim appname as String
    Dim topic as String
    Dim item as String
    Dim pcommand as String
    Dim msgtext as String
    appname="Pbrush"
    topic="System"
    item="Topics"
    x=Shell(appname & ".EXE")
    channel = DDEInitiate(appname, topic)
    If channel=0 then
        msgtext="Unable to open Paintbrush."
    Else
        On Error Resume Next
        msgtext="The Paintbrush topics available are:" & Chr$(13)
        msgtext=msgtext & Chr$(13) & DDERequest(channel,item)
        DDETerminate channel
        If Err<>0 then
            msgtext="DDE Access failed."
        End If
    End If
    MsgBox msgtext
End Sub
```

See Also

- DDEAppReturnCode()
- DDEExecute
- DDEPoke
- DDERequest()
- DDETerminate

' DDEPoke Statement Example

Close | Copy All | Copy | Print

'This example opens Microsoft Write, uses DDEPoke to write the text "Hello, world" to the open document (Untitled) and uses DDEExecute to save the text to the file TEMP001.

```
Sub main
    Dim channel as Integer
    Dim appname as String
    Dim topic as String
    Dim testtext as String
    Dim item as String
    Dim pcommand as String
    Dim msgtext as String
    Dim x as Integer
    appname="Write"
    topic="Untitled"
    item="Page1"
    testtext="Hello, world."
    On Error Goto Errhandler
    x=Shell(appname & ".EXE")
    channel = DDEInitiate(appname, topic)
    If channel=0 then
        MsgBox "Unable to open Write."
        Exit Sub
    End If
    DDEPoke channel, item, testtext
    pcommand="[%FS(" + Chr$(34) + "C:\TEMP001" + Chr$(34) + ")]"
    DDEExecute channel, pcommand
    msgtext="The text: " & testtext & " saved to C:\TEMP001." & Chr$(13)
    msgtext=msgtext & Chr$(13) & "Delete? (Y/N)"
    answer=InputBox(msgtext)
    If answer="Y" or answer="y" then
        Kill "C:\TEMP001"
    End If
    DDETerminate channel
    Exit Sub
Errhandler:
    If Err<>0 then
      MsgBox "DDE Access failed."
    End If
End Sub
```

See Also

- DDEAppReturnCode()
- DDEExecute
- DDEInitiate()
- DDERequest()
- DDETerminate

' DDERequest Function Example

Close | Copy All | Copy | Print

```
'This example opens a channel to the Paintbrush application and uses DDERequest to display the
list of topics available (using the System topic).

Sub main
    Dim channel as Integer
    Dim appname as String
    Dim topic as String
    Dim item as String
    Dim pcommand as String
    Dim msgtext as String
    appname="Pbrush"
    topic="System"
    item="Topics"
    x=Shell(appname & ".EXE")
    channel = DDEInitiate(appname, topic)
    If channel=0 then
        msgtext="Unable to open Paintbrush."
    Else
        On Error Resume Next
        msgtext="The Paintbrush topics available are:" & Chr$(13)
        msgtext=msgtext & Chr$(13) & DDERequest(channel,item)
        DDETerminate channel
        If Err<>0 then
            msgtext="DDE Access failed."
        End If
    End If
    MsgBox msgtext
End Sub
```

See Also

- DDEAppReturnCode()
- DDEExecute
- DDEInitiate()
- DDEPoke
- DDETerminate

' DDETerminate Statement Example

Close | Copy All | Copy | Print

'This example opens a channel to the Paintbrush application, displays the list of topics available
(using the System topic) and then terminates the channel using DDETerminate.

```
Sub main
    Dim channel as Integer
    Dim appname as String
    Dim topic as String
    Dim item as String
    Dim pcommand as String
    Dim msgtext as String
    appname="Pbrush"
    topic="System"
    item="Topics"
    x=Shell(appname & ".EXE")
    channel = DDEInitiate(appname, topic)
    If channel=0 then
        msgtext="Unable to open Paintbrush."
    Else
        On Error Resume Next
        msgtext="The Paintbrush topics available are:" & Chr$(13)
        msgtext=msgtext & Chr$(13) & DDERequest(channel,item)
        DDETerminate channel
        If Err<>0 then
            msgtext="DDE Access failed."
        End If
    End If
    MsgBox msgtext
End Sub
```

See Also

- DDEAppReturnCode()
- DDEExecute
- DDEInitiate()
- DDEPoke
- DDERequest()

Close | Copy All | Copy | Print

' Declare Statement Example

```
'This example returns the square of a number.

Declare Sub MessageBoxA Lib "user32.dll" _
                (ByVal h%, ByVal t$, ByVal c$, ByVal u%)

Declare Function ForwardRefFunc(y%)

Sub main
    Dim x As Integer, z As Integer

    z = InputBox("Type a number to be squared.")

    x = ForwardRefFunc(z)
    MessageBoxA 0, "The answer is: " & x, "Square", 64
End Sub

Function ForwardRefFunc(y As Integer)
    ForwardRefFunc = y^2
End Function
```

See Also

- Call
- Const
- Def*type*
- Dim
- $Include
- Static
- Type

' Deftype Statement Example

Close | Copy All | Copy | Print

'This example finds the average of bowling scores entered by the user. Since the variable *average* begins with A, it is automatically defined as a single-precision floating point number. The other variables will be defined as Integers.

```
DefInt c,s,t
DefSng a
Sub main
    Dim count
    Dim total
    Dim score
    Dim average
    Dim msgtext
    For count=0 to 4
        score=InputBox("Enter bowling score #" & count+1 &":")
        total=total+score
    Next count
    average=total/count
    msgtext="Your average is: " &average
    MsgBox msgtext
End Sub
```

See Also

- Declare
- Dim
- Let
- Type

' Dialog Function Example

Close | Copy All | Copy | Print

'This example creates a dialog box with a drop down combo box in it and three buttons: OK, Cancel, and Help. The Dialog function used here enables the subroutine to trap when the user clicks on any of these buttons.

```
Sub main
    Dim cchoices as String
    cchoices="All"+Chr$(9)+"Nothing"
     Begin Dialog UserDialog 180, 95, "OPEN Script Dialog Box"
         ButtonGroup .ButtonGroup1
         Text   9, 3, 69, 13, "Filename:", .Text1
         ComboBox   9, 17, 111, 41, cchoices, .ComboBox1
         OKButton   131, 8, 42, 13
         CancelButton   131, 27, 42, 13
         PushButton 132, 48, 42, 13, "Help", .Push1
     End Dialog
    Dim mydialogbox As UserDialog
    answer= Dialog(mydialogbox)
    Select Case answer
        Case -1
            MsgBox "You pressed OK"
        Case 0
            MsgBox "You pressed Cancel"
        Case 1
            MsgBox "You pressed Help"
    End Select
End Sub
```

See Also

Begin Dialog...End Dialog
Dialog Statement

' Dialog Statement Example

Close | Copy All | Copy | Print

'This example defines and displays a dialog box defined as *UserDialog* and named *mydialogbox*. If the user presses the Cancel button, an error code of 102 is returned and is trapped by the If...Then statement listed after the Dialog statement.

```
Sub main
    Dim cchoices as String
    On Error Resume Next
    cchoices="All"+Chr$(9)+"Nothing"
     Begin Dialog UserDialog 180, 95, "OPEN Script Dialog Box"
         ButtonGroup .ButtonGroup1
         Text   9, 3, 69, 13, "Filename:", .Text1
         ComboBox   9, 17, 111, 41, cchoices, .ComboBox1
         OKButton   131, 8, 42, 13
         CancelButton   131, 27, 42, 13
End Dialog
    Dim mydialogbox As UserDialog
    Dialog mydialogbox
    If Err=102 then
        MsgBox "You pressed Cancel."
    Else
        MsgBox "You pressed OK."
    End If
End Sub
```

See Also

Begin Dialog...End Dialog
Dialog Function

Close | Copy All | Copy | Print

' Dim Statement Example

```
'This example shows a Dim statement for each of the possible data types.

Rem Must define a record type before you can declare a record variable
    Type Testrecord
        Custno As Integer
        Custname As String
    End Type

Sub main
    Dim counter As Integer
    Dim fixedstring As String*25
    Dim varstring As String
    Dim myrecord As Testrecord
    Dim ole2var As Object
    Dim F(1 to 10), A()
'   ...(code here)...
End Sub
```

See Also

- Global
- Option Base
- ReDim
- Set
- Static
- Type

Close | Copy All | Copy | Print

' Dir Function Example

```
'This example lists the contents of the diskette in drive A.

Sub main
    Dim msgret
    Dim directory, count
    Dim x, msgtext
    Dim A()
    msgret=MsgBox("Insert a disk in drive A.")
    count=1
    ReDim A(100)
    directory=Dir ("A:\*.*")
    Do While directory<>""
        A(count)=directory
        count=count+1
        directory=Dir
    Loop
    msgtext="Contents of drive A:\ is:" & Chr(10) & Chr(10)
    For x=1 to count
        msgtext=msgtext & A(x) & Chr(10)
    Next x
    MsgBox msgtext
End Sub
```

See Also

- ChDir
- ChDrive
- CurDir
- MkDir
- RmDir

Close | Copy All | Copy | Print

' DlgControlID Function Example

'This example displays a dialog box similar to File Open.

```
Declare Sub ListFiles(str1$)
Declare Function FileDlgFunction(identifier$, action, suppvalue)

Sub main
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Dim filetypes as String
    Dim exestr$()
    Dim button as Integer
    Dim x as Integer
    Dim directory as String
    filetypes="Program files (*.exe)"+Chr$(9)+"All Files (*.*)"
    Begin Dialog newdlg 230, 145, "Open", .FileDlgFunction
        '$CStrings Save
        Text   8, 6, 60, 11, "&Filename:"
        TextBox   8, 17, 76, 13, .TextBox1
        ListBox   9, 36, 75, 61, exestr$(), .ListBox1
        Text   8, 108, 61, 9, "List Files of &Type:"
        DropListBox   7, 120, 78, 30, filetypes, .DropListBox1
        Text   98, 7, 43, 10, "&Directories:"
        Text   98, 20, 46, 8, "c:\\windows"
        ListBox   99, 34, 66, 66, "", .ListBox2
        Text   98, 108, 44, 8, "Dri&ves:"
        DropListBox   98, 120, 68, 12, "", .DropListBox2
        OKButton   177, 6, 50, 14
        CancelButton   177, 24, 50, 14
        PushButton 177, 42, 50, 14, "&Help"
        '$CStrings Restore
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Sub ListFiles(str1$)
    DlgText 1,str1$
    x=0
    Redim exestr$(x)
    directory=Dir$("c:\windows\" & str1$,16)
    If directory<>"" then
      Do
        exestr$(x)=LCase$(directory)
        x=x+1
        Redim Preserve exestr$(x)
        directory=Dir
        Loop Until directory=""
    End If
    DlgListBoxArray 2,exestr$()
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
      Case 1
        str1$="*.exe"                          'dialog box initialized
        ListFiles str1$
      Case 2                                   'button or control value changed
        If DlgControlId(identifier$) = 4 Then
            If DlgText(4)="All Files (*.*)" then
                str1$="*.*"
            Else
                str1$="*.exe"
```

```
            End If
          ListFiles str1$
          End If
        Case 3                          'text or combo box changed
          str1$=DlgText$(1)
          ListFiles str1$
        Case 4                          'control focus changed

        Case 5                          'idle
    End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgEnable Function
- DlgEnable Statement
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Function
- DlgText Statement
- DlgValue Function
- DlgValue Statement
- DlgVisible Function
- DlgVisible Statement

' DlgEnable Statement Example

Close | Copy All | Copy | Print

```
'This example displays a dialog box with two check boxes, one labeled Either, the other labeled Or.
If the user clicks on Either, the Or option is grayed. Likewise, if Or is selected, Either is grayed.
'This example uses the DlgEnable statement to toggle the state of the buttons.

Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92,"DlgEnable example", .FileDlgFunction
        OKButton   130, 6, 50, 14
        CancelButton   130, 23, 50, 14
        CheckBox   34, 25, 75, 19, "Either", .CheckBox1
        CheckBox   34, 43, 73, 25, "Or", .CheckBox2
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 2                              'button or control value changed
         If DlgControlId(identifier$) = 2 Then
            DlgEnable 3
         Else
            DlgEnable 2
            End If
    End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Function
- DlgText Statement
- DlgValue Function
- DlgValue Statement
- DlgVisible Function
- DlgVisible Statement

' DlgEnable Function Example

'This example displays a dialog box with one check box, labeled Show More, and a group box, labeled More, with two option buttons, Option 1 and Option 2. It uses the DlgEnable function to enable the More group box and its options if the Show More check box is selected.

```
Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgEnable example", .FileDlgFunction
        OKButton   130, 6, 50, 14
        CancelButton   130, 23, 50, 14
        CheckBox   13, 6, 75, 19, "Show more", .CheckBox1
        GroupBox   16, 28, 94, 50, "More"
        OptionGroup .OptionGroup1
            OptionButton   23, 40, 56, 12, "Option 1", .OptionButton1
            OptionButton   24, 58, 61, 13, "Option 2", .OptionButton2
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 1
            DlgEnable 3,0
            DlgEnable 4,0
            DlgEnable 5,0
        Case 2                              'button or control value changed
         If DlgControlID(identifier$) = 2 Then
            If DlgEnable (3)=0 then
                  DlgEnable 3,1
                  DlgEnable 4,1
                  DlgEnable 5,1
            Else
                DlgEnable 3,0
                DlgEnable 4,0
                DlgEnable 5,0
            End If
         End If
    End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Statement
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Function
- DlgText Statement
- DlgValue Function
- DlgValue Statement
- DlgVisible Function
- DlgVisible Statement

' DlgEnd Statement Example

Close | Copy All | Copy | Print

'This example displays a dialog box with the message "You have 30 seconds to cancel." The dialog box counts down from 30 seconds to 0. If the user clicks OK or Cancel during the countdown, the dialog box closes. If the countdown reaches 0, however, the DlgEnd statement closes the dialog box.

```
Function timeout(id$,action%,suppvalue&)
   Static timeoutStart as Long
   Static currentSecs as Long
   Dim thisSecs as Long
   Select Case action%
      Case 1
         ' initialize the dialog box. Set the ticker value to 30
         ' and remember when we put up the dialog box
         DlgText "ticker", "30"
         timeoutStart = timer
         currentSecs = 30
      Case 5
         ' this is an idle message - set thisSecs to the number of
         ' seconds left until timeout
         thisSecs = timer
         If thisSecs < timeoutStart Then thisSecs = thisSecs + 24*60*60
         thisSecs = 30 - (thisSecs - timeoutStart)
         ' if there are negative seconds left, timeout!
         If thisSecs < 0 Then DlgEnd -1
         ' If the seconds left has changed since last time,
         ' update the dialog box
         If thisSecs <> currentSecs Then
            DlgText "ticker", trim$(str$(thisSecs))
            currentSecs = thisSecs
         End If
         ' make sure to return non-zero so we keep getting idle messages
         timeout = 1
    End Select
End Function

Sub main
   Begin Dialog newdlg 167, 78, "Do You Want to Continue?", .timeout
      '$CStrings Save
      OKButton   27, 49, 50, 14
      CancelButton   91, 49, 50, 14
      Text   24, 14, 119, 8, "This is your last chance to bail out."
      Text   27, 30, 35, 8, "You have"
      Text   62, 30, 13, 8, "30", .ticker
      Text   74, 30, 66, 8, "seconds to cancel."
      '$CStrings Restore
   End Dialog
   Dim dlgVar As newdlg
   If dialog(dlgvar) = 0 Then
      Exit Sub                  ' abort
   End If
   ' do whatever it is we want to do
End Sub
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgEnable Statement
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Function
- DlgText Statement
- DlgValue Function
- DlgValue Statement
- DlgVisible Function
- DlgVisible Statement

' DlgFocus Function Example

Close | Copy All | Copy | Print

'This example displays a dialog box with a check box, labeled Check1, and a text box, labeled Text Box 1, in it. When the box is initialized, the focus is set to the text box. As soon as the user clicks the check box, the focus goes to the OK button.

```
Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgFocus Example", .FileDlgFunction
        OKButton   130, 6, 50, 14
        CancelButton   130, 23, 50, 14
        TextBox   15, 37, 82, 12, .TextBox1
        Text   15, 23, 57, 10, "Text Box 1"
        CheckBox   15, 6, 75, 11, "Check1", .CheckBox1
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 1
            DlgFocus 2
        Case 2                          'user changed control or clicked a button
            If DlgFocus() <> "OKButton" then
                    DlgFocus 0
            End If
        End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgEnable Statement
- DlgFocus Statement
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Function
- DlgText Statement
- DlgValue Function
- DlgValue Statement
- DlgVisible Function
- DlgVisible Statement

' DlgFocus Statement Example

Close | Copy All | Copy | Print

'This example displays a dialog box with a check box, labeled Check1, and a text box, labeled Text Box 1, in it. When the box is initialized, the focus is set to the text box. As soon as the user clicks the check box, the focus goes to the OK button.

```
Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgFocus Example", .FileDlgFunction
        OKButton   130, 6, 50, 14
        CancelButton   130, 23, 50, 14
        TextBox   15, 37, 82, 12, .TextBox1
        Text   15, 23, 57, 10, "Text Box 1"
        CheckBox   15, 6, 75, 11, "Check1", .CheckBox1
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 1
            DlgFocus 2
        Case 2                          'user changed control or clicked a button
            If DlgFocus() <> "OKButton" then
                DlgFocus 0
            End If
        End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgEnable Statement
- DlgFocus Function
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Function
- DlgText Statement
- DlgValue Function
- DlgValue Statement
- DlgVisible Function
- DlgVisible Statement

' DlgListBoxArray Function Example

Close | Copy All | Copy | Print

```
'This example displays a dialog box with a check box, labeled "Display List", and an empty list box.
If the user clicks the check box, the list box is filled with the contents of the array called
"myarray". The DlgListBox Array function makes sure the list box is empty.

Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgListBoxArray Example", .FileDlgFunction
        '$CStrings Save
        OKButton   130, 6, 50, 14
        CancelButton   130, 23, 50, 14
        ListBox   19, 26, 74, 59, "", .ListBox1
        CheckBox   12, 4, 86, 13, "Display List", .CheckBox1
        '$CStrings Restore
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
Dim myarray$(3)
Dim msgtext as Variant
Dim x as Integer
For x= 0 to 2
    myarray$(x)=Chr$(x+65)
Next x
    Select Case action
        Case 1
        Case 2                          'user changed control or clicked a button
            If DlgControlID(identifier$)=3 then
                If DlgListBoxArray(2)=0 then
                        DlgListBoxArray 2, myarray$()
                End If
            End If
        End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgEnable Statement
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Function
- DlgText Statement
- DlgValue Function
- DlgValue Statement
- DlgVisible Function
- DlgVisible Statement

' DlgListBoxArray Statement Example

Close | Copy All | Copy | Print

'This example displays a dialog box similar to File Open.

```
Declare Sub ListFiles(str1$)
Declare Function FileDlgFunction(identifier$, action, suppvalue)

Sub main
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Dim filetypes as String
    Dim exestr$()
    Dim button as Integer
    Dim x as Integer
    Dim directory as String
    filetypes="Program files (*.exe)"+Chr$(9)+"All Files (*.*)"
    Begin Dialog newdlg 230, 145, "Open", .FileDlgFunction
        '$CStrings Save
        Text   8, 6, 60, 11, "&Filename:"
        TextBox   8, 17, 76, 13, .TextBox1
        ListBox   9, 36, 75, 61, exestr$(), .ListBox1
        Text   8, 108, 61, 9, "List Files of &Type:"
        DropListBox   7, 120, 78, 30, filetypes, .DropListBox1
        Text   98, 7, 43, 10, "&Directories:"
        Text   98, 20, 46, 8, "c:\\windows"
        ListBox   99, 34, 66, 66, "", .ListBox2
        Text   98, 108, 44, 8, "Dri&ves:"
        DropListBox   98, 120, 68, 12, "", .DropListBox2
        OKButton   177, 6, 50, 14
        CancelButton   177, 24, 50, 14
        PushButton 177, 42, 50, 14, "&Help"
        '$CStrings Restore
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Sub ListFiles(str1$)
    DlgText 1,str1$
    x=0
    Redim exestr$(x)
    directory=Dir$("c:\windows\" & str1$,16)
    If directory<>"" then
      Do
      exestr$(x)=LCase$(directory)
      x=x+1
      Redim Preserve exestr$(x)
      directory=Dir
      Loop Until directory=""
    End If
    DlgListBoxArray 2,exestr$()
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
      Case 1
        str1$="*.exe"                          'dialog box initialized
        ListFiles str1$
      Case 2                                   'button or control value changed
        If DlgControlId(identifier$) = 4 Then
            If DlgText(4)="All Files (*.*)" then
                str1$="*.*"
            Else
                str1$="*.exe"
```

```
            End If
          ListFiles str1$
          End If
        Case 3                          'text or combo box changed
          str1$=DlgText$(1)
          ListFiles str1$
        Case 4                          'control focus changed

        Case 5                          'idle
    End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Function
- DlgEnable
- DlgSetPicture
- DlgText Function
- DlgText Statement
- DlgValue Function
- DlgValue Statement
- DlgVisible Function
- DlgVisible Statement

' DlgSetPicture Statement Example

Close | Copy All | Copy | Print

'This example displays a picture in a dialog box and changes the picture if the user clicks the check box labeled "Change Picture".

```
Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgSetPicture Example", .FileDlgFunction
        OKButton   130, 6, 50, 14
        CancelButton   130, 23, 50, 14
        Picture   43, 28, 49, 31, "C:\WINDOWS\THATCH.BMP", 0
        CheckBox   30, 8, 62, 15, "Change Picture", .CheckBox1
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 1
        Case 2                          'user changed control or clicked a button
            If DlgControlID(identifier$)=3 then
                If suppvalue=1 then
                    DlgSetPicture 2, "C:\WINDOWS\WINLOGO.BMP",0
                Else
                    DlgSetPicture 2, "C:\WINDOWS\THATCH.BMP",0
                End If
            End If
    End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgEnable Statement
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgText Function
- DlgText Statement
- DlgValue Function
- DlgValue Statement
- DlgVisible Function
- DlgVisible Statement

Close | Copy All | Copy | Print

' DlgText Function Example

```
'This example displays a dialog box similar to File Open. It uses DlgText to determine what group
of files to display.

Declare Sub ListFiles(str1$)
Declare Function FileDlgFunction(identifier$, action, suppvalue)

Sub main
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Dim filetypes as String
    Dim exestr$()
    Dim button as Integer
    Dim x as Integer
    Dim directory as String
    filetypes="Program files (*.exe)"+Chr$(9)+"All Files (*.*)"
    Begin Dialog newdlg 230, 145, "Open", .FileDlgFunction
        '$CStrings Save
        Text   8, 6, 60, 11, "&Filename:"
        TextBox   8, 17, 76, 13, .TextBox1
        ListBox   9, 36, 75, 61, exestr$(), .ListBox1
        Text   8, 108, 61, 9, "List Files of &Type:"
        DropListBox   7, 120, 78, 30, filetypes, .DropListBox1
        Text   98, 7, 43, 10, "&Directories:"
        Text   98, 20, 46, 8, "c:\\windows"
        ListBox   99, 34, 66, 66, "", .ListBox2
        Text   98, 108, 44, 8, "Dri&ves:"
        DropListBox   98, 120, 68, 12, "", .DropListBox2
        OKButton   177, 6, 50, 14
        CancelButton   177, 24, 50, 14
        PushButton 177, 42, 50, 14, "&Help"
        '$CStrings Restore
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Sub ListFiles(str1$)
    DlgText 1,str1$
    x=0
    Redim exestr$(x)
    directory=Dir$("c:\windows\" & str1$,16)
    If directory<>"" then
      Do
      exestr$(x)=LCase$(directory)
      x=x+1
      Redim Preserve exestr$(x)
      directory=Dir
      Loop Until directory=""
    End If
    DlgListBoxArray 2,exestr$()
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
      Case 1
        str1$="*.exe"                          'dialog box initialized
        ListFiles str1$
      Case 2                              'button or control value changed
        If DlgControlId(identifier$) = 4 Then
            If DlgText(4)="All Files (*.*)" then
                str1$="*.*"
            Else
```

```
            str1$="*.exe"
          End If
        ListFiles str1$
        End If
      Case 3                        'text or combo box changed
        str1$=DlgText$(1)
        ListFiles str1$
      Case 4                        'control focus changed

      Case 5                        'idle
    End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgEnable Statement
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Statement
- DlgValue Function
- DlgValue Statement
- DlgVisible Function
- DlgVisible Statement

' DlgText Statement Example

Close | Copy All | Copy | Print

'This example displays a dialog box similar to File Open. It uses the DlgText statement to display the list of files in the Filename list box.

```
Declare Sub ListFiles(str1$)
Declare Function FileDlgFunction(identifier$, action, suppvalue)

Sub main
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Dim filetypes as String
    Dim exestr$()
    Dim button as Integer
    Dim x as Integer
    Dim directory as String
    filetypes="Program files (*.exe)"+Chr$(9)+"All Files (*.*)"
    Begin Dialog newdlg 230, 145, "Open", .FileDlgFunction
        '$CStrings Save
        Text   8, 6, 60, 11, "&Filename:"
        TextBox   8, 17, 76, 13, .TextBox1
        ListBox   9, 36, 75, 61, exestr$(), .ListBox1
        Text   8, 108, 61, 9, "List Files of &Type:"
        DropListBox   7, 120, 78, 30, filetypes, .DropListBox1
        Text   98, 7, 43, 10, "&Directories:"
        Text   98, 20, 46, 8, "c:\\windows"
        ListBox   99, 34, 66, 66, "", .ListBox2
        Text   98, 108, 44, 8, "Dri&ves:"
        DropListBox   98, 120, 68, 12, "", .DropListBox2
        OKButton   177, 6, 50, 14
        CancelButton   177, 24, 50, 14
        PushButton 177, 42, 50, 14, "&Help"
        '$CStrings Restore
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Sub ListFiles(str1$)
    DlgText 1,str1$
    x=0
    Redim exestr$(x)
    directory=Dir$("c:\windows\" & str1$,16)
    If directory<>"" then
      Do
      exestr$(x)=LCase$(directory)
      x=x+1
      Redim Preserve exestr$(x)
      directory=Dir
      Loop Until directory=""
    End If
    DlgListBoxArray 2,exestr$()
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
      Case 1
        str1$="*.exe"                          'dialog box initialized
        ListFiles str1$
      Case 2                              'button or control value changed
        If DlgControlId(identifier$) = 4 Then
            If DlgText(4)="All Files (*.*)" then
                str1$="*.*"
            Else
```

```
            str1$="*.exe"
          End If
        ListFiles str1$
        End If
      Case 3                        'text or combo box changed
        str1$=DlgText$(1)
        ListFiles str1$
      Case 4                        'control focus changed

      Case 5                        'idle
  End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgEnable Statement
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Function
- DlgValue Function
- DlgValue Statement
- DlgVisible Function
- DlgVisible Statement

' DlgValue Function Example

Close | Copy All | Copy | Print

'This example changes the picture in the dialog box if the check box is selected and changes the picture to its original bitmap if the checkbox is turned off.

```
Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgSetPicture Example", .FileDlgFunction
        OKButton   130, 6, 50, 14
        CancelButton   130, 23, 50, 14
        Picture   43, 28, 49, 31, "C:\WINDOWS\THATCH.BMP", 0
        CheckBox   30, 8, 62, 15, "Change Picture", .CheckBox1
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 1
        Case 2                          'user changed control or clicked a button
            If DlgControlID(identifier$)=3 then
                If DlgValue(3)=1 then
                    DlgSetPicture 2, "C:\WINDOWS\WINLOGO.BMP",0
                Else
                    DlgSetPicture 2, "C:\WINDOWS\THATCH.BMP",0
                End If
            End If
    End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgEnable Statement
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Function
- DlgText Statement
- DlgValue Statement
- DlgVisible Function
- DlgVisible Statement

' DlgValue Statement Example

Close | Copy All | Copy | Print

'This example displays a dialog box with a checkbox, labeled Change Option, and a group box with two option buttons, labeled Option 1 and Option 2. When the user clicks the Change Option button, Option 2 is selected.

```
Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgValue Example", .FileDlgFunction
        OKButton   130, 6, 50, 14
        CancelButton   130, 23, 50, 14
        CheckBox   30, 8, 62, 15, "Change Option", .CheckBox1
        GroupBox   28, 34, 79, 47, "Group"
        OptionGroup .OptionGroup1
            OptionButton   41, 47, 52, 10, "Option 1", .OptionButton1
            OptionButton   41, 62, 58, 11, "Option 2", .OptionButton2
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 1
        Case 2                          'user changed control or clicked a button
          If DlgControlID(identifier$)=2 then
              If DlgValue(2)=1 then
                  DlgValue 4,1
              Else
                  DlgValue 4,0
              End If
          End If
    End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgEnable Statement
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Function
- DlgText Statement
- DlgValue Function
- DlgVisible Function
- DlgVisible Statement

' DlgVisible Function Example

Close | Copy All | Copy | Print

'This example displays Option 2 in the Group box if the user clicks the check box labeled "Show Option 2". If the user clicks the box again, Option 2 is hidden.

```
Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgVisible Example", .FileDlgFunction
        OKButton   130, 6, 50, 14
        CancelButton   130, 23, 50, 14
        CheckBox   30, 8, 62, 15, "Show Option 2", .CheckBox1
        GroupBox   28, 34, 79, 47, "Group"
        OptionGroup .OptionGroup1
            OptionButton   41, 47, 52, 10, "Option 1", .OptionButton1
            OptionButton   41, 62, 58, 11, "Option 2", .OptionButton2
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 1
            DlgVisible 6,0
        Case 2                          'user changed control or clicked a button
          If DlgControlID(identifier$)=2 then
              If DlgVisible(6)<>1 then
                  DlgVisible 6
              End If
          End If
    End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgEnable Statement
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Function
- DlgText Statement
- DlgValue Function
- DlgValue Statement
- DlgVisible Statement

' DlgVisible Statement Example

Close | Copy All | Copy | Print

'This example displays Option 2 in the Group box if the user clicks the check box. labeled "Show Option 2". If the user clicks the box again, Option 2 is hidden.

```
Declare Function FileDlgFunction(identifier$, action, suppvalue)
Sub Main
    Dim button as integer
    Dim identifier$
    Dim action as Integer
    Dim suppvalue as Integer
    Begin Dialog newdlg 186, 92, "DlgVisible Example", .FileDlgFunction
        OKButton   130, 6, 50, 14
        CancelButton   130, 23, 50, 14
        CheckBox   30, 8, 62, 15, "Show Option 2", .CheckBox1
        GroupBox   28, 34, 79, 47, "Group"
        OptionGroup .OptionGroup1
            OptionButton   41, 47, 52, 10, "Option 1", .OptionButton1
            OptionButton   41, 62, 58, 11, "Option 2", .OptionButton2
    End Dialog
    Dim dlg As newdlg
    button = Dialog(dlg)
End Sub

Function FileDlgFunction(identifier$, action, suppvalue)
    Select Case action
        Case 1
            DlgVisible 6,0
        Case 2                          'user changed control or clicked a button
          If DlgControlID(identifier$)=2 then
              If DlgVisible(6)<>1 then
                  DlgVisible 6
              End If
          End If
    End Select
End Function
```

See Also

- BeginDialog...End Dialog
- DlgControlID Function
- DlgEnable Function
- DlgEnable Statement
- DlgFocus Function
- DlgFocus Statement
- DlgListBoxArray Function
- DlgListBoxArray Statement
- DlgSetPicture
- DlgText Function
- DlgText Statement
- DlgValue Function
- DlgVisible Function

Close | Copy All | Copy | Print

' Do...Loop Statement Example

```
'This example lists the contents of the diskette in drive A.

Sub main
Dim msgret
    Dim directory, count
    Dim x, msgtext
    Dim A()
    msgret=MsgBox("Insert a disk in drive A.")
    count=1
    ReDim A(100)
    directory=Dir ("A:\*.*")
    Do While directory<>""
        A(count)=directory
        count=count+1
        directory=Dir
    Loop
    msgtext="Directory of drive A:\ is:" & Chr(10)
    For x=1 to count
        msgtext=msgtext & A(x) & Chr(10)
    Next x
    MsgBox msgtext
End Sub
```

See Also

- Exit
- For...Next
- Stop
- While...Wend

' DoEvents Statement Example

Close | Copy All | Copy | Print

'This example activates the Windows Terminal application, dials the number and then allows the operating system to process events.

```
Sub main
    Dim phonenumber, msgtext
    Dim x
    phonenumber=InputBox("Type telephone number to call:")
    x=Shell("Terminal.exe",1)
    SendKeys "%PD" & phonenumber & "{Enter}",1
    msgtext="Dialing..."
    MsgBox msgtext
    DoEvents
End Sub
```

See Also

- AppActivate
- SendKeys
- Shell

' DropComboBox Statement Example

Close | Copy All | Copy | Print

```
'This example defines a dialog box with a drop combo box and the OK and Cancel buttons.

Sub main
    Dim cchoices as String
    On Error Resume Next
    cchoices="All"+Chr$(9)+"Nothing"
    Begin Dialog UserDialog 180, 95, "OPEN Script Dialog Box"
        ButtonGroup .ButtonGroup1
        Text   9, 3, 69, 13, "Filename:", .Text1
        DropComboBox   9, 17, 111, 41, cchoices, .ComboBox1
        OKButton   131, 8, 42, 13
        CancelButton   131, 27, 42, 13
    End Dialog
    Dim mydialogbox As UserDialog
    Dialog mydialogbox
    If Err=102 then
        MsgBox "You pressed Cancel."
    Else
        MsgBox "You pressed OK."
    End If
End Sub
```

See Also

- Begin Dialog...End Dialog Statement
- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- DropListBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

' DropListBox Statement Example

Close | Copy All | Copy | Print

'This example defines a dialog box with a drop list box and the OK and Cancel buttons.

```
Sub main
    Dim DropListBox1() as String
    ReDim DropListBox1(3)
    For x=0 to 2
      DropListBox1(x)=Chr(65+x) & ":"
    Next x
    Begin Dialog UserDialog 186, 62, "OPEN Script Dialog Box"
        Text   8, 4, 42, 8, "Drive:", .Text3
        DropListBox   8, 16, 95, 44, DropListBox1(), .DropListBox1
        OKButton   124, 6, 54, 14
        CancelButton   124, 26, 54, 14
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin Dialog...End Dialog Statement
- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- DropComboBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

' Environ Statement Example

Close | Copy All | Copy | Print

'This example lists all the strings from the operating system environment table.

```
Sub main
    Dim str1(100)
    Dim msgtext
    Dim count, x
    Dim newline
    newline=Chr(10)
    x=1
    str1(x)= Environ(x)
    Do While Environ(x)<>""
        str1(x)= Environ(x)
        x=x+1
        str1(x)=Environ(x)
    Loop
    msgtext="The Environment Strings are:" & newline & newline
    count=x
    For x=1 to count
        msgtext=msgtext & str1(x) & newline
    Next x
    MsgBox msgtext
End Sub
```

See Also

(None)

' Eof Function Example

Close | Copy All | Copy | Print

'This example uses the Eof function to read records from a Random file, using a Get statement.
The Eof function keeps the Get statement from attempting to read beyond the end of the file. The
subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram.

```
Declare Sub createfile()
Sub main
    Dim acctno
    Dim msgtext as String
    newline=Chr(10)
    Call createfile
    Open "C:\temp001" For Input As #1
    msgtext="The account numbers are:" & newline
    Do While Not Eof(1)
            Input #1,acctno
            msgtext=msgtext & newline & acctno & newline
    Loop
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the numbers 1-10 into a file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=1 to 10
        Write #1, x
    Next x
    Close #1
End Sub
```

See Also

- Get
- Input Function
- Input Statement
- Line Input
- Loc
- Lof
- Open

Close | Copy All | Copy | Print

' Erase Statement Example

'This example prompts for a list of item numbers to put into an array and clears array if the user wants to start over.

```
Sub main
    Dim msgtext
    Dim inum(100) as Integer
    Dim x, count
    Dim newline
    newline=Chr(10)
    x=1
    count=x
    inum(x)=0
    Do
      inum(x)=InputBox("Enter item #" & x & " (99=start over;0=end):")
      If inum(x)=99 then
          Erase inum()
          x=0
      ElseIf inum(x)=0 then
          Exit Do
      End If
      x=x+1
  Loop
  count=x-1
  msgtext="You entered the following numbers:" & newline
  For x=1 to count
      msgtext=msgtext & inum(x) & newline
  Next x
  MsgBox msgtext
End Sub
```

See Also

- Dim
- ReDim
- LBound
- UBound

Close | Copy All | Copy | Print

' Erl Function Example

'This example prints the error number using the Err function and the line number using the Erl statement if an error occurs during an attempt to open a file. Line numbers are automatically assigned, starting with 1, which is the **Sub main** statement.

```
Sub main
    Dim msgtext, userfile
    On Error GoTo Debugger
    msgtext="Enter the filename to use:"
    userfile=InputBox$(msgtext)
    Open userfile For Input As #1
    MsgBox "File opened for input."
'   ....etc....
    Close #1
done:
    Exit Sub
Debugger:
    msgtext="Error number " & Err & " occurred at line: " & Erl
    MsgBox msgtext
    Resume done
End Sub
```

See Also

- Err Function
- Err Statement
- Error Function
- Error Statement
- On Error
- Resume
- Trappable Errors

' Err Function Example

Close | Copy All | Copy | Print

'This example prints the error number using the Err function and the line number using the Erl statement if an error occurs during an attempt to open a file. Line numbers are automatically assigned, starting with 1, which is the **Sub main** statement.

```
Sub main
    Dim msgtext, userfile
    On Error GoTo Debugger
    msgtext="Enter the filename to use:"
    userfile=InputBox$(msgtext)
    Open userfile For Input As #1
    MsgBox "File opened for input."
'    ....etc....
    Close #1
done:
    Exit Sub
Debugger:
    msgtext="Error number " & Err & " occurred at line: " & Erl
    MsgBox msgtext
    Resume done
End Sub
```

See Also

- Erl
- Err Statement
- Error Function
- Error Statement
- On Error
- Resume
- Trappable Errors

' Err Statement Example

Close | Copy All | Copy | Print

'This example generates an error code of 10000 and displays an error message if a user does not enter a customer name when prompted for it. It uses the Err statement to clear any previous error codes before running the loop the first time and it also clears the error to allow the user to try again.

```
Sub main
    Dim custname as String
    On Error Resume Next
    Do
        Err=0
        custname=InputBox$("Enter customer name:")
        If custname="" then
            Error 10000
        Else
            Exit Do
        End If
        Select Case Err
            Case 10000
                MsgBox "You must enter a customer name."
            Case Else
                MsgBox "Undetermined error. Try again."
        End Select
    Loop Until custname<>""
    MsgBox "The name is: " & custname
End Sub
```

See Also

- Erl
- Err Function
- Error Function
- Error Statement
- On Error
- Resume
- Trappable Errors

Close | Copy All | Copy | Print

' Error Function Example

'This example prints the error number, using the Err function, and the text of the error, using the Error$ function, if an error occurs during an attempt to open a file.

```
Sub main
    Dim msgtext, userfile
    On Error GoTo Debugger
    msgtext="Enter the filename to use:"
    userfile=InputBox$(msgtext)
    Open userfile For Input As #1
    MsgBox "File opened for input."
'   ....etc....
    Close #1
done:
    Exit Sub
Debugger:
    msgtext="Error " & Err & ": " & Error$
    MsgBox msgtext
    Resume done
End Sub
```

See Also

- Erl
- Err Function
- Err Statement
- Error Statement
- On Error
- Resume
- Trappable Errors

' Error Statement Example

Close | Copy All | Copy | Print

'This example generates an error code of 10000 and displays an error message if a user does not enter a customer name when prompted for it.

```
Sub main
    Dim custname as String
    On Error Resume Next
    Do
        Err=0
        custname=InputBox$("Enter customer name:")
        If custname="" then
            Error 10000
        Else
            Exit Do
        End If
        Select Case Err
            Case 10000
                MsgBox "You must enter a customer name."
            Case Else
                MsgBox "Undetermined error. Try again."
        End Select
    Loop Until custname<>""
    MsgBox "The name is: " & custname
End Sub
```

See Also

- Erl
- Err Function
- Err Statement
- Error Function
- On Error
- Resume
- Trappable Errors

Close | Copy All | Copy | Print

' Exit Statement Example

'This example uses the On Error statement to trap run-time errors. If there is an error, the program execution continues at the label "Debugger". The example uses the Exit statement to skip over the debugging code when there is no error.

```
Sub main
    Dim msgtext, userfile
    On Error GoTo Debugger
    msgtext="Enter the filename to use:"
    userfile=InputBox$(msgtext)
    Open userfile For Input As #1
    MsgBox "File opened for input."
'   ....etc....
    Close #1
done:
    Exit Sub
Debugger:
    msgtext="Error " & Err & ": " & Error$
    MsgBox msgtext
    Resume done
End Sub
```

See Also

- Do...Loop
- For...Next
- Function...End Function
- Stop
- Sub...End Sub

' Exp Function Example

Close | Copy All | Copy | Print

'This example estimates the value of a factorial of a number entered by the user. A factorial (notated with an exclamation mark, !) is the product of a number and each integer between it and the number 1. For example, 5 factorial, or 5!, is the product of 5*4*3*2*1, or the value 120.

```
Sub main
    Dim x as Single
    Dim msgtext, PI
    Dim factorial as Double
    PI=3.14159
i: x=InputBox("Enter an integer between 1 and 88: ")
    If x<=0 then
        Exit Sub
    ElseIf x>88 then
        MsgBox "The number you entered is too large.   Try again."
        Goto i
    End If
    factorial=Sqr(2*PI*x)*(x^x/Exp(x))
    msgtext="The estimated factorial is: " & Format(factorial, "Scientific")
    MsgBox msgtext
End Sub
```

See Also

- Abs
- Fix
- Int
- Log
- Rnd
- Sgn
- Sqr

' FileAttr Function Example

Close | Copy All | Copy | Print

'This example closes an open file if it is open for Input or Output. If open for Append, it writes a range of numbers to the file. The second subprogram, CREATEFILE, creates the file and leaves it open.

```
Declare Sub createfile()
Sub main
    Dim filemode as Integer
    Dim attrib as Integer
    Call createfile
    attrib=1
    filemode=FileAttr(1,attrib)
    If filemode=1 or 2 then
        MsgBox "File was left open. Closing now."
        Close #1
    Else
        For x=11 to 15
            Write #1, x
        Next x
        Close #1
    End If
    Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the numbers 1-10 into a file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=1 to 10
        Write #1, x
    Next x
End Sub
```

See Also

- GetAttr
- Open
- SetAttr

' FileCopy Statement Example

Close | Copy All | Copy | Print

'This example copies one file to another. Both filenames are specified by the user.

```
Sub main
    Dim oldfile, newfile
    On Error Resume Next
    oldfile= InputBox("Copy which file?")
    newfile= InputBox("Copy to?")
    FileCopy oldfile,newfile
    If Err<>0 then
        msgtext="Error during copy. Rerun program."
    Else
        msgtext="Copy successful."
    End If
    MsgBox msgtext
End Sub
```

See Also

- FileAttr
- FileDateTime
- GetAttr
- Kill
- Name

' FileDateTime Function Example

Close | Copy All | Copy | Print

'This example writes data to a file if it hasn't been saved within the last 2 minutes.

```
Sub main
    Dim tempfile
    Dim filetime, curtime
    Dim msgtext
    Dim acctno(100) as Single
    Dim x, I
    tempfile="C:\TEMP001"
    Open tempfile For Output As #1
    filetime=FileDateTime(tempfile)
    x=1
    I=1
    acctno(x)=0
    Do
        curtime=Time
        acctno(x)=InputBox("Enter an account number (99 to end):")
        If acctno(x)=99 then
            For I=1 to x-1
                Write #1, acctno(I)
            Next I
            Exit Do
        ElseIf (Minute(filetime)+2)<=Minute(curtime) then
            For I=I to x
                Write #1, acctno(I)
            Next I
        End If
        x=x+1
    Loop
    Close #1
    x=1
    msgtext="Contents of C:\TEMP001 is:" & Chr(10)
    Open tempfile for Input as #1
    Do While Eof(1)<>-1
        Input #1, acctno(x)
        msgtext=msgtext & Chr(10) & acctno(x)
        x=x+1
    Loop
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub
```

See Also

FileLen
GetAttr

Close | Copy All | Copy | Print

' FileLen Function Example

'This example returns the length of a file.

```
Sub main
    Dim length as Long
    Dim userfile as String
    Dim msgtext
    On Error Resume Next
    msgtext="Enter a filename:"
    userfile=InputBox(msgtext)
    length=FileLen(userfile)
    If Err<>0 then
        msgtext="Error occurred.   Rerun program."
    Else
        msgtext="The length of " & userfile & " is: " & length
    End If
    MsgBox msgtext
End Sub
```

See Also

- FileDateTime
- FileLen
- GetAttr
- Lof

Close Copy All Copy Print

' Fix Function Example

'This example returns the integer portion of a number provided by the user.

```
Sub main
    Dim usernum
    Dim intvalue
    usernum=InputBox("Enter a number with decimal places:")
    intvalue=Fix(usernum)
    MsgBox "The integer portion of " & usernum & " is: " & intvalue
End Sub
```

See Also

- Abs
- CInt
- Exp
- Int
- Log
- Rnd
- Sgn
- Sqr

' For...Next Statement Example

Close | Copy All | Copy | Print

```
'This example calculates the factorial of a number. A factorial (notated with an exclamation
mark, !) is the product of a number and each integer between it and the number 1. For example, 5
factorial, or 5!, is the product of 5*4*3*2*1, or the value 120.

Sub main
    Dim number as Integer
    Dim factorial as Double
    Dim msgtext
    number=InputBox("Enter an integer between 1 and 170:")
    If number<=0 then
        Exit Sub
    End If
    factorial=1
    For x=number to 2 step -1
        factorial=factorial*x
    Next x
Rem If number<= 35, then its factorial is small enough
Rem to be stored as a single-precision number
    If number<35 then
        factorial=CSng(factorial)
    End If
    msgtext="The factorial of " & number & " is: " & factorial
    MsgBox msgtext
End Sub
```

See Also

- Do...Loop
- Exit
- While...Wend

' Format Function Example

Close | Copy All | Copy | Print

'This example calculates the square root of 2 as a double-precision floating point value and displays it in scientific notation.

```
Sub main
    Dim value
    Dim msgtext
    value=CDbl(Sqr(2))
    msgtext= "The square root of 2 is: " & Format(Value,"Scientific")
    MsgBox msgtext
End Sub
```

See Also

- Asc
- CCur
- CDbl
- Chr
- CInt
- CLng
- CSng
- CStr
- CVar
- CVDate
- Str

' FreeFile Function Example

Close | Copy All | Copy | Print

'This example opens a file and assigns to it the next file number available.

```
Sub main
    Dim filenumber
    Dim filename as String
    filenumber=FreeFile
    filename=InputBox("Enter a file to open: ")
    On Error Resume Next
    Open filename For Input As filenumber
    If Err<>0 then
        MsgBox "Error loading file.   Re-run program."
        Exit Sub
    End If
    MsgBox "File " & filename & " opened as number: " & filenumber
    Close #filenumber
    MsgBox "File now closed."
End Sub
```

See Also

Open

' Function...End Function Example

Close | Copy All | Copy | Print

```
'This example declares a function that is later called by the main subprogram. The function does
nothing but set its return value to 1.

Declare Function OPEN_Script_exfunction()
Sub main
    Dim y as Integer
    Call OPEN_Script_exfunction
    y=OPEN_Script_exfunction
    MsgBox "The value returned by the function is: " & y
End Sub

Function OPEN_Script_exfunction()
    OPEN_Script_exfunction=1
End Function
```

See Also

- Call
- Declare
- Dim
- Global
- IsMissing
- Option Explicit
- Static
- Sub...End Sub

' FV Function Example

Close | Copy All | Copy | Print

'This example finds the future value of an annuity, based on terms specified by the user.

```
Sub main
    Dim aprate, periods
    Dim payment, annuitypv
    Dim due, futurevalue
    Dim msgtext
    annuitypv=InputBox("Enter present value of the annuity: ")
    aprate=InputBox("Enter the annual percentage rate: ")
    If aprate >1 then
        aprate=aprate/100
    End If
    periods=InputBox("Enter the total number of pay periods: ")
    payment=InputBox("Enter the initial amount paid to you: ")
Rem Assume payments are made at end of month
    due=0
    futurevalue=FV(aprate/12,periods,-payment,-annuitypv,due)
    msgtext= "The future value is: " & Format(futurevalue, "Currency")
    MsgBox msgtext
End Sub
```

See Also

- IPmt
- IRR
- NPV
- Pmt
- PPmt
- PV
- Rate

' Get Statement Example

Close | Copy All | Copy | Print

'This example opens a file for Random access, gets its contents, and closes the file again. The second subprogram, CREATEFILE, creates the C:\TEMP001 file used by the main subprogram.

```
Declare Sub createfile()
Sub main
    Dim acctno as String*3
    Dim recno as Long
    Dim msgtext as String
    Call createfile
    recno=1
    newline=Chr(10)
    Open "C:\TEMP001" For Random As #1 Len=3
    msgtext="The account numbers are:" & newline
    Do Until recno=11
          Get #1,recno,acctno
          msgtext=msgtext & acctno
          recno=recno+1
    Loop
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the numbers 1-10 into a file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=1 to 10
       Write #1, x
    Next x
    Close #1
End Sub
```

See Also

- Open
- Put
- Type

' GetAttr Function Example

Close | Copy All | Copy | Print

```
'This example tests the attributes for a file and if it is hidden, changes it to a non-hidden file.

Sub main
    Dim filename as String
    Dim attribs, saveattribs as Integer
    Dim answer as Integer
    Dim archno as Integer
    Dim msgtext as String
    archno=32
    On Error Resume Next
    msgtext="Enter name of a file:"
    filename=InputBox(msgtext)
    attribs=GetAttr(filename)
    If Err<>0 then
        MsgBox "Error in filename. Re-run Program."
        Exit Sub
    End If
    saveattribs=attribs
    If attribs>= archno then
        attribs=attribs-archno
    End If
    Select Case attribs
        Case 2,3,6,7
            msgtext="   File: " &filename & " is hidden." & Chr(10)
            msgtext=msgtext & Chr(10) & "     Change it?"
            answer=Msgbox(msgtext,308)
            If answer=6 then
                SetAttr filename, saveattribs-2
                Msgbox "File is no longer hidden."
                Exit Sub
            End If
            MsgBox "Hidden file not changed."
        Case Else
            MsgBox "File was not hidden."
    End Select
End Sub
```

See Also

FileAttr
SetAttr

' GetField Function Example

Close | Copy All | Copy | Print

'This example finds the third value in a string, delimited by plus signs (+).

```
Sub main
    Dim teststring,retvalue
    Dim msgtext
    teststring="9+8+7+6+5"
    retvalue=GetField(teststring,3,"+")
    MsgBox "The third field in: " & teststring & " is: " & retvalue
End Sub
```

See Also

- Left
- LTrim
- Mid Function
- Mid Statement
- Right
- RTrim
- SetField
- StrComp
- Trim

' GetObject Function Example

Close | Copy All | Copy | Print

'This example displays a list of open files in the software application, VISIO. It uses the GetObject function to access VISIO. To see how this example works, you need to start VISIO and open one or more documents.

```
Sub main
    Dim visio as Object
    Dim doc as Object
    Dim msgtext as String
    Dim i as Integer, doccount as Integer

'Initialize Visio
    Set visio = GetObject(,"visio.application")          ' find Visio
    If (visio Is Nothing) then
        Msgbox "Couldn't find Visio!"
        Exit Sub
    End If
'Get # of open Visio files
    doccount = visio.documents.count        'OLE Automation call to Visio
    If doccount=0 then
        msgtext="No open Visio documents."
    Else
        msgtext="The open files are: " & Chr$(13)
        For i = 1 to doccount
            Set doc = visio.documents(i)        ' access Visio's document method
            msgtext=msgtext & Chr$(13)& doc.name
        Next i
    End If
    MsgBox msgtext
End Sub
```

See Also

- CreateObject
- Is
- Me
- New
- Nothing
- Object Class
- Typeof

Close | Copy All | Copy | Print

' Global Statement Example

```
'This example contains two subroutines that share the variables TOTAL and ACCTNO, and the
record GRECORD.

Type acctrecord
    acctno As Integer
End Type

Global acctno as Integer
Global total as Integer
Global grecord as acctrecord
Declare Sub createfile

Sub main
    Dim msgtext
    Dim newline as String
    newline=Chr$(10)
    Call createfile
    Open "C:\TEMP001" For Input as #1
    msgtext="The new account numbers are: " & newline
    For x=1 to total
        Input #1, grecord.acctno
        msgtext=msgtext   & newline & grecord.acctno
    Next x
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub

Sub createfile
    Dim x
    x=1
    grecord.acctno=1
    Open "C:\TEMP001" For Output as #1
    Do While grecord.acctno<>0
        grecord.acctno=InputBox("Enter 0 or new account #" & x & ":")
        If grecord.acctno<>0 then
            Print #1, grecord.acctno
            x=x+1
        End If
    Loop
    total=x-1
    Close #1
End Sub
```

See Also

- Const
- Dim
- Option Base
- ReDim
- Static
- Type

Close | Copy All | Copy | Print

' GoTo Statement Example

'This example displays the date for one week from the date entered by the user. If the date is invalid, the Goto statement sends program execution back to the beginning.

```
Sub main
    Dim str1 as String
    Dim nextweek
    Dim msgtext
i: str1=InputBox$("Enter a date:")
    answer=IsDate(str1)
    If answer=-1 then
        str1=CVDate(str1)
        nextweek=DateValue(str1)+7
        msgtext="One week from the date entered is:"
        msgtext=msgtext & Format(nextweek,"dddddd")
        MsgBox msgtext
    Else
        MsgBox "Invalid date or format. Try again."
        Goto i
    End If
End Sub
```

See Also

- Do...Loop
- For...Next
- If...Then...Else
- Select Case
- While...Wend

Close | Copy All | Copy | Print

' GroupBox Statement Example

'This example creates a dialog box with two group boxes.

```
Sub main
    Begin Dialog UserDialog 242, 146, "Print Dialog Box"
        '$CStrings Save
        GroupBox   115, 14, 85, 57, "Page Range"
        OptionGroup .OptionGroup2
            OptionButton   123, 30, 46, 12, "All Pages", .OptionButton1
            OptionButton   123, 50, 67, 8, "Current Page", .OptionButton2
        GroupBox   14, 12, 85, 76, "Include"
        CheckBox   26, 17, 54, 25, "Pictures", .CheckBox1
        CheckBox   26, 36, 54, 25, "Links", .CheckBox2
        CheckBox   26, 58, 63, 25, "Header/Footer", .CheckBox3
        PushButton   34, 115, 54, 14, "Print"
        PushButton   136, 115, 54, 14, "Cancel"
        '$CStrings Restore
    End Dialog
    Dim mydialog as UserDialog
    Dialog mydialog
End Sub
```

See Also

- Begin Dialog...End Dialog
- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- Dialog
- DropComboBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

Close | Copy All | Copy | Print

' Hex Function Example

```
'This example returns the hex value for a number entered by the user.

Sub main
    Dim usernum as Integer
    Dim hexvalue
    usernum=InputBox("Enter a number to convert to hexidecimal:")
    hexvalue=Hex(usernum)
    Msgbox "The HEX value is: " & hexvalue
End Sub
```

See Also

Oct
Format

Close | Copy All | Copy | Print

' Hour Function Example

'This example extracts just the time (hour, minute, and second) from a file's last modification date and time.

```
Sub main
    Dim filename as String
    Dim ftime
    Dim hr, min
    Dim sec
    Dim msgtext as String
i: msgtext="Enter a filename:"
    filename=InputBox(msgtext)
    If filename="" then
        Exit Sub
    End If
    On Error Resume Next
    ftime=FileDateTime(filename)
    If Err<>0 then
        MsgBox "Error in filename. Try again."
        Goto i:
    End If
    hr=Hour(ftime)
    min=Minute(ftime)
    sec=Second(ftime)
    Msgbox "The file's time is: " & hr &":" &min &":" &sec
End Sub
```

See Also

- DateSerial
- DateValue
- Day
- Minute
- Month
- Now
- Second
- Time Function
- Time Statement
- TimeSerial
- TimeValue
- Weekday
- Year

' If...Then...Else Function Example

Close | Copy All | Copy | Print

'This example checks the time and the day of the week, and returns an appropriate message.

```
Sub main
    Dim h, m, m2, w
    h = hour(now)
    If h > 18 then
        m= "Good evening, "
    Elseif h >12 then
        m= "Good afternoon, "
    Else
        m= "Good morning, "
    End If
        w = weekday(now)
    If w = 1 or w = 7 then m2 = "the office is closed." else m2 = "please hold for company operator."
    Msgbox m & m2
End Sub
```

See Also

- Do...Loop
- For...Next
- Goto
- On...Goto
- Select Case
- While...Wend

' $Include Metacommand Example

Close | Copy All | Copy | Print

'This example includes a file containing the list of global variables, called GLOBALS.OSS. For this example to work correctly, you must create the GLOBALS.OSS file with at least the following statement: Dim gtext as String. The Option Explicit statement is included in this example to prevent OPEN Script from automatically dimensioning the variable as a Variant.

```
Option Explicit
Sub main
    Dim msgtext as String
    '$Include: "c:\globals.oss"
    gtext=InputBox("Enter a string for the global variable:")
    msgtext="The variable for the string '"
    msgtext=msgtext & gtext & "' was DIM'ed in GLOBALS.OSS."
    MsgBox msgtext
End Sub
```

See Also

- $CStrings
- Global
- $NoCStrings

Close | Copy All | Copy | Print

' Input Function Example

'This example opens a file and prints its contents to the screen.

```
Sub main
    Dim fname
    Dim fchar()
    Dim x as Integer
    Dim msgtext
    Dim newline
    newline=Chr(10)
    On Error Resume Next
    fname=InputBox("Enter a filename to print:")
    If fname="" then
        Exit Sub
    End If
    Open fname for Input as #1
    If Err<>0 then
        MsgBox "Error loading file.   Re-run program."
        Exit Sub
    End If
    msgtext="The contents of " & fname & " is: " & newline &newline
    Redim fchar(Lof(1))
     For x=1 to Lof(1)
          fchar(x)=Input(1,#1)
          msgtext=msgtext & fchar(x)
    Next x
    MsgBox msgtext
    Close #1
End Sub
```

See Also

- Get
- Input Statement
- Line Input
- Open
- Write

' Input Statement Example

Close | Copy All | Copy | Print

'This example prompts a user for an account number, opens a file, searches for the account number and displays the matching letter for that number. It uses the Input statement to increase the value of x and at the same time get the letter associated with each value. The second subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram.

```
Declare Sub createfile()
Global x as Integer
Global y(100) as String

Sub main
    Dim acctno as Integer
    Dim msgtext
    Call createfile
i: acctno=InputBox("Enter an account number from 1-10:")
    If acctno<1 Or acctno>10 then
        MsgBox "Invalid account number. Try again."
        Goto i:
    End if
    x=1
    Open "C:\TEMP001" for Input as #1
    Do Until x=acctno
        Input #1, x,y(x)
    Loop
        msgtext="The letter for account number " & x & " is: " & y(x)
    Close #1
    MsgBox msgtext
    Kill "C:\TEMP001"
End Sub

Sub createfile()
' Put the numbers 1-10 and letters A-J into a file
    Dim startletter
    Open "C:\TEMP001" for Output as #1
    startletter=65
    For x=1 to 10
        y(x)=Chr(startletter)
        startletter=startletter+1
    Next x
    For x=1 to 10
        Write #1, x,y(x)
    Next x
    Close #1
End Sub
```

See Also

- Get
- Input Function
- Line Input
- Open
- Write

' InputBox Function Example

Close | Copy All | Copy | Print

'This example uses InputBox to prompt for a filename and then prints the filename using MsgBox.

```
Sub main
    Dim filename
    Dim msgtext
    msgtext="Enter a filename:"
    filename=InputBox$(msgtext)
    MsgBox "The filename you entered is: " & filename
End Sub
```

See Also

- Dialog Boxes
- Input Function
- Input Statement
- MsgBox Function
- MsgBox Statement
- PasswordBox

' InStr Function Example

Close | Copy All | Copy | Print

'This example generates a random string of characters then uses InStr to find the position of a single character within that string.

```
Sub main
    Dim x as Integer
    Dim y
    Dim str1 as String
    Dim str2 as String
    Dim letter as String
    Dim randomvalue
    Dim upper, lower
    Dim position as Integer
    Dim msgtext, newline
    upper=Asc("z")
    lower=Asc("a")
    newline=Chr(10)
    For x=1 to 26
        Randomize
        randomvalue=Int(((upper - (lower+1)) * Rnd) +lower)
        letter=Chr(randomvalue)
        str1=str1 & letter
'Need to waste time here for fast processors
        For y=1 to 1000
        Next y
    Next x
    str2=InputBox("Enter a letter to find")
    position=InStr(str1,str2)
    If position then
        msgtext="The position of " & str2 & " is: " & position & newline
        msgtext=msgtext & "in string: " & str1
    Else
        msgtext="The letter: " & str2 & " was not found in: " & newline
        msgtext=msgtext & str1
    End If
    MsgBox msgtext
End Sub
```

See Also

- GetField
- Left
- Mid Function
- Mid Statement
- Option Compare
- Right
- Str
- StrComp

' Int Function Example

Close | Copy All | Copy | Print

'This example uses Int to generate random numbers in the range between the ASCII values for lowercase a and z (97 and 122). The values are converted to letters and displayed as a string.

```
Sub main
    Dim x as Integer
    Dim y
    Dim str1 as String
    Dim letter as String
    Dim randomvalue
    Dim upper, lower
    Dim msgtext, newline
    upper=Asc("z")
    lower=Asc("a")
    newline=Chr(10)
    For x=1 to 26
        Randomize
        randomvalue=Int(((upper - (lower+1)) * Rnd) +lower)
        letter=Chr(randomvalue)
        str1=str1 & letter
'Need to waste time here for fast processors
        For y=1 to 1500
        Next y
    Next x
    msgtext="The string is:" & newline
    msgtext=msgtext & str1
    MsgBox msgtext
End Sub
```

See Also

- Exp
- Fix
- Log
- Rnd
- Sgn
- Sqr

' IPmt Function Example

Close | Copy All | Copy | Print

'This example finds the interest portion of a loan payment amount for payments made in last month of the first year. The loan is for $25,000 to be paid back over 5 years at 9.5% interest.

```
Sub main
    Dim aprate, periods
    Dim payperiod
    Dim loanpv, due
    Dim loanfv, intpaid
    Dim msgtext
    aprate=.095
    payperiod=12
    periods=120
    loanpv=25000
    loanfv=0
Rem Assume payments are made at end of month
    due=0
    intpaid=IPmt(aprate/12,payperiod,periods,-loanpv,loanfv,due)
    msgtext="For a loan of $25,000 @ 9.5% for 10 years," & Chr(10)
    msgtext=msgtext+ "the interest paid in month 12 is: "
    msgtext=msgtext + Format(intpaid, "Currency")
    MsgBox msgtext
End Sub
```

See Also

- FV
- IRR
- NPV
- Pmt
- PPmt
- PV
- Rate

Close | Copy All | Copy | Print

' IRR Function Example

'This example calculates an internal rate of return (expressed as an interest rate percentage) for a series of business transactions (income and costs). The first value entered must be a negative amount, or IRR generates an "Illegal Function Call" error.

```
Sub main
    Dim cashflows() as Double
    Dim guess, count as Integer
    Dim i as Integer
    Dim intnl as Single
    Dim msgtext as String
    guess=.15
    count=InputBox("How many cash flow amounts do you have?")
    ReDim cashflows(count+1)
    For i=0 to count-1
        cashflows(i)=InputBox("Enter income value for month " & i+1 & ":")
    Next i
    intnl=IRR(cashflows(),guess)
    msgtext="The IRR for your cash flow amounts is: "
    msgtext=msgtext & Format(intnl, "Percent")
    MsgBox msgtext
End Sub
```

See Also

- FV
- IPmt
- NPV
- Pmt
- PPmt
- PV
- Rate

' Is Operator Example

Close | Copy All | Copy | Print

'This example displays a list of open files in the software application, VISIO. It uses the Is operator to determine whether VISIO is available. To see how this example works, you need to start VISIO and open one or more documents.

```
Sub main
    Dim visio as Object
    Dim doc as Object
    Dim msgtext as String
    Dim i as Integer, doccount as Integer

'Initialize Visio
    Set visio = GetObject(,"visio.application")          ' find Visio
    If (visio Is Nothing) then
        Msgbox "Couldn't find Visio!"
        Exit Sub
    End If
'Get # of open Visio files
    doccount = visio.documents.count       'OLE Automation call to Visio
    If doccount=0 then
        msgtext="No open Visio documents."
    Else
        msgtext="The open files are: " & Chr$(13)
        For i = 1 to doccount
            Set doc = visio.documents(i)       ' access Visio's document method
            msgtext=msgtext & Chr$(13)& doc.name
        Next i
    End If
    MsgBox msgtext
End Sub
```

See Also

- Create Object
- Get Object
- Me
- Nothing
- Object
- Typeof

' IsDate Function Example

Close | Copy All | Copy | Print

'This example adds a number to today's date value and checks to see if it is still a valid date
(within the range January 1, 100AD through December 31, 9999AD).

```
Sub main
    Dim curdatevalue
    Dim yrs
    Dim msgtext
    curdatevalue=DateValue(Date$)
    yrs=InputBox("Enter a number of years to add to today's date")
    yrs=yrs*365
    curdatevalue=curdatevalue+yrs
    If IsDate(curdatevalue)=-1 then
        MsgBox "The new date is: " & Format(CVDate(curdatevalue), "dddddd")
    Else
        MsgBox "The date is not valid."
    End If
End Sub
```

See Also

- CVDate
- IsEmpty
- IsNull
- IsNumeric
- VarType

' IsEmpty Function Example

Close | Copy All | Copy | Print

'This example prompts for a series of test scores and uses IsEmpty to determine whether the maximum allowable limit has been hit. (IsEmpty determines when to exit the Do...Loop.)

```
Sub main
    Dim arrayvar(10)
    Dim x as Integer
    Dim tscore as Single
    Dim total as Integer
    x=1
    Do
        tscore=InputBox("Enter test score #" & x & ":")
        arrayvar(x)=tscore
        x=x+1
    Loop Until IsEmpty(arrayvar(10))<>-1
    total=x-1
    msgtext="You entered: " & Chr(10)
    For x=1 to total
         msgtext=msgtext & Chr(10) & arrayvar(x)
    Next x
    MsgBox msgtext
End Sub
```

See Also

- IsDate
- IsNull
- IsNumeric
- VarType

' IsMissing Function Example

Close | Copy All | Copy | Print

'This example prints a list of letters. The number printed is determined by the user. If the user wants to print all letters, the Function myfunc is called without any argument. The function uses IsMissing to determine whether to print all the letters or just the number specified by the user.

```
Function myfunc(Optional arg1)
    If IsMissing(arg1)=-1 then
        arg1=26
    End If
    msgtext="The letters are: " & Chr$(10)
    For x= 1 to arg1
        msgtext=msgtext & Chr$(x+64) & Chr$(10)
    Next x
    MsgBox msgtext
End Function

Sub main
    Dim arg1
    arg1=InputBox("How many letters do you want to print? (0 for all)")
    If arg1=0 then
        myfunc()
    Else
        myfunc(arg1)
    End If
End Sub
```

See Also

Function...End Function

' IsNull Function Example

Close | Copy All | Copy | Print

'This example asks for ten test score values and calculates the average. If any score is negative, the value is set to Null. Then IsNull is used to reduce the total count of scores (originally 10) to just those with positive values before calculating the average.

```
Sub main
    Dim arrayvar(10)
    Dim count as Integer
    Dim total as Integer
    Dim x as Integer
    Dim tscore as Single
    count=10
    total=0
    For x=1 to count
        tscore=InputBox("Enter test score #" & x & ":")
        If tscore<0 then
            arrayvar(x)=Null
        Else
            arrayvar(x)=tscore
            total=total+arrayvar(x)
        End If
    Next x
    Do While x<>0
        x=x-1
        If IsNull(arrayvar(x))=-1 then
            count=count-1
        End If
    Loop
    msgtext="The average (excluding negative values) is: " & Chr(10)
    msgtext=msgtext & Format (total/count, "##.##")
    MsgBox msgtext
End Sub
```

See Also

- IsDate
- IsEmpty
- IsNumeric
- VarType

' IsNumeric Function Example

Close | Copy All | Copy | Print

'This example uses IsNumeric to determine whether a user selected an option (1-3) or typed "Q"
to quit.

```
Sub main
    Dim answer
    answer=InputBox("Enter a choice (1-3) or type Q to quit")
    If IsNumeric(answer)=-1 then
        Select Case answer
            Case 1
                MsgBox "You chose #1."
            Case 2
                MsgBox "You chose #2."
            Case 3
                MsgBox "You chose #3."
        End Select
    Else
        MsgBox "You typed Q."
    End If
End Sub
```

See Also

- IsDate
- IsEmpty
- IsNull
- VarType

' Kill Function Example

Close | Copy All | Copy | Print

'This example prompts a user for an account number, opens a file, searches for the account number and displays the matching letter for that number. The second subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram. After processing is complete, the first subroutine uses Kill to delete the file.

```
Declare Sub createfile()
Global x as Integer
Global y(100) as String

Sub main
    Dim acctno as Integer
    Dim msgtext
    Call createfile
i: acctno=InputBox("Enter an account number from 1-10:")
    If acctno<1 Or acctno>10 then
        MsgBox "Invalid account number. Try again."
        Goto i:
    End if
    x=1
    Open "C:\TEMP001" for Input as #1
    Do Until x=acctno
        Input #1, x,y(x)
    Loop
        msgtext="The letter for account number " & x & " is: " & y(x)
    Close #1
    MsgBox msgtext
    Kill "C:\TEMP001"
End Sub

Sub createfile()
' Put the numbers 1-10 and letters A-J into a file
    Dim startletter
    Open "C:\TEMP001" for Output as #1
    startletter=65
    For x=1 to 10
        y(x)=Chr(startletter)
        startletter=startletter+1
    Next x
    For x=1 to 10
        Write #1, x,y(x)
    Next x
    Close #1
End Sub
```

See Also

- FileAttr
- FileDateTime
- GetAttr
- RmDir

' LBound Function Example

Close | Copy All | Copy | Print

'This example resizes an array if the user enters more data than can fit in the array. It uses LBound and UBound to determine the existing size of the array and ReDim to resize it. Option Base sets the default lower bound of the array to 1.

```
Option Base 1
Sub main
    Dim arrayvar() as Integer
    Dim count as Integer
    Dim answer as String
    Dim x, y as Integer
    Dim total
    total=0
    x=1
    count=InputBox("How many test scores do you have?")
    ReDim arrayvar(count)
start:
    Do until x=count+1
      arrayvar(x)=InputBox("Enter test score #" &x & ":")
      x=x+1
    Loop
    answer=InputBox$("Do you have more scores? (Y/N)")
    If answer="Y" or answer="y" then
      count=InputBox("How many more do you have?")
      If count<>0 then
        count=count+(x-1)
        ReDim Preserve arrayvar(count)
        Goto start
      End If
    End If
    x=LBound(arrayvar,1)
    count=UBound(arrayvar,1)
    For y=x to count
          total=total+arrayvar(y)
    Next y
    MsgBox "The average of " & count & " scores is: " & Int(total/count)
End Sub
```

See Also

- Dim
- Global
- Option Base
- ReDim
- Static
- UBound

Close | Copy All | Copy | Print

' LCase Function Example

```
'This example converts a string entered by the user to lowercase.

Sub main
Dim userstr as String
    userstr=InputBox$("Enter a string in upper and lowercase letters")
    userstr=LCase$(userstr)
    Msgbox "The string now is: " & userstr
End Sub
```

See Also

UCase

' Left Function Example

Close | Copy All | Copy | Print

'This example extracts a user's first name from the entire name entered.

```
Sub main
    Dim username as String
    Dim count as Integer
    Dim firstname as String
    Dim charspace
    charspace=Chr(32)
    username=InputBox("Enter your first and last name")
    count=InStr(username,charspace)
    firstname=Left(username,count)
    Msgbox "Your first name is: " &firstname
End Sub
```

See Also

- GetField
- Len
- LTrim
- Mid Function
- Mid Statement
- Right
- RTrim
- Str
- StrComp
- Trim

Close | Copy All | Copy | Print

' Len Function Example

```
'This example returns the length of a name entered by the user (including spaces).

Sub main
    Dim username as String
    username=InputBox("Enter your name")
    count=Len(username)
    Msgbox "The length of your name is: " &count
End Sub
```

See Also

Instr

' Let (Assignment Statement) Example

Close | Copy All | Copy | Print

'This example uses the Let statement for the variable sum. The subroutine finds an average of 10 golf scores.

```
Sub main
    Dim score As Integer
    Dim x, sum
    Dim msgtext
    Let sum=0
    For x=1 to 10
        score=InputBox("Enter your last ten golf scores #" & x & ":")
        sum=sum+score
    Next x
    msgtext="Your average is: " & CInt(sum/(x-1))
    MsgBox msgtext
End Sub
```

See Also

- Const
- Lset
- Set

Close | Copy All | Copy | Print

' Like Operator Example

```
'This example tests whether a letter is lowercase.

Sub main
    Dim userstr as String
    Dim revalue as Integer
    Dim msgtext as String
    Dim pattern
    pattern="[a-z]"
    userstr=InputBox$("Enter a letter:")
    retvalue=userstr LIKE pattern
    If retvalue=-1 then
        msgtext="The letter " & userstr & " is lowercase."
    Else
        msgtext="Not a lowercase letter."
    End If
    Msgbox msgtext
End Sub
```

See Also

- Expressions
- Instr
- Option Compare
- StrComp

' Line Input Statement Example

Close | Copy All | Copy | Print

'This example reads the contents of a sequential file line by line (to a carriage return) and displays the results. The second subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram.

```
Declare Sub createfile()
Sub main
    Dim testscore as String
    Dim x
    Dim y
    Dim newline
    Call createfile
    Open "c:\temp001" for Input as #1
    x=1
    newline=Chr(10)
    msgtext= "The contents of c:\temp001 is: " & newline
    Do Until x=Lof(1)
        Line Input #1, testscore
        x=x+1
        y=Seek(1)
        If y>Lof(1) then
            x=Lof(1)
        Else
            Seek 1,y
        End If
        msgtext=msgtext & testscore & newline
    Loop
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the numbers 1-10 into a file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=1 to 10
        Write #1, x
    Next x
    Close #1
End Sub
```

See Also

- DialogBoxes
- Get
- Input Function
- Input Statement
- InputBox
- Open

Close | Copy All | Copy | Print

' ListBox Statement Example

```
'This example defines a dialog box with list box and two buttons.

Sub main
    Dim ListBox1() as String
    ReDim ListBox1(0)
    ListBox1(0)="C:\"
    Begin Dialog UserDialog 133, 66, 171, 65, "OPEN Script Dialog Box"
        Text   3, 3, 34, 9, "Directory:", .Text2
        ListBox   3, 14, 83, 39, ListBox1(), .ListBox2
        OKButton   105, 6, 54, 14
        CancelButton   105, 26, 54, 14
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin...End Dialog
- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- Dialog
- DialogBoxes
- DropComboBox
- GroupBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

Close | Copy All | Copy | Print

' Loc Function Example

'This example creates a file of account numbers as entered by the user. When the user finishes,
the example displays the offset in the file of the last entry made.

```
Sub main
    Dim filepos as Integer
    Dim acctno() as Integer
    Dim x as Integer
    x=0
    Open "c:\TEMP001" for Random as #1
    Do
      x=x+1
      Redim Preserve acctno(x)
      acctno(x)=InputBox("Enter account #" & x & " or 0 to end:")
      If acctno(x)=0 then
          Exit Do
      End If
      Put #1,, acctno(x)
    Loop
    filepos=Loc(1)
    Close #1
    MsgBox "The offset is: " & filepos
    Kill "C:\TEMP001"
End Sub
```

See Also

- Eof
- Lof
- Open

' Lock Function Example

Close | Copy All | Copy | Print

'This example locks a file that is shared by others on a network, if the file is already in use. The second subprogram, CREATEFILE, creates the file used by the main subprogram.

```
Declare Sub createfile
Sub main
    Dim btngrp, icongrp
    Dim defgrp
    Dim answer
    Dim noaccess as Integer
    Dim msgabort
    Dim msgstop as Integer
    Dim acctname as String
    noaccess=70
    msgstop=16
    Call createfile
    On Error Resume Next
    btngrp=1
    icongrp=64
    defgrp=0
    answer=MsgBox("Open the account file?" & Chr(10), btngrp+icongrp+defgrp)
    If answer=1 then
        Open "C:\TEMP001" for Input as #1
        If Err=noaccess then
            msgabort=MsgBox("File Locked",msgstop,"Aborted")
        Else
            Lock #1
            Line Input #1, acctname
            MsgBox "The first account name is: " & acctname
            Unlock #1
        End If
        Close #1
    End If
    Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the letters A-J into the file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=1 to 10
        Write #1, Chr(x+64)
    Next x
    Close #1
End Sub
```

See Also

Open
Unlock

Close | Copy All | Copy | Print

' Lof Function Example

'This example opens a file and prints its contents to the screen.

```
Sub main
    Dim fname
    Dim fchar()
    Dim x as Integer
    Dim msgtext
    Dim newline
    newline=Chr(10)
    fname=InputBox("Enter a filename to print:")
    On Error Resume Next
    Open fname for Input as #1
    If Err<>0 then
        MsgBox "Error loading file. Re-run program."
        Exit Sub
    End If
    msgtext="The contents of " & fname & " is: " & newline &newline
    Redim fchar(Lof(1))
     For x=1 to Lof(1)
          fchar(x)=Input(1,#1)
          msgtext=msgtext & fchar(x)
    Next x
    MsgBox msgtext
    Close #1
End Sub
```

See Also

- Eof
- FileLen
- Loc
- Open

Close | Copy All | Copy | Print

' Log Function Example

'This example uses the Log function to determine which number is larger: 999^1000 (999 to the 1000 power) or 1000^999 (1000 to the 999 power). Note that you cannot use the exponent (^) operator for numbers this large.

```
Sub main
    Dim x
    Dim y
    x=999
    y=1000
    a=y*(Log(x))
    b=x*(Log(y))
    If a>b then
        MsgBox "999^1000 is greater than 1000^999"
    Else
        MsgBox "1000^999 is greater than 999^1000"
    End If
End Sub
```

See Also

- Exp
- Fix
- Int
- Rnd
- Sgn
- Sqr

' Lset Statement Example

Close | Copy All | Copy | Print

'This example puts a user's last name into the variable LASTNAME. If the name is longer than the size of LASTNAME, then the user's name is truncated. If you have a long last name and you get lots of junk mail, you've probably seen how this works already.

```
Sub main
    Dim lastname as String
    Dim strlast as String*8
    lastname=InputBox("Enter your last name")
    Lset strlast=lastname
    msgtext="Your last name is: " &strlast
    MsgBox msgtext
End Sub
```

See Also

Rset

' LTrim Function Example

Close | Copy All | Copy | Print

'This example trims the leading spaces from a string padded with spaces on the left.

```
Sub main
    Dim userinput as String
    Dim numsize
    Dim str1 as String*50
    Dim strsize
    strsize=50
    userinput=InputBox("Enter a string of characters:")
    numsize=Len(userinput)
    str1=Space(strsize-numsize) & userinput
' Str1 has a variable number of leading spaces.
    MsgBox "The string is: " &str1
    str1=LTrim$(str1)
' Str1 now has no leading spaces.
    MsgBox "The string now has no leading spaces: " & str1
End Sub
```

See Also

- GetField
- Left
- Mid Function
- Mid Statement
- Right
- RTrim
- Trim

' Me Example

Close | Copy All | Copy | Print

(None)

See Also

- Create Object
- Get Object
- New
- Nothing
- Object
- Typeof

Close | Copy All | Copy | Print

' Mid Statement Example

'This example uses the Mid statement to replace the last name in a user-entered string to asterisks(*).

```
Sub main
    Dim username as String
    Dim position as Integer
    Dim count as Integer
    Dim uname as String
    Dim replacement as String
    username=InputBox("Enter your full name:")
    uname=username
    replacement="*"
    Do
        position=InStr(username," ")
        If position=0 then
            Exit Do
        End If
        username=Mid(username,position+1)
        count=count+position
    Loop
    For x=1 to Len(username)
        count=count+1
        Mid(uname,count)=replacement
    Next x
    MsgBox "Your name now is: " & uname
End Sub
```

See Also

- GetField
- Left
- Len
- LTrim
- Mid Function
- Right
- RTrim
- Trim

Close | Copy All | Copy | Print

' Mid Function Example

'This example uses the Mid function to find the last name in a string. entered by the user.

```
Sub main
    Dim username as String
    Dim position as Integer
    username=InputBox("Enter your full name:")
    Do
      position=InStr(username," ")
      If position=0 then
        Exit Do
      End If
      position=position+1
      username=Mid(username,position)
    Loop
    MsgBox "Your last name is: " & username
End Sub
```

See Also

- GetField
- LCase
- Left
- Len
- LTrim
- Mid Statement
- Right
- RTrim
- Trim

' Minute Function Example

Close | Copy All | Copy | Print

```
'This example extracts just the time (hour, minute, and second) from a file's last modification date
and time.

Sub main
    Dim filename as String
    Dim ftime
    Dim hr, min
    Dim sec
    Dim msgtext as String
i: msgtext="Enter a filename:"
    filename=InputBox(msgtext)
    If filename="" then
        Exit Sub
    End If
    On Error Resume Next
    ftime=FileDateTime(filename)
    If Err<>0 then
        MsgBox "Error in filename. Try again."
        Goto i:
    End If
    hr=Hour(ftime)
    min=Minute(ftime)
    sec=Second(ftime)
    Msgbox "The file's time is: " & hr &":" &min &":" &sec
End Sub
```

See Also

- DateSerial
- DateValue
- Day
- Hour
- Month
- Now
- Second
- Time Function
- Time Statement
- TimeSerial
- TimeValue
- Weekday
- Year

' MkDir Statement Example

Close | Copy All | Copy | Print

```
'This example makes a new temporary directory in C:\ and then deletes it.

Sub main
    Dim path as String
    On Error Resume Next
    path=CurDir(C)
    If path<>"C:\" then
        ChDir "C:\"
    End If
    MkDir "C:\TEMP01"
    If Err=75 then
        MsgBox "Directory already exists"
    Else
        MsgBox "Directory C:\TEMP01 created"
        MsgBox "Now removing directory"
        RmDir "C:\TEMP01"
    End If
End Sub
```

See Also

- ChDir
- ChDrive
- CurDir
- Dir
- RmDir

Close | Copy All | Copy | Print

' Month Function Example

'This example finds the month (1-12) and day (1-31) values for this Thursday.

```
Sub main
    Dim x, today
    Dim msgtext
    Today=DateValue(Now)
    Let x=0
    Do While Weekday(Today+x)<> 5
        x=x+1
    Loop
    msgtext="This Thursday is: " & Month(Today+x)&"/"&Day(Today+x)
    MsgBox msgtext
End Sub
```

See Also

- Date Function
- Date Statement
- DateSerial
- DateValue
- Day
- Hour
- Minute
- Now
- Second
- TimeSerial
- TimeValue
- Weekday
- Year

Close | Copy All | Copy | Print

' Msgbox Function Example

'This example displays one of each type of message box.

```
Sub main
    Dim btngrp as Integer
    Dim icongrp as Integer
    Dim defgrp as Integer
    Dim msgtext as String
    icongrp=16
    defgrp=0
    btngrp=0
    Do Until btngrp=6
      Select Case btngrp
          Case 1, 4, 5
              defgrp=0
          Case 2
              defgrp=256
          Case 3
              defgrp=512
        End Select
        msgtext="   Icon group = " & icongrp & Chr(10)
        msgtext=msgtext + "    Button group = " & btngrp & Chr(10)
        msgtext=msgtext + "    Default group = " & defgrp & Chr(10)
        msgtext=msgtext + Chr(10) + "    Continue?"
        answer=MsgBox(msgtext, btngrp+icongrp+defgrp)
        Select Case answer
          Case 2,3,7
             Exit Do
        End Select
        If icongrp<>64 then
           icongrp=icongrp+16
        End If
        btngrp=btngrp+1
    Loop
End Sub
```

See Also

- Dialog Boxes
- InputBox
- MsgBox Statement
- PasswordBox

Close | Copy All | Copy | Print

' Msgbox Statement Example

'This example finds the future value of an annuity, whose terms are defined by the user. It uses the MsgBox statement to display the result.

```
Sub main
    Dim aprate, periods
    Dim payment, annuitypv
    Dim due, futurevalue
    Dim msgtext
    annuitypv=InputBox("Enter present value of the annuity: ")
    aprate=InputBox("Enter the annual percentage rate: ")
    If aprate >1 then
        aprate=aprate/100
    End If
    periods=InputBox("Enter the total number of pay periods: ")
    payment=InputBox("Enter the initial amount paid to you: ")
Rem Assume payments are made at end of month
    due=0
    futurevalue=FV(aprate/12,periods,-payment,-annuitypv,due)
    msgtext="The future value is: " & Format(futurevalue, "Currency")
    MsgBox msgtext
End Sub
```

See Also

- InputBox
- MsgBox Function
- PasswordBox

' Name Statement Example

Close | Copy All | Copy | Print

'This example creates a temporary file, C:\TEMP001, renames the file to C:\TEMP002, then deletes them both. It calls the subprogram, CREATEFILE, to create the C:\TEMP001 file.

```
Declare Sub createfile()
Sub main
    Call createfile
    On Error Resume Next
    Name "C:\TEMP001" As "C:\TEMP002"
    MsgBox "The file has been renamed"
    MsgBox "Now deleting both files"
    Kill "TEMP001"
    Kill "TEMP002"
End Sub

Sub createfile()
    Rem Put the numbers 1-10 into a file
    Dim x as Integer
    Dim y()
    Dim startletter
    Open "C:\TEMP001" for Output as #1
    For x=1 to 10
        Write #1, x
    Next x
    Close #1
End Sub
```

See Also

- FileAttr
- FileCopy
- GetAttr
- Kill

' New Operator Example

Close | Copy All | Copy | Print

(None)

See Also

- Dim
- Global
- Set
- Static

' $NoCStrings Metacommand Example

Close | Copy All | Copy | Print

'This example displays two lines, the first time using the C-language characters "\n" for a carriage return and line feed.

```
Sub main
    '$CStrings
    MsgBox "This is line 1\n This is line 2 (using C Strings)"
    '$NoCStrings
    MsgBox "This is line 1" +Chr$(13)+Chr$(10)+"This is line 2 (using Chr)"
End Sub
```

See Also

- $CStrings
- $Include
- Rem

' Nothing Function Example

Close | Copy All | Copy | Print

'This example displays a list of open files in the software application VISIO. It uses the Nothing function to determine whether VISIO is available. To see how this example works, you need to start VISIO and open one or more documents.

```
Sub main
    Dim visio as Object
    Dim doc as Object
    Dim msgtext as String
    Dim i as Integer, doccount as Integer

'Initialize Visio
    Set visio = GetObject(,"visio.application")          ' find Visio
    If (visio Is Nothing) then
        Msgbox "Couldn't find Visio!"
        Exit Sub
    End If
'Get # of open Visio files
    doccount = visio.documents.count        'OLE Automation call to Visio
    If doccount=0 then
        msgtext="No open Visio documents."
    Else
        msgtext="The open files are: " & Chr$(13)
        For i = 1 to doccount
            Set doc = visio.documents(i)        ' access Visio's document method
            msgtext=msgtext & Chr$(13)& doc.name
        Next i
    End If
    MsgBox msgtext
End Sub
```

See Also

- ls
- New

' Now Function Example

Close | Copy All | Copy | Print

'This example finds the month (1-12) and day (1-31) values for this Thursday.

```
Sub main
    Dim x, today
    Dim msgtext
    Today=DateValue(Now)
    Let x=0
    Do While Weekday(Today+x)<> 5
        x=x+1
    Loop
    msgtext="This Thursday is: " &Month(Today+x)&"/"&Day(Today+x)
    MsgBox msgtext
End Sub
```

See Also

- Date Function
- Date Statement
- Day
- Hour
- Minute
- Month
- Second
- Time Function
- Time Statement
- Weekday
- Year

Close | Copy All | Copy | Print

' NPV Function Example

'This example finds the net present value of an investment, given a range of cash flows by the user.

```
Sub main
    Dim aprate as Single
    Dim varray() as Double
    Dim cflowper as Integer
    Dim x as Integer
    Dim netpv as Double
    cflowper=InputBox("Enter number of cash flow periods")
    ReDim varray(cflowper)
    For x= 1 to cflowper
        varray(x)=InputBox("Enter cash flow amount for period #" & x & ":")
    Next x
    aprate=InputBox("Enter discount rate: ")
    If aprate>1 then
        aprate=aprate/100
    End If
    netpv=NPV(aprate,varray())
    MsgBox "The net present value is: " & Format(netpv, "Currency")
End Sub
```

See Also

- FV
- IPmt
- IRR
- Pmt
- PPmt
- PV
- Rate

' Null Function Example

Close | Copy All | Copy | Print

'This example asks for ten test score values and calculates the average. If any score is negative, the value is set to Null. Then IsNull is used to reduce the total count of scores (originally 10) to just those with positive values before calculating the average.

```
Sub main
    Dim arrayvar(10)
    Dim count as Integer
    Dim total as Integer
    Dim x as Integer
    Dim tscore as Single
    count=10
    total=0
    For x=1 to count
        tscore=InputBox("Enter test score #" & x & ":")
        If tscore<0 then
            arrayvar(x)=Null
        Else
            arrayvar(x)=tscore
            total=total+arrayvar(x)
        End If
    Next x
    Do While x<>0
        x=x-1
        If IsNull(arrayvar(x))=-1 then
            count=count-1
        End If
    Loop
    msgtext="The average (excluding negative values) is: " & Chr(10)
    msgtext=msgtext & Format (total/count, "##.##")
    MsgBox msgtext
End Sub
```

See Also

- IsEmpty
- IsNull
- VarType

' Object Class Example

Close | Copy All | Copy | Print

'This example displays a list of open files in the software application VISIO. It uses the Object class to declare the variables used for accessing VISIO and its document files and methods.

```
Sub main
    Dim visio as Object
    Dim doc as Object
    Dim msgtext as String
    Dim i as Integer, doccount as Integer

'Initialize Visio
    Set visio = GetObject(,"visio.application")          ' find Visio
    If (visio Is Nothing) then
        Msgbox "Couldn't find Visio!"
        Exit Sub
    End If
'Get # of open Visio files
    doccount = visio.documents.count       'OLE Automation call to Visio
    If doccount=0 then
        msgtext="No open Visio documents."
    Else
        msgtext="The open files are: " & Chr$(13)
        For i = 1 to doccount
            Set doc = visio.documents(i)       ' access Visio's document method
            msgtext=msgtext & Chr$(13)& doc.name
        Next i
    End If
    MsgBox msgtext
End Sub
```

See Also

- Create Object
- Get Object
- New
- Nothing
- Typeof

Close | Copy All | Copy | Print

' Oct Function Example

'This example prints the octal values for the numbers from 1 to 15.

```
Sub main
    Dim x,y
    Dim msgtext
    Dim nofspaces
    msgtext="Octal numbers from 1 to 15:" & Chr(10)
    For x=1 to 15
      nofspaces=10
      y=Oct(x)
      If Len(x)=2 then
          nofspaces=nofspaces-2
      End If
      msgtext=msgtext & Chr(10) & x & Space(nofspaces) & y
    Next x
    MsgBox msgtext
End Sub
```

See Also

Hex

' OKButton Statement Example

Close | Copy All | Copy | Print

'This example defines a dialog box with a dropcombo box and the OK and Cancel buttons.

```
Sub main
    Dim cchoices as String
    On Error Resume Next
    cchoices="All"+Chr$(9)+"Nothing"
     Begin Dialog UserDialog 180, 95, "OPEN Script Dialog Box"
        ButtonGroup .ButtonGroup1
        Text   9, 3, 69, 13, "Filename:", .Text1
        DropComboBox   9, 17, 111, 41, cchoices, .ComboBox1
        OKButton   131, 8, 42, 13
        CancelButton   131, 27, 42, 13
     End Dialog
        Dim mydialogbox As UserDialog
        Dialog mydialogbox
        If Err=102 then
            MsgBox "You pressed Cancel."
        Else
            MsgBox "You pressed OK."
        End If
End Sub
```

See Also

- Begin...End Dialog
- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- Dialog
- DropComboBox
- GroupBox
- ListBox
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

' On ..Goto Statement Example

Close | Copy All | Copy | Print

'This example sets the current system time to the user's entry. If the entry cannot be converted to a valid time value, this subroutine sets the variable to Null. It then checks the variable and if it is Null, uses the On...Goto statement to ask again.

```
Sub main
     Dim answer as Integer
     answer=InputBox("Enter a choice (1-3) or 0 to quit")
     On answer Goto c1, c2, c3
     MsgBox("You typed 0.")
     Exit Sub
c1:  MsgBox("You picked choice 1.")
     Exit Sub
c2:  MsgBox("You picked choice 2.")
     Exit Sub
c3:  MsgBox("You picked choice 3.")
     Exit Sub
End Sub
```

See Also

- Goto
- Select Case

' On Error Statement Example

Close | Copy All | Copy | Print

```
'This example prompts the user for a drive and directory name and uses On Error to trap invalid
entries.

Sub main
        Dim userdrive, userdir, msgtext
in1:    userdrive=InputBox("Enter drive:",,"C:")
        On Error Resume Next
        ChDrive userdrive
        If Err=68 then
            MsgBox "Invalid Drive. Try again."
            Goto in1
        End If
in2:    On Error Goto Errhdlr1
        userdir=InputBox("Enter directory path:")
        ChDir userdrive & userdir
        Msgbox "New default directory is: " & userdrive & userdir
        Exit Sub
Errhdlr1:
        Select Case Err
            Case 75
                msgtext="Path is invalid."
            Case 76
                msgtext="Path not found."
            Case 70
                msgtext="Permission denied."
            Case Else
                msgtext="Error " & Err & ": " & Error$ & "occurred."
        End Select
        MsgBox msgtext & " Try again."
        Resume in2
End Sub
```

See Also

- Erl
- Err Function
- Err Statement
- Error Function
- Error Statement
- Resume

' Open Statement Example

Close | Copy All | Copy | Print

'This example opens a file for Random access, gets the contents of the file, and closes the file again. The second subprogram, CREATEFILE, creates the file C:\TEMP001 used by the main subprogram.

```
Declare Sub createfile()
Sub main
    Dim acctno as String*3
    Dim recno as Long
    Dim msgtext as String
    Call createfile
    recno=1
    newline=Chr(10)
    Open "C:\TEMP001" For Random As #1 Len=3
    msgtext="The account numbers are:" & newline
    Do Until recno=11
            Get #1,recno,acctno
            msgtext=msgtext & acctno
            recno=recno+1
    Loop
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the numbers 1-10 into a file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=1 to 10
        Write #1, x
    Next x
    Close #1
End Sub
```

See Also

Close
FreeFile

' OptionButton Statement Example

Close | Copy All | Copy | Print

'This example creates a dialog box with a group box with two option buttons: "All pages" and "Range of pages".

```
Sub main
    Begin Dialog UserDialog 183, 70, "OPEN Script Dialog Box"
        GroupBox   5, 4, 97, 57, "File Range"
        OptionGroup .OptionGroup2
            OptionButton   16, 12, 46, 12, "All pages", .OptionButton3
            OptionButton   16, 28, 67, 8, "Range of pages", .OptionButton4
        Text   22, 39, 20, 10, "From:", .Text6
        Text   60, 39, 14, 9, "To:", .Text7
        TextBox   76, 39, 13, 12, .TextBox4
        TextBox   44, 39, 12, 11, .TextBox5
        OKButton   125, 6, 54, 14
        CancelButton   125, 26, 54, 14
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin...End Dialog
- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- Dialog
- DropComboBox
- GroupBox
- ListBox
- OKButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

' OptionGroup Statement Example

Close | Copy All | Copy | Print

'This example creates a dialog box with a group box with two option buttons: "All pages" and "Range of Pages".

```
Sub main
    Begin Dialog UserDialog 192, 71, "OPEN Script Dialog Box"
        GroupBox   7, 6, 97, 57, "File Range"
        OptionGroup .OptionGroup2
            OptionButton   18, 14, 46, 12, "All pages", .OptionButton3
            OptionButton   18, 30, 67, 8, "Range of pages", .OptionButton4
        Text   24, 41, 20, 10, "From:", .Text6
        Text   62, 41, 14, 9, "To:", .Text7
        TextBox   78, 41, 13, 12, .TextBox4
        TextBox   46, 41, 12, 11, .TextBox5
        OKButton   126, 6, 54, 14
        CancelButton   126, 26, 54, 14
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin...End Dialog
- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- Dialog
- DropComboBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- Picture
- StaticComboBox
- Text
- TextBox

' Option Base Statement Example

Close | Copy All | Copy | Print

'This example resizes an array if the user enters more data than can fit in the array. It uses LBound and UBound to determine the existing size of the array and ReDim to resize it. Option Base sets the default lower bound of the array to 1.

```
Option Base 1
Sub main
    Dim arrayvar() as Integer
    Dim count as Integer
    Dim answer as String
    Dim x, y as Integer
    Dim total
    total=0
    x=1
    count=InputBox("How many test scores do you have?")
    ReDim arrayvar(count)
start:
    Do until x=count+1
      arrayvar(x)=InputBox("Enter test score #" &x & ":")
      x=x+1
    Loop
    answer=InputBox$("Do you have more scores? (Y/N)")
    If answer="Y" or answer="y" then
      count=InputBox("How many more do you have?")
      If count<>0 then
        count=count+(x-1)
        ReDim Preserve arrayvar(count)
        Goto start
      End If
    End If
    x=LBound(arrayvar,1)
    count=UBound(arrayvar,1)
    For y=x to count
          total=total+arrayvar(y)
    Next y
    MsgBox "The average of " & count & " scores is: " & Int(total/count)
End Sub
```

See Also

- Dim
- Global
- LBound
- ReDim
- Static

' Option Compare Statement Example

Close | Copy All | Copy | Print

'This example compares two strings: "Jane Smith" and "jane smith". When Option Compare is Text, the strings are considered the same. If Option Compare is Binary, they will not be the same. Binary is the default. To see the difference, run the example once, then run it again, commenting out the Option Compare statement.

```
Option Compare Text
Sub main
    Dim strg1 as String
    Dim strg2 as String
    Dim retvalue as Integer
    strg1="JANE SMITH"
    strg2="jane smith"
i:
    retvalue=StrComp(strg1,strg2)
    If retvalue=0 then
        MsgBox "The strings are identical"
    Else
        MsgBox "The strings are not identical"
        Exit Sub
    End If
End Sub
```

See Also

Instr
StrComp

' Option Explicit Statement Example

Close | Copy All | Copy | Print

'This example specifies that all variables must be explicitly declared, thus preventing any mistyped variable names.

```
Option Explicit
Sub main
    Dim counter As Integer
    Dim fixedstring As String*25
    Dim varstring As String
'...(code here)...
End Sub
```

See Also

- Const
- Def*type*
- Dim
- Function...End Function
- Global
- ReDim
- Static
- Sub...End Sub

Close | Copy All | Copy | Print

' PasswordBox Function Example

```
'This example asks the user for a password.

Sub main
    Dim retvalue
    Dim a
    retvalue=PasswordBox("Enter your login password",Password)
    If retvalue<>"" then
        MsgBox "Verifying password"
'       (continue code here)
    Else
        MsgBox "Login cancelled"
    End If
End Sub
```

See Also

- InputBox
- MsgBox Function
- MsgBox Statement

' Picture Statement Example

Close | Copy All | Copy | Print

'This example defines a dialog box with a picture, and the OK and Cancel buttons.

```
Sub main
    Begin Dialog UserDialog 148, 73, "OPEN Script Dialog Box"
        Picture   8, 7, 46, 46, "C:\WINDOWS\ARCADE.BMP", 0
        OKButton   80, 10, 54, 14
        CancelButton   80, 30, 54, 14
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin...End Dialog
- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- Dialog
- DropComboBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- StaticComboBox
- Text
- TextBox

Close | Copy All | Copy | Print

' Pmt Function Example

```
'This example finds the monthly payment on a given loan.

Sub main
    Dim aprate, totalpay
    Dim loanpv, loanfv
    Dim due, monthlypay
    Dim yearlypay, msgtext
    loanpv=InputBox("Enter the loan amount: ")
    aprate=InputBox("Enter the loan rate percent: ")
    If aprate >1 then
        aprate=aprate/100
    End If
    totalpay=InputBox("Enter the total number of monthly payments: ")
    loanfv=0
'Assume payments are made at end of month
    due=0
    monthlypay=Pmt(aprate/12,totalpay,-loanpv,loanfv,due)
    msgtext="The monthly payment is: " & Format(monthlypay, "Currency")
    MsgBox msgtext
End Sub
```

See Also

- FV
- IPmt
- IRR
- NPV
- PV
- PPmt
- Rate

' PPmt Function Example

Close | Copy All | Copy | Print

'This example finds the principal portion of a loan payment amount for payments made in last month of the first year. The loan is for $25,000 to be paid back over 5 years at 9.5% interest.

```
Sub main
    Dim aprate, periods
    Dim payperiod
    Dim loanpv, due
    Dim loanfv, principal
    Dim msgtext
    aprate=9.5/100
    payperiod=12
    periods=120
    loanpv=25000
    loanfv=0
Rem Assume payments are made at end of month
    due=0
    principal=PPmt(aprate/12,payperiod,periods,-loanpv,loanfv,due)
    msgtext="Given a loan of $25,000 @ 9.5% for 10 years," & Chr(10)
    msgtext=msgtext & " the principal paid in month 12 is: "
    MsgBox msgtext & Format(principal, "Currency")
End Sub
```

See Also

- FV
- IPmt
- IRR
- NPV
- Pmt
- PV
- Rate

Close | Copy All | Copy | Print

' Print Statement Example

'This example prints the octal values for the numbers from 1 to 25.

```
Sub main
    Dim x as Integer
    Dim y
    For x=1 to 25
        y=Oct$(x)
        Print x Tab(10) y
    Next x
End Sub
```

See Also

- Open
- Spc
- Tab
- Write

Close | Copy All | Copy | Print

' PushButton Statement Example

'This example defines a dialog box with a combination list box and three buttons.

```
Sub main
    Dim fchoices as String
    fchoices="File1" & Chr(9) & "File2" & Chr(9) & "File3"
    Begin Dialog UserDialog 185, 94, "OPEN Script Dialog Box"
        Text   9, 5, 69, 10, "Filename:", .Text1
        DropComboBox   9, 17, 88, 71, fchoices, .ComboBox1
        ButtonGroup .ButtonGroup1
        OKButton   113, 14, 54, 13
        CancelButton   113, 33, 54, 13
        PushButton 113, 57, 54, 13, "Help", .Push1
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin Dialog...End Dialog Statement
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- DropComboBox
- DropListBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

' Put Statement Example

Close | Copy All | Copy | Print

'This example opens a file for Random access, puts the values 1-10 in it, prints the contents, and closes the file again.

```
Sub main
' Put the numbers 1-10 into a file
    Dim x, y
    Open "C:\TEMP001" as #1
    For x=1 to 10
        Put #1,x, x
    Next x
    msgtext="The contents of the file is:" & Chr(10)
    For x=1 to 10
        Get #1,x, y
        msgtext=msgtext & y & Chr(10)
    Next x
    Close #1
    MsgBox msgtext
    Kill "C:\TEMP001"
End Sub
```

See Also

- Close
- Get
- Open
- Write

Close | Copy All | Copy | Print

' PV Function Example

'This example finds the present value of a 10-year $25,000 annuity that will pay $1,000 a year at 9.5%.

```
Sub main
    Dim aprate, periods
    Dim payment, annuityfv
    Dim due, presentvalue
    Dim msgtext
    aprate=9.5
    periods=120
    payment=1000
    annuityfv=25000
Rem Assume payments are made at end of month
    due=0
    presentvalue=PV(aprate/12,periods,-payment, annuityfv,due)
    msgtext= "The present value for a 10-year $25,000 annuity @ 9.5%"
    msgtext=msgtext & " with a periodic payment of $1,000 is: "
    msgtext=msgtext & Format(presentvalue, "Currency")
    MsgBox msgtext
End Sub
```

See Also

- FV
- IPmt
- IRR
- NPV
- Pmt
- PPmt
- Rate

' Randomize Statement Example

Close | Copy All | Copy | Print

'This example generates a random string of characters using the Randomize statement and Rnd function. The second For...Next loop is to slow down processing in the first For...Next loop so that Randomize can be seeded with a new value each time from the Timer function.

```
Sub main
    Dim x as Integer
    Dim y
    Dim str1 as String
    Dim str2 as String
    Dim letter as String
    Dim randomvalue
    Dim upper, lower
    Dim msgtext
    upper=Asc("z")
    lower=Asc("a")
    newline=Chr(10)
    For x=1 to 26
        Randomize
        randomvalue=Int(((upper - (lower+1)) * Rnd) +lower)
        letter=Chr(randomvalue)
        str1=str1 & letter
        For y = 1 to 1500
        Next y
    Next x
    msgtext=str1
    MsgBox msgtext
End Sub
```

See Also

Rnd
Timer

' Rate Function Example

Close | Copy All | Copy | Print

```
'This example finds the interest rate on a 10-year $25,000 annuity, that pays $100 per month.

Sub main
    Dim aprate
    Dim periods
    Dim payment, annuitypv
    Dim annuityfv, due
    Dim guess
    Dim msgtext as String
    periods=120
    payment=100
    annuitypv=0
    annuityfv=25000
    guess=.1
Rem Assume payments are made at end of month
    due=0
    aprate=Rate(periods,-payment,annuitypv,annuityfv, due, guess)
    aprate=(aprate*12)
    msgtext= "The percentage rate for a 10-year $25,000 annuity "
    msgtext=msgtext & "that pays $100/month has "
    msgtext=msgtext & "a rate of: " & Format(aprate, "Percent")
    MsgBox msgtext
End Sub
```

See Also

- FV
- IPmt
- IRR
- NPV
- Pmt
- PPmt
- PV

' ReDim Statement Example

Close | Copy All | Copy | Print

'This example finds the net present value for a series of cash flows. The array variable that holds the cash flow amounts is initially a dynamic array that is redimensioned after the user enters the number of cash flow periods they have.

```
Sub main
    Dim aprate as Single
    Dim varray() as Double
    Dim cflowper as Integer
    Dim x as Integer
    Dim netpv as Double
    cflowper=InputBox("Enter number of cash flow periods:")
    ReDim varray(cflowper)
    For x= 1 to cflowper
        varray(x)=InputBox("Enter cash flow amount for period #" &x &":")
    Next x
    aprate=InputBox ("Enter discount rate:")
    If aprate>1 then
        aprate=aprate/100
    End If
    netpv=NPV(aprate,varray())
    MsgBox "The Net Present Value is: " & Format(netpv,"Currency")
End Sub
```

See Also

- Dim
- Global
- Option Base
- Static

' Rem Statement Example

Close | Copy All | Copy | Print

'This example defines a dialog box with a combination list box and two buttons. The Rem statements describe each block of definition code.

```
Sub main
    Dim fchoices as String
    fchoices="File1" & Chr(9) & "File2" & Chr(9) & "File3"
    Begin Dialog UserDialog 185, 94, "OPEN Script Dialog Box"
Rem The next two lines create the combo box
        Text   9, 5, 69, 10, "Filename:", .Text1
        DropComboBox   9, 17, 88, 71, fchoices, .ComboBox1
Rem The next two lines create the command buttons
        OKButton   113, 14, 54, 13
        CancelButton   113, 33, 54, 13
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

(None)

' Reset Statement Example

Close | Copy All | Copy | Print

'This example creates a file, puts the numbers 1-10 in it, then attempts to Get past the end of the file. The On Error statement traps the error and execution goes to the Debugger code which uses Reset to close the file before exiting.

```
Sub main
' Put the numbers 1-10 into a file
    Dim x as Integer
    Dim y as Integer
    On Error Goto Debugger
    Open "C:\TEMP001" as #1 Len=2
    For x=1 to 10
        Put #1,x, x
    Next x
    Close #1
    msgtext="The contents of the file is:" & Chr(10)
    Open "C:\TEMP001" as #1 Len=2
    For x=1 to 10
        Get #1,x, y
        msgtext=msgtext & Chr(10) & y
    Next x
    MsgBox msgtext
done:
    Close #1
    Kill "C:\TEMP001"
    Exit Sub
Debugger:
    MsgBox "Error " & Err & " occurred. Closing open file."
    Reset
    Resume done
End Sub
```

See Also

Close

' Resume Statement Example

Close | Copy All | Copy | Print

'This example prints an error message if an error occurs during an attempt to open a file. The Resume statement jumps back into the program code at the label, done. From here, the program exits.

```
Sub main
    Dim msgtext, userfile
    On Error GoTo Debugger
    msgtext="Enter the filename to use:"
    userfile=InputBox$(msgtext)
    Open userfile For Input As #1
    MsgBox "File opened for input."
'   ....etc....
    Close #1
done:
    Exit Sub
Debugger:
    msgtext="Error number " & Err & " occurred at line: " & Erl
    MsgBox msgtext
    Resume done
End Sub
```

See Also

- Erl
- Err Function
- Err Statement
- Error
- Error Function
- On Error
- Trappable Errors

' Right Function Example

Close | Copy All | Copy | Print

'This example checks for the extension .BMP in a filename entered by a user and activates the Paintbrush application if the file is found. Note this uses the Option Compare statement to accept either uppercase or lowercase letters for the filename extension.

```
Option Compare Text
Sub main
    Dim filename as String
    Dim x
    filename=InputBox("Enter a .BMP file and path: ")
    extension=Right(filename,3)
    If extension="BMP" then
        x=Shell("PBRUSH.EXE",1)
        Sendkeys "%FO" & filename & "{Enter}", 1
    Else
        MsgBox "File not found or extension not .BMP."
    End If
End Sub
```

See Also

- GetField
- Instr
- Left
- Len
- LTrim
- Mid Function
- Mid Statement
- RTrim
- Trim

' RmDir Statement Example

Close | Copy All | Copy | Print

'This example makes a new temporary directory in C:\ and then deletes it.

```
Sub main
    Dim path as String
    On Error Resume Next
    path=CurDir(C)
    If path<>"C:\" then
        ChDir "C:\"
    End If
    MkDir "C:\TEMP01"
    If Err=75 then
        MsgBox "Directory already exists"
    Else
        MsgBox "Directory C:\TEMP01 created"
        MsgBox "Now removing directory"
        RmDir "C:\TEMP01"
    End If
End Sub
```

See Also

- ChDir
- ChDrive
- CurDir
- Dir
- MkDir

' Rnd Function Example

Close | Copy All | Copy | Print

'This example generates a random string of characters within a range. The Rnd function is used to set the range between lowercase "a" and "z". The second For...Next loop is to slow down processing in the first For...Next loop so that Randomize can be seeded with a new value each time from the Timer function.

```
Sub main
    Dim x as Integer
    Dim y
    Dim str1 as String
    Dim str2 as String
    Dim letter as String
    Dim randomvalue
    Dim upper, lower
    Dim msgtext
    upper=Asc("z")
    lower=Asc("a")
    newline=Chr(10)
    For x=1 to 26
        Randomize
        randomvalue=Int(((upper - (lower+1)) * Rnd) +lower)
        letter=Chr(randomvalue)
        str1=str1 & letter
        For y = 1 to 1500
        Next y
    Next x
    msgtext=str1
    MsgBox msgtext
End Sub
```

See Also

- Exp
- Fix
- Int
- Log
- Randomize
- Sgn
- Sqr

' Rset Statement Example

Close | Copy All | Copy | Print

'This example uses Rset to right-align an amount entered by the user in a field that is 15 characters long. It then pads the extra spaces with asterisks (*) and adds a dollar sign ($) and decimal places (if necessary).

```
Sub main
    Dim amount as String*15
    Dim x
    Dim msgtext
    Dim replacement
    replacement="*"
    amount=InputBox("Enter an amount:")
    position=InStr(amount,".")
    If Right(amount,3)<>".00" then
        amount=Rtrim(amount) & ".00"
    End If
    Rset amount="$" & Rtrim(amount)
    length=15-Len(Ltrim(amount))
    For x=1 to length
        Mid(amount,x)=replacement
    Next x
    Msgbox "Formatted amount: " & amount
End Sub
```

See Also

Lset

' RTrim Function Example

Close | Copy All | Copy | Print

'This example asks for an amount and then right-aligns it in a field that is 15 characters long. It uses Rtrim to trim any trailing spaces in the amount string, if the number entered by the user is less than 15 digits.

```
Sub main
    Dim amount as String*15
    Dim x
    Dim msgtext
    Dim replacement
    replacement="X"
    amount=InputBox("Enter an amount:")
    position=InStr(amount,".")
    If position=0 then
        amount=Rtrim(amount) & ".00"
    End If
    Rset amount="$" & Rtrim(amount)
    length=15-Len(Ltrim(amount))
    For x=1 to length
        Mid(amount,x)=replacement
    Next x
    Msgbox "Formatted amount: " & amount
End Sub
```

See Also

- GetField
- Left
- Len
- LTrim
- Mid Function
- Mid Statement
- Right
- Trim

' Second Function Example

Close | Copy All | Copy | Print

'This example displays the last saved date and time for a file whose name is entered by the user.

```
Sub main
    Dim filename as String
    Dim ftime
    Dim hr, min
    Dim sec
    Dim msgtext as String
i: msgtext="Enter a filename:"
    filename=InputBox(msgtext)
    If filename="" then
        Exit Sub
    End If
    On Error Resume Next
    ftime=FileDateTime(filename)
    If Err<>0 then
        MsgBox "Error in filename. Try again."
        Goto i:
    End If
    hr=Hour(ftime)
    min=Minute(ftime)
    sec=Second(ftime)
    Msgbox "The file's time is: " & hr &":" &min &":" &sec
End Sub
```

See Also

- Day
- Hour
- Minute
- Month
- Now
- Time Function
- Time Statement
- Weekday
- Year

' Seek Function Example

Close | Copy All | Copy | Print

'This example reads the contents of a sequential file line by line (to a carriage return) and displays the results. The second subprogram, CREATEFILE, creates the file "C:\TEMP001" used by the main subprogram.

```
Declare Sub createfile
Sub main
    Dim testscore as String
    Dim x
    Dim y
    Dim newline
    Call createfile
    Open "C:\TEMP001" for Input as #1
    x=1
    newline=Chr(10)
    msgtext= "The test scores are: " & newline
    Do Until x=Lof(1)
        Line Input #1, testscore
        x=x+1
        y=Seek(1)
        If y>Lof(1) then
            x=Lof(1)
        Else
            Seek 1,y
        End If
        msgtext=msgtext & newline & testscore
    Loop
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the numbers 10-100 into a file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=10 to 100 step 10
        Write #1, x
    Next x
    Close #1
End Sub
```

See Also

- Get
- Open
- Put
- Seek Statement

' Seek Statement Example

Close | Copy All | Copy | Print

'This example reads the contents of a sequential file line by line (to a carriage return) and displays the results. The second subprogram, CREATEFILE, creates the file "C:\TEMP001" used by the main subprogram.

```
Declare Sub createfile
Sub main
    Dim testscore as String
    Dim x
    Dim y
    Dim newline
    Call createfile
    Open "C:\TEMP001" for Input as #1
    x=1
    newline=Chr(10)
    msgtext= "The test scores are: " & newline
    Do Until x=Lof(1)
        Line Input #1, testscore
        x=x+1
        y=Seek(1)
        If y>Lof(1) then
            x=Lof(1)
        Else
            Seek 1,y
        End If
        msgtext=msgtext & newline & testscore
    Loop
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the numbers 10-100 into a file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=10 to 100 step 10
        Write #1, x
    Next x
    Close #1
End Sub
```

See Also

- Get
- Open
- Put
- Seek Function

' Select Case Statement Example

Close | Copy All | Copy | Print

'This example tests the attributes for a file and if it is hidden, changes it to a non-hidden file.

```
Sub main
    Dim filename as String
    Dim attribs, saveattribs as Integer
    Dim answer as Integer
    Dim archno as Integer
    Dim msgtext as String
    archno=32
    On Error Resume Next
    msgtext="Enter name of a file:"
    filename=InputBox(msgtext)
    attribs=GetAttr(filename)
    If Err<>0 then
        MsgBox "Error in filename. Re-run Program."
        Exit Sub
    End If
    saveattribs=attribs
    If attribs>= archno then
        attribs=attribs-archno
    End If
    Select Case attribs
        Case 2,3,6,7
            msgtext="   File: " &filename & " is hidden." & Chr(10)
            msgtext=msgtext & Chr(10) & "     Change it?"
            answer=Msgbox(msgtext,308)
            If answer=6 then
                SetAttr filename, saveattribs-2
                Msgbox "File is no longer hidden."
                Exit Sub
            End If
            MsgBox "Hidden file not changed."
        Case Else
            MsgBox "File was not hidden."
    End Select
End Sub
```

See Also

If...Then...Else
On...Goto
Option Compare

' SendKeys Statement Example

```
'This example starts the Windows Terminal application and dials a phone number entered by the
user.

Sub main
    Dim phonenumber, msgtext
    Dim x
    phonenumber=InputBox("Type telephone number to call:")
    x=Shell("Terminal.exe",1)
    SendKeys "%PD" & phonenumber & "{Enter}",1
    msgtext="Dialing..."
    MsgBox msgtext
End Sub
```

See Also

- AppActivate
- DoEvents
- Shell

' Set Statement Example

Close | Copy All | Copy | Print

'This example displays a list of open files in the software application, VISIO. It uses the Set statement to assign VISIO and its document files to object variables. To see how this example works, you need to start VISIO and open one or more documents.

```
Sub main
    Dim visio as Object
    Dim doc as Object
    Dim msgtext as String
    Dim i as Integer, doccount as Integer

'Initialize Visio
    Set visio = GetObject(,"visio.application")          ' find Visio
    If (visio Is Nothing) then
        Msgbox "Couldn't find Visio!"
        Exit Sub
    End If
'Get # of open Visio files
    doccount = visio.documents.count        'OLE Automation call to Visio
    If doccount=0 then
        msgtext="No open Visio documents."
    Else
        msgtext="The open files are: " & Chr$(13)
        For i = 1 to doccount
            Set doc = visio.documents(i)        ' access Visio's document method
            msgtext=msgtext & Chr$(13)& doc.name
        Next i
    End If
    MsgBox msgtext
End Sub
```

See Also

- CreateObject
- Is
- Me
- New
- Nothing
- Object Class
- Typeof

' SetAttr Statement Example

Close | Copy All | Copy | Print

```
'This example tests the attributes for a file and if it is hidden, changes it to a normal (not hidden)
file.

Sub main
    Dim filename as String
    Dim attribs, saveattribs as Integer
    Dim answer as Integer
    Dim archno as Integer
    Dim msgtext as String
    archno=32
    On Error Resume Next
    msgtext="Enter name of a file:"
    filename=InputBox(msgtext)
    attribs=GetAttr(filename)
    If Err<>0 then
        MsgBox "Error in filename. Re-run Program."
        Exit Sub
    End If
    saveattribs=attribs
    If attribs>= archno then
        attribs=attribs-archno
    End If
    Select Case attribs
        Case 2,3,6,7
            msgtext="   File: " &filename & " is hidden." & Chr(10)
            msgtext=msgtext & Chr(10) & "     Change it?"
            answer=Msgbox(msgtext,308)
            If answer=6 then
                SetAttr filename, saveattribs-2
                Msgbox "File is no longer hidden."
                Exit Sub
            End If
            MsgBox "Hidden file not changed."
        Case Else
            MsgBox "File was not hidden."
    End Select
End Sub
```

See Also

FileAttr
GetAttr

' SetField Function Example

Close | Copy All | Copy | Print

'This example extracts the last name from a full name entered by the user.

```
Sub main
    Dim username as String
    Dim position as Integer
    username=InputBox("Enter your full name:")
    Do
      position=InStr(username," ")
      If position=0 then
        Exit Do
      End If
      username=SetField(username,1," "," ")
      username=Ltrim(username)
    Loop
    MsgBox "Your last name is: " & username
End Sub
```

See Also

GetField

' Sgn Function Example

Close | Copy All | Copy | Print

'This example tests the value of the variable profit and displays 0 for profit if it is a negative number. The subroutine uses Sgn to determine whether profit is positive, negative or zero.

```
Sub main
    Dim profit as Single
    Dim expenses
    Dim sales
    expenses=InputBox("Enter total expenses: ")
    sales=InputBox("Enter total sales: ")
    profit=Val(sales)-Val(expenses)
    If Sgn(profit)=1 then
        MsgBox "Yeah! We turned a profit!"
    ElseIf Sgn(profit)=0 then
        MsgBox "Okay. We broke even."
    Else
        MsgBox "Uh, oh. We lost money."
    End If
End Sub
```

See Also

- Exp
- Fix
- Int
- Log
- Rnd
- Sqr

' Shell Function Example

Close | Copy All | Copy | Print

'This example activates the Terminal application and dials a number entered by the user.

```
Sub main
    Dim phonenumber, msgtext
    Dim x
    phonenumber=InputBox("Type telephone number to call:")
    x=Shell("Terminal.exe",1)
    SendKeys "%PD" & phonenumber & "{Enter}",1
    msgtext="Dialing..."
    MsgBox msgtext
End Sub
```

See Also

- AppActivate
- Command
- SendKeys

' Sin Function Example

```
'This example finds the height of the building, given the length of a roof and the roof pitch.

Sub main
    Dim height, rooflength
    Dim pitch
    Dim msgtext
    Const PI=3.14159
    Const conversion= PI/180
    pitch=InputBox("Enter the roof pitch in degrees:")
    pitch=pitch*conversion
    rooflength=InputBox("Enter the length of the roof in feet:")
    height=Sin(pitch)*rooflength
    msgtext="The height of the building is "
    msgtext=msgtext & Format(height, "##.##") & " feet."
    MsgBox msgtext
End Sub
```

See Also

- Atn
- Cos
- Tan
- Derived Trigonometric Functions

Close | Copy All | Copy | Print

' Space Function Example

'This example prints the octal numbers from 1 to 15 as a two-column list and uses Space to separate the columns.

```
Sub main
    Dim x,y
    Dim msgtext
    Dim nofspaces
    msgtext="Octal numbers from 1 to 15:" & Chr(10)
    For x=1 to 15
      nofspaces=10
      y=Oct(x)
      If Len(x)=2 then
          nofspaces=nofspaces-2
      End If
      msgtext=msgtext & Chr(10) & x & Space(nofspaces) & y
    Next x
    MsgBox msgtext
End Sub
```

See Also

- Spc
- String

' Spc Function Example

'This example puts five spaces and the string "ABCD" to a file. The five spaces are derived by taking 15 MOD 10, or the remainder of dividing 15 by 10.

```
Sub main
    Dim str1 as String
    Dim x as String*10
    str1="ABCD"
    Open "C:\TEMP001" For Output As #1
    Width #1, 10
    Print #1, Spc(15); str1
    Close #1
    Open "C:\TEMP001" as #1 Len=12
    Get #1, 1,x
    Msgbox "The contents of the file is: " & x
    Close #1
    Kill "C:\TEMP001"
End Sub
```

See Also

- Print
- Space
- Tab
- Width

' SQLClose Function Example

Close | Copy All | Copy | Print

```
'This example opens the data source named SqlTest, gets the names in the ODBC data sources,
and closes the connection.

Sub main
'    Declarations
'
     Dim outputStr As String
     Dim connection As Long
     Dim prompt As Integer
     Dim datasources(1 To 50) As Variant
     Dim retcode As Variant

     prompt = 5
'    Open the datasource "SqlTest"
     connection = SQLOpen("DSN=SqlTest", outputStr, prompt:=5)

     action1 = 1   ' Get the names of the ODBC datasources
     retcode = SQLGetSchema(connection:=connection,action:=1, qualifier:=qualifier,
ref:=datasources())

'    Close the datasource connection
     retcode = SQLClose(connection)

End Sub
```

See Also

- SQLError
- SQLExecQuery
- SQLGetSchema
- SQLOpen
- SQLRequest
- SQLRetrieve
- SQLRetrieveToFile

Close | Copy All | Copy | Print

' SQLError Function Example

```
'This example forces an error to test SQLError function.

sub main
'   Declarations
    Dim connection As long
    Dim prompt as integer
    Dim retcode as long
    Dim errors(1 To 3, 1 To 10) as Variant

'   Open the datasource
    connection = SQLOpen("DSN=SQLTESTW;UID=DBA;PWD=SQL",outputStr,prompt:=3)

'   force an error to test SQLError select a nonexistent table
    retcode = SQLExecQuery(connection:=connection,query:="select * from notable ")

'   Retrieve the detailed error message information into the errors array
    SQLError destination:=errors
    retcode = SQLClose(connection)
end sub
```

See Also

- SQLClose
- SQLExecQuery
- SQLGetSchema
- SQLOpen
- SQLRequest
- SQLRetrieve
- SQLRetrieveToFile

Close | Copy All | Copy | Print

' SQLExecQuery Function Example

```
'This example performs a query on the data source.

Sub main
'    Declarations
'
     Dim connection As Long
     Dim destination(1 To 50, 1 To 125)   As Variant
     Dim retcode As long

'    open the connection
     connection = SQLOpen("DSN=SqlTest",outputStr,prompt:=3)
'
'    Execute the query
     query = "select * from customer"
     retcode = SQLExecQuery(connection,query)
'
'    retrieve the first 50 rows with the first 6 columns of each row into
'    the array destination, omit row numbers and put column names in the
'    first row of the array
'
     retcode = SQLRetrieve(connection:=connection,destination:=destination,
columnNames:=1,rowNumbers:=0,maxRows:=50, maxColumns:=6,fetchFirst:=0)

'    Get the next 50 rows of from the result set
     retcode = SQLRetrieve(connection:=connection,destination:=destination,
columnNames:=1,rowNumbers:=0,maxRows:=50, maxColumns:=6)

'    Close the connection
     retcode = SQLClose(connection)

End Sub
```

See Also

- SQLClose
- SQLError
- SQLGetSchema
- SQLOpen
- SQLRequest
- SQLRetrieve
- SQLRetrieveToFile

' SQLGetSchema Function Example

Close | Copy All | Copy | Print

```
'This example opens the data source named SqlTest, gets the names in the ODBC data sources,
and closes the connection.

Sub main
'    Declarations
'
     Dim outputStr As String
     Dim connection As Long
     Dim prompt As Integer
     Dim datasources(1 To 50) As Variant
     Dim retcode As Variant

     prompt = 5
'    Open the datasource "SqlTest"
     connection = SQLOpen("DSN=SqlTest", outputStr, prompt:=5)

     action1 = 1   ' Get the names of the ODBC datasources
     retcode = SQLGetSchema(connection:=connection,action:=1, qualifier:=qualifier,
ref:=datasources())

'    Close the datasource connection
     retcode = SQLClose(connection)

End Sub
```

See Also

- SQLClose
- SQLError
- SQLExecQuery
- SQLOpen
- SQLRequest
- SQLRetrieve
- SQLRetrieveToFile

' SQLOpen Function Example

Close | Copy All | Copy | Print

```
'This example opens the data source named SqlTest, gets the names in the ODBC data sources,
and closes the connection.

Sub main
'    Declarations
'
     Dim outputStr As String
     Dim connection As Long
     Dim prompt As Integer
     Dim datasources(1 To 50) As Variant
     Dim retcode As Variant

     prompt = 5
'    Open the datasource "SqlTest"
     connection = SQLOpen("DSN=SqlTest", outputStr, prompt:=5)

     action1 = 1   ' Get the names of the ODBC datasources
     retcode = SQLGetSchema(connection:=connection,action:=1, qualifier:=qualifier,
ref:=datasources())

'    Close the datasource connection
     retcode = SQLClose(connection)

End Sub
```

See Also

- SQLClose
- SQLError
- SQLExecQuery
- SQLGetSchema
- SQLRequest
- SQLRetrieve
- SQLRetrieveToFile

' SQLRequest Function Example

Close | Copy All | Copy | Print

'This example will open the datasource SQLTESTW and execute the query specified by query and return the results in destination

```
Sub main
'   Declarations
'
    Dim destination(1 To 50, 1 To 125)   As Variant
    Dim prompt As integer

'   The following will open the datasource SQLTESTW and execute the query
'   specified by query and return the results in destination
'
    query = "select * from class"
    retcode =
SQLRequest("DSN=SQLTESTW;UID=DBA;PWD=SQL",query,outputStr,prompt,0,destination())

End Sub
```

See Also

- SQLClose
- SQLError
- SQLExecQuery
- SQLGetSchema
- SQLOpen
- SQLRetrieve
- SQLRetrieveToFile

Close | Copy All | Copy | Print

' SQLRetrieve Function Example

```
'This example retrieves information from a data source.

Sub main
'    Declarations
'
     Dim connection As Long
     Dim destination(1 To 50, 1 To 125)   As Variant
     Dim retcode As long

'    open the connection
     connection = SQLOpen("DSN=SqlTest",outputStr,prompt:=3)
 '
 '   Execute the query
     query = "select * from customer"
     retcode = SQLExecQuery(connection,query)

 '   retrieve the first 50 rows with the first 6 columns of each row into
 '   the array destination, omit row numbers and put column names in the
 '   first row of the array

     retcode = SQLRetrieve(connection:=connection,destination:=destination,
columnNames:=1,rowNumbers:=0,maxRows:=50, maxColumns:=6,fetchFirst:=0)

'    Get the next 50 rows of from the result set
     retcode = SQLRetrieve(connection:=connection,destination:=destination,
columnNames:=1,rowNumbers:=0,maxRows:=50, maxColumns:=6)

'    Close the connection
     retcode = SQLClose(connection)
End Sub
```

See Also

- SQLClose
- SQLError
- SQLExecQuery
- SQLGetSchema
- SQLOpen
- SQLRequest
- SQLRetrieveToFile

' SQLRetrieveToFile Function Example

Close | Copy All | Copy | Print

'This example opens a connection to a data source and retrieves information to a file.

```
Sub main
'    Declarations
'
     Dim connection As Long
     Dim destination(1 To 50, 1 To 125)   As Variant
     Dim retcode As long

'    open the connection

     connection = SQLOpen("DSN=SqlTest",outputStr,prompt:=3)
 '
 '   Execute the query
 '
     query = "select * from customer"
     retcode = SQLExecQuery(connection,query)

 '   Place the results of the previous query in the file named by
 '   filename and put the column names in the file as the first row.
 '   The field delimiter is %
 '
     filename = "c:\myfile.txt"
     columnDelimiter = "%"
     retcode = SQLRetrieveToFile(connection:=connection,destination:=filename,
columnNames:=1,columnDelimiter:=columnDelimiter)

     retcode = SQLClose(connection)

End Sub
```

See Also

- SQLClose
- SQLError
- SQLExecQuery
- SQLGetSchema
- SQLOpen
- SQLRequest
- SQLRetrieve

' Sqr Function Example

Close | Copy All | Copy | Print

```
'This example calculates the square root of 2 as a double-precision floating point value and
displays it in scientific notation.

Sub main
    Dim value as Double
    Dim msgtext
    value=CDbl(Sqr(2))
    msgtext= "The square root of 2 is: " & Format(Value,"Scientific")
    MsgBox msgtext
End Sub
```

See Also

- Exp
- Fix
- Int
- Log
- Rnd
- Sgn

' Static Statement Example

Close | Copy All | Copy | Print

```
'This example puts account numbers to a file using the record variable GRECORD and then prints
them again.

Type acctrecord
    acctno as Integer
End Type

Sub main
    Static grecord as acctrecord
    Dim x
    Dim total
    x=1
    grecord.acctno=1
    On Error Resume Next
    Open "C:\TEMP001" For Output as #1
    Do While grecord.acctno<>0
i:      grecord.acctno=InputBox("Enter 0 or new account #" & x & ":")
        If Err<>0 then
            MsgBox "Error occurred.   Try again."
            Err=0
            Goto i
        End If
        If grecord.acctno<>0 then
            Print #1, grecord.acctno
            x=x+1
        End If
    Loop
    Close #1
    total=x-1
    msgtext="The account numbers are: " & Chr(10)
    Open "C:\TEMP001" For Input as #1
    For x=1 to total
        Input #1, grecord.acctno
        msgtext=msgtext & Chr(10) & grecord.acctno
    Next x
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub
```

See Also

- Dim
- Function...End Function
- Global
- Option Base
- ReDim
- Sub...End Sub

' StaticComboBox Statement Example

Close | Copy All | Copy | Print

'This example defines a dialog box with a static combo box labeled "Installed Drivers" and the OK and Cancel buttons.

```
Sub main
    Dim cchoices as String
    cchoices="MIDI Mapper"+Chr$(9)+"Timer"
    Begin Dialog UserDialog 182, 116, "OPEN Script Dialog Box"
        StaticComboBox   7, 20, 87, 49, cchoices, .StaticComboBox1
        Text   6, 3, 83, 10, "Installed Drivers", .Text1
        OKButton   118, 12, 54, 14
        CancelButton   118, 34, 54, 14
    End Dialog
    Dim mydialogbox As UserDialog
    Dialog mydialogbox
    If Err=102 then
        MsgBox "You pressed Cancel."
    Else
        MsgBox "You pressed OK."
    End If
End Sub
```

See Also

- Begin Dialog...End Dialog
- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- Dialog
- DropComboBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text
- TextBox

Close | Copy All | Copy | Print

' Stop Statement Example

'This example stops program execution at the user's request.

```
Sub main
    Dim str1
    str1=InputBox("Stop program execution? (Y/N):")
    If str1="Y" or str1="y" then
        Stop
    End If
    MsgBox "Program complete."
End Sub
```

See Also

(None)

' Str Function Example

Close | Copy All | Copy | Print

'This example prompts for two numbers, adds them, then shows them as a concatenated string.

```
Sub main
    Dim x as Integer
    Dim y as Integer
    Dim str1 as String
    Dim value1 as Integer
    x=InputBox("Enter a value for x: ")
    y=InputBox("Enter a value for y: ")
    MsgBox "The sum of these numbers is: " & x+y
    str1=Str(x) & Str(y)
    MsgBox "The concatenated string for these numbers is: " & str1
End Sub
```

See Also

Format
Val

Close | Copy All | Copy | Print

' StrComp Function Example

'This example compares a user-entered string to the string "Smith".

```
Option Compare Text
Sub main
    Dim lastname as String
    Dim smith as String
    Dim x as Integer
    smith="Smith"
    lastname=InputBox("Type your last name")
    x=StrComp(lastname,smith,1)
    If x=0 then
        MsgBox "You typed 'Smith' or 'smith'."
    Else
        MsgBox "You typed: " & lastname & " not 'Smith'."
    End If
End Sub
```

See Also

Instr
Option Compare

' String Function Example

Close | Copy All | Copy | Print

```
'This example places asterisks (*) in front of a string that is printed as a payment amount..

Sub main
    Dim str1 as String
    Dim size as Integer
i: str1=InputBox("Enter an amount up to 999,999.99: ")
    If Instr(str1,".")=0 then
        str1=str1+".00"
    End If
    If Len(str1)>10 then
        MsgBox "Amount too large.   Try again."
        Goto i
    End If
    size=10-Len(str1)
'Print amount in a space on a check allotted for 10 characters
    str1=String(size,Asc("*")) & str1
    Msgbox "The amount is: $" & str1
End Sub
```

See Also

- Space
- Str

Close | Copy All | Copy | Print

' Sub...End Sub Function Example

'This example is a subroutine that uses the Sub...End Sub function.

```
Sub main
    MsgBox "Hello, World."
End Sub
```

See Also

- Call
- Declare
- Dim
- Function...End Function
- Global
- Option Explicit
- Static

' Tab Function Statement Example

Close | Copy All | Copy | Print

'This example prints the octal values for the numbers from 1 to 25. It uses Tab to put five character spaces between the values.

```
Sub main
    Dim x as Integer
    Dim y
    For x=1 to 25
        y=Oct$(x)
        Print x Tab(10) y
    Next x
End Sub
```

See Also

- Print
- Space
- Spc
- Width

' Tan Function Example

Close | Copy All | Copy | Print

'This example finds the height of the exterior wall of a building, given its roof pitch and the length of the building.

```
Sub main
    Dim bldglen, wallht
    Dim pitch
    Dim msgtext
    Const PI=3.14159
    Const conversion= PI/180
    On Error Resume Next
    pitch=InputBox("Enter the roof pitch in degrees:")
    pitch=pitch*conversion
    bldglen=InputBox("Enter the length of the building in feet:")
    wallht=Tan(pitch)*(bldglen/2)
    msgtext="The height of the building is: " & Format(wallht, "##.00")
    MsgBox msgtext
End Sub
```

See Also

- Atn
- Cos
- Sin
- Derived Trigonometric Functions

' Text Statement Example

Close | Copy All | Copy | Print

'This example defines a dialog box with a combination list and text box and three buttons.

```
Sub main
    Dim ComboBox1() as String
    ReDim ComboBox1(0)
    ComboBox1(0)=Dir("C:\*.*")
    Begin Dialog UserDialog 166, 142, "OPEN Script Dialog Box"
        Text   9, 3, 69, 13, "Filename:", .Text1
        DropComboBox   9, 14, 81, 119, ComboBox1(), .ComboBox1
        OKButton   101, 6, 54, 14
        CancelButton   101, 26, 54, 14
        PushButton 101, 52, 54, 14, "Help", .Push1
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin Dialog...End Dialog
- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- Dialog
- DropComboBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- TextBox

' TextBox Statement Example

Close | Copy All | Copy | Print

'This example creates a dialog box with a group box, and two buttons.

```
Sub main
    Begin Dialog UserDialog 194, 76, "OPEN Script Dialog Box"
        GroupBox   9, 8, 97, 57, "File Range"
        OptionGroup .OptionGroup2
            OptionButton   19, 16, 46, 12, "All pages", .OptionButton3
            OptionButton   19, 32, 67, 8, "Range of pages", .OptionButton4
        Text   25, 43, 20, 10, "From:", .Text6
        Text   63, 43, 14, 9, "To:", .Text7
        TextBox   79, 43, 13, 12, .TextBox4
        TextBox   47, 43, 12, 11, .TextBox5
        OKButton   135, 6, 54, 14
        CancelButton   135, 26, 54, 14
    End Dialog
    Dim mydialog as UserDialog
    On Error Resume Next
    Dialog mydialog
    If Err=102 then
        MsgBox "Dialog box canceled."
    End If
End Sub
```

See Also

- Begin Dialog...End Dialog
- Button
- ButtonGroup
- CancelButton
- Caption
- CheckBox
- ComboBox
- Dialog
- DropComboBox
- GroupBox
- ListBox
- OKButton
- OptionButton
- OptionGroup
- Picture
- StaticComboBox
- Text

' Time Function Example

Close | Copy All | Copy | Print

'This example writes data to a file if it hasn't been saved within the last 2 minutes.

```
Sub main
    Dim tempfile
    Dim filetime, curtime
    Dim msgtext
    Dim acctno(100) as Single
    Dim x, I
    tempfile="C:\TEMP001"
    Open tempfile For Output As #1
    filetime=FileDateTime(tempfile)
    x=1
    I=1
    acctno(x)=0
    Do
        curtime=Time
        acctno(x)=InputBox("Enter an account number (99 to end):")
        If acctno(x)=99 then
            For I=1 to x-1
                Write #1, acctno(I)
            Next I
            Exit Do
        ElseIf (Minute(filetime)+2)<=Minute(curtime) then
            For I=I to x
                Write #1, acctno(I)
            Next I
        End If
        x=x+1
    Loop
    Close #1
    x=1
    msgtext="Contents of C:\TEMP001 is:" & Chr(10)
    Open tempfile for Input as #1
    Do While Eof(1)<>-1
        Input #1, acctno(x)
        msgtext=msgtext & Chr(10) & acctno(x)
        x=x+1
    Loop
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub
```

See Also

- Date Function
- Date Statement
- Time Statement
- Timer
- TimeSerial
- TimeValue

Close | Copy All | Copy | Print

' Time Statement Example

'This example changes the time on the system clock.

```
Sub main
    Dim newtime as String
    Dim answer as String
    On Error Resume Next
i: newtime=InputBox("What time is it?")
    answer=InputBox("Is this AM or PM?")
    If answer="PM" or answer="pm" then
        newtime=newtime &"PM"
    End If
    Time=newtime
    If Err<>0 then
        MsgBox "Invalid time.   Try again."
        Err=0
        Goto i
    End If
End Sub
```

See Also

- Date Function
- Date Statement
- Time Function
- TimeSerial
- TimeValue

Close | Copy All | Copy | Print

' Timer Function Example

'This example uses Timer Function to find a Megabucks number.

```
Sub main
    Dim msgtext
    Dim value(9)
    Dim nextvalue
    Dim x
    Dim y
    msgtext="Your Megabucks numbers are: "
    For x=1 to 8
        Do
            value(x)=Timer
            value(x)=value(x)*100
            value(x)=Str(value(x))
            value(x)=Val(Right(value(x),2))
        Loop Until value(x)>1 and value(x)<36
        For y=1 to 1500
        Next y
    Next x
    For y=1 to 8
      For x= 1 to 8
        If y<>x then
            If value(y)=value(x) then
                value(x)=value(x)+1
            End If
        End If
      Next x
    Next y
    For x=1 to 8
        msgtext=msgtext & value(x) & " "
    Next x
    MsgBox msgtext
End Sub
```

See Also

Randomize

Close | Copy All | Copy | Print

' TimeSerial Function Example

'This example displays the current time using Time Serial.

```
Sub main
    Dim y
    Dim msgtext
    Dim nowhr
    Dim nowmin
    Dim nowsec
    nowhr=Hour(Now)
    nowmin=Minute(Now)
    nowsec=Second(Now)
    y=TimeSerial(nowhr,nowmin,nowsec)
    msgtext="The time is: " & y
    MsgBox msgtext
End Sub
```

See Also

- DateSerial
- Date Value
- Hour
- Minute
- Now
- Second
- TimeValue

' TimeValue Function Example

Close | Copy All | Copy | Print

'This example writes a variable to a disk file based on a comparison of its last saved time and the current time. Note that all the variables used for the TimeValue function are dimensioned as Double, so that calculations based on their values will work properly.

```
Sub main
    Dim tempfile
    Dim ftime
    Dim filetime as Double
    Dim curtime as Double
    Dim minutes as Double
    Dim acctno(100) as Integer
    Dim x, I
    tempfile="C:\TEMP001"
    Open tempfile For Output As 1
    ftime=FileDateTime(tempfile)
    filetime=TimeValue(ftime)
    minutes= TimeValue("00:02:00")
    x=1
    I=1
    acctno(x)=0
    Do
        curtime= TimeValue(Time)
        acctno(x)=InputBox("Enter an account number (99 to end):")
        If acctno(x)=99 then
            For I=I to x-1
                Write #1, acctno(I)
            Next I
            Exit Do
        ElseIf filetime+minutes<=curtime then
            For I=I to x
                Write #1, acctno(I)
            Next I
        End If
        x=x+1
    Loop
    Close #1
    x=1
    msgtext="You entered:" & Chr(10)
    Open tempfile for Input as #1
    Do While Eof(1)<>-1
        Input #1, acctno(x)
        msgtext=msgtext & Chr(10) & acctno(x)
        x=x+1
    Loop
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub
```

See Also

- DateSerial
- Date Value
- Hour
- Minute
- Now
- Second
- TimeSerial

' Trim Function Example

Close | Copy All | Copy | Print

```
'This example removes leading and trailing spaces from a string entered by the user.

Sub main
    Dim userstr as String
    userstr=InputBox("Enter a string with leading/trailing spaces")
    MsgBox "The string is: " & Trim(userstr) & " with nothing after it."
End Sub
```

See Also

- GetField
- Left
- Len
- LTrim
- Mid Function
- Mid Statement
- Right
- RTrim

' Type Statement Example

Close | Copy All | Copy | Print

'This example shows a Type and Dim statement for a record. You must define a record type before you can declare a record variable. The subroutine then references a field within the record.

```
Type Testrecord
    Custno As Integer
    Custname As String
End Type
Sub main
    Dim myrecord As Testrecord
i: myrecord.custname=InputBox("Enter a customer name:")
    If myrecord.custname="" then
        Exit Sub
    End If
    answer=InputBox("Is the name: " & myrecord.custname &" correct? (Y/N)")
    If answer="Y" or answer="y" then
        MsgBox "Thank you."
    Else
        MsgBox "Try again."
        Goto i
    End If
End Sub
```

See Also

Def*type*
Dim

Close | Copy All | Copy | Print

' Typeof Statement Example

(None)

See Also

- CreateObject
- GetObject
- Is
- Me
- New
- Nothing
- Object Class

' UBound Function Example

Close | Copy All | Copy | Print

'This example resizes an array if the user enters more data than can fit in the array. It uses LBound and UBound to determine the existing size of the array and ReDim to resize it. Option Base sets the default lower bound of the array to 1.

```
Option Base 1
Sub main
    Dim arrayvar() as Integer
    Dim count as Integer
    Dim answer as String
    Dim x, y as Integer
    Dim total
    total=0
    x=1
    count=InputBox("How many test scores do you have?")
    ReDim arrayvar(count)
start:
    Do until x=count+1
      arrayvar(x)=InputBox("Enter test score #" &x & ":")
      x=x+1
    Loop
    answer=InputBox$("Do you have more scores? (Y/N)")
    If answer="Y" or answer="y" then
      count=InputBox("How many more do you have?")
      If count<>0 then
        count=count+(x-1)
        ReDim Preserve arrayvar(count)
        Goto start
      End If
    End If
    x=LBound(arrayvar,1)
    count=UBound(arrayvar,1)
    For y=x to count
          total=total+arrayvar(y)
    Next y
    MsgBox "The average of the " & count & " scores is: " & Int(total/count)
End Sub
```

See Also

- Dim
- Global
- LBound
- Option Base
- ReDim
- Static

' UCase Function Example

Close | Copy All | Copy | Print

'This example converts a filename entered by a user to all uppercase letters.

```
Option Base 1
Sub main
    Dim filename as String
    filename=InputBox("Enter a filename: ")
    filename=UCase(filename)
    MsgBox "The filename in uppercase is: " & filename
End Sub
```

See Also

- Asc
- LCase

' Unlock Function Example

Close | Copy All | Copy | Print

'This example locks a file that is shared by others on a network, if the file is already in use. The second subprogram, CREATEFILE, creates the file used by the main subprogram.

```
Declare Sub createfile
Sub main
    Dim btngrp, icongrp
    Dim defgrp
    Dim answer
    Dim noaccess as Integer
    Dim msgabort
    Dim msgstop as Integer
    Dim acctname as String
    noaccess=70
    msgstop=16
    Call createfile
    On Error Resume Next
    btngrp=1
    icongrp=64
    defgrp=0
    answer=MsgBox("Open the account file?" & Chr(10), btngrp+icongrp+defgrp)
    If answer=1 then
        Open "C:\TEMP001" for Input as #1
        If Err=noaccess then
            msgabort=MsgBox("File Locked",msgstop,"Aborted")
        Else
            Lock #1
            Line Input #1, acctname
            MsgBox "The first account name is: " & acctname
            Unlock #1
        End If
        Close #1
    End If
    Kill "C:\TEMP001"
End Sub

Sub createfile()
    Rem Put the letters A-J into the file
    Dim x as Integer
    Open "C:\TEMP001" for Output as #1
    For x=1 to 10
        Write #1, Chr(x+64)
    Next x
    Close #1
End Sub
```

See Also

- Lock
- Open

' Val Function Example

Close | Copy All | Copy | Print

'This example tests the value of the variable profit and displays 0 for profit if it is a negative number. The subroutine uses Sgn to determine whether profit is positive, negative or zero.

```
Sub main
    Dim profit as Single
    Dim expenses
    Dim sales
    expenses=InputBox("Enter total expenses: ")
    sales=InputBox("Enter total sales: ")
    profit=Val(sales)-Val(expenses)
    If Sgn(profit)=1 then
        MsgBox "Yeah! We turned a profit!"
    ElseIf Sgn(profit)=0 then
        MsgBox "Okay. We broke even."
    Else
        MsgBox "Uh, oh. We lost money."
    End If
End Sub
```

See Also

- CCur
- CDbl
- CInt
- CLng
- CSng
- CStr
- CVar
- CVDate
- Format
- Str

Close | Copy All | Copy | Print

' VarType Function Example

'This example returns the type of a variant.

```
Sub main
    Dim x
    Dim myarray(8)
    Dim retval
    Dim retstr
    myarray(1)=Null
    myarray(2)=0
    myarray(3)=39000
    myarray(4)=CSng(10^20)
    myarray(5)=10^300
    myarray(6)=CCur(10.25)
    myarray(7)=Now
    myarray(8)="Five"
    For x=0 to 8
        retval=Vartype(myarray(x))
        Select Case retval
            Case 0
                retstr=" (Empty)"
            Case 1
                retstr=" (Null)"
            Case 2
                retstr=" (Integer)"
            Case 3
                retstr=" (Long)"
            Case 4
                retstr=" (Single)"
            Case 5
                retstr=" (Double)"
            Case 6
                retstr=" (Currency)"
            Case 7
                retstr=" (Date)"
            Case 8
                retstr=" (String)"
        End Select
        If retval=1 then
            myarray(x)="[null]"
        ElseIf retval=0 then
            myarray(x)="[empty]"
        End If
        MsgBox "The variant type for " &myarray(x) & " is: " &retval &retstr
    Next x
End Sub
```

See Also

- IsDate
- IsEmpty
- IsNull
- IsNumeric

' Weekday Function Example

Close | Copy All | Copy | Print

'This example finds the day of the week on which New Year's Day will fall in the year 2000.

```
Sub main
    Dim newyearsday
    Dim daynumber
    Dim msgtext
    Dim newday as Variant
    Const newyear=2000
    Const newmonth=1
    Let newday=1
    newyearsday=DateSerial(newyear,newmonth,newday)
    daynumber=Weekday(newyearsday)
    msgtext="New Year's day 2000 falls on a " & Format(daynumber, "dddd")
    MsgBox msgtext
End Sub
```

See Also

- Date Function
- Date Statement
- Day
- Hour
- Minute
- Month
- Now
- Second
- Year

' While...Wend Structure Example

Close | Copy All | Copy | Print

'This example opens a series of customer files and checks for the string "*Overdue*" in each file. It uses While...Wend to loop through the C:\TEMP00? files. These files are created by the subroutine CREATEFILES.

```
Declare Sub createfiles
Sub main
    Dim custfile as String
    Dim aline as String
    Dim pattern as String
    Dim count as Integer
    Call createfiles
    Chdir "C:\"
    custfile=Dir$("TEMP00?")
    pattern="*" + "Overdue" + "*"
    While custfile <> ""
        Open custfile for input as #1
        On Error goto atEOF
        Do
            Line Input #1, aline
            If aline Like pattern Then
                count=count+1
            End If
        Loop
nxtfile:
        On Error GoTo 0
        Close #1
        custfile = Dir$
    Wend
    If count<>0 then
        Msgbox "Number of overdue accounts: " & count
    Else
        Msgbox "No accounts overdue"
    End If
    Kill "C:\TEMP001"
    Kill "C:\TEMP002"
    Exit Sub
atEOF:
    Resume nxtfile
End Sub

Sub createfiles()
  Dim odue as String
  Dim ontime as String
  Dim x
  Open "C:\TEMP001" for OUTPUT as #1
  odue="*" + "Overdue" + "*"
  ontime="*" + "On-Time" + "*"
  For x=1 to 3
      Write #1, odue
  Next x
  For x=4 to 6
      Write #1, ontime
  Next x
  Close #1
  Open "C:\TEMP002" for Output as #1
  Write #1, odue
  Close #1
End Sub
```

See Also

Do...Loop

' Width Statement Example

Close | Copy All | Copy | Print

'This example puts five spaces and the string "ABCD" to a file. The five spaces are derived by taking 15 MOD 10, or the remainder of dividing 15 by 10.

```
Sub main
    Dim str1 as String
    Dim x as String*10
    str1="ABCD"
    Open "C:\TEMP001" For Output As #1
    Width #1, 10
    Print #1, Spc(15); str1
    Close #1
    Open "C:\TEMP001" as #1 Len=12
    Get #1, 1,x
    Msgbox "The contents of the file is: " & x
    Close #1
    Kill "C:\TEMP001"
End Sub
```

See Also

Open
Print

' With Statement Example

Close | Copy All | Copy | Print

'This example creates a user-defined record type, custrecord and uses the With statement to fill in values for the record fields, for the record called "John".

```
Type custrecord
    name as String
    ss as String
    salary as Single
    dob as Variant
    street as String
    apt as Variant
    city as String
    state as String
End Type
Sub main
    Dim John as custrecord
    Dim msgtext
    John.name="John"
    With John
        .ss="037-67-2947"
        .salary=60000
        .dob=#10-09-65#
        .street="15 Chester St."
        .apt=28
        .city="Cambridge"
        .state="MA"
    End With
    msgtext=Chr(10) & "Name:" & Space(5) & John.name & Chr(10)
    msgtext=msgtext & "SS#: " & Space(6) & john.ss & chr(10)
    msgtext=msgtext & "D.O.B:" & Space(4) & john.dob
    Msgbox "Done with: " & Chr(10) & msgtext
End Sub
```

See Also

Type...End Type

' Write Statement Example

Close | Copy All | Copy | Print

'This example writes a variable to a disk file based on a comparison of its last saved time and the current time.

```
Sub main
    Dim tempfile
    Dim filetime, curtime
    Dim msgtext
    Dim acctno(100) as Single
    Dim x, I
    tempfile="C:\TEMP001"
    Open tempfile For Output As #1
    filetime=FileDateTime(tempfile)
    x=1
    I=1
    acctno(x)=0
    Do
        curtime=Time
        acctno(x)=InputBox("Enter an account number (99 to end):")
        If acctno(x)=99 then
            If x=1 then Exit Sub
            For I=1 to x-1
                Write #1, acctno(I)
            Next I
            Exit Do
        ElseIf (Minute(filetime)+2)<=Minute(curtime) then
            For I=I to x-1
                Write #1, acctno(I)
            Next I
        End If
        x=x+1
    Loop
    Close #1
    x=1
    msgtext="Contents of C:\TEMP001 is:" & Chr(10)
    Open tempfile for Input as #1
    Do While Eof(1)<>-1
        Input #1, acctno(x)
        msgtext=msgtext & Chr(10) & acctno(x)
        x=x+1
    Loop
    MsgBox msgtext
    Close #1
    Kill "C:\TEMP001"
End Sub
```

See Also

- Close
- Open
- Print
- Put

Close | Copy All | Copy | Print

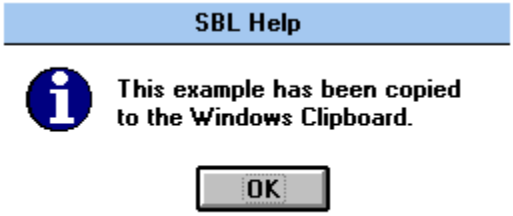
' Year Function Example

```
'This example returns the year for today.

Sub main
    Dim nowyear
    nowyear=Year(Now)
    MsgBox "The current year is: " &nowyear
End Sub
```

See Also

- Date Function
- Date Statement
- Day
- Hour
- Minute
- Month
- Now
- Time Function
- Second
- Weekday


SBL Help
This example has been copied
to the Windows Clipboard.
OK

Bitmap Viewer

Displays a series of bitmaps in a dialog box

```
Declare Sub GetWindowsDirectory Lib "kernel32" Alias _
                  "GetWindowsDirectoryA" (ByVal buf$, ByVal buflen%)
Dim fname$, WinDir$
Const IdleLoop = 5

'Dialog Box Function.   Find and display next bitmap.
Function DlgFunc% (id$, action%, svalue&)
    If action = IdleLoop And (svalue Mod 800 = 799) Then
        fname = dir$
        If fname = ""   Then
            SendKeys "{enter}"
            Exit Function
        End If
        ' load next picture
        DlgSetPicture "p1", WinDir & fname, 0
        DlgText DlgControlID("FileName"), fname
    End If
    If action = IdleLoop Then DlgFunc = 1
End Function

Sub Main
    Dim WinDirBuf as String * 150
'Find Windows bitmap files
    Call GetWindowsDirectory   (WinDirBuf, Len(WinDirBuf) )
    WinDir = Left(WinDirBuf, InStr(WinDirBuf, Chr$(0))-1) & "\"
    fname   = Dir$(WinDir & "*.bmp")
    If (fname = "") Then Exit Sub

    Begin Dialog PictureBoxType 25, 25, 210, 240, "Picture" , .DlgFunc
        Picture           5,    5, 200, 200, WinDir & fname, 0, .p1
        Text             15, 225,   70,   15, fname, .FileName
        PushButton 145, 220,   45,   15, "Stop"
    End Dialog
    Dim PictureBox as PictureBoxType
    Dialog PictureBox
End Sub
```

Find Files

Finds a test file containing a specified string, based on pattern matching

```
Option compare binary
Dim count                       '   Number of files searched.
Const DialogInit     = 1        '   Used to determine why the
Const ButtonPush     = 2        '   dialog box function was called.
Const TextBoxEnter = 3
Const IdleLoop       = 5
'
' Function searchFiles finds the files and does the comparison.
' According to user defined flags, it will either use string
' comparison (the InStr function) or regular expressions
' comparison (the Like operator).   The user also chooses
' whether the comparison will be case sensitive or insensitive.
'
Function searchFiles$(fileSpec$, subPattern$, caseSensitive%, regexp%)
     Dim aLine$
     retVal     = ""
     thisFile = dir$(fileSpec)
     pattern   = subPattern
     While thisFile <> ""
          count = count+1
          Open thisFile for input as #1
          Do While Not Eof(1)
               Line Input #1, aLine
               If regexp Then
                    On Error Goto badRegexp
                    If Left$(pattern,1) <> "*" Then pattern = "*"+pattern+"*"
                    If Not caseSensitive Then       ' convert to upper case
                         pattern = UCase(pattern)
                         aLine = UCase$(aLine)
                    End If
                    If aLine Like pattern Then
                         retVal = retVal + thisFile + chr$(13)
                         Exit Do
                    End If
               ElseIf InStr(1, aLine, pattern, 1 - caseSensitive) Then
                    retVal = retVal + thisFile + chr$(13)
                    Exit Do
               End If
          Loop
          Close #1
          thisFile = dir$
     Wend
     searchFiles = retVal
     Exit Function
badRegexp:
     MsgBox "Error: Bad regular expression"
End Function
'
' Dialog Box Procedure
'
Function DlgProc%(Control$, action%, values&)
    CR = Chr(13) : TabC = Chr(9)
    HelpText = "Regular expression pattern matching rules:" & CR & CR & _
          "?"   & TabC & _
          "match any single character" & CR & _
          "*"   & TabC & _
          "match any set of zero or more characters" & CR & _
          "#"   & TabC & _
          "match any single digit character (0-9)" & CR & _
          "[chars]" & TabC & _
          "match any single character in chars" & CR & _
```

```
            "[!chars]" & TabC & _
            "match any single character not in chars" & CR & CR & _
            "Note: rules are per Visual Basic"
    DlgProc = 0
    Select Case action
        ' disable find button until a search string is entered.
        Case DialogInit : DlgEnable 7, 0
        Case ButtonPush And (values=18)
            ' display help message
            MsgBox HelpText, 0, "Help"
            DlgProc = 1
        Case TextBoxEnter And (Contol="searchPattern")
            ' search string entered, enable find button.
            DlgEnable 7, 1
        Case IdleLoop
            ' whenever the searchpattern is empty, disable the find button
            ' whenever it becomes nonempty, enable the find button
            patternID = DlgControlID("searchPattern")
            If DlgText(patternID) <> "" Then DlgEnable 7,1 Else DlgEnable 7,0
            DlgProc = 1
    End Select
End Function
'
' Prompt user for keyword and filespec.
'
Sub main
    Begin dialog listboxd 30, 50, 165, 110, "Document Search", .DlgProc
        text             10, 10, 60, 15, "&Files to Search:"
        textbox           70,   7, 75, 15, .files
        text             10, 27, 60, 15, "&Search Pattern:"
        textbox           70, 24, 75, 15, .searchPattern
        checkbox          25, 75, 85, 15, "Match Case", .xcase
        checkbox          25, 90, 85, 15, "Use Pattern Matching", .regexp
        buttongroup .but
        button            25, 55, 60, 15, "Find"
        button           110, 90, 40, 15, "Help"
        cancelbutton      90, 55, 60, 15
    End dialog
    On Error Goto Cancelled
    Dim SearchBox as listboxd
    SearchBox.files = "*.oss"
    SearchBox.xcase = 0
    Dialog SearchBox
    fileList = searchFiles(SearchBox.files, SearchBox.searchPattern, _
               SearchBox.xcase, SearchBox.regexp)
    If fileList = "" Then
        MsgBox "Pattern " & """ & SearchBox.searchPattern & """ _
        & " not found in " & count & " file(s)"
    Else
        MsgBox fileList
    End If
Cancelled:
    Exit Sub
    Resume
End Sub
```

Greatest Common Factor

Updates a dialog box dynamically, based on user input

```
Dim msg$                ' Module-level variable, visible to all functions below
Const ButtonPush    = 2       ' Dialog box actions
Const TextBoxEnter = 3

'
' In this function, the greatest common factor is computed.
'
Function gcf% ( u%, v% )
     dim t%

     If ( u < v ) Then t=u   Else t=v
     While ( (u mod t) <> 0) OR ( (v mod t) <> 0)
        t=t-1
     Wend
     gcf = t
End Function

'
' CheckNumbers verifies both numbers are positive.
'
Function CheckNumbers% (Control$, action%, values&)

    If action = TextBoxEnter Then
        If Val(DlgText$(Control)) < 1 Then
            RetVal = 1
            DlgText "errmsg", "Bad number, please reenter"
            DlgFocus Control
        End If
    ElseIf action = ButtonPush and values = 16 Then
        a = Val(DlgText$("num1"))
        b = Val(DlgText$("num2"))
        If a < 1 Or a <> Int(a) Then
            RetVal = 1
            MsgBox "Bad number, please reenter"
            DlgFocus "num1"
        ElseIf (b < 1 Or b <> Int (b)) And RetVal = 0 Then
            RetVal = 1
            MsgBox "Bad number, please reenter"
            DlgFocus "num2"
        Else                            ' no error found, ok to print out answer
            DlgText "errmsg", "The answer is " & gcf(a,b)
            RetVal = 1
        End If
    End If
    CheckNumbers = RetVal           ' if RetVal = 0, dialog box will be exited
End Function

'
' Showdlg creates and displays the dialog box, prompting the
' user to input the two numbers.
'
Sub Showdlg

    Begin dialog enter2num 60,60,150,50, " ** G C F **",.CheckNumbers
        text       3,   4,   40, 10,    "first number"
        textbox 60,   2,   25, 12,                        .num1$
        text       3, 18,   70, 10,    "second number"
        textbox 60, 18,   25, 12,                        .num2$
        text       5, 35, 130, 10,     msg$,              .errmsg

        OptionGroup   .but
```

```
        PushButton 100,   1,   40, 15,    "OK",              .okBut
        PushButton 100, 18,   40, 15,    "Cancel",          .cancelBut
    End dialog

    Dim InputDlg as enter2num
    InputDlg.num1$="0"
    InputDlg.num2$="0"
    Dialog InputDlg
End Sub

Sub Main
    Call Showdlg
End Sub
```

Hello World

Demonstrates calls to subroutines and functions

```
'   MessageBox and GetTickCount are calls to functions defined in
'   user32.dll and kernel32.dll.

Declare Sub MessageBox LIB "user32.dll" Alias "MessageBoxA" (BYVAL h%, _
                    BYVAL t$, BYVAL c$, BYVAL u%)
Declare Function GetTickCount& LIB "kernel32.dll" ()

'
' Function CAT$ concatenates two strings with a space between them
'
Function Cat$(a$, b$)
     Cat = a & " " & b
End Function

'
' Subprogram Say computes the time and displays a message box.
'
Sub Say(what$)
     Dim min, sec, hrs

     sec = GetTickCount() /1000
     min = sec / 60 : sec = sec mod 60
     hrs = min / 60 : min = min mod 60

     Dim eTime as variant                          ' DIM can now be anywhere
     eTime = Format$(hrs,"00") & ":" & Format$(min,"00") & ":" & Format$(sec,"00")
     MessageBox 0, what, "Elapsed Time is " & eTime, 64
End Sub

Sub Main
     Dim msg$
     If (Command$ = "") Then msg$ = "world" Else msg$ = Command$
     Say Cat("Hello", msg$)
End Sub
```

Quicksort

Performs a recursive quicksort

```
Const max% = 5000               ' Maximum length of data to be sorted.
Const ButtonPush     = 2        ' Used to determine why a
Const TextBoxEnter = 3        ' dialog box function was called.
Const IdleLoop        = 5
Dim a(MAX) as Double
Dim count%, StarField%, Flag%, R%, Graphics%
'
' Display stars indicating recursion depth.
'
Sub Display
     For i%=1 To 1000 : Next i                  ' Delay loop
     DlgText StarField, String$(R,"*")
End Sub

'   Sort the array of numbers.   Note that OPEN Script allows recursion.
'
Sub QuickSort(LeftSide%, RightSide%)
     Dim v#, t as Double
     Dim i as integer, j%

     If Graphics Then
          R = R+1 : Call Display      ' display recursion level
     End If
     If (RightSide>LeftSide) Then
          v=a(RightSide) : i=LeftSide-1 : j=RightSide : a(0) = v
          Do
               Do : i=i+1 :     Loop Until a(i)>=v
               Do : j=j-1 :     Loop Until a(j)<=v
               t=a(i) : a(i)=a(j) : a(j)=t
          Loop Until (j<=i)
          a(j)=a(i) : a(i)= a(RightSide) : a(RightSide)=t
          Call QuickSort(LeftSide,i-1)
          Call QuickSort(i+1,RightSide)
     End If
     If Graphics Then
          R = R-1 : Call Display
     End If
End Sub

'   Dialog Box Function for star display dialog box.
'   Every dialog box can have its own dialog box function.
'
Function DlgFunc%(Control$, action%, values&)
     ' a sneaky way to make a dialog box with no button:
     ' create a button but make it invisible.
     If action = 1 Then DlgVisible DlgControlID("Stop"), 0
     If action = IdleLoop Then
          DlgFunc = 1
          If Flag = 0 Then
               Flag = 1
               ' get the ID of the field which will contain stars
               StarField = DlgControlID("Stars")
               Call QuickSort(1, count)
               ' when sorting is done, close the dialog box.
               SendKeys "{enter}"
               Exit Function
          End If
     End If
End Function

'   Verify array size.
```

```
'
Function InputFunc%(Control$, action%, values&)
    If (action = ButtonPush) And (Control = "OkBut") Then
       If (Val(DlgText("Data")) <= 0) Or (Val(DlgText("Data")) > 1000) Then
          MsgBox "Invalid list size"
          DlgFocus DlgControlID("Data")
          InputFunc = 1
       End If
    End If
End Function

Sub Main
    Begin Dialog StarBoxType   106, 20, "Recursion Level", .DlgFunc
      Text      5, 8, 101, 10, "Text", .Stars
      PushButton 1, 3,     1,   1, "Stop", .Stop
    End Dialog
    Begin Dialog DataBoxType 20, 30, 186, 47, "Quicksort Parameters", .InputFunc
      TextBox           83,   9, 25, 11, .Data
      OKButton          130,   6, 50, 14, .OkBut
      CancelButton   130, 23, 50, 14
      Text                4, 10, 75, 10, "Size of list (0 - 1000)"
      CheckBox            6, 25, 98,   8, "Animation", .Graphics
    End Dialog

    On Error Goto Done
    Randomize
    Dim DataBox as DataBoxType
    DataBox.Data = "500"                ' Default array size
    Dialog DataBox
    count = Val(DataBox.Data)           ' Actual array size
    t0      = timer
    For i=1 To count : a(i) = Rnd(0.5) : Next i ' make random data
    Dim StarBox as StarBoxType
    Graphics = DataBox.Graphics
    If Graphics Then
        Dialog StarBox
    Else
        Call QuickSort(1, count)
    End If
    t1 = timer
    Msgbox "elapsed time = "+str(t1-t0), ,"Quicksort Finished"
Done:
    Exit Sub
    Resume Next
End Sub
```

Abs Function

See Also **Example**

Returns the absolute value of a number.

Syntax Abs(*number*)

where:	**is:**
number	Any valid numeric expression.

Comments The data type of the return value matches the type of the *number*. If *number* is a **Variant** string (vartype 8), the return value will be converted to vartype 5 (Double). If the absolute value evaluates to vartype 0 (Empty), the return value will be vartype 3 (Long).

AppActivate Statement

See Also **Example**

Activates an application window.

Syntax **AppActivate** *title*

where:	**is:**
title	A string expression for the title-bar name of the application window to activate.

Comments *Title* must match the name of the window character for character, but comparison is not case-sensitive, e.g., "File Manager" is the same as "file manager" or "FILE MANAGER". If there is more than one window with a name matching *title,* a window is chosen at random.

AppActivate changes the focus to the specified window but does not change whether the window is minimized or maximized. Use **AppActivate** with the **SendKeys** statement to send keys to another application.

Asc Function

See Also **Example**

Returns an integer corresponding to the ANSI code of the first character in the specified string.

Syntax **Asc(** *string$* **)**

where:	**is:**
string$	A string expression of one or more characters.

Comments To change an ANSI code to string characters, use **Chr**.

Atn Function

See Also **Example**

Returns the angle (in radians) for the arc tangent of the specified number.

Syntax **Atn(** *number* **)**

where:	**is:**
number	Any valid numeric expression.

Comments The **Atn** function assumes *number* is the ratio of two sides of a right triangle: the side opposite the angle to find and the side adjacent to the angle. The function returns a single-precision value for a ratio expressed as an integer, a currency, or a single-precision numeric expression. The return value is a double-precision value for a long, Variant or double-precision numeric expression.

To convert radians to degrees, multiply by (180/PI). The value of PI is approximately 3.14159.

Beep Statement

See Also **Example**

Produces a tone through the computer speaker.

Syntax **Beep**

Comments The frequency and duration of the tone depends on the hardware.

Begin Dialog ... End Dialog Statement

See Also **Example** **Overview**

Begins and ends a dialog-box declaration.

Syntax

Begin Dialog *dialogName* [*x , y* **,**] *dx , dy* [, *caption$*] [, **.***dialogfunction*]
' dialog box definition statements
End Dialog

where:	**is:**
dialogName	The record name for the dialog box definition.
x , y	The coordinates for the upper left corner of the dialog box.
dx , dy	The width and height of the dialog box (relative to *x* and *y*).
caption$	The title for the dialog box.
.dialogfunction	A Basic function to process user actions in the dialog box.

Comments

To display the dialog box, you create a dialog record variable with the **Dim** statement, and then display the dialog box using the **Dialog function** or **Dialog statement** with the variable name as its argument. In the **Dim** statement, this variable is defined **As** *dialogName*.

The *x* and *y* coordinates are relative to the upper left corner of the client area of the parent window. The *x* argument is measured in units that are 1/4 the average width of the system font. The *y* argument is measured in units 1/8 the height of the system font. For example, to position a dialog box 20 characters in, and 15 characters down from the upper left hand corner, enter 80, 120 as the *x , y* coordinates. If these arguments are omitted, the dialog box is centered in the client area of the parent window.

The *dx* argument is measured in 1/4 system-font character-width units. The *dy* argument is measured in 1/8 system-font character-width units. For example, to create a dialog box 80 characters wide, and 15 characters in height, enter 320, 120 for the *dx , dy* coordinates.

If the *caption$* argument is omitted, a standard default caption is used.

The optional *.dialogfunction* function must be defined (using the **Function** statement) or declared (using **Dim**) before being used in the **Begin Dialog** statement. Define the *dialogfunction* with the following three arguments:

Function *dialogfunction%* **(** *id$, action% , suppvalue&* **)**
' function body
End Function

id$	The text string that identifies the dialog control that triggered the call to the dialog function (usually because the user changed this control).
action%	An integer from 1 to 5 identifying the reason why the dialog function was called.
suppvalue&	Gives more specific information about why the dialog function was called.

As with any Basic function, these arguments can have different names. The arguments of the dialog function can also be Variants. (Click the underlined argument above to see more about it.)

In most cases, the return value of *dialogfunction* is ignored. The exceptions are a return value of 2 or 5 for *action%*. If the user clicks the OK button, Cancel button, or a command button (as indicated by an *action%* return value of 2 and the corresponding *id$* for the button clicked), and the dialog function returns a non-zero value, the dialog box will *not* be closed.

Unless the **Begin Dialog** statement is followed by at least one other dialog-box definition statement and the **End Dialog** statement, an error will result. The definition statements must include an **OKButton**, **CancelButton** or **Button** statement. If this statement is left out, there will be no way to close the dialog box, and the procedure will be unable to continue executing.

Id$* is the same value for the dialog control that you use in the definition of that control. For example, the *id$ value for a text box is Text1 if it is defined this way:

```
Textbox 271 , 78, 33, 18, .Text1
```

The following table summarizes the possible *action%* values and their meanings:

action%	Meaning
1	Dialog box initialization. This value is passed before the dialog box becomes visible.
2	Command button selected or dialog box control changed (except typing in a text box or combo box).
3	Change in a text box or combo box. This value is passed when the control loses the input focus: the user presses the TAB key or clicks another control.
4	Change of control focus. *Id$* is the id of the dialog control gaining focus. *Suppvalue&* contains the numeric id of the control losing focus. A dialog function cannot display a message box or dialog box in response to an action value 4.
5	An idle state. As soon as the dialog box is initialized (*action%* = 1), the dialog function will be continuously called with *action%* = 5 if no other action occurs. If *dialog function* wants to receive this message continuously while the dialog box is idle, return a non-zero value. If 0 (zero) is returned, *action%* = 5 will be passed only while the user is moving the mouse. For this action, *Id$* is equal to empty string ("") and *suppvalue&* is equal to the number of times action 5 was passed before.

If the user clicks a command button or changes a dialog box control, *action%* returns 2 or 3 and *suppvalue&* identifies the control affected. The value returned depends on the type of control or button the user changed or clicked. The following table summarizes the possible values for *suppvalue&*:

Control	***suppvalue&***
List box	Number of the item selected, 0-based.
Check box	1 if selected, 0 if cleared, -1 if filled with gray.
Option button	Number of the option button in the option group, 0-based.
Text box	Number of characters in the text box.
Combo box	The number of the item selected (0-based) for action 2, the number of characters in its text box for action 3.
OK button	1
Cancel button	2

Button Statement

See Also **Example**

Defines a custom push button.

Syntax A **Button** *x , y , dx , dy , text$* [, **.***id*]

Syntax B **PushButton** *x , y , dx , dy , text$* [, **.***id*]

where:	**is:**
x , y	The position of the button relative to the upper left corner of the dialog box.
dx , dy	The width and height of the button.
text$	The name for the push button. If the width of this string is greater than *dx*, trailing characters are truncated.
.id	An optional identifier used by the dialog statements that act on this control.

Comments A *dy* value of 14 typically accommodates text in the system font.

Use this statement to create buttons other than OK and Cancel. Use this statement in conjunction with the **ButtonGroup** statement. The two forms of the statement (**Button** and **PushButton**) are equivalent.

Use the **Button** statement only between a **Begin Dialog** and an **End Dialog** statement.

ButtonGroup Statement

See Also **Example**

Begins the definition of a group of custom buttons for a dialog box.

Syntax **ButtonGroup** *.field*

where:	**is:**
.field	The field to contain the user's custom button selection.

Comments If **ButtonGroup** is used, it must appear before any **PushButton** (or **Button**) statement that creates a custom button (one other than OK or Cancel). Only one **ButtonGroup** statement is allowed within a dialog box definition.

Use the **ButtonGroup** statement only between a **Begin Dialog** and an **End Dialog** statement.

Call Statement

See Also **Example**

Transfers control to a subprogram or function.

Syntax A **Call** *subprogram-name* [**(** *argumentlist* **)**]

Syntax B *subprogram-name argumentlist*

where:	**is:**
subprogram-name	The name of the subroutine or function to call.
argumentlist	The arguments for the subroutine or function (if any).

Comments Use the Call statement to call a subprogram or function written in Basic or to call C procedures in a DLL. These C procedures must be described in a **Declare** statement or be implicit in the application.

If a procedure accepts named arguments, you can use the names to specify the argument and its value. Order is not important. For example, if a procedure is defined as follows:

```
Sub mysub(aa, bb, optional cc, optional dd)
```

the following calls to this procedure are all equivalent:

```
call mysub(1, 2, , 4)
mysub aa := 1, bb := 2, dd :=4
call mysub(aa := 1, dd:=4, bb := 2)
mysub 1, 2, dd:=4
```

Note that the syntax for named arguments is as follows:

argname := *argvalue*

where *argname* is the name for the argument as supplied in the **Sub** or **Function** statement and *argvalue* is the value to assign to the argument when you call it. The advantage to using named arguments is that you do not have to remember the order specified in the procedure's original definition, and if the procedure takes optional arguments, you do not need to include commas (,) for arguments that you leave out.

The procedures that use named arguments include:

1. All functions defined with the **Function** statement.
2. All subprograms defined with the **Sub** statement.
3. All procedures declared with **Declare** statement.
4. Many built-in functions and statements (such as **InputBox**).
5. Some externally registered DLL functions and methods.

Arguments are passed by reference to procedures written in Basic. If you pass a variable to a procedure that modifies its corresponding formal parameter, and you do not want to have your variable modified, enclose the variable in parentheses in the Call statement. This will tell OPEN Script to pass a copy of the variable. Note that this will be less efficient, and should not be done unless necessary.

When a variable is passed to a procedure that expects its argument by reference, the variable must match the exact type of the formal parameter of the function. (This restriction does not apply to expressions or Variants.)

When calling an external DLL procedure, arguments can be passed by value rather than by reference. This is specified either in the **Declare** statement, the **Call** itself, or both, using the **ByVal** keyword. If **ByVal** is specified in the declaration, then the **ByVal** keyword is optional in the call. If present, it must precede the value. If **ByVal** was not specified in the declaration, it is illegal in

the call unless the data type specified in the declaration was **Any**.

CancelButton Statement

See Also **Example**

Sets the position and size of a Cancel button in a dialog box.

Syntax **CancelButton** *x , y , dx , dy* [, **.***id*]

where:	**is:**
x , y	The position of the Cancel button relative to the upper left corner of the dialog box.
dx , dy	The width and height of the button.
.id	An optional identifier for the button.

Comments A *dy* value of 14 can usually accommodate text in the system font.

.Id is used by the dialog statements that act on this control.

If you use the **Dialog** statement to display the dialog box and the user clicks Cancel, the box is removed from the screen and an Error 102 is triggered. If you use the **Dialog** function to display the dialog box, the function will return 0 and no error occurs.

Use the **CancelButton** statement only between a **Begin Dialog** and an **End Dialog** statement.

Caption Statement

See Also **Example**

Defines the title of a dialog box.

Syntax **Caption** *text$*

where:	**is:**
text$	A string expression containing the title of the dialog box.

Comments Use the **Caption** statement only between a **Begin Dialog** and an **End Dialog** statement. The default caption for this statement is OPEN Script.

If no **Caption** statement is specified for the dialog box, a default caption is used.

CCur Function

See Also **Example**

Converts an expression to the data type **Currency**.

Syntax **CCur(** *expression* **)**

where:	**is:**
expression	Any expression that evaluates to a number.

Comments **CCur** accepts any type of *expression.* Numbers that do not fit in the Currency data type result in an "Overflow" error. Strings that cannot be converted result in a "Type Mismatch" error. Variants containing null result in an "Illegal Use of Null" error.

CDbl Function

See Also **Example**

Converts an expression to the data type **Double**.

Syntax **CDbl(** *expression* **)**

where:	**is:**
expression	Any expression that evaluates to a number.

Comments **CDbl** accepts any type of *expression.* Strings that cannot be converted to a double-precision floating point result in a "Type Mismatch" error. Variants containing null result in an "Illegal Use of Null" error.

ChDir Statement

See Also **Example**

Changes the default directory for the specified drive.

Syntax **ChDir** *path$*

where:	**is:**
path$	A string expression identifying the new default directory.

Comments The syntax for *path$* is:

[*drive*:] [\] *directory* [*directory*]

If the drive argument is omitted, **ChDir** changes the default directory on the current drive. The **ChDir** statement does not change the default drive. To change the default drive, use **ChDrive**.

ChDrive Statement

See Also **Example**

Changes the default drive.

Syntax **ChDrive** *drive$*

where:	**is:**
drive$	A string expression designating the new default drive.

Comments This drive must exist and must be within the range specified by the LASTDRIVE statement in the CONFIG.SYS file. If a null argument (" ") is supplied, the default drive remains the same. If the *drive$* argument is a string, **ChDrive** uses the first letter only. If the argument is omitted, an error message is produced. To change the current directory on a drive, use **ChDir**.

CheckBox Statement

See Also **Example**

Creates a check box in a dialog box.

Syntax **CheckBox** *x , y , dx , dy , text$, .field*

where:	**is:**
x , y	The upper left corner coordinates of the check box, relative to the upper left corner of the dialog box.
dx	The sum of the widths of the check box and *text$.*
dy	The height of *text$*.
text$	The title shown to the right of the check box.
.field	The name of the dialog-record field that will hold the current check box setting (0=unchecked, -1=grey, 1=checked).

Comments The *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-height units. (See **Begin Dialog** for more information.)

Because proportional spacing is used, the *dx* argument width will vary with the characters used. To approximate the width, multiply the number of characters in the *text$* field (including blanks and punctuation) by 4 and add 12 for the checkbox.

A *dy* value of 12 is standard, and should cover typical default fonts. If larger fonts are used, the value should be increased. As the *dy* number grows, the checkbox and the accompanying text will move down within the dialog box.

If the width of the *text$* field is greater than *dx*, trailing characters will be truncated. If you want to include underlined characters so that the check box selection can be made from the keyboard, precede the character to be underlined with an ampersand (&).

OPEN Script treats any other value of *.field* the same as a 1. The *.field* argument is also used by the dialog statements that act on this control.

Use the **CheckBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Chr Function

See Also **Example**

Returns the one-character string corresponding to an ANSI code.

Syntax **Chr**[**$**]**(** *charcode* **)**

where:	**is:**
charcode	An integer between 0 and 255.

Comments The dollar sign, "$", in the function name is optional. If specified, the return type is String. If omitted, the function will return a **Variant** of vartype 8 (string).

CInt Function

See Also **Example**

Converts an expression to the data type **Integer** by rounding.

Syntax **CInt(** *expression* **)**

where:	**is:**
expression	Any expression that can evaluate to a number.

Comments After rounding, the resulting number must be within the range of -32767 to 32767, or an error occurs.

Strings that cannot be converted to an integer result in a "Type Mismatch" error. Variants containing null result in an "Illegal Use of Null" error.

Clipboard

Example

The Windows Clipboard can be accessed directly in your program to enable you to get text from and put text into other applications that support the Clipboard.

Syntax **Clipboard.Clear**

Clipboard.GetText()

Clipboard.SetText *string$*

Clipboard.GetFormat()

where:	**is:**
string$	A string or string expression containing the text to send to the Clipboard.

The Clipboard methods supported are as follows:

Method:	**What it does:**
Clear	Clears the contents of the Clipboard.
GetText	Returns a text string from the Clipboard.
SetText	Puts a text string to the Clipboard.
GetFormat	Returns TRUE (non-0) if the format of the item on the Clipboard is text. Otherwise, returns FALSE (0).

Note: Data on the Clipboard is lost when another set of data of the same format is placed on the Clipboard (either through code or a menu command).

CLng Function

See Also **Example**

Converts an expression to the data type **Long** by rounding.

Syntax **CLng(** *expression* **)**

where:	**is:**
expression	Any expression that can evaluate to a number.

Comments After rounding, the resulting number must be within the range of -2,147,483,648 to 2,147,483,647, or an error occurs.

Strings that cannot be converted to a long result in a "Type Mismatch" error. Variants containing null result in an "Illegal Use of Null" error.

Close Statement

See Also **Example**

Closes a file, concluding input/output to that file.

Syntax **Close** [[#] *filenumber%* [, [#] *filenumber%* ...]]

where:	**is:**
filenumber%	An integer expression identifying the file to close.

Comments *Filenumber%* is the number assigned to the file in the **Open** statement and can be preceded by a pound sign (#). If this argument is omitted, all open files are closed. Once a **Close** statement is executed, the association of a file with *filenumber%* is ended, and the file can be reopened with the same or a different file number.

When the **Close** statement is used, the final output buffer is written to the operating system buffer for that file. **Close** frees all buffer space associated with the closed file. Use the **Reset** statement so that the operating system will flush its buffers to disk.

ComboBox Statement

See Also **Example**

Creates a combination text box and list box in a dialog box.

Syntax A **ComboBox** *x , y , dx , dy , text$, .field*

Syntax B **ComboBox** *x , y , dx , dy , stringarray$, .field*

where:	**is:**
x , y	The upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
dx , dy	The width and height of the combo box in which the user enters or selects text.
text$	A string containing the selections for the combo box.
stringarray$	An array of dynamic strings for the selections in the combo box.
.field	The name of the dialog-record field that will hold the text string entered in the text box or chosen from the list box.

Comments The *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-width units. (See **Begin Dialog** for more information.)

The *text$* argument must be defined, using a **Dim** Statement, before the **Begin Dialog** statement is executed. The arguments in the *text$* string are entered as shown in the following example:

dimname = "*listchoice*"+Chr$(9)+"*listchoice*"+Chr$(9)+"*listchoice*"...

The string in the text box will be recorded in the field designated by the *.field* argument when the OK button (or any pushbutton other than Cancel) is pushed. The *field* argument is also used by the dialog statements that act on this control.

Use the **ComboBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Command Function

See Also **Example**

Returns the command line specified when the MAIN subprogram was invoked.

Syntax **Command**[**$**]

Comments After the MAIN subprogram returns, further calls to the **Command** function will yield an empty string.

The dollar sign, "$", in the function name is optional. If specified, the return type is String. If omitted, the function returns a **Variant** of vartype 8 (string).

Const Statement

See Also **Example**

Declares symbolic constants for use in a Basic program.

Syntax **[Global] Const** *constantName* [**As** *type*]= *expression* [,*constantName* [**As** *type*]= *expression*] **...**

where:	**is:**
constantName	The variable name to contain a constant value.
type	The data type of the constant (**Number** or **String**).
expression	Any expression that evaluates to a constant number.

Comments Instead of using the **As** clause, the type of the constant can be specified by using a type character as a suffix (# for numbers, $ for strings) to the *constantName*. If no type character is specified, the type of the *constantName* is derived from the type of the expression.

If **Global** is specified, the constant is validated at module load time. If the constant has already been added to the run-time global area, the constant's type and value are compared to the previous definition, and the load fails if a mismatch is found. This is useful as a mechanism for detecting version mismatches between modules.

Cos Function

See Also **Example**

Returns the cosine of an angle.

Syntax **Cos(** *number* **)**

where:	**is:**
number	An angle in radians.

Comments The return value will be between -1 and 1. The return value is a single-precision number if the angle has a data type **Integer**, **Currency**, or is a single-precision value. The return value will be a double precision value if the angle has a data type **Long**, **Variant** or is a double-precision value.

The angle can be either positive or negative. To convert degrees to radians, multiply by (PI/180). The value of PI is approximately 3.14159.

CreateObject Function

See Also **Example** **Overview**

Creates a new OLE Automation object.

Syntax **CreateObject(** *class* **)**

where:	**is:**
class	The name of the application, a period, and the name of the object to be used.

Comments To create an object, you first must declare an object variable, using **Dim**, and then **Set** the variable equal to the new object, as follows:

Dim *OLEobj* **As Object**

Set *OLEobj* **= CreateObject(**"spoly.cpoly"**)**

To refer to a method or property of the newly created object, use the syntax *objectvar.property* or *objectvar.method*, as follows:

OLEobj.reset

Refer to the documentation provided with your OLE Automation server application for correct application and object names.

CSng Function

See Also **Example**

Converts an expression to the data type **Single**.

Syntax **CSng(** *expression* **)**

where:	**is:**
expression	Any expression that can evaluate to a number.

Comments The *expression* must have a value within the range allowed for the **Single** data type, or an error occurs.

Strings that cannot be converted to an integer result in a "Type Mismatch" error. Variants containing null result in an "Illegal Use of Null" error.

CStr Function

See Also **Example**

Converts an expression to the data type **String**.

Syntax **CStr(** *expression* **)**

where:	**is:**
expression	Any expression that can evaluate to a number.

Comments The **CStr** statement accepts any type of *expression*:

***expression* is:**	**CStr returns:**
Boolean	A String containing "True" or "False".
Date	A String containing a date.
Empty	A zero-length String ("").
Error	A String containing "Error", followed by the error number.
Null	A run-time error.
Other Numeric	A String containing the number.

$CStrings Metacommand [OPEN Script Extension]

See Also **Example**

Tells the compiler to treat a backslash character inside a string (\) as an escape character.

Syntax **'$CStrings** [**Save** | **Restore**]

where:	**means:**
Save	Saves the current $Cstrings setting.
Restore	Restores a previously saved $CStrings setting.

Comments This treatment of a backslash in a string is based on the 'C' language.

Save and **Restore** operate as a stack and allow the user to change the setting for a range of the program without impacting the rest of the program.

The supported special characters are:

Newline (Linefeed)	\n
Horizontal Tab	\t
Vertical Tab	\v
Backspace	\b
Carriage Return	\r
Formfeed	\f
Backslash	\\
Single Quote	\'
Double Quote	\"
Null Character	\0

The instruction "Hello\r World" is the equivalent of "Hello" + Chr$(13)+"World".

In addition, any character can be represented as a 3-digit octal code or a 3-digit hexadecimal code:

Octal Code	\ddd
Hexadecimal Code	\xddd

For both hexadecimal and octal, fewer than 3 characters can be used to specify the code as long as the subsequent character is not a valid (hex or octal) character.

To tell the compiler to return to the default string processing mode, where the backslash character has no special meaning, use the '**$NoCStrings** Metacommand.

CurDir Function

See Also **Example**

Returns the default directory (and drive) for the specified drive.

Syntax **CurDir**[**$**] [**(** *drive$* **)**]

where:	**is:**
drive$	A string expression containing the drive to search.

Comments The drive must exist, and must be within the range specified in the LASTDRIVE statement of the CONFIG.SYS file. If a null argument (" ") is supplied, or if no *drive$* is indicated, the path for the default drive is returned.

The dollar sign, "$", in the function name is optional. If specified, the return type is string. If omitted, the function will return a **Variant** of vartype 8 (string).

To change the current drive, use **ChDrive**. To change the current directory, use **ChDir**.

CVar Function

See Also **Example**

Converts an expression to the data type **Variant**.

Syntax **CVar(** *expression* **)**

where:	**is:**
expression	Any expression that can evaluate to a number.

Comments **CVar** accepts any type of *expression*.

CVar generates the same result as you would get by assigning the *expression* to a **Variant** variable.

CVDate Function

See Also **Example**

Converts an expression to the data type **Variant Date**.

Syntax **CVDate(** *expression* **)**

where:	**is:**
expression	Any expression that can evaluate to a number.

Comments **CVDate** accepts both string and numeric values.

The **CVDate** function returns a **Variant** of vartype 7 (date) that represents a date from January 1, 100 through December 31, 9999. A value of 2 represents January 1, 1900. Times are represented as fractional days.

Date Function

See Also **Example**

Returns a string representing the current date.

Syntax **Date**[**$**]

Comments The **Date** function returns a ten character string.

The dollar sign, "$", in the function name is optional. If specified, the return type is string. If omitted, the function will return a **Variant** of vartype 8 (string).

Date Statement

See Also **Example**

Sets the system date.

Syntax **Date**[**$**] **=** *expression*

where:	**is:**
expression	A string in one of the following forms: *mm-dd-yy* *mm-dd-yyyy* *mm/dd/yy* *mm/dd/yyyy* where *mm* denotes a month (01-12), *dd* denotes a day (01-31), and *yy* or *yyyy* denotes a year (1980-2099).

Comments If the dollar sign, "$", is omitted, *expression* can be a string containing a valid date, a **Variant** of vartype 7 (date), or a **Variant** of vartype 8 (string).

If *expression* is not already a **Variant** of vartype 7 (date), **Date** attempts to convert it to a valid date from January 1, 1980 through December 31, 2099. **Date** uses the Short Date format in the International section of Windows Control Panel to recognize day, month, and year if a string contains three numbers delimited by valid date separators. In addition, **Date** recognizes month names in either full or abbreviated form.

DateSerial Function

See Also **Example**

Returns a date value for year, month, and day specified.

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

where:	**is:**
year%	A year between 100 and 9999, or a numeric expression.
month%	A month between 1 and 12, or a numeric expression.
day%	A day between 1 and 31, or a numeric expression.

Comments The **DateSerial** function returns a **Variant** of vartype 7 (date) that represents a date from January 1, 100 through December 31, 9999, where January 1, 1900 is 2.

A numeric expression can be used for any of the arguments to specify a relative date: a number of days, months, or years before or after a certain date.

DateValue Function

See Also **Example**

Returns a date value for the string specified.

Syntax **DateValue(** *date$* **)**

where:	**is:**
date$	A string representing a valid date.

Comments The **DateValue** function returns a **Variant** of vartype 7 (date) that represents a date from January 1, 100 through December 31, 9999, where January 1, 1900 is 2.

DateValue accepts several different string representations for a date. It makes use of the operating system's international settings for resolving purely numeric dates.

Day Function

See Also **Example**

Returns the day of the month (1-31) of a date-time value.

Syntax **Day(** *date* **)**

where:	**is:**
date	Any expression that can evaluate to a date.

Comments **Day** attempts to convert the input value of *date* to a date value. The return value is a **Variant** of vartype 2 (integer). If the value of *date* is null, a Variant of vartype 1 (null) is returned.

DDEAppReturnCode Function

See Also

Returns a code received from an application on an open dynamic data exchange (DDE) channel.

Syntax **DDEAppReturnCode()**

Comments To open a DDE channel, use **DDEInitiate**. Use **DDEAppReturnCode** to check for error return codes from the server application after using **DDEExecute**, **DDEPoke** or **DDERequest**.

DDEExecute Statement

See Also **Example**

Sends one or more commands to an application via a dynamic-data exchange (DDE) channel.

Syntax **DDEExecute** *channel% , cmd$*

where:	**is:**
channel%	An integer or expression for the channel number of the DDE conversation as returned by **DDEInitiate**.
cmd$	One or more commands recognized by the application.

Comments If *channel* doesn't correspond to an open channel, an error occurs.

You can also use the format described under **SendKeys** to send specific key sequences. If the server application cannot perform the specified command, an error occurs.

In many applications that support DDE, *cmd$* can be one or more statements or functions in the application's macro language. Note that some applications require that each command received through a DDE channel be enclosed in brackets and quotation marks.

You can use a single **DDEExecute** instruction to send more than one command to an application.

Many commands require arguments in the form of strings enclosed in quotation marks. Because quotation marks indicate the beginning and end of a string in OPEN Script, you must use Chr$(34) to include a quotation mark in a command string. For example, the following instruction tells Microsoft Excel to open MYFILE.XLS:

DDEExecute channelno, "[OPEN(" + Chr$(34) + "MYFILE.XLS" + Chr$(34) + ")]"

DDEInitiate Function

See Also **Example**

Opens a dynamic-data exchange (DDE) channel and returns the DDE channel number (1,2, etc.).

Syntax **DDEInitiate(** *appname$, topic$* **)**

where:	**is:**
appname$	A string or expression for the name of the DDE application to talk to.
topic$	A string or expression for the name of a topic recognized by *appname$.*

Comments If **DDEInitiate** is unable to open a channel, it returns zero (0).

Appname$ is usually the name of the application's .EXE file without the .EXE filename extension. If the application is not running, **DDEInitiate** cannot open a channel and returns an error. Use **Shell** to start an application.

Topic$* is usually an open filename. If *appname$ doesn't recognize *topic$*, **DDEInitiate** generates an error. Many applications that support DDE recognize a topic named **System**, which is always available and can be used to find out which other topics are available. For more information on the **System** topic, see **DDERequest**.

The maximum number of channels that can be open simultaneously is determined by the operating system and your system's memory and resources. If you aren't using an open channel, you should conserve resources by closing it using **DDETerminate**.

DDEPoke Statement

See Also **Example**

Sends data to an application on an open dynamic-data exchange (DDE) channel.

Syntax **DDEPoke** *channel%*, *item$*, *data$*

where:	**is:**
channel%	An integer or expression for the open DDE channel number.
item$	A string or expression for the name of an item in the currently opened topic.
data$	A string or expression for the information to send to the topic.

Comments If *channel%* doesn't correspond to an open channel, an error occurs.

When you open a channel to an application using **DDEInitiate**, you also specify a topic, such as a filename, to communicate with. The *item$* is the part of the topic you want to send data to. **DDEPoke** sends data as a text string; you cannot send text in any other format, nor can you send graphics.

If the server application doesn't recognize *item$*, an error occurs.

DDERequest Function

See Also **Example**

Returns data from an application through an open dynamic data exchange (DDE) channel.

Syntax **DDERequest**[**$**] **(** *channel%*, *item$* **)**

where:	**is:**
channel%	An integer or expression for the open DDE channel number.
item$	A string or expression for the name of an item in the currently opened topic to get information about.

Comments If *channel%* doesn't correspond to an open channel, an error occurs.

If the server application doesn't recognize *item$*, an error occurs.

If **DDERequest** is unsuccessful, it returns an empty string ("").

When you open a channel to an application using **DDEInitiate**, you also specify a topic, such as a filename, to communicate with. The *item$* is the part of the topic whose contents you are requesting.

DDERequest returns data as a text string. Data in any other format cannot be transferred, nor can graphics.

Many applications that support DDE recognize a topic named **System**. Three standard items in the **System** topic are described in the following table:

Item:	**Returns:**
SysItems	A list of all items in the **System** topic
Topics	A list of available topics
Formats	A list of all the Clipboard formats supported

DDETerminate Statement

See Also **Example**

Closes the specified dynamic data exchange (DDE) channel.

Syntax **DDETerminate** *channel%*

where:	**is:**
channel%	An integer or expression for the open DDE channel number.

Comments To free system resources, you should close channels you aren't using. If *channel%* doesn't correspond to an open channel, an error occurs.

Declare Statement

See Also **Example**

Declares a procedure in a module or dynamic link library (DLL).

Syntax A **Declare Sub** *name* **[** *libSpecification* **]** **[(** *parameter* **[As** *type* **])]**

Syntax B **Declare Function** *name* **[** *libSpecification* **]** **[(** *parameter* **[As** *type* **])] [As** *functype* **]**

where:	**is:**
name	The subprogram or function procedure to declare.
libSpecification	The location of the procedure (module or DLL).
parameter	The arguments to pass to the procedure, separated by commas.
type	The type for the arguments.
functype	The type of the return value for a function procedure.

Comments A **Sub** procedure does not return a value. A **Function** procedure returns a value, and can be used in an expression. To specify the data type for the return value of a function, end the Function name with a type character or use the **As** *functype* clause shown above. If no type is provided, the function defaults to data type **Variant**.

If the *libSpecification* is of the format:

BasicLib *libName* **[Alias "***aliasname***"]**

the procedure is in another Basic module named *libName*. The **Alias** keyword specifies that the procedure in *libName* is called *aliasname*. The other module will be loaded on demand whenever the procedure is called. OPEN Script will not automatically unload modules that are loaded in this fashion. OPEN Script will detect errors of mis-declaration.

If the *libSpecification* is of the format:

Lib *libName* **[Alias ["]***ordinal***["]]** or

Lib *libName* **[Alias "***aliasname***"]**

the procedure is in a Dynamic Link Library (DLL) named *libName*. The *ordinal* argument specifies the ordinal number of the procedure within the external DLL. Alternatively, *aliasname* specifies the name of the procedure within the external DLL. If neither *ordinal* nor *aliasname* is specified, the DLL function is accessed by name. It is recommended that the *ordinal* be used whenever possible, since accessing functions by name might cause the module to load more slowly.

A forward declaration is needed only when a procedure in the current module is referenced before it is defined. In this case, the **BasicLib**, **Lib** and **Alias** clauses are not used.

The data type of a parameter can be specified by using a type character or by using the **As** clause. Record parameters are declared by using an **As** clause and a *type* that has previously been defined using the **Type** statement. Array parameters are indicated by using empty parentheses after the *parameter*: array dimensions are not specified in the **Declare** statement.

External DLL procedures are called with the PASCAL calling convention (the actual arguments are pushed on the stack from left to right). By default, the actual arguments are passed by Far reference. For external DLL procedures, there are two additional keywords, **ByVal** and **Any**, that can be used in the parameter list.

When **ByVal** is used, it must be specified before the parameter it modifies.

When applied to numeric data types, **ByVal** indicates that the parameter is passed by value, not by reference. When applied to string parameters, **ByVal** indicates that the string is passed by Far pointer to the string data. By default, strings are passed by Far pointer to a string descriptor.

Any can be used as a type specification, and permits a call to the procedure to pass a value of any datatype. When **Any** is used, type checking on the actual argument used in calls to the procedure is disabled (although other arguments not declared as type **Any** are fully type-safe). The actual argument is passed by Far reference, unless **ByVal** is specified, in which case the actual value is placed on the stack (or a pointer to the string in the case of string data). **ByVal** can also be used in the call. It is the external DLL procedure's responsibility to determine the type and size of the passed-in value.

When an empty string ("") is passed **ByVal** to an external procedure, the external procedure will receive a valid (non-NULL) pointer to a character of 0. To send a NULL pointer, **Declare** the procedure argument as **ByVal As Any**, and call the procedure with an argument of **0**.

You can also declare a function in one module and then call it from another module. To call such an external function, you declare the function in one module and then $Include that module in the module from which you call the function.

Def*type* Statement

See Also **Example**

Specifies the default data type for one or more variables.

Syntax

DefCur *varTypeLetters*
DefInt *varTypeLetters*
DefLng *varTypeLetters*
DefSng *varTypeLetters*
DefDbl *varTypeLetters*
DefStr *varTypeLetters*
DefVar *varTypeLetters*

where:	**is:**
varTypeLetters	A first letter of the variable name to use.

Comments

VarTypeLetters can be a single letter, a comma-separated list of letters, or a range of letters. For example, a-d indicates the letters a, b, c and d.

The case of the letters is not important, even in a letter range. The letter range a-z is treated as a special case: it denotes all alpha characters, including the international characters.

The **Def*type*** statement affects only the module in which it is specified. It must precede any variable definition within the module.

Variables defined using the Global or Dim can override the **Def*type*** statement by using an **As** clause or a type character.

Dialog Function

See Also **Example** **Overview**

Displays a dialog box and returns a number for the button selected (-1= OK, 0=Cancel).

Syntax **Dialog (** *recordName* **)**

where:	**is:**
recordName	A variable name declared as a dialog box record.

Comments If the dialog box contains additional command buttons (for example, Help), the **Dialog** function returns a number greater than 0. 1 corresponds to the first command button, 2 to the second, and so on.

The dialog box *recordName* must have been declared using the **Dim** statement with the **As** parameter followed by a dialog box definition name. This name comes from the name argument used in the Begin Dialog statement.

To trap a user's selections within a dialog box, you must create a function and specify it as the last argument to the Begin Dialog statement. See **Begin Dialog** for more information.

The **Dialog** function does not return until the dialog box is closed.

Dialog Statement

See Also **Example** **Overview**

Displays a dialog box.

Syntax **Dialog** *recordName*

where:	**is:**
recordName	A variable name declared as a dialog box record.

Comments The dialog box *recordName* must have been declared using the **Dim** statement with the **As** parameter followed by a dialog box definition name. This name comes from the name argument used in the **Begin Dialog** statement.

If the user exits the dialog box by pushing the Cancel button, the run-time error 102 is triggered, which can be trapped using **On Error**.

To trap a user's selections within a dialog box, you must create a function and specify it as the last argument to the Begin Dialog statement. See **Begin Dialog** for more information.

The **Dialog** statement does not return until the dialog box is closed.

Dim Statement

See Also **Example** **Overview**

Declares variables for use in a Basic program.

Syntax **Dim** [**Shared**] *variableName* [**As** [**New**] *type*] [,*variableName* [**As** [**New**] *type*]] **...**

where:	**is:**
variableName	The name of the variable to declare.
type	The data type of the variable.

Comments *VariableName* must begin with a letter and contain only letters, numbers and underscores. A name can also be delimited by brackets, and any character can be used inside the brackets, except for other brackets.

Dim *my_1st_variable* **As String**
Dim [one long and strange! variable name] **As String**

If the **As** clause is not used, the *type* of the variable can be specified by using a type character as a suffix to *variableName*. The two different type-specification methods can be intermixed in a single **Dim** statement (although not on the same variable).

Basic is a strongly typed language: all variables must be given a data type or they will be automatically assigned the data type **Variant**. The available data types are:

Arrays

Numbers

Objects

Records

Strings

Variants

Variables can be shared across modules. A variable declared inside a procedure has scope Local to that procedure. A variable declared outside a procedure has scope Local to the module. If you declare a variable with the same name as a module variable, the module variable is not accessible. See the **Global** statement for details.

The **Shared** keyword is included for backward compatibility with older versions of Basic. It is not allowed in **Dim** statements inside a procedure. It has no effect.

It is considered good programming practice to declare all variables. To force all variables to be explicitly declared use the **Option Explicit** statement. It is also recommended that you place all procedure-level **Dim** statements at the beginning of the procedure.

Regardless of which mechanism you use to declare a variable, you can choose to use or omit the type character when referring to the variable in the rest of your program. The type suffix is not considered part of the variable name.

Arrays

The available data types for arrays are: numbers, strings, variants, objects and records. Arrays of arrays, dialog box records, and objects are not supported.

Array variables are declared by including a subscript list as part of the *variableName*. The syntax to use for *variableName* is:

Dim *variable*([*subscriptRange*, ...]) **As** *typeName* or
Dim *variable_with_suffix*([*subscriptRange*, ...])

where *subscriptRange* is of the format:

[*startSubscript* **To**] *endSubscript*

If *startSubscript* is not specified, 0 is used as the default. The **Option Base** statement can be used to change the default.

Both the *startSubscript* and the *endSubscript* are valid subscripts for the array. The maximum number of subscripts that can be specified in an array definition is 60. The maximum total size for an array is only limited by the amount of memory available.

If no *subscriptRange* is specified for an array, the array is declared as a dynamic array. In this case, the **ReDim** statement must be used to specify the dimensions of the array before the array can be used.

Numbers

Numeric variables can be declared using the **As** clause and one of the following numeric types: **Currency**, **Integer**, **Long**, **Single**, **Double**. Numeric variables can also be declared by including a type character as a suffix to the name. Numeric variables are initialized to 0.

Objects

Object variables are declared using an **As** clause and a *typeName* of **Object**. Object variables can be **Set** to refer to an object, and then used to access members and methods of the object using dot notation.

```
Dim OLEobj   As Object
Set OLEobj = CreateObject("spoly.cpoly")
OLEobj.reset
```

An object can be declared as **New** for some classes. In such instances, the object variable does not need to be **Set**; a new object will be allocated when the variable is used. Note: The class **Object** does not support the **New** operator.

```
Dim variableName As New className
variableName.methodName
```

Records

Record variables are declared by using an **As** clause and a *typeName* that has been defined previously using the **Type** statement. The syntax to use is:

Dim *variableName* **As** *typeName*

Records are made up of a collection of data elements called fields. These fields can be of any numeric, string, Variant, or previously-defined record type. See **Type** for details on accessing fields within a record.

You can also use the **Dim** statement to declare a dialog box record. In this case, *type* is specified as *dialogName*, where *dialogName* matches a dialog box name previously defined using **Begin Dialog**. The dialog record variable can then be used in a **Dialog** statement.

Dialog box records have the same behavior as regular records; they differ only in the way they are defined. Some applications might provide a number of predefined dialog boxes.

Strings

OPEN Script supports two types of strings: fixed-length and dynamic. Fixed-length strings are declared with a specific length (between 1 and 32767) and cannot be changed later. Use the following syntax to declare a fixed-length string:

Dim *variableName* **As String*** *length*

Dynamic strings have no declared length, and can vary in length from 0 to 32,767. The initial length for a dynamic string is 0. Use the following syntax to declare a dynamic string:

Dim *variableName$* or
Dim *variableName* **As String**

When initialized, fixed-length strings are filled with zeros. Dynamic strings are initialized as zero-length strings.

Variants

Declare variables as Variants when the type of the variable is not known at the start of, or might change during, the procedure. For example, a Variant is useful for holding input from a user when valid input can be either text or numbers. Use the following syntax to declare a Variant:

Dim *variableName* or
Dim *variableName* **As Variant**

Variant variables are initialized to vartype **Empty**.

Dir Function

See Also **Example**

Returns a filename that matches the specified pattern.

Syntax **Dir**[**$**] [(*pathname$* [,*attributes%*)]

where:	**is:**
pathname$	A string expression identifying a path or filename.
attributes%	An integer expression specifying the file attributes to select.

Comments *Pathname$* can include a drive specification and wildcard characters ('?' and '*'). **Dir** returns the first filename that matches the *pathname$* argument. To retrieve additional matching filenames, call the **Dir** function again, omitting the *pathname$* and *attributes%* arguments. If no file is found, an empty string ("") is returned.

The default value for *attributes%* is 0. In this case, **Dir** returns only files without directory, hidden, system, or volume label attributes set.

Here are the possible values for *attributes%*:

Value	**Meaning**
0	return normal files
2	add hidden files
4	add system files
8	return volume label
16	add directories

The values in the table can be added together to select multiple attributes. For example, to list hidden and system files in addition to normal files set *attributes%* to 6 (6=2+4).

If *attributes%* is set to 8, the **Dir** function returns the volume label of the drive specified in the *pathname$*, or of the current drive if drive is not explicitly specified. If volume label attribute is set, all other attributes are ignored.

The dollar sign, "$", in the function name is optional. If specified the return type is string. If omitted the function will return a **Variant** of vartype 8 (string).

DlgControlID Function

See Also **Example** **Overview**

Returns the numeric ID of a dialog box control with the specified *Id$* in the active dialog box.

Syntax **DlgControlID (** *Id$* **)**

where:	**is:**
Id$	The string ID for a dialog control.

Comments The **DlgControlID** function translates a string *Id$* into a numeric ID. This function can only be used from within a dialog box function. The value of the numeric identifier is based on the position of the dialog box control with the dialog; it will be 0 (zero) for the first control, 1 (one) for the second control, and so on.

Given the following example, the statement DlgControlID(doGo) will return the value 1.

```
Begin Dialog newdlg 200, 200
    PushButton 40, 50, 80, 20, "&Stop", .doStop
    PushButton 40, 80, 80, 20, "&Go", .doGo
End Dialog
```

The advantage of using a dialog box controls numeric ID is that it is more efficient, and numeric values can sometimes be more easily manipulated.

Rearranging the order of a control within a dialog box will change its numeric ID. For example, if a PushButton control originally had a numeric value of 1, and a textbox control is added before it, the PushButton controls new numeric value will be 2. This is shown in the following example:

```
CheckBox   40, 110, 80, 20, "CheckBox", .CheckBox1
TextBox    40, 20, 80, 20, .TextBox1       this is the new added control
PushButton 40, 80, 80, 20, "&Go", .doGo
```

The string IDs come from the last argument in the dialog definition statement that created the dialog control, such as the **TextBox** or **ComboBox** statements. The string ID does not include the period (.) and is case-sensitive.

Use **DlgControlID** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgEnable Function

See Also **Example** **Overview**

Returns the enable state for the specified dialog control (-1=enabled, 0=disabled).

Syntax **DlgEnable (** *Id* **)**

where:	**is:**
Id	The control ID for the dialog control.

Comments If a dialog box control is enabled, it is accessible to the user. You might want to disable a control if its use depends on the selection of other controls.

Use the **DlgControlID** function to find the numeric ID for a dialog control, based on its string identifier.

Use **DlgEnable** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgEnable Statement

Enables, disables, or toggles the state of the specified dialog control.

Syntax **DlgEnable** *Id* [, *mode*]

where:	**is:**
Id	The control ID for the dialog control to change.
mode	An integer representing the enable state (1=enable, 0=disable)

Comments If *mode* is omitted, the **DlgEnable** toggles the state of the dialog control specified by *Id*. If a dialog box control is enabled, it is accessible to the user. You might want to disable a control if its use depends on the selection of other controls.

Use the **DlgControlID** function to find the numeric ID for a dialog control, based on its string identifier. The string IDs come from the last argument in the dialog definition statement that created the dialog control, such as the **TextBox** or **ComboBox** statements.

Use **DlgEnable** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgEnd Statement

See Also **Example** **Overview**

Closes the active dialog box.

Syntax **DlgEnd** *exitCode*

where:	**is:**
exitCode	The return value after closing the dialog box (-1=OK, 0=Cancel).

Comments *ExitCode* contains a return value only if the dialog box was displayed using the **Dialog** function. That is, if you used the **Dialog** statement, *exitCode* is ignored.

If the dialog box contains additional command buttons (for example, Help), the **Dialog** function returns a number greater than 0. 1 corresponds to the first command button, 2 to the second, and so on.

Use **DlgEnd** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgFocus Function

Returns the control ID of the dialog control having the input focus.

Syntax **DlgFocus**[**$**]**( )**

Comments A control has focus when it is active and responds to keyboard input.

Use **DlgFocus** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgFocus Statement

See Also **Example** **Overview**

Sets the focus for the specified dialog control.

Syntax **DlgFocus** *Id*

where:	**is:**
Id	The control ID for the dialog control to make active.

Comments Use the **DlgControlID** function to find the numeric ID for a dialog control, based on its string identifier. The string IDs come from the last argument in the dialog definition statement that created the dialog control, such as the **TextBox** or **ComboBox** statements.

Use **DlgFocus** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgListBoxArray Function

See Also **Example** **Overview**

Returns the number of elements in a list or combo box.

Syntax **DlgListBoxArray (** *Id*[, *Array$*] **)**

where:	**is:**
Id	The control ID for the list or combo box.
Array$	The entries in the list box or combo box returned.

Comments *Array$* is a one-dimensional array of dynamic strings. If *array$* is dynamic, its size is changed to match the number of strings in the list or combo box. If *array$* is not dynamic and it is too small, an error occurs. If *array$* is omitted, the function returns the number of entries in the specified dialog control.

Use the **DlgControlID** function to find the numeric ID for a dialog control, based on its string identifier. The string IDs come from the last argument in the dialog definition statement that created the dialog control, such as the **TextBox** or **ComboBox** statements.

Use **DlgListBoxArray** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgListBoxArray Statement

See Also **Example** **Overview**

Fills a list or combo box with an array of strings.

Syntax **DlgListBoxArray** *Id*, *Array$*

where:	**is:**
Id	The control ID for the list or combo box.
Array$	The entries for the list box or combo box.

Comments *Array$* has to be a one-dimensional array of dynamic strings. One entry appears in the list box for each element of the array. If the number of strings changes depending on other selections made in the dialog box, you should use a dynamic array and **ReDim** the size of the array whenever it changes.

Use **DlgListBoxArray** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgSetPicture Statement

See Also **Example** **Overview**

Changes the picture in a picture dialog control for the current dialog box.

Syntax **DlgSetPicture** *Id*, *filename$, type*

where:	**is:**
Id	The control ID for the picture dialog control.
filename$	The name of the bitmap file (.BMP) to use.
type	An integer representing the location of the file (0=*filename$*, 3=Clipboard)

Comments Use the **DlgControlID** function to find the numeric ID for a dialog control, based on its string identifier. The string IDs come from the last argument in the dialog definition statement that created the dialog control, such as the **TextBox** or **ComboBox** statements.

Use **DlgListBoxArray** only while a dialog box is running. See the **Begin Dialog** statement for more information.

See the **Picture** statement for more information about displaying pictures in dialog boxes.

DlgText Function

See Also **Example** **Overview**

Returns the text associated with a dialog control for the current dialog box.

Syntax **DlgText**[**$**] **(** *Id* **)**

where:	**is:**
Id	The control ID for a dialog control.

Comments If the control is a text box or a combo box, **DlgText** function returns the text that appears in the text box. If it is a list box, the function returns its current selection. If it is a text box, **DlgText** returns the text. If the control is a command button, option button, option group, or a check box, the function returns its label.

Use **DlgText** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgText Statement

See Also **Example** **Overview**

Changes the text associated with a dialog control for the current dialog box.

Syntax **DlgText** *Id*, *text$*

where:	**is:**
Id	The control ID for a dialog control.
text$	The text to use for the dialog control.

Comments If the dialog control is a text box or a combo box, **DlgText** sets the text that appears in the text box. If it is a list box, a string equal to *text$* or beginning with *text$* is selected. If the dialog control is a text control, **DlgText** sets it to *text$*. If the dialog control is a command button, option button, option group, or a check box, the statement sets its label.

The **DlgText** statement does not change the identifier associated with the control.

Use **DlgText** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgValue Function

See Also **Example** **Overview**

Returns a numeric value for the state of a dialog control for the current dialog box.

Syntax **DlgValue (** *Id* **)**

where:	**is:**
Id	The control ID for a dialog control.

Comments The values returned depend on the type of dialog control:

Control	**Value Returned**
Checkbox	1 = Selected, 0=Cleared, -1=Grayed
Option Group	0 = 1st button selected, 1 = 2nd button selected, etc.
Listbox	0 = 1st item, 1= 2nd item, etc.
Combobox	0 = 1st item, 1 = 2nd item, etc.

Text, Textbox, Button Error occurs

Use **DlgValue** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgValue Statement

See Also **Example** **Overview**

Changes the value associated with the dialog control for the current dialog box.

Syntax **DlgValue** *Id*, *value%*

where:	**is:**
Id	The control ID for a dialog control.
value%	The new value for the dialog control.

Comments The values you use to set the control depend on the type of the control:

Control	**Value Returned**
Checkbox	1 = Select, 0=Clear, -1=Gray.
Option Group	0 = Select 1st button, 1 = Select 2nd button.
Listbox	0 = Select 1st item, 1= Select 2nd item, etc.
Combobox	0 = Select 1st item, 1 = Select 2nd item, etc.
Text, Textbox, Button	Error occurs

Use **DlgValue** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgVisible Function

See Also **Example** **Overview**

Returns -1 if a dialog control is visible, 0 if it is hidden.

Syntax **DlgVisible (** *Id* **)**

where:	**is:**
Id	The control ID for a dialog control.

Use **DlgVisible** only while a dialog box is running. See the **Begin Dialog** statement for more information.

DlgVisible Statement

See Also **Example** **Overview**

Hides or displays a dialog control for the current dialog box.

Syntax **DlgVisible** *Id* [, *mode*]

where:	**is:**
Id	The control ID for a dialog control.
mode	Value to use to set the dialog control state: 1 = Display a previously hidden control. 0 = Hide the control.

Comments If you omit the *mode*, the dialog box state is toggled between visible and hidden.

Use **DlgVisible** only while a dialog box is running. See the **Begin Dialog** statement for more information.

Do...Loop Statement

See Also **Example**

Repeats a series of program lines as long as (or until) an expression is TRUE.

Syntax A

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

Syntax B

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

where:	**is:**
Condition	Any expression that evaluates to TRUE (nonzero) or FALSE (0).
statementblock(s)	Program lines to repeat while (or until) *condition* is TRUE.

Comments When an **Exit Do** statement is executed, control goes to the statement after the Loop statement. When used within a nested loop, an **Exit Do** statement moves control out of the immediately enclosing loop.

DoEvents Statement

See Also **Example**

Yields execution to Windows for processing operating system events.

Syntax **DoEvents**

Comments **DoEvents** does not return until Windows has finished processing all events in the queue and all keys sent by SendKeys statement..

DoEvents should not be used if other tasks can interact with the running program in unforeseen ways. Since OPEN Script yields control to the operating system at regular intervals, **DoEvents** should only be used to force OPEN Script to allow other applications to run at a known point in the program.

DropComboBox Statement

See Also **Example**

Creates a combination of a drop-down list box and a text box.

Syntax A **DropComboBox** *x , y , dx , dy , text$, .field*

Syntax B **DropComboBox** *x , y , dx , dy , stringarray$() , .field*

where:	**is:**
x , y	The upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
dx , dy	The width and height of the combo box in which the user enters or selects text.
text$	A string containing the selections for the combo box.
stringarray$	An array of dynamic strings for the selections in the combo box.
.field	The name of the dialog-record field that will hold the text string entered in the text box or chosen from the list box.

Comments The *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-width units. (See **Begin Dialog** for more information.)

The *text$* argument must be defined, using a **Dim** Statement, before the **Begin Dialog** statement is executed. The arguments in the *text$* string are entered as shown in the following example:

dimname = "*listchoice*"+Chr$(9)+"*listchoice*"+Chr$(9)+"*listchoice*"...

The string in the text box will be recorded in the field designated by the *.field* argument when the OK button (or any pushbutton other than Cancel) is pushed. The *field* argument is also used by the dialog statements that act on this control.

You use a drop combo box when you want the user to be able to edit the contents of the list box (such as filenames or their paths). You use a drop list box when the items in the list should remain unchanged.

Use the **DropComboBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

DropListBox Statement

See Also **Example**

Creates a drop-down list of choices.

Syntax A **DropListBox** *x , y , dx , dy , text$, .field*

Syntax B **DropListBox** *x , y , dx , dy , stringarray$() , .field*

where:	**is:**
x , y	The upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
dx , dy	The width and height of the combo box in which the user enters or selects text.
text$	A string containing the selections for the combo box.
stringarray$	An array of dynamic strings for the selections in the combo box.
.field	The name of the dialog-record field that will hold the text string entered in the text box or chosen from the list box.

Comments The *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-width units. (See **Begin Dialog** for more information.)

The *text$* argument must be defined, using a **Dim** Statement, before the **Begin Dialog** statement is executed. The arguments in the *text$* string are entered as shown in the following example:

dimname = "*listchoice*"+Chr$(9)+"*listchoice*"+Chr$(9)+"*listchoice*"...

The string in the text box will be recorded in the field designated by the *.field* argument when the OK button (or any pushbutton other than Cancel) is pushed. The *field* argument is also used by the dialog statements that act on this control.

A drop list box is different from a list box. The drop list box only displays its list when the user selects it; the list box also displays its entire list in the dialog box.

Use the **DropListBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Environ Function

Example

Returns the string setting for a keyword in the operating system's environment table.

Syntax A **Environ**[**$**]**(** *environment-string$* **)**

Syntax B **Environ**[**$**]**(** *numeric expression%* **)**

where:	**is:**
Environment-string$	The name of a keyword in the operating system environment.
Numeric expression%	A number for the position of the string in the environment table. (1st, 2nd, 3rd, etc.)

Comments If you use the *environment-string$* parameter, enter it in uppercase, or **Environ** returns a null string (""). The return value for Syntax A is the string associated with the keyword requested.

If you use the *numeric expression%* parameter, the numeric expression is automatically rounded to a whole number, if necessary. The return value for Syntax B is a string in the form "keyword=value."

Environ returns a null string if the specified argument cannot be found.

The dollar sign, "$", in the function name is optional. If specified the return type is string. If omitted the function will return a **Variant** of vartype 8 (string).

Eof Function

See Also **Example**

Returns the value -1 if the end of the specified open file has been reached, 0 otherwise.

Syntax **Eof(** *filenumber%* **)**

where:	**is:**
filenumber%	An integer expression identifying the open file to use.

Comments See the **Open** statement for more information about assigning numbers to files when they are opened.

Erase Statement

See Also **Example**

Reinitializes the contents of a fixed array or frees the storage associated with a dynamic array.

Syntax **Erase** *Array* [, *Array*]

where:	**is:**
Array	The name of the array variable to re-initialize.

Comments The effect of using **Erase** on the elements of a fixed array varies with the type of the element:

Element Type	**Erase Effect**
numeric	Each element set to zero.
variable length string	Each element set to zero length string.
fixed length string	Each element's string is filled with zeros.
Variant	Each element set to **Empty.**
user-defined type	Members of each element are cleared as if the members were array elements, i.e. numeric members have their value set to zero, etc.
object	Each element is set to the special value **Nothing.**

Erl Function

See Also **Example** **Overview**

Returns the line number where an error was trapped.

Syntax **Erl**

Comments If you use a **Resume** or **On Error** statement after **Erl**, the return value for **Erl** is reset to 0. To maintain the value of the line number returned by **Erl**, assign it to a variable.

The value of the **Erl** function can be set indirectly through the **Error** statement.

Err Function

See Also **Example** **Overview**

Returns the run-time error code for the last error trapped.

Syntax **Err**

Comments If you use a **Resume** or **On Error** statement after **Erl**, the return value for **Err** is reset to 0. To maintain the value of the line number returned by **Erl**, assign it to a variable.

The value of the **Err** function can be set directly through the **Err** statement, and indirectly through the **Error** statement.

For a list of trappable errors, see **Trappable Errors**.

Err Statement

See Also **Example** **Overview**

Sets a run-time error code.

Syntax **Err** = *n%*

where:	**is:**
n%	An integer expression for the error code (between 1 and 32,767) or 0 for no run-time error.

Comments The **Err** statement is used to send error information between procedures.

Error Function

See Also **Example** **Overview**

Returns the error message that corresponds to the specified error code.

Syntax **Error**[**$**] [**(** *errornumber%* **)**]

where:	is:
errornumber%	An integer between 1 and 32,767 for the error code.

Comments If this argument is omitted, OPEN Script returns the error message for the run-time error that has occurred most recently.

If no error message is found to match the errorcode, "" (a null string) is returned.

The dollar sign, "$", in the function name is optional. If specified the return type is string. If omitted the function will return a **Variant** of vartype 8 (string).

For a list of trappable errors, see **Trappable Errors**.

Error Statement

See Also **Example** **Overview**

Simulates the occurrence of an OPEN Script or user-defined error.

Syntax **Error** *errornumber%*

where:	is:
errornumber%	An integer between 1 and 32,767 for the error code.

Comments If an *errornumber%* is one that OPEN Script already uses, the **Error** statement will simulate an occurrence of that error.

User-defined error codes should employ values greater than those used for standard OPEN Script error codes. To help ensure that non-OPEN Script error codes are chosen, user-defined codes should work down from 32,767.

If an **Error** statement is executed, and there is no error-handling routine enabled, OPEN Script produces an error message and halts program execution. If an **Error** statement specifies an error code not used by OPEN Script, the message "User-defined error" is displayed.

Exit Statement

See Also **Example**

Terminates Loop statements or transfers control to a calling procedure.

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

Comments Use **Exit Do** inside a **Do...Loop** statement. Use **Exit For** inside a **For...Next** statement. When the **Exit** statement is executed, control transfers to the statement after the Loop or Next statement. When used within a nested loop, an Exit statement moves control out of the immediately enclosing loop.

Use **Exit Function** inside a **Function...End Function** procedure. Use **Exit Sub** inside a **Sub...End Sub** procedure.

Exp Function

See Also **Example**

Returns the value *e* (the base of natural logarithms) raised to a power.

Syntax **Exp(** *number* **)**

where:	**is:**
number	The exponent value for *e*.

Comments If the variable to contain the return value has a data type **Integer**, **Currency**, or **Single**, the return value is a single-precision value. If the variable has a date type of **Long**, **Variant**, or **Double**, the value returned is a double-precision number.

The constant *e* is approximately 2.718282.

FileAttr Function

See Also **Example**

Returns the file mode or the operating system handle for the open file.

Syntax **FileAttr(** *filenumber% , returntype* **)**

where:	**is:**
filenumber%	An integer expression identifying the open file to use.
returntype	1=Return file mode, 2=Return operating system handle

Comments The argument *filenumber%* is the number used in the **Open** statement to open the file.

The following table lists the return values and corresponding file modes if *returntype* is 1:

Value	**Mode**
1	**Input**
2	**Output**
8	**Append**

FileCopy Statement

See Also **Example**

Copies a file.

Syntax **FileCopy** *source$, destination$*

where:	**is:**
source$	A string expression for the name (and path) of the file to copy.
destination$	A string expression for the name (and path) for the copied file.

Comments Wildcards (* or ?) are not allowed for either the *source$* or *destination$*. The *source$* file cannot be copied if it is opened by OPEN Script for anything other than **Read** access.

FileDateTime Function

See Also **Example**

Returns the last modification date and time for the specified file.

Syntax **FileDateTime**(*pathname$*)

where:	**is:**
pathname$	A string expression for the name of the file to query.

Comments *Pathname$* can contain path and disk information, but cannot include wildcards (* and ?).

FileLen Function

See Also **Example**

Returns the length of the specified file.

Syntax **FileLen**(*pathname$*)

where:	**is:**
pathname$	A string expression that contains the name of the file to query.

Comments *Pathname$* can contain path and disk information, but cannot include wildcards (* and ?).

If the specified file is open, **FileLen** returns the length of the file before it was opened.

Fix Function

See Also **Example**

Returns the integer part of a number.

Syntax **Fix (** *number* **)**

where:	**is:**
number	Any valid numeric expression.

Comments The return value's data type matches the type of the numeric expression. This includes **Variant** expressions, unless the numeric expression is a string (vartype 8) that evaluates to a number, in which case the data type for its return value is vartype 5 (double). If the numeric expression is vartype 0 (empty), the data type for the return value is vartype 3 (long).

For both positive and negative *numbers,* **Fix** removes the fractional part of the expression and returns the integer part only. For example, **Fix** (6.2) returns 6; **Fix** (-6.2) returns -6.

For...Next Statement

See Also **Example**

Repeats a series of program lines a fixed number of times.

Syntax

For *counter* = *start* **TO** *end* [**STEP** *increment*]
[*statementblock*]
[**Exit For**]
[*statementblock*]
Next [*counter*]

where:	**is:**
counter	A numeric variable for the loop counter.
start	The beginning value of the counter.
end	The ending value of the counter.
increment	The amount by which the counter is changed each time the loop is run. (The default is one.)
statementblock	Basic functions, statements, or methods to be executed.

Comments

The *start* and *end* values must be consistent with *increment*: If *end* is greater than *start*, *increment* must be positive. If *end* is less than *start*, *increment* must be negative. OPEN Script compares the sign of (*start-end*) with the sign of *increment.* If the signs are the same, and *end* does not equal *start*, the **For...Next** loop is started. If not, the loop is omitted in its entirety.

With a **For...Next** loop, the program lines following the **For** statement are executed until the **Next** statement is encountered. At this point, the **Step** amount is added to the *counter* and compared with the final value, *end*. If the beginning and ending values are the same, the loop executes once, regardless of the **Step** value. Otherwise, the **Step** value controls the loop as follows:

Step Value	**Loop Execution**
Positive	If *counter* is less than or equal to *end*, the **Step** value is added to *counter*. Control returns to the statement after the **For** statement and the process repeats. If *counter* is greater than *end*, the loop is exited; execution resumes with the statement following the **Next** statement.
Negative	The loop repeats until *counter* is less than *end*.
Zero	The loop repeats indefinitely.

Within the loop, the value of the *counter* should not be changed, as changing the *counter* will make programs more difficult to maintain and debug.

For...Next loops can be nested within one another. Each nested loop should be given a unique variable name as its *counter*. The **Next** statement for the inside loop must appear before the **Next** statement for the outside loop. The **Exit For** statement can be used as an alternative exit from **For...Next** loops.

If the variable is left out of a **Next** statement, the **Next** statement will match the most recent **For** statement. If a **Next** statement occurs prior to its corresponding **For** statement, OPEN Script will return an error message.

Multiple consecutive **Next** statements can be merged together. If this is done, the counters must appear with the innermost counter first and the outermost counter last. For example:

For i = 1 **To** 10
[*statementblock*]
For j = 1 **To** 5

[*statementblock*]

Next j, i

Format Function

See Also **Example**

Returns a formatted string of an expression based on a given format.

Syntax **Format**[**$**]**(** *expression* [, *format*] **)**

where:	**is:**
expression	The value to be formatted. It can be a number, Variant, or string.
format	A string expression representing the format to use. Select one of the topics below for a detailed description of format strings.

Comments **Format** formats the *expression* as a number, date, time, or string depending upon the *format* argument. The dollar sign, "$", in the function name is optional. If specified the return type is string. If omitted the function will return a **Variant** of vartype 8 (string). As with any string, you must enclose the *format* argument in quotation marks ("").

Numeric values are formatted as either numbers or date/times. If a numeric expression is supplied and the *format* argument is omitted or null, the number will be converted to a string without any special formatting.

Both numeric values and Variants can be formatted as dates. When formatting numeric values as dates, the value is interpreted according the standard Basic date encoding scheme. The base date, December 30, 1899, is represented as zero, and other dates are represented as the number of days from the base date.

Strings are formatted by transferring one character at a time from the input *expression* to the output string.

For more information, see these topics:

Formatting Numbers

Formatting Dates and Times

Formatting Strings

Formatting Numbers

The predefined numeric formats with their meanings are as follows:

Format	Description
General Number	Display the number without thousand separator.
Fixed	Display the number with at least one digit to the left and at least two digits to the right of the decimal separator.
Standard	Display the number with thousand separator and two digits to the right of decimal separator.
Scientific	Display the number using standard scientific notation.
Currency	Display the number using a currency symbol as defined in the International section of the Control Panel. Use thousand separator and display two digits to the right of decimal separator. Enclose negative value in parentheses.
Percent	Multiply the number by 100 and display with a percent sign appended to the right; display two digits to the right of decimal separator.
True/False	Display FALSE for 0, TRUE for any other number.
Yes/No	Display No for 0, Yes for any other number.
On/Off	Display Off for 0, On for any other number.

To create a user-defined numeric format, follow these guidelines:

For a simple numeric format, use one or more digit characters and (optionally) a decimal separator. The two format digit characters provided are zero, "0", and number sign, "#". A zero forces a corresponding digit to appear in the output; while a number sign causes a digit to appear in the output if it is significant (in the middle of the number or non-zero).

Number	Fmt	Result
1234.56	#	1235
1234.56	#.##	1234.56
1234.56	#.#	1234.6
1234.56	######.##	1234.56
1234.56	00000.000	01234.560
0.12345	#.##	.12
0.12345	0.##	0.12

A comma placed between digit characters in a format causes a comma to be placed between every three digits to the left of the decimal separator.

Number	Fmt	Result
1234567.8901	#,#.##	1,234,567.89
1234567.8901	#,#.### #	1,234,567.8901

Note: Although a comma and period are used in the *format* to denote separators for thousands and decimals, the output string will contain the appropriate character, based upon the current international settings for your machine.

Numbers can be scaled either by inserting one or more commas before the decimal separator or by including a percent sign in the *format* specification. Each comma preceding the decimal separator (or after all digits if no decimal separator is supplied) will scale (divide) the number by 1000. The commas will not appear in the output string. The percent sign will cause the number to be

multiplied by 100. The percent sign will appear in the output string in the same position as it appears in *format*.

Number	Fmt	Result
1234567.8901	#,.##	1234.57
1234567.8901	#,,.####	1.2346
1234567.8901	#,#,.##	1,234.57
0.1234	#0.00%	12.34%

Characters can be inserted into the output string by being included in the *format* specification. The following characters will be automatically inserted in the output string in a location matching their position in the *format* specification:

- + $ ( ) space : /

Any set of characters can be inserted by enclosing them in double quotes. Any single character can be inserted by preceding it with a backslash, "\".

Number	Fmt	Result
1234567.89	$#,0.00	$1,234,567.89
1234567.89	"TOTAL:" $#,#.00	TOTAL: $1,234,567.89
1234	\=\>#,#\<\=	=>1,234<=

You can use the OPEN Script **'$CSTRINGS** metacommand or the **Chr** function if you need to embed quotation marks in a format specification. The character code for a quotation mark is 34.

Numbers can be formatted in scientific notation by including one of the following exponent strings in the *format* specification:

E- E+ e- e+

The exponent string should be preceded by one or more digit characters. The number of digit characters following the exponent string determines the number of exponent digits in the output. *Format* specifications containing an upper case E will result in an upper case E in the output. Those containing a lower case e will result in a lower case e in the output. A minus sign following the E will cause negative exponents in the output to be preceded by a minus sign. A plus sign in the *format* will cause a sign to always precede the exponent in the output.

Number	Fmt	Result
1234567.89	###.##E-00	123.46E04
1234567.89	###.##e+#	123.46e+4
0.12345	0.00E-00	1.23E-01

A numeric *format* can have up to four sections, separated by semicolons. If you use only one section, it applies to all values. If you use two sections, the first section applies to positive values and zeros, the second to negative values. If you use three sections, the first applies to positive values, the

second to negative values, and the third to zeros. If you include semicolons with nothing between them, the undefined section is printed using the format of the first section. The fourth section applies to Null values. If it is omitted and the input expression results in a NULL value, **Format** will return an empty string.

Number	Fmt	Result
1234567.89	#,0.00;(#,0.00);"Zero";"NA"	1,234,567.89
-1234567.89	#,0.00;(#,0.00);"Zero";"NA"	(1,234,567.89)
0.0	#,0.00;(#,0.00);"Zero";"NA#"	Zero
0.0	#,0.00;(#,0.00);;"NA"	0.00
Null	#,0.00;(#,0.00);"Zero";"NA"	NA
Null	"The value is: "	0.00

Formatting Dates and Times

As with numeric formats, there are several predefined formats for formatting dates and times:

Format	Description
General Date	If the number has both integer and real parts, display both date and time. (e.g., 11/8/93 1:23:45 PM); if the number has only integer part, display it as a date; if the number has only fractional part, display it as time.
Long Date	Display a Long Date. Long Date is defined in the International section of the Control Panel.
Medium Date	Display the date using the month abbreviation and without the day of the week. (e.g, 08-Nov-93).
Short Date	Display a Short Date. Short Date is defined in the International section of the Control Panel.
Long Time	Display Long Time. Long Time is defined in the International section of the Control Panel and includes hours, minutes, and seconds.
Medium Time	Do not display seconds; display hours in 12-hour format and use the AM/PM designator.
Short Time	Do not display seconds; use 24-hour format and no AM/PM designator.

When using a user-defined format for a date, the *format* specification contains a series of tokens. Each token is replaced in the output string by its appropriate value.

A complete date can be output using the following tokens:

Token	Output
c	The date time as if the *format* was: "ddddd ttttt". See the definitions below.
ddddd	The date including the day, month, and year according to the machine's current Short Date setting. The default Short Date setting for the United States is m/d/yy.
dddddd	The date including the day, month, and year according to the machine's current Long Date setting. The default Long Date setting for the United States is mmmm dd, yyyy.
ttttt	The time including the hour, minute, and second using the machine's current time settings The default time format is h:mm:ss AM/PM.

Finer control over the output is available by including *format* tokens that deal with the individual components of the date time. These tokens are:

Token	Output
d	The day of the month as a one or two digit number (1-31).
dd	The day of the month as a two digit number (01-31).
ddd	The day of the week as a three letter abbreviation (Sun-Sat).
dddd	The day of the week without abbreviation (Sunday-Saturday).
w	The day of the week as a number (Sunday as 1, Saturday as 7).
ww	The week of the year as a number (1-53).
m	The month of the year or the minute of the hour as a one or two digit number. The minute will be output if the

	preceding token was an hour; otherwise, the month will be output.
mm	The month or the year or the minute of the hour as a two digit number. The minute will be output if the preceding token was an hour; otherwise, the month will be output.
mmm	The month of the year as a three letter abbreviation (Jan-Dec).
mmmm	The month of the year without abbreviation(January-December).
q	The quarter of the year as a number (1-4).
y	The day of the year as a number (1-366).
yy	The year as a two-digit number (00-99).
yyyy	The year as a four-digit number (100-9999).
h	The hour as a one or two digit number (0-23).
hh	The hour as a two digit number (00-23).
n	The minute as a one or two digit number (0-59).
nn	The minute as a two digit number (00-59).
s	The second as a one or two digit number (0-59).
ss	The second as a two digit number (00-59).

By default, times will be displayed using a military (24-hour) clock. Several tokens are provided in date time *format* specifications to change this default. They all cause a 12 hour clock to be used. These are:

Token	**Output**
AM/PM	An uppercase AM with any hour before noon; an uppercase PM with any hour between noon and 11:59 PM.
am/pm	A lowercase am with any hour before noon; a lowercase pm with any hour between noon and 11:59 PM.
A/P	An uppercase A with any hour before noon; an uppercase P with any hour between noon and 11:59 PM.
a/p	A lowercase a with any hour before noon; a lowercase p with any hour between noon and 11:59 PM.
AMPM	The contents of the 1159 string (s1159) in the WIN.INI file with any hour before noon; the contents of the 2359 string (s2359) with any hour between noon and 11:59 PM. Note, ampm is equivalent to AMPM.

Any set of characters can be inserted into the output by enclosing them in double quotes. Any single character can be inserted by preceding it with a backslash, "\". See number formatting above for more details.

Formatting Strings

By default, string formatting transfers characters from left to right. The exclamation point, "!", when added to the *format* specification causes characters to be transferred from right to left.

By default, characters being transferred will not be modified. The less than, "<", and the greater than, ">", characters can be used to force case conversion on the transferred characters. Less than forces output characters to be in lowercase. Greater than forces output characters to be in uppercase.

Character transfer is controlled by the at sign, "@", and ampersand, "&", characters in the *format* specification. These operate as follows:

Character	Interpretation
@	Output a character or a space. If there is a character in the string being formatted in the position where the @ appears in the format string, display it; otherwise, display a space in that position.
&	Output a character or nothing. If there is a character in the string being formatted in the position where the & appears, display it; otherwise, display nothing.

A *format* specification for strings can have one or two sections separated by a semicolon. If you use one section, the format applies to all string data. If you use two sections, the first section applies to string data, the second to Null values and zero-length strings.

FreeFile Function

See Also **Example**

Returns the lowest unused file number.

Syntax **FreeFile**

Comments The **FreeFile** function is used when you need to supply a file number and want to make sure that you are not choosing a file number that is already in use.

The value returned can be used in a subsequent **Open** statement.

Function ... End Function Statement

See Also **Example**

Defines a function procedure.

Syntax **[Static] [Private] Function** *name* **[(** [**Optional**]*parameter* [**As** *type*] ... **)]** **[As** *functype* **]**

*name***=** *expression*

End Function

where:	**is:**
name	A function name.
parameter	The argument(s) to pass to the function when it is called.
type	The data type for the function arguments.
functype	The data type for the return value.
name=expression	The expression that sets the return value for the function.

Comments The purpose of a function is to produce and return a single value of a specified type. Recursion is supported.

The data type of *name* determines the type of the return value. Use a type character as part of the *name*, or use the **As** *functype* clause to specify the data type. If omitted, the default data type is **Variant**. When calling the function, you need not specify the type character.

The *parameters* are specified as a comma-separated list of variable names. The data type of a parameter can be specified by using a type character or by using the **As** clause. Record parameters are declared using an **As** clause and a *type* that has previously been defined using the **Type** statement. Array parameters are indicated by using empty parentheses after the *parameter*. The array dimensions are not specified in the **Function** statement. All references to an array parameter within the body of the function must have a consistent number of dimensions.

You specify the return value for the function name using the *name=expression* assignment, where name is the name of the function and expression evaluates to a return value. If omitted, the value returned is 0 for numeric functions and an empty string ("") for string functions and vartype 0 (Empty) is returned for a return type of Variant. The function returns to the caller when the **End Function** statement is reached or when an **Exit Function** statement is executed.

If you declare a parameter as **Optional**, a procedure can omit its value when calling the function. Only parameters with **Variant** data types can be declared as optional, and all optional arguments must appear after all required arguments in the **Function** statement. The function **IsMissing** must be used to check whether an optional parameter was omitted by the user or not. Named parameters are described under the **Call** statement heading, but they can be used when the function is used in an expression as well.

The **Static** keyword specifies that all the variables declared within the function will retain their values as long as the program is running, regardless of the way the variables are declared.

The **Private** keyword specifies that the function will not be accessible to functions and subprograms from other modules. Only procedures defined in the same module will have access to a **Private** function.

Basic procedures use the call by reference convention. This means that if a procedure assigns a value to a parameter, it will modify the variable passed

by the caller. This feature should be used with great care.
Use **Sub** to define a procedure with no return value.

FV Function

See Also **Example**

Returns the future value for a constant periodic stream of cash flows as in an annuity or a loan.

Syntax **FV** *(rate , nper , pmt , pv , due)*

where:	**is:**
rate	Interest rate per period.
nper	Total number of payment periods.
pmt	Constant periodic payment per period.
pv	Present value or the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).
due	An integer value for when the payments are due (0=end of each period, 1= beginning of the period).

Comments The given interest rate is assumed constant over the life of the annuity.

If payments are on a monthly schedule and the annual percentage rate on the annuity or loan is 9%, the *rate* is 0.0075 (.0075=.09/12).

Get Statement

See Also **Example**

Reads data from a file opened in **Random** or **Binary** mode and puts it in a variable.

Syntax **Get** [#] *filenumber%*, [*recnumber&*], *varname*

where:	**is:**
filenumber%	An integer expression identifying the open file to use.
recnumber&	A **Long** expression containing the number of the record (for **Random** mode) or the offset of the byte (for **Binary** mode) at which to start reading.
varname	The name of the variable into which **Get** reads file data. *Varname* can be any variable except **Object** or **Array** variables (single array elements can be used).

Comments For more information about how files are numbered when they're opened, see the **Open** statement.

Recnumber& is in the range 1 to 2,147,483,647. If omitted, the next record or byte is read.

Note: The commas before and after the *recnumber&* are required, even if you do not supply a *recnumber&*.

For **Random** mode, the following rules apply:

Blocks of data are read from the file in chunks whose size is equal to the size specified in the Len clause of the **Open** statement. If the size of *varname* is smaller than the record length, the additional data is discarded. If the size of *varname* is larger than the record length, an error occurs.

For variable length **String** variables, Get reads two bytes of data that indicate the length of the string, then reads the data into *varname*.

For **Variant** variables, Get reads two bytes of data that indicate the type of the Variant, then it reads the body of the Variant into *varname*. Note that Variants containing strings contain two bytes of data type information followed by two bytes of length followed by the body of the string.

User defined types are read as if each member were read separately, except no padding occurs between elements.

Files opened in **Binary** mode behave similarly to those opened in **Random** mode, except:

Get reads variables from the disk without record padding.

Variable length **Strings** that are not part of user defined types are not preceded by the two-byte string length. Instead, the number of bytes read is equal to the length of *varname*.

GetAttr Function

See Also **Example**

Returns the attributes of a file, directory or volume label.

Syntax **GetAttr**(*pathname$*)

where:	**is:**
pathname$	A **String** expression for the name of the file, directory, or label to query.

Comments *Pathname$* cannot contain wildcards (* and ?).

The file attributes returned by **GetAttr** are as follows:

Value	**Meaning**
0	Normal file
1	Read-only file
2	Hidden file
4	System file
8	Volume label
16	Directory
32	Archive - file has changed since last backup

GetField Function [OPEN Script Extension]

See Also **Example**

Returns a substring from a source string.

Syntax **GetField**[**$**]**(** *string$, field_number% , separator_chars$* **)**

where:	**is:**
string$	A list of fields, divided by separator characters.
field_number%	The number of the field to return, starting with 1.
separator_chars$	The characters separating each field.

Comments Multiple separator characters can be specified. If *field_number* is greater than the number of fields in the string, an empty string ("") is returned.

GetObject Function

See Also **Example** **Overview**

Returns an OLE Automation object associated with the filename or the application name.

Syntax A **GetObject(** *pathname***)**
Syntax B **GetObject(** *pathname , class* **)**
Syntax C **GetObject(** *, class* **)**

where:	**is:**
pathname	The path and filename for the object to retrieve.
class	A string containing the class of the object.

Comments Use GetObject with the **Set** statement to assign a variable to the object for use in a Basic procedure. The variable used must first be dimensioned as an **Object**.

Syntax A of **GetObject** accesses an OLE Automation object stored in a file. For example, the following two lines dimension the variable, FILEOBJECT as an Object and assign the object file "PAYABLES" to it. PAYABLES is located in the subdirectory SPREDSHT:

```
Dim FileObject As Object
Set FileObject = GetObject("\spredsht\payables")
```

If the application supports accessing component OLE Automation objects within the file, you can append an exclamation point and a component object name to the filename, as follows:

```
Dim ComponentObject As Object
Set ComponentObject = GetObject("\spredsht\payables!R1C1:R13C9")
```

Syntax B of **GetObject** accesses an OLE Automation object of a particular class that is stored in a file. *Class* uses the syntax: "*appname.objtype*", where *appname* is the name of the application that provides the object, and *objtype* is the type or class of the object. For example:

```
Dim ClassObject As Object
Set ClassObject = GetObject("\spredsht\payables",
turbosht.spreadsheet)
```

The third form of **GetObject** accesses the active OLE Automation object of a particular class. For example:

```
Dim ActiveSheet As Object
SetActiveSheet = GetObject( , turbosht.spreadsheet)
```

Global Statement

See Also **Example**

Declare Global variables for use in a Basic program.

Syntax **Global** *variableName* [**As** *type*] [,*variableName* [**As** *type*]] **...**

where:	**is:**
variableName	A variable name
type	The data type for a variable.

Comments Global data is shared across all loaded modules. If an attempt is made to load a module that has a global variable declared that has a different data type than an existing global variable of the same name, the module load will fail.

Basic is a strongly typed language: all variables must be given a data type or they will be automatically assigned a type of **Variant**.

If the **As** clause is not used, the type of the global variable can be specified by using a type character as a suffix to *variableName*. The two different type-specification methods can be intermixed in a single **Global** statement (although not on the same variable).

Regardless of which mechanism you use to declare a global variable, you can choose to use or omit the type character when referring to the variable in the rest of your program. The type suffix is not considered part of the variable name.

The available data types are:

Arrays

Numbers

Records

Strings

Variants

Arrays

The available data types for arrays are: numbers, strings, Variants and records. Arrays of arrays, dialog box records, and objects are not supported.

Array variables are declared by including a subscript list as part of the *variableName*. The syntax to use for *variableName* is:

Global *variable*([*subscriptRange*, ...]) [**As** *typeName*]

where *subscriptRange* is of the format:

[*startSubscript* **To**] *endSubscript*

If *startSubscript* is not specified, 0 is used as the default. The **Option Base** statement can be used to change the default.

Both the *startSubscript* and the *endSubscript* are valid subscripts for the array. The maximum number of subscripts that can be specified in an array definition is 60.

If no *subscriptRange* is specified for an array, the array is declared as a dynamic array. In this case, the **ReDim** statement must be used to specify the dimensions of the array before the array can be used.

Numbers

Numeric variables can be declared using the **As** clause and one of the following numeric types: **Currency**, **Integer**, **Long**, **Single**, **Double**. Numeric variables can also be declared by including a type character as a suffix to the name.

Records

Record variables are declared by using an **As** clause and a *type* that has previously been defined using the **Type** statement. The syntax to use is:

Global *variableName* **As** *typeName*

Records are made up of a collection of data elements called fields. These fields can be of any numeric, string, Variant or previously-defined record type. See **Type** for details on accessing fields within a record.

You cannot use the **Global** statement to declare a dialog record.

Strings

OPEN Script supports two types of strings, fixed-length and dynamic. Fixed-length strings are declared with a specific length (between 1 and 32767) and cannot be changed later. Use the following syntax to declare a fixed-length string:

Global *variableName* **As String*** *length*

Dynamic strings have no declared length, and can vary in length from 0 to 32767. The initial length for a dynamic string is 0. Use the following syntax to declare a dynamic string:

Global *variableName$* or
Global *variableName* **As String**

Variants

Declare variables as Variants when the type of the variable is not known at the start of, or might change during, the procedure. For example, a Variant is useful for holding input from a user when valid input can be either text or numbers. Use the following syntax to declare a Variant:

Global *variableName* or
Global*variableName* **As Variant**

Variant variables are initialized to vartype **Empty**.

GoTo Statement

See Also **Example**

Transfers program control to the label specified.

Syntax **GoTo** { *label* }

where:	**is:**
label	A name beginning in the first column of a line of code and ending with a colon (:).

Comments A *label* has the same format as any other Basic name. Reserved words are not valid labels.

GoTo cannot be used to transfer control out of the current Function or Subprogram.

GroupBox Statement [OPEN Script Extension]

See Also **Example**

Defines and draws a box that encloses sets of dialog box items, such as option boxes and check boxes.

Syntax **GroupBox** *x , y , dx , dy , text$* [, **.***id*]

where:	**is:**
x , y	The upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
dx , dy	The width and height of the combo box in which the user enters or selects text.
text$	A string containing the title for the top border of the group box.
.id	The optional string ID for the groupbox, used by the dialog statements that act on this control.

Comments The *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-width units. (See **Begin Dialog** for more information.)

If *text$* is wider than *dx*, the additional characters are truncated. If *text$* is an empty string (""), the top border of the group box will be a solid line.

Use the **GroupBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Hex Function

See Also **Example**

Returns the hexadecimal representation of a number, as a string.

Syntax **Hex**[**$**]**(** *number* **)**

where:	**is:**
number	Any numeric expression that evaluates to a number.

Comments If *number* is an integer, the return string contains up to four hexadecimal digits; otherwise, the value will be converted to a **Long** Integer, and the string can contain up to 8 hexadecimal digits.

To represent a hexadecimal number directly, precede the hexadecimal value with **&H**. For example, &H10 equals decimal 16 in hexadecimal notation.

The dollar sign, "$", in the function name is optional. If specified the return type is string. If omitted the function will return a **Variant** of vartype 8 (string).

Hour Function

See Also **Example**

Returns the hour of day component (0-23) of a date-time value.

Syntax **Hour(** *time* **)**

where:	**is:**
time	Any numeric or string expression that can evaluate to a date and time.

Comments **Hour** accepts any type of *time* including strings and will attempt to convert the input value to a date value.

The return value is a **Variant** of vartype 2 (integer). If the value of *time* is Null, a Variant of vartype 1 (null) is returned.

Time is a double-precision value. The numbers to the left of the decimal point denote the date and the decimal value denotes the time (from 0 to .99999). Use the **TimeValue** function to obtain the correct value for a specific time.

If ... Then ... Else

See Also **Example**

Executes alternative blocks of program code based on one or more expressions.

Syntax A **If** *condition* **Then** *then_statement* **[Else** *else_statement* **]**

Syntax B **If** *condition* **Then**
statement_block
[ElseIf *expression* **Then**
*statement_block***]...**
[Else
statement_block **]**
End If

where:	**is:**
condition	Any expression that evaluates to TRUE (non-zero) or FALSE (zero).
then_statement	Any valid single expression.
else_statement	Any valid single expression.
expression	Any expression that evaluates to TRUE (non-zero) or FALSE (zero).
statement_block	0 or more valid expressions, separated by colons (:), or on different lines.

Comments When multiple statements are required in either the **Then** or **Else** clauses, use the block version (Syntax B) of the **If** statement.

'$Include Metacommand [OPEN Script Extension]

See Also **Example**

Includes statements from the specified file.

Syntax '**$Include: "***filename***"**

where:	**is:**
filename	The name and location of the file to include.

Comments We recommend (although it is not required) that you use the file extension .OSS for your script files, as in *filename.oss*.

All metacommands must begin with an apostrophe (') and are recognized by the compiler only if the command starts at the beginning of a line. For compatibility with other versions of Basic, you can enclose the *filename* in single quotation marks (').

If no directory or drive is specified, the compiler will search for *filename* on the source file search path.

Input Function

See Also **Example**

Returns a string containing the characters read.

Syntax **Input**[**$**]**(** *number% ,* [#]*filenumber%***)**

where:	**is:**
number%	The number of characters (bytes) to read from the file.
filenumber%	An integer expression identifying the open file to use.

Comments The file pointer is advanced the number of characters read. Unlike the **Input** statement, **Input** returns all characters it reads, including carriage returns, line feeds, and leading spaces.

The dollar sign, "$", in the function name is optional. If specified the return type is string. If omitted the function will return a **Variant** of vartype 8 (string).

Input Statement

See Also **Example**

Reads data from a sequential file and assigns the data to variables.

Syntax A **Input** [#] *filenumber% , variable* [*, variable*]...

Syntax B **Input** [*prompt$*,] *variable* [, *variable*]...

where:	**is:**
filenumber%	An integer expression identifying the open file to read from
variable	The variable(s) to contain the value(s) read from the file.
prompt$	An optional string that prompts for keyboard input.

Comments The *filenumber%* is the number used in the **Open** statement to open the file. The list of *variables* is separated by commas.

If *filenumberr%* is not specified, the user is prompted for keyboard input, either with *prompt$* or with a "?", if *prompt$* is omitted.

InputBox Function

See Also **Example**

Displays a dialog box containing a prompt and returns a string entered by the user.

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

The dollar sign, "$", in the function name is optional. If specified the return type is string. If omitted the function will return a **Variant** of vartype 8 (string).

where:	**is:**
prompt$	A string expression containing the text to show in the dialog box.
title$	The caption to display in the dialog box's title bar.
default$	The string expression to display in the edit box as the default response.
xpos%, *ypos%*	Numeric expressions, specified in dialog box units, that determine the position of the dialog box.

Comments The length of *prompt$* is restricted to 255 characters. This figure is approximate and depends on the width of the characters used. Note that a carriage return and a line-feed character must be included in *prompt$* if a multiple-line prompt is used.

If either *prompt$* or *default$* is omitted, nothing is displayed.

Xpos% determines the horizontal distance between the left edge of the screen and the left border of the dialog box. *Ypos%* determines the horizontal distance from the top of the screen to the dialog box's upper edge. If these arguments are not entered, the dialog box is centered roughly one third of the way down the screen. A horizontal dialog box unit is 1/4 of the average character width in the system font; a vertical dialog box unit is 1/8 of the height of a character in the system font.

Note: If you want to specify the dialog box's position, you must enter both of these arguments. If you enter one without the other, the default positioning is set.

If the user presses Enter, or selects the OK button, **InputBox** returns the text contained in the input box. If the user selects Cancel, the **InputBox** function returns a null string ("").

InStr Function

See Also **Example**

Returns the position of the first occurrence of one string within another string.

Syntax A **InStr(** [*start%***,**] *string1$***,** *string2$* **)**

Syntax B **InStr(** *start , string1$***,** *string2$*[, *compare*]**)**

where:	**is:**
start%	The position in *string1$* to begin the search. (1=first character in string.)
string1$	The string to search.
string2$	The string to find.
compare	An integer expression for the method to use to compare the strings. (0=case-sensitive, 1=case-insensitive.)

Comments If not specified, the search starts at the beginning of the string (equivalent to a *start%* of 1). These arguments can be of any type. They will be converted to strings.

InStr returns a zero under the following conditions:

1. *start%* is greater than the length of *string2$*.
2. *string1$* is a null string.
3. *string2$* is not found.

If either *string1$* or *string2$* is a null Variant , **Instr** returns a null Variant.

If *string2$* is a null string (""), **Instr** returns the value of *start%*.

If *compare* is 0, a case-sensitive comparison based on the ANSI character set sequence is performed. If *compare* is 1, a case-insensitive comparison is done based upon the relative order of characters as determined by the country code setting for your system. If *compare* is omitted, the module level default, as specified with **Option Compare**, is used.

Int Function

See Also **Example**

Returns the integer part of a *number*.

Syntax **Int(** *number* **)**

where:	**is:**
number	Any numeric expression.

Comments For positive *numbers*, **Int** removes the fractional part of the expression and returns the integer part only. For negative *numbers*, **Int** returns the largest integer less than or equal to the expression. For example, **Int** (6.2) returns 6; **Int**(-6.2) returns -7.

The return type matches the type of the numeric expression. This includes **Variant** expressions that will return a result of the same vartype as input except vartype 8 (string) will be returned as vartype 5 (double) and vartype 0 (empty) will be returned as vartype 3 (long).

IPmt Function

See Also **Example**

Returns the interest portion of a payment for a given period of an annuity.

Syntax **IPmt(** *rate , per , nper , pv , fv , due* **)**

where:	**is:**
rate	Interest rate per period.
per	Particular payment period in the range 1 through *nper*.
nper	Total number of payment periods.
pv	Present value of the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).
fv	Future value of the final lump sum amount required (as in the case of a savings plan) or paid (0 as in the case of a loan).
due	0 if payments are due at the end of each payment period, and 1 if they are due at the beginning of the period.

Comments The given interest rate is assumed constant over the life of the annuity. If payments are on a monthly schedule, then *rate* will be 0.0075 if the annual percentage rate on the annuity or loan is 9%.

IRR Function

See Also **Example**

Returns the internal rate of return for a stream of periodic cash flows.

Syntax **IRR(** *valuearray( ) , guess* **)**

where:	**is:**
valuearray( )	An array containing cash flow values.
guess	A ballpark estimate of the value returned by **IRR**.

Comments *valuearray()* must have at least one positive value (representing a receipt) and one negative value (representing a payment). All payments and receipts must be represented in the exact sequence. The value returned by **IRR** will vary with the change in the sequence of cash flows.

In general, a *guess* value of between 0.1 (10 percent) and 0.15 (15 percent) would be a reasonable estimate.

IRR is an iterative function. It improves a given guess over several iterations until the result is within 0.00001 percent. If it does not converge to a result within 20 iterations, it signals failure.

Is Operator

See Also **Example** **Overview**

Compares two object expressions and returns -1 if they refer to the same object, 0 otherwise.

Syntax *objectExpression* **Is** *objectExpression*

where:	**is:**
objectexpression	Any valid object expression.

Comments **Is** can also be used to test if an object variable has been **Set** to **Nothing**.

IsDate Function

See Also **Example**

Returns -1 (TRUE) if an expression is a legal date, 0 (FALSE) if it is not.

Syntax **IsDate(** *expression* **)**

where:	**is:**
expression	Any valid expression.

Comments **IsDate** returns -1 (TRUE) if the expression is of vartype 7 (date) or a string that can be interpreted as a date.

IsEmpty Function

See Also **Example**

Returns -1 (TRUE) if a Variant has been initialized. 0 (FALSE) otherwise.

Syntax **IsEmpty(** *expression* **)**

where:	**is:**
expression	Any expression with a data type of **Variant**.

Comments **IsEmpty** returns -1 (TRUE) if the Variant is of vartype 0 (empty). Any newly-defined Variant defaults to being of Empty type, to signify that it contains no initialized data. An Empty Variant converts to zero when used in a numeric expression, or an empty string ("") in a string expression.

IsMissing Function

See Also **Example**

Returns -1 (TRUE) if an optional parameter was not supplied by the user, 0 (FALSE) otherwise.

Syntax **IsMissing(** *argname* **)**

where:	**is:**
argname	An optional argument for a subprogram, function, OPEN Script statement, or OPEN Script function.

Comments **IsMissing** is used in procedures that have optional arguments to find out whether the arguments value was supplied or not.

IsNull Function

See Also **Example**

Returns -1 (TRUE) if a Variant expression contains the Null value, 0 (FALSE) otherwise.

Syntax **IsNull(** *expression***)**

where:	**is:**
expression	Any expression with a data type of **Variant**.

Comments Null Variants have no associated data and serve only to represent invalid or ambiguous results. Null is not the same as Empty, which indicates that a Variant has not yet been initialized.

IsNumeric Function

See Also **Example**

Returns -1 (TRUE) if an expression has a data type of **Numeric**, 0 (FALSE) otherwise.

Syntax **IsNumeric(** *expression* **)**

where:	**is:**
expression	Any valid expression.

Comments **IsNumeric** returns -1 (TRUE) if the expression is of vartypes 2-6 (numeric) or a string that can be interpreted as a number.

Kill Statement

See Also **Example**

Deletes files from a hard disk or floppy drive.

Syntax **Kill** *pathname$*

where:	**is:**
pathname$	A String expression that specifies a valid DOS file specification.

Comments The *pathname$* specification can contain paths and wildcards. **Kill** deletes files only, not directories. Use the **RmDir** function to delete directories.

LBound Function

See Also **Example**

Returns the lower bound of the subscript range for the specified array.

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

where:	**is:**
arrayname	The name of the array to use.
dimension	The dimension to use.

Comments The dimensions of an array are numbered starting with 1. If the *dimension* is not specified, 1 is used as a default.

LBound can be used with **UBound** to determine the length of an array.

LCase Function

See Also **Example**

Returns a copy of a string, with all uppercase letters converted to lowercase.

Syntax **LCase**[**$**]**(** *string$* **)**

where:	**is:**
string$	A string, or an expression containing the string to use.

Comments The translation is based on the country specified in the Windows Control Panel. **LCase** accepts expressions of type String. **LCase** accepts any type of argument and will convert the input value to a string.

The dollar sign, "$", in the function name is optional. If specified the return type is String. If omitted the function will typically return a **Variant** of vartype 8 (string). If the value of *string$* is NULL, a Variant of vartype 1 (Null) is returned.

Left Function

See Also **Example**

Returns a string of a specified length copied from the beginning of another string.

Syntax **Left**[**$**]**(** *string$*, *length%* **)**

where:	**is:**
string$	A string or an expression containing the string to copy.
length%	The number of characters to copy.

Comments If *length%* is greater than the length of *string$,* **Left** returns the whole string.

Left accepts expressions of type String. **Left** accepts any type of *string$,* including numeric values, and will convert the input value to a string.

The dollar sign, "$", in the function name is optional. If specified, the return type is string. If omitted, the function will typically return a **Variant** of vartype 8 (string). If the value of *string$* is NULL, a Variant of vartype 1 (Null) is returned.

Len Function

See Also **Example**

Returns the length of a string or variable.

Syntax A **Len(** *string$* **)**

Syntax B **Len(** *varname* **)**

where:	**is:**
string$	A string or an expression that evaluates to a string.
varname	A variable that contains a string.

Comments If the argument is a string, the number of characters in the string is returned. If the argument is a Variant variable, **Len** returns the number of bytes required to represent its value as a string; otherwise, the length of the built-in data type or user-defined type is returned.

If syntax B is used, and *varname* is a **Variant** containing a NULL, **Len** will return a Null Variant.

Let (Assignment Statement)

See Also **Example**

Assigns an expression to a Basic variable.

Syntax **[Let]** *variable* **=** *expression*

where:	**is:**
variable	The name of a variable to assign to the *expression*.
expression	The expression to assign to the variable.

Comments The keyword **Let** is optional.

The **Let** statement can be used to assign a value or expression to a variable with a data type of **Numeric**, **String**, **Variant** or **Record** variable. You can also use the **Let** statement to assign to a record field or to an element of an array.

When assigning a value to a numeric or string variable, standard conversion rules apply.

Let differs from **Set** in that Set assigns a variable to an OLE Automation object. For example,

Set o1 = o2 will set the object reference.

Let o1 = o2 will set the value of the default member.

Like Operator

See Also **Example**

Returns the value -1 (TRUE) if a string matches a pattern, 0 (FALSE) otherwise.

Syntax *string$* **LIKE** *pattern$*

where:	**is:**
string$	Any string expression.
pattern$	Any string expression to match to *string$*.

Comments *pattern$* can include the following special characters:

Character:	**Matches:**
?	A single character
*	A set of zero or more characters
#	A single digit character (0-9)
[*chars*]	A single character in *chars*
[!*chars*]	A single character not in *chars*
[*schar-echar*]	A single character in range *schar* to *echar*
[!*schar-echar*]	A single character not in range *schar* to *echar*

Both ranges and lists can appear within a single set of square brackets. Ranges are matched according to their ANSI values. In a range, *schar* must be less than *echar*.

If either *string$* or *pattern$* is NULL then the result value is NULL.

The **Like** operator respects the current setting of **Option Compare**.

For more information about operators, see **Expressions**.

Line Input Statement

See Also **Example**

Reads a line from a sequential file into a string variable.

Syntax A **Line Input** [#] *filenumber% , varname$*

Syntax B **Line Input** [*prompt$*,] *varname$*

where:	**is:**
filenumber%	An integer expression identifying the open file to use.
prompt$	An optional string that can be used to prompt for keyboard input; it must be a literal string.
varname$	A string variable to contain the line read.

Comments If specified, the *filenumber%* is the number used in the **Open** statement to open the file. If *filenumber%* is not provided, the line is read from the keyboard.

If *prompt$* is not provided, a prompt of "?" is used.

ListBox Statement

See Also **Example**

Defines a list box of choices for a dialog box.

Syntax A **ListBox** *x , y , dx , dy , text$, .field*

Syntax B **ListBox** *x , y , dx , dy , stringarray$() , .field*

where:	**is:**
x , y	The upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
dx , dy	The width and height of the list box.
text$	A string containing the selections for the list box.
stringarray$	An array of dynamic strings for the selections in the list box.
.field	The name of the dialog-record field that will hold a number for the choice made in the list box.

Comments The *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-width units. (See **Begin Dialog** for more information.)

The *text$* argument must be defined, using a **Dim** Statement, before the **Begin Dialog** statement is executed. The arguments in the *text$* string are entered as shown in the following example:

dimname = "*listchoice*"+Chr$(9)+"*listchoice*"+Chr$(9)+"*listchoice*"...

A number representing the selection's position in the *text$* string is recorded in the field designated by the *.field* argument when the OK button (or any pushbutton other than Cancel) is pushed. The numbers begin at 0. If no item is selected, it is -1. The *field* argument is also used by the dialog statements that act on this control.

Use the **ListBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Loc Function

See Also **Example**

Returns the current offset within an open file.

Syntax **Loc(** *filenumber%* **)**

where:	**is:**
filenumber%	An integer expression identifying the open file to query.

Comments The *filenumber%* is the number used in the **Open** statement of the file.

For files opened in **Random** mode, **Loc** returns the number of the last record read or written. For files opened in **Append**, **Input**, or **Output** mode, **Loc** returns the current byte offset divided by 128. For files opened in **Binary** mode, **Loc** returns the offset of the last byte read or written.

Lock Statement

See Also **Example**

Controls access to an open file.

Syntax **Lock** [#]*filenumber%* [, [*start&*] [**To** *end&*]]

where:	**is:**
filenumber%	An integer expression identifying the open file.
start&	Number of the first record or byte offset to lock/unlock.
end&	Number of the last record or byte offset to lock/unlock.

Comments The *filenumber%* is the number used in the **Open** statement of the file.

For **Binary** mode, *start&*, and *end&* are byte offsets. For **Random** mode, *start&*, and *end&* are record numbers. If *start&* is specified without *end&*, then only the record or byte at *start&* is locked. If **To** *end&* is specified without *start&*, then all records or bytes from record number or offset 1 to *end&* are locked.

For **Input**, **Output** and **Append** modes, *start&,* and *end&* are ignored and the whole file is locked.

Lock and **Unlock** always occur in pairs with identical parameters. All locks on open files must be removed before closing the file, or unpredictable results will occur.

Lof Function

See Also **Example**

Returns the length in bytes of an open file.

Syntax **Lof(** *filenumber%* **)**

where:	**is:**
filenumber%	An integer expression identifying the open file.

Comments The *filenumber%* is the number used in the **Open** statement of the file.

Log Function

See Also **Example**

Returns the natural logarithm of a number.

Syntax **Log(** *number* **)**

where:	**is:**
number	Any valid numeric expression.

Comments The return value is single-precision for an integer, currency or single-precision numeric expression, double precision for a long, Variant or double-precision numeric expression.

Lset Statement

See Also **Example**

Copies one string to another, or assigns a user-defined type variable to another.

Syntax A **Lset** *string$* **=** *string-expression*

Syntax B **Lset** *variable1* = *variable2*

where:	**is:**
string$	A string or string expression to contain the copied characters.
string-expression	An expression containing the string to copy.
variable1	A variable with a user-defined type to contain the copied variable.
variable2	A variable with a user-defined type to copy.

Comments If *string$* is shorter than *string-expression*, **Lset** copies the leftmost character of *string-expression* into *string$.* The number of characters copied is equal to the length of *string$.*

If *string* is longer than *string-expression,* all characters of *string-expression* are copied into *string$*, filling it from left to right. All leftover characters of *string$* are replaced with spaces.

In Syntax B, the number of characters copied is equal to the length of the shorter of *variable1* and *variable2.*

Lset cannot be used to assign variables of different user-defined types if either contains a **Variant** or a variable-length string.

LTrim Function

See Also **Example**

Returns a copy of a string with all leading space characters removed.

Syntax **LTrim**[**$**]**(** *string$* **)**

where:	**is:**
string$	A string or expression containing a string to copy.

Comments **LTrim** accepts any type of *string$*, including numeric values, and will convert the input value to a string.

The dollar sign, "$", in the function name is optional. If specified, the return type is string. If omitted, the function typically returns a **Variant** of vartype 8 (string). If the value of *string$* is NULL, a Variant of vartype 1 (Null) is returned.

Me

See Also

Refers to the currently used OLE Automation object.

Syntax **Me**

Comments Some Basic modules are attached to application objects and Basic subroutines are invoked when that application object encounters events. A good example is a user visible button that triggers a Basic routine when the user clicks the mouse on the button.

Subroutines in such contexts can use the variable **Me** to refer to the object that triggered the event (i.e., which button was clicked). The programmer can use **Me** in all the same ways as any other object variable except that **Me** cannot be **Set**.

Mid Function

See Also **Example**

Returns a portion of a string, starting at a specified location within the string.

Syntax **Mid**[**$**]**(** *string$***,** *start%*[**,** *length%*] **)**

where:	**is:**
string$	A string or expression that contains the string to change.
start%	The starting position in *string$* to begin replacing characters.
length%	The number of characters to replace.

Comments **Mid** accepts any type of *string$*, including numeric values, and will convert the input value to a string. If the *length%* argument is omitted, or if *string$* is smaller than *length%*, then **Mid** returns all characters in *string$*. If *start%* is larger than *string$*, then **Mid** returns a null string ("").

The index of the first character in a string is 1.

The dollar sign, "$", in the function name is optional. If specified, the return type is string. If omitted, the function typically returns a **Variant** of vartype 8 (string). If the value of *string$* is Null, a Variant of vartype 1 (Null) is returned. **Mid$** requires the string argument to be of type string or variant. **Mid** allows the string argument to be of any datatype.

To modify a portion of a string value, see **Mid Statement**.

Mid Statement

See Also **Example**

Replaces part (or all) of one string with another, starting at a specified location.

Syntax **Mid (** *stringvar$***,** *start%*[**,** *length%*] **) =** *string$*

where:	**is:**
stringvar$	The string to change.
start%	An expression for the position to begin replacing characters.
length%	An expression for the number of characters to replace.
string$	The string to place into another string.

Comments If the *length%* argument is omitted, or if there are fewer characters in *string$* than specified in *length%*, then **Mid** replaces all the characters from the *start%* to the end of the *string$*. If *start%* is larger than the number of characters in the indicated *stringvar$*, then **Mid** appends *string%* to *stringvar$.*

If *length%* is greater than the length of *string$*, then *length%* is set to the length of *string$.* If *start%* is greater than the number of characters in *stringvar$*, an illegal function call error will occur at runtime. If *length%* plus *start%* is greater than the length of *stringvar$*, then only the characters up to the end of *stringvar$* are replaced.

Mid never changes the number of characters in *stringvar$*.

The index of the first character in a string is 1.

Minute Function

See Also **Example**

Returns an integer for the minute component (0-59) of a date-time value.

Syntax **Minute(** *time* **)**

where:	**is:**
time	Any expression that can evaluate to a date-time value.

Comments **Minute** accepts any type of *time*, including strings, and will attempt to convert the input value to a date value.

The return value is a **Variant** of vartype 2 (Integer). If the value of *time* is null, a Variant of vartype 1 (null) is returned.

MkDir Statement

See Also **Example**

Creates a new directory.

Syntax **MkDir** *path$*

where:	**is:**
path$	A string expression identifying the new default directory to create.

Comments The syntax for *path$* is:

[*drive*:] [\] *directory* [*directory*]

The *drive* argument is optional. If *drive* is omitted, **MkDir** makes a new directory on the current drive. The *directory* argument is any directory name.

Month Function

See Also **Example**

Returns an integer for the month component (1-12) of a date-time value.

Syntax **Month(** *date* **)**

where:	**is:**
date	Any expression that evaluates to a date-time value.

Comments It accepts any type of *date*, including strings, and will attempt to convert the input value to a date value.

The return value is a **Variant** of vartype 2 (integer). If the value of *date* is null, a Variant of vartype 1 (null) is returned.

Msgbox Function

See Also **Example**

Displays a message dialog box and returns a value (1-7) indicating which button the user selected.

Syntax **Msgbox(** *prompt$* ,[*buttons%*][, *title$*] **)**

where:	**is:**
prompt$	The text to display in a dialog box.
buttons%	An integer value for the buttons, the icon, and the default button choice to display in a dialog box.
title$	A string expression containing the title for the message box.

Comments *prompt$* must be no more than 1,024 characters long. A message string greater than 255 characters without intervening spaces will be truncated after the 255th character.

buttons% is the sum of three values, one from each of the following groups:

	Value	**Description**
Group 1:	0	OK only
Buttons	1	OK, Cancel
	2	Abort, Retry, Ignore
	3	Yes, No, Cancel
	4	Yes, No
	5	Retry, Cancel
Group 2:	16	Critical Message (STOP)
Icons	32	Warning Query (?)
	48	Warning Message (!)
	64	Information Message (i)
Group 3:	0	First button
Defaults	256	Second button
	512	Third button

If *buttons%* is omitted, Msgbox displays a single OK button.

After the user clicks a button, **Msgbox** returns a value indicating the user's choice. The return values for the Msgbox function are:

Value	**Button Pressed**
1	OK
2	Cancel
3	Abort
4	Retry
5	Ignore
6	Yes
7	No

Msgbox Statement

See Also **Example**

Displays a prompt in a message dialog box.

Syntax **MsgBox** *prompt$,* [*buttons%*][, *title$*]

where:	**is:**
prompt$	The text to display in a dialog box.
buttons%	An integer value for the buttons, the icon, and the default button choice to display in a dialog box.
title$	A string expression containing the title for the message box.

Comments *Prompt$* must be no more than 1,024 characters long. A message string greater than 255 characters without intervening spaces will be truncated after the 255th character.

buttons% is the sum of three values, one from each of the following groups:

	Value	**Description**
Group 1:	0	OK only
Buttons	1	OK, Cancel
	2	Abort, Retry, Ignore
	3	Yes, No, Cancel
	4	Yes, No
	5	Retry, Cancel
Group 2:	16	Critical Message (STOP)
Icons	32	Warning Query (?)
	48	Warning Message (!)
	64	Information Message (i)
Group 3:	0	First button
Defaults	256	Second button
	512	Third button

If *buttons%* is omitted, Msgbox displays a single OK button.

Name Statement

See Also **Example**

Renames a file or moves a file from one directory to another.

Syntax **Name** *oldfilename$* **As** *newfilename$*

where:	**is:**
oldfilename$	A string expression containing the file to rename.
newfilename$	A string expression containing the name for the file.

Comments A path can be part of either filename argument. If the paths are different, the file is moved to the new directory.

A file must be closed in order to be renamed. If the file *oldfilename$* is open or if the file *newfilename$* already exists, Basic generates an error message.

New Operator

See Also

Allocates and initializes a new OLE Automation object of the named class.

Syntax

Set *objectVar* **= New** *className*

Dim *objectVar* **As New** *className*

where:	**is:**
objectVar	The OLE Automation object to allocate and initialize.
className	The class to assign to the object.

Comments

In the **Dim** statement, **New** marks *objectVar* so that a new object will be allocated and initialized when *objectVar* is first used. If *objectVar* is not referenced, then no new object will be allocated.

Note: An object variable that was declared with **New** will allocate a second object if *objectVar* is **Set** to **Nothing** and referenced again.

$NoCStrings Metacommand [OPEN Script Extension]

See Also **Example**

Tells the compiler to treat a backslash (\) inside a string as a normal character.

Syntax **'$NoCStrings** [**Save**]

where:	**means:**
Save	Saves the current **'$CStrings** setting before restoring the treatment of the backslash (\) to a normal character.

Comments Use the **'$CStings Restore** command to restore a previously saved setting. Save and Restore operate as a stack and allow the user to change the **'$CStrings** setting for a range of the program without impacting the rest of the program.

Use the **'$CStrings** metacommand to tell the compiler to treat a backslash (\) inside of a string as an Escape character.

Nothing Function

See Also **Example** **Overview**

Returns an object value that doesn't refer to an object.

Syntax **Set** *variableName* **= Nothing**

where:	**is:**
variableName	The name of the object variable to set to nothing.

Comments **Nothing** is the value object variables have when they do not refer to an object, either because the have not been initialized yet or because they were explicitly **Set** to **Nothing**. For example:

```
If Not objectVar Is Nothing then
        objectVar.Close
        Set objectVar = Nothing
End If
```

Now Function

See Also **Example**

Returns the current date and time.

Syntax **Now( )**

Comments The **Now** function returns a **Variant** of vartype 7 (date) that represents the current date and time according to the setting of the computer's system date and time.

NPV Function

See Also **Example**

Returns the net present value of a investment based on a stream of periodic cash flows and a constant interest rate.

Syntax **NPV** *(rate , valuearray( ))*

where:	**is:**
rate	Discount rate per period.
valuearray( )	An array containing cash flow values.

Comments *Valuearray( )* must have at least one positive value (representing a receipt) and one negative value (representing a payment). All payments and receipts must be represented in the exact sequence. The value returned by **NPV** will vary with the change in the sequence of cash flows.

If the discount rate is 12% per period, *rate* is the decimal equivalent, i.e. 0.12.

NPV uses future cash flows as the basis for the net present value calculation. If the first cash flow occurs at the beginning of the first period, its value should be added to the result returned by **NPV** and must not be included in *valuearray()*.

Null Function

See Also **Example**

Returns a **Variant** value set to NULL.

Syntax **Null**

Comments **Null** is used to set a Variant to the Null value explicitly, as follows:
variableName **= Null**

Note that Variants are initialized by Basic to the empty value, which is different from the null value.

Object Class

See Also **Example**

A class that provides access to OLE Automation objects.

Syntax **Dim** *variableName* **As Object**

where:	**is:**
variableName	The name of the object variable to declare.

Comments To create a new object, first dimension a variable, using the **Dim** statement, then **Set** the variable to the return value of **CreateObject** or **GetObject**, as follows:

```
Dim OLEobj As Object

Set OLEobj = CreateObject("spoly.cpoly")
```

To refer to a method or property of the newly created object, use the syntax: *objectvar.property* or *objectvar.method*, as follows:

```
OLEobj.reset
```

Oct Function

See Also **Example**

Returns the octal representation of a number, as a string.

Syntax **Oct**[**$**]**(** *number* **)**

where:	**is:**
number	A numeric expression for the number to convert to octal.

Comments If the numeric expression has a data type of **Integer**, the string contains up to six octal digits; otherwise, the expression will be converted to a data type of **Long**, and the string can contain up to 11 octal digits.

To represent an octal number directly, precede the octal value with **&O**. For example, &O10 equals decimal 8 in octal notation.

The dollar sign, "$", in the function name is optional. If specified the return data type is **String**. If omitted the function will return a **Variant** of vartype 8 (string).

OKButton Statement

See Also **Example**

Determines the position and size of an OK button in a dialog box.

Syntax **OKButton** *x , y , dx , dy* [, **.***id*]

where:	**is:**
x , y	The position of the Cancel button relative to the upper left corner of the dialog box.
dx , dy	The width and height of the button.
.id	An optional identifier for the button.

Comments A *dy* value of 14 typically accommodates text in the system font.

.id is an optional identifier used by the dialog statements that act on this control.

Use the **OKButton** statement only between a **Begin Dialog** and an **End Dialog** statement.

On...Goto Statement

See Also **Example**

Branch to a label in the current procedure based on the value of a numeric expression.

Syntax **ON** *numeric-expression* **GoTo** *label1* [,*label2 , ...*]

where:	**is:**
numeric-expression	Any numeric expression that evaluates to a positive number.
label1 , label2	A label in the current procedure to branch to if *numeric-expression* evaluates to 1, 2, etc.

Comments If *numeric expression* evaluates to 0 or to a number greater than the number of labels following **GoTo**, the program continues at the next statement. If *numeric-expression* evaluates to a number less than 0 or greater than 255, an "Illegal function call" error is issued.

On Error Statement

See Also **Example** **Overview**

Specifies the location of an error-handling routine within the current procedure.

Syntax **ON [Local] Error** {**GoTo** *label* [**Resume Next**] **GoTo 0**}

where:	**is:**
label	A string used as a label in the current procedure to identify the lines of code that process errors.

Comments **On Error** can also be used to disable an error-handling routine. Unless an **On Error** statement is used, any run-time error will be fatal, that is, OPEN Script will terminate the execution of the program.

An **On Error** statement is composed of the following parts:

Part	**Definition**
Local	Keyword allowed in error-handling routines at the procedure level. Used to ensure compatibility with other Variants of Basic.
GoTo label	Enables the error-handling routine that starts at label. If the designated label is not in the same procedure as the On Error statement, OPEN Script generates an error message.
Resume Next	Designates that error-handling code is handled by the statement that immediately follows the statement that caused an error. At this point, use the Err function to retrieve the error-code of the run-time error.
GoTo 0	Disables any error handler that has been enabled.

When it is referenced by an **On Error GoTo** *label* statement, an error-handler is enabled. Once this enabling occurs, a run-time error will result in program control switching to the error-handling routine and "activating" the error handler. The error handler remains active from the time the run-time error has been trapped until a **Resume** statement is executed in the error handler.

If another error occurs while the error handler is active, OPEN Script will search for an error handler in the procedure that called the current procedure (if this fails, OPEN Script will look for a handler belonging to the caller's caller, and so on). If a handler is found, the current procedure will terminate, and the error handler in the calling procedure will be activated.

It is an error (No Resume) to execute an **End Sub** or **End Function** statement while an error handler is active. The **Exit Sub** or **Exit Function** statement can be used to end the error condition and exit the current procedure.

Open Statement

See Also **Example**

Opens a file or device for input or output.

Syntax **Open** *filename$* [**For** *mode*] [**Access** *access*] [*lock*] **As [#]** *filenumber%* [**Len =** *reclen*]

where:	**is:**	
filename$	A string or string expression for the name of the file to open.	
mode	One of the following keywords:	
	Input	Put data into the file sequentially.
	Output	Read data from the file sequentially.
	Append	Add data to the file sequentially.
	Random	Get data from the file by random access.
	Binary	Get binary data from the file.
access	One of the following keywords:	
	Read	Read data from the file only.
	Write	Write data the file only.
	Read Write	Read or write data to the file.
lock	One of the following keywords to designate access by other processes:	
	Shared	Read or write available on the file.
	Lock Read	Read data only.
	Lock Write	Write data only.
	Lock Read Write	No read or write available.
filenumber%	An integer or expression containing the integer to assign to the open file (between 1 and 255).	
reclen	The length of the records (for Random or Binary files only).	

Comments A file must be opened before any input/output operation can be performed on it.

If *filename$* does not exist, it is created when opened in **Append**, **Binary**, **Output** or **Random** modes.

If *mode* is not specified, it defaults to **Random**.

If *access* is not specified for **Random** or **Binary** modes, *access* is attempted in the following order: **Read Write**, **Write**, **Read**.

If *lock* is not specified, *filename$* can be opened by other processes that do not specify a *lock*, although that process cannot perform any file operations on the file while the original process still has the file open.

Use the **FreeFile** function to find the next available value for *filenumber%*.

Reclen is ignored for **Input**, **Output**, and **Append** *modes*.

OptionButton Statement

See Also **Example**

Defines the position and text associated with an option button in a dialog box.

Syntax **OptionButton** *x , y , dx , dy , text$* [, **.***id*]

where:	**is:**
x , y	The position of the button relative to the upper left corner of the dialog box.
dx , dy	The width and height of the button.
text$	A string to display next to the option button. If the width of this string is greater than *dx*, trailing characters are truncated.
.id	An optional identifier used by the dialog statements that act on this control.

Comments You must have at least two **OptionButton** statements in a dialog box. You use these statements in conjunction with the **OptionGroup** statement.

A *dy* value of 12 typically accommodates text in the system font.

To enable the user to select an option button by typing a character from the keyboard, precede the character in *text$* with an ampersand (&).

Use the **OptionButton** statement only between a **Begin Dialog** and an **End Dialog** statement.

OptionGroup Statement

See Also **Example**

Groups a series of option buttons under one heading in a dialog box.

Syntax **OptionGroup** *.field*

where:	**is:**
.field	A value for the option button selected by the user: 0 for the first option button, 1 for the second button, and so on.

Comments The **OptionGroup** statement is used in conjunction with **OptionButton** statements to set up a series of related options. The **OptionGroup** Statement begins the definition of the option buttons and establishes the dialog-record field that will contain the option selection.

Use the **OptionGroup** statement only between a **Begin Dialog** and an **End Dialog** statement.

Option Base Statement

See Also **Example** **Overview**

Specifies the default lower bound to use for array subscripts.

Syntax **Option Base** *lowerBound%*

where:	**is:**
lowerBound	A number or expression containing a number for the default lower bound: either 0 or 1.

Comments If no **Option Base** statement is specified, the default lower bound for array subscripts will be 0.

The **Option Base** statement is ***not*** allowed inside a procedure, and must precede any use of arrays in the module. Only one **Option Base** statement is allowed per module.

Option Compare Statement

See Also **Example**

Specifies the default method for string comparisons: either case-sensitive or case-insensitive.

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

where:	**means:**
Binary	Comparisons are case-sensitive (i.e., lowercase and uppercase letters are different).
Text	Comparisons are not case-sensitive.

Comments **Binary** comparisons compare strings based upon the ANSI character set. **Text** comparisons are based upon the relative order of characters as determined by the country code setting for your system.

Option Explicit Statement

See Also **Example**

Specifies that all variables in a module ***must*** be explicitly declared.

Syntax **Option Explicit**

Comments By default, Basic automatically declares any variables that do not appear in a **Dim**, **Global**, **Redim**, or **Static** statement. **Option Explicit** causes such variables to produce a "Variable Not Declared" error.

PasswordBox Function

See Also **Example**

Returns a string entered by the user without echoing it to the screen.

Syntax **PasswordBox**[**$**]**(** *prompt$* ,[*title$*] ,[*default$*] [*,xpos% , ypos%*] **)**

where:	**is:**
prompt$	A string expression containing the text to show in the dialog box
title$	The caption for the dialog box's title bar
default$	The string expression shown in the edit box as the default response.
xpos% , ypos%	The position of the dialog box, relative to the upper left corner of the screen.

Comments The **PasswordBox** function displays a dialog box containing a prompt. Once the user has entered text, or made the button choice being prompted for, the contents of the box are returned.

The length of *prompt$* is restricted to 255 characters. This figure is approximate and depends on the width of the characters used. Note that a carriage return and a line-feed character must be included in *prompt$* if a multiple-line prompt is used.

If either *prompt$* or *default$* is omitted, nothing is displayed.

Xpos% determines the horizontal distance between the left edge of the screen and the left border of the dialog box, measured in dialog box units. *Ypos%* determines the horizontal distance from the top of the screen to the dialog box's upper edge, also in dialog box units. If these arguments are not entered, the dialog box is centered roughly one third of the way down the screen. A horizontal dialog box unit is 1/4 of the average character width in the system font; a vertical dialog box unit is 1/8 of the height of a character in the system font.

Note: To specify the dialog box's position, you must enter both of these arguments. If you enter one without the other, the default positioning is used.

Once the user presses Enter, or selects the OK button, **PasswordBox** returns the text contained in the password box. If the user selects Cancel, the **PasswordBox** function returns a null string ("").

The dollar sign, "$", in the function name is optional. If specified the return type is string. If omitted, the function will return a **Variant** of vartype 8 (string).

Picture Statement

See Also **Example**

Defines a picture control in a dialog box

Syntax **Picture** *x , y , dx , dy , filename$, type* [, **.***id*]

where:	**is:**
x , y	The position of the picture relative to the upper left corner of the dialog box.
dx , dy	The width and height of the picture.
filename$	The name of the bitmap file (a file with .BMP extension) where the picture is located.
type	An integer for the location of the bitmap (0=*filename$*, 3=Windows Clipboard).
.id	An optional identifier used by the dialog statements that act on this control.

Comments The **Picture** statement can only be used between a **Begin Dialog** and an **End Dialog** statement.

Note: The picture will be scaled equally in both directions and centered if the dimensions of the picture are not proportional to *dx* and *dy.*

If *type%* is 3, *filename$* is ignored.

If the picture is not available (the file *filename$* doesn't exist, doesn't contain a bitmap, or there is no bitmap on the Clipboard), the picture control will display the picture frame and the text "(missing picture)". This behavior can be changed by adding 16 to the value of *type%.* If *type%* is 16 or 19 and the picture is not available, a runtime error occurs.

Pmt Function

See Also **Example**

Returns a constant periodic payment amount for an annuity or a loan.

Syntax **Pmt** *(rate , nper , pv , fv , due)*

where:	**is:**
rate	Interest rate per period.
nper	Total number of payment periods.
pv	Present value of the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).
fv	Future value of the final lump sum amount required (as in the case of a savings plan) or paid (0 as in the case of a loan).
due	An integer value for when the payments are due (0=end of each period, 1= beginning of the period).

Comments *Rate* is assumed to be constant over the life of the loan or annuity. If payments are on a monthly schedule, then *rate* will be 0.0075 if the annual percentage rate on the annuity or loan is 9%.

PPmt Function

See Also **Example**

Returns the principal portion of the payment for a given period of an annuity.

Syntax **PPmt** *(rate , per , nper , pv , fv , due)*

where:	**is:**
rate	Interest rate per period.
per	Particular payment period in the range 1 through *nper*.
nper	Total number of payment periods.
pv	Present value of the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).
fv	Future value of the final lump sum amount required (as in the case of a savings plan) or paid (0 as in the case of a loan).
due	An integer value for when the payments are due (0=end of each period, 1= beginning of the period).

Comments *Rate* is assumed to be constant over the life of the loan or annuity. If payments are on a monthly schedule, then *rate* will be 0.0075 if the annual percentage rate on the annuity or loan is 9%.

Print Statement

See Also **Example**

Prints data to an open file or to the screen.

Syntax **Print** [*filenumber% ,*] *expressionlist* [{ ; | , }]

where:	**is:**
filenumber%	An integer expression identifying the open file to use.
expressionlist	A numeric, string, and Variant expression containing the list of values to print.

Comments The **Print** statement outputs data to the specified *filenumber%*. *filenumber%* is the number assigned to the file when it was opened. See the **Open** statement for more information. If this argument is omitted, the **Print** statement outputs data to the screen.

If the *expressionlist* is omitted, a blank line is written to the file.

The values in *expressionlist* are separated by either a semi-colon (";") or a comma (",") . A semi-colon indicates that the next value should appear immediately after the preceding one without intervening white space. A comma indicates that the next value should be positioned at the next print zone. Print zones begin every 14 spaces.

The optional [{;|,}] argument at the end of the **Print** statement determines where output for the next **Print** statement to the same output file should begin. A semi-colon will place output immediately after the output from this **Print** statement on the current line; a comma will start output at the next print zone on the current line. If neither separator is specified, a CR-LF pair will be generated and the next **Print** statement will print to the next line.

Special functions **Spc** and **Tab** can be used inside **Print** statement to insert a given number of spaces and to move the print position to a desired column.

The **Print** statement supports only elementary Basic data types. See **Input** for more information on parsing this statement.

PushButton Statement

See Also **Example**

Defines a custom push button.

Syntax A **PushButton** *x , y , dx , dy , text$* [, **.***id*]

Syntax B **Button** *x, y, dx, dy, text$* [, **.***id*]

where:	**is:**
x , y	The position of the button relative to the upper left corner of the dialog box.
dx , dy	The width and height of the button.
text$	The name for the push button. If the width of this string is greater than *dx*, trailing characters are truncated.
.id	An optional identifier used by the dialog statements that act on this control.

Comments A *dy* value of 14 typically accommodates text in the system font.

Use this statement to create buttons other than OK and Cancel. Use this statement in conjunction with the **ButtonGroup** statement. The two forms of the statement (**Button** and **PushButton**) are equivalent.

Use the **Button** statement only between a **Begin Dialog** and an **End Dialog** statement.

Put Statement

See Also **Example**

Writes a variable to a file opened in **Random** or **Binary** mode.

Syntax **Put** [#] *filenumber%*, [*recnumber&*], *varname*

where:	**is:**
filenumber%	An integer expression identifying the open file to use.
recnumber&	A **Long** expression containing the record number or the byte offset at which to start writing.
varname	The name of the variable containing the data to write.

Comments *Filenumber%* is the number assigned to the file when it was opened. See the **Open** statement for more information.

Recnumber& is in the range 1 to 2,147,483,647. If *recnumber&* is omitted, the next record or byte is written.

Note: The commas before and after *recnumber%* are ***required***, even if no *recnumber&* is specified.

Varname can be any variable except **Object**, **Application Data Type** or **Array** variables (single array elements can be used).

For **Random** mode, the following apply:

Blocks of data are written to the file in chunks whose size is equal to the size specified in the **Len** clause of the **Open** statement. If the size of *varname* is smaller than the record length, the record is padded to the correct record size. If the size of variable is larger than the record length, an error occurs.

For variable length String variables, **Put** writes two bytes of data that indicate the length of the string, then writes the string data.

For Variant variables, **Put** writes two bytes of data that indicate the type of the Variant, then it writes the body of the Variant into the variable. Note that Variants containing strings contain two bytes of type information, followed by two bytes of length, followed by the body of the string.

User defined types are written as if each member were written separately, except no padding occurs between elements.

Files opened in **Binary** mode behave similarly to those opened in **Random** mode except:

Put writes variables to the disk without record padding.

Variable length **Strings** that are not part of user defined types are not preceded by the two byte string length.

PV Function

See Also **Example**

Returns the present value of a constant periodic stream of cash flows as in an annuity or a loan.

Syntax **PV** *(rate , nper , pmt , fv , due)*

where:	**is:**
rate	Interest rate per period.
nper	Total number of payment periods.
pmt	Constant periodic payment per period.
fv	Future value of the final lump sum amount required (in the case of a savings plan) or paid (0 in the case of a loan).
due	An integer value for when the payments are due (0=end of each period, 1= beginning of the period).

Comments *Rate* is assumed constant over the life of the annuity. If payments are on a monthly schedule, then *rate* will be 0.0075 if the annual percentage rate on the annuity or loan is 9%.

Randomize Statement

See Also **Example**

Seeds the random number generator.

Syntax **Randomize** [*number%*]

where:	**is:**
number%	An integer value between -32768 and 32767.

Comments If no *number%* argument is given, Basic uses the **Timer** function to initialize the random number generator.

Rate Function

See Also **Example**

Returns the interest rate per period for an annuity or a loan.

Syntax **Rate** *(nper , pmt , pv , fv , due , guess)*

where:	**is:**
nper	Total number of payment periods.
pmt	Constant periodic payment per period.
pv	Present value of the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan).
fv	Future value of the final lump sum amount required (in the case of a savings plan) or paid (0 in the case of a loan).
due	An integer value for when the payments are due (0=end of each period, 1= beginning of the period)
guess	A ballpark estimate for the rate returned.

Comments In general, a guess of between 0.1 (10 percent) and 0.15 (15 percent) would be a reasonable value for *guess*.

Rate is an iterative function: it improves the given value of *guess* over several iterations until the result is within 0.00001 percent. If it does not converge to a result within 20 iterations, it signals failure.

ReDim Statement

See Also **Example** **Overview**

Changes the upper and lower bounds of a dynamic array's dimensions.

Syntax **ReDim** [**Preserve**] *variableName* **(** *subscriptRange , ...* **)** [**As** [**New**] *type*] , ...

where:	**is:**
variableName	The variable array name to redimension.
subscriptRange	The new upper and lower bounds for the array.
type	The type for the data elements in the array.

Comments **ReDim** re-allocates memory for the dynamic array to support the specified dimensions, and can optionally re-initialize the array elements. **ReDim** cannot be used at the module level; it must be used inside of a procedure.

The **Preserve** option is used to change the last dimension in the array while maintaining its contents. If **Preserve** is not specified, the contents of the array are re-initialized. Numbers will be set to zero (0). Strings and Variants will be set to empty ("").

The *subscriptRange* is of the format:

[*startSubscript* **To**] *endSubscript*

If *startSubscript* is not specified, 0 is used as the default. The **Option Base** statement can be used to change the default.

A dynamic array is normally created by using **Dim** to declare an array without a specified *subscriptRange*. The maximum number of dimensions for a dynamic array created in this fashion is 8. If you need more than 8 dimensions, you can use the **ReDim** statement inside of a procedure to declare an array that has not previously been declared using **Dim** or **Global**. In this case, the maximum number of dimensions allowed is 60.

The available data types for arrays are: numbers, strings, Variants, records and objects. Arrays of arrays, dialog box records, and objects are not supported.

If the **As** clause is not used, the type of the variable can be specified by using a type character as a suffix to the name. The two different type-specification methods can be intermixed in a single **ReDim** statement (although not on the same variable).

The **ReDim** statement cannot be used to change the number of dimensions of a dynamic array once the array has been given dimensions. It can only change the upper and lower bounds of the dimensions of the array. The **LBound** and **UBound** functions can be used to query the current bounds of an array variable's dimensions.

Care should be taken to avoid **ReDim**'ing an array in a procedure that has received a reference to an element in the array in an argument; the result is unpredictable.

Rem Statement

Example

Identifies a line of code as a comment in a Basic program.

Syntax **Rem** *comment*

where:	**is:**
comment	The text of the comment.

Comments Everything from **Rem** to the end of the line is ignored.

The single quote (') can also be used to initiate a comment. Metacommands (e.g., **$CSTRINGS**) must be preceded by the single quote comment form.

Reset Statement

See Also **Example**

Closes all open disk files and writes any data in the operating system buffers to disk.

Syntax **Reset**

Resume Statement

See Also **Example** **Overview**

Halts an error-handling routine.

Syntax A **Resume Next**

Syntax B **Resume** *label*

Syntax C **Resume** [**0**]

where:	**is:**
label	The label that identifies the statement to go to after handling an error.

Comments When the **Resume Next** statement is used, control is passed to the statement that immediately follows the statement in which the error occurred.

When the **Resume [0]** statement is used, control is passed to the statement in which the error occurred.

The location of the error handler that has caught the error determines where execution will resume. If an error is trapped in the same procedure as the error handler, program execution will resume with the statement that caused the error. If an error is located in a different procedure from the error handler, program control reverts to the statement that last called out the procedure containing the error handler.

Right Function

See Also **Example**

Returns a string of a specified length copied from the end of another string.

Syntax **Right**[**$**]**(** *string$***,** *length%* **)**

where:	**is:**
string$	A string or expression containing the string to copy.
length%	The number of characters to copy.

Comments If *length%* is greater than the length of *string$,* **Right** returns the whole string.

Right accepts any type of *string$,* including numeric values, and will convert the input value to a string.

The dollar sign, "$", in the function name is optional. If specified, the return type is string. If omitted, the function will typically return a **Variant** of vartype 8 (string). If the value of *string$* is NULL, a Variant of vartype 1 (Null) is returned.

RmDir Statement

See Also **Example**

Removes a directory.

Syntax **RmDir** *path$*

where:	**is:**
path$	A string expression identifying the directory to remove.

Comments The syntax for *path$* is:

[*drive*:] [\] *directory* [*directory*]

The *drive* argument is optional. The *directory* argument is a directory name.

The directory to be removed must be empty, except for the working (.) and parent (..) directories.

Rnd Function

See Also **Example**

Returns a single precision random number between 0 and 1.

Syntax **Rnd** [**(** *number!* **)**]

where:	**is:**
number!	A numeric expression to specify how to generate the random numbers. (<0=use the number specified, >0=use the next number in the sequence, 0=use the number most recently generated.)

Comments If *number!* is omitted, **Rnd** uses the next number in the sequence to generate a random number. The same sequence of random numbers is generated whenever **Rnd** is run, unless the random number generator is re-initialized by the **Randomize** statement.

Rset Statement

See Also **Example**

Right aligns one string inside another string.

Syntax **Rset** *string$* **=** *string-expression*

where:	**is:**
string$	The string to contain the right-aligned characters.
string-expression	The string containing the characters to put into *string$*.

Comments If *string$* is longer than *string-expression,* the leftmost characters of *string$* are replaced with spaces.

If *string$* is shorter than *string-expression,* only the leftmost characters of *string-expression* are copied.

Rset cannot be used to assign variables of different user-defined types.

RTrim Function

See Also **Example**

Copies a string and removes any trailing spaces.

Syntax **RTrim**[**$**]**(** *string$* **)**

where:	**is:**
string$	An expression that evaluates to a string.

Comments **RTrim** accepts any type of *string* including numeric values and will convert the input value to a string.

The dollar sign, "$", in the function name is optional. If specified the return type is string. If omitted the function will typically return a **Variant** of vartype 8 (string). If the value of *string* is NULL, a Variant of vartype 1 (Null) is returned.

Second Function

See Also **Example**

Returns the second component (0-59) of a date-time value.

Syntax **Second(** *time* **)**

where:	**is:**
time	An expression containing a date time value.

Comments **Second** accepts any type of *time* including strings and will attempt to convert the input value to a date value.

The return value is a **Variant** of vartype 2 (integer). If the value of *time* is NULL, a Variant of vartype 1 (Null) is returned.

Seek Function

See Also **Example**

Returns the current file position for an open file.

Syntax **Seek(** *filenumber%* **)**

where:	**is:**
filenumber%	An integer expression identifying an open file to query.

Comments *Filenumber%* is the number assigned to the file when it was opened. See the **Open** statement for more information.

For files opened in **Random** mode, **Seek** returns the number of the next record to be read or written. For all other modes, **Seek** returns the file offset for the next operation. The first byte in the file is at offset 1, the second byte is at offset 2, etc. The return value is a **Long**.

Seek Statement

See Also **Example**

Sets the position within an open file for the next read or write operation.

Syntax **Seek** [#] *filenumber% , position&*

where:	**is:**
filenumber%	An integer expression identifying an open file to query.
position&	A numeric expression for the starting position of the next read or write operation (record number or byte offset).

Comments The **Seek** statement. If you write to a file after seeking beyond the end of the file, the file's length is extended. Basic will return an error message if a **Seek** operation is attempted that specifies a negative or zero position.

Filenumber% is an integer expression identifying the open file to **Seek** in. See the **Open** statement for more details.

For files opened in **Random** mode, *position&* is a record number; for all other modes, *position&* is a byte offset. *Position&* is in the range 1 to 2,147,483,647. The first byte or record in the file is at position 1, the second is at position 2, etc.

Select Case Statement

See Also **Example**

Executes a series of statements, depending on the value of an expression.

Syntax

Select Case *testexpression*
[**Case** *expressionlist*
[*statement_block*]]
[**Case** *expressionlist*
[*statement_block*]]
.
.
[**Case Else**
[*statement_block*]]
End Select

where:	**is:**
testexpression	Any expression containing a variable to test.
expressionlist	One or more expressions that contain a possible value for *testexpression* .
statement_block	The statements to execute if *testexpression* equals *expressionlist*.

Comments

When there is a match between *testexpression* and one of the values in *expressionlist*, the *statement_block* following the **Case** clause is executed. When the next **Case** clause is reached, execution control goes to the statement following the **End Select** statement.

The *expressionlist(s)* can be a comma-separated list of expressions of the following forms:

expression

expression **To** *expression*

Is *comparison_operator expression*

The type of each *expression* must be compatible with the type of *testexpression*.

Note that when the **To** keyword is used to specify a range of values, the smaller value must appear first. The *comparison_operator* used with the **Is** keyword is one of: <, >, =, <=, >=, <>.

Each *statement_block* can contain any number of statements on any number of lines.

SendKeys Statement

See Also **Example**

Send keystrokes to an active Windows application.

Syntax **SendKeys** *string$* [, *wait%*]

where:	**is:**
string$	An expression containing the characters to send.
wait%	A numeric expression to determine whether to wait until all keys are processed before continuing program execution (-1=wait, 0=don't wait).

Comments The keystrokes are represented by characters of *string.*

The default value for *wait* is 0 (FALSE).

To specify an ordinary character, enter this character in the *string*. For example, to send character 'a' use "a" as *string.* Several characters can be combined in one string: *string* "abc" means send 'a', 'b', and 'c'.

To specify that Shift, Alt, or Control keys should be pressed simultaneously with a character, prefix the character with

+ to specify Shift

% to specify Alt

^ to specify Control.

Parentheses can be used to specify that the Shift, Alt, or Control key should be pressed with a group of characters. For example, "%(abc)" is equivalent to "%a%b%c".

Since '+', '%', '^' ,'(' and ')' characters have special meaning to **SendKeys**, they must be enclosed in braces if they need to be sent with **SendKeys**. For example *string* "{%}" specifies a percent character '%'.

The other characters that need to be enclosed in braces are '~' which stands for a newline or "Enter" if used by itself and braces themselves: use {{} to send '{' and {}} to send '}'. Brackets '[' and ']' do not have special meaning to **SendKeys** but might have special meaning in other applications, therefore, they need to be enclosed inside braces as well.

To specify that a key needs to be sent several times, enclose the character in braces and specify the number of keys sent after a space: for example, use {X 20} to send 20 'X' characters.

To send one of the non-printable keys use a special keyword inside braces:

Key	**Keyword**
Backspace	{BACKSPACE} or {BKSP} or {BS}
Break	{BREAK}
Caps Lock	{CAPSLOCK}
Clear	{CLEAR}
Delete	{DELETE} or {DEL}
Down Arrow	{DOWN}
End	{END}
Enter	{ENTER}
Esc	{ESCAPE} or {ESC}
Help	{HELP}
Home	{HOME}
Insert	{INSERT}
Left Arrow	{LEFT}
Num Lock	{NUMLOCK}

Page Down	{PGDN}
Page Up	{PGUP}
Right Arrow	{RIGHT}
Scroll Lock	{SCROLLLOCK}
Tab	{TAB}
Up Arrow	{UP}

To send one of function keys F1-F15, simply enclose the name of the key inside braces. For example, to send F5 use "{F5}"

Note that special keywords can be used in combination with +, %, and ^. For example: %{TAB} means Alt-Tab. Also, you can send several special keys in the same way as you would send several normal keys: {UP 25} sends 25 Up arrows.

SendKeys can send keystrokes only to the currently active application. Therefore, you have to use the **AppActivate** statement to activate an application before sending keys (unless it is already active).

SendKeys cannot be used to send keys to an application that was not designed to run under Windows.

Set Statement

See Also **Example** **Overview**

Assigns a variable to an OLE Automation object.

Syntax **Set** *variableName* **=** *expression*

where:	**is:**
variableName	An object variable or a Variant variable.
expression	An expression that evaluates to an object--typically a function, an object member, or **Nothing**.

Comments The following example shows the syntax for the **Set** statement:

```
Dim OLEobj As Object
Set OLEobj = CreateObject("spoly.cpoly")
OLEobj.reset
```

Note: If you omit the keyword **Set** when assigning an object variable, Basic will try to copy the default member of one object to the default member of another. This usually results in a runtime error:

```
' Incorrect code - tries to copy default member!
OLEobj = GetObject( ,"spoly.cpoly")
```

Set differs from **Let** in that Let assigns an expression to a Basic variable. For example,

Set o1 = o2	will set the object reference.
Let o1 = o2	will set the value of the default member.

SetAttr Statement

See Also **Example**

Sets the attributes for a file.

Syntax **SetAttr** *pathname$, attributes%*

where:	**is:**
pathname$	A string expression containing the filename to modify.
attributes %	An integer containing the new attributes for the file.

Comments Wildcards are not allowed in *pathname$*. If the file is open, you can modify its attributes, but only if it is opened for **Read** access. Here is a description of attributes that can be modified:

Value	**Meaning**
0	Normal file
1	Read-only file
2	Hidden file
4	System file
32	Archive - file has changed since last backup

SetField Function [OPEN Script Extension]

See Also **Example**

Replaces a field within a string and returns the modified string.

Syntax **SetField**[**$**]**(** *string$***,** *field_number%***,** *field$***,** *separator_chars$* **)**

where:	**is:**
string$	A string consisting of a series of fields, separated by *separator_char$*.
field_number%	An integer for the field to replace within *string$*.
field$	An expression containing the new value for the field.
separator_char$	A string containing the character(s) used to separate the fields in *string$*.

Comments *separator_char$* can contain multiple separator characters, although the first one will be used as the separator character.

The *field_number%* starts with 1. If *field_number%* is greater than the number of fields in the string, the returned string will be extended with separator characters to produce a string with the proper number of fields.

It is legal for the new *field$* value to be a different size than the old value.

Sgn Function

See Also **Example**

Returns a value indicating the sign of a number.

Syntax **Sgn(** *number* **)**

where:	**is:**
number	An expression for the number to use.

Comments The value that the **Sgn** function returns depends on the sign of *number.*

For *numbers* > 0, **Sgn** (*number*) returns 1.

For *numbers* = 0, **Sgn** (*number*) returns 0.

For *numbers* < 0, **Sgn** (*number*) returns -1.

Shell Function

See Also **Example**

Starts a Windows application and returns its task ID.

Syntax **Shell(** *pathname$,* [*windowstyle%*] **)**

where:	**is:**
pathname$	The name of the program to execute
windowstyle%	An integer value for the style of the program's window (1-7).

Comments **Shell** runs an executable program. *Pathname$* can be the name of any valid .COM, .EXE., .BAT, or .PIF file. Arguments or command line switches can be included. If *pathname$* is not a valid executable filename, or if **Shell** cannot start the program, an error message occurs.

Windowstyle% is one of the following values:

Value	**Window Style**
1	Normal window with focus
2	Minimized with focus
3	Maximized with focus
4	Normal window without focus
7	Minimized without focus

If *windowstyle%* is not specified, the default of *windowstyle%* = 1 is assumed (normal window with focus).

Shell returns the task ID for the program, a unique number that identifies the running program.

Sin Function

See Also **Example**

Returns the sine of an angle specified in radians.

Syntax **Sin**(*number*)

where:	**is:**
number	An expression containing the angle in radians.

Comments The return value will be between -1 and 1. The return value is single-precision if the angle is an integer, currency or single-precision value, double precision for a long, Variant or double-precision value. The angle is specified in radians, and can be either positive or negative.

To convert degrees to radians, multiply by (PI/180). The value of PI is 3.14159.

Space Function

See Also **Example**

Returns a string of spaces.

Syntax **Space**[**$**]**(** *number* **)**

where:	**is:**
number	A numeric expression for the number of spaces to return.

Comments *number* can be any numeric data type, but will be rounded to an integer. *number* must be between 0 and 32,767.

The dollar sign, "$", in the function name is optional. If specified the return type is String. If omitted, the function will return a **Variant** of vartype 8 (String).

Spc Function

See Also **Example**

Prints a number of spaces.

Syntax **Spc (** *n* **)**

where:	**is:**
n	An integer for the number of spaces to output.

Comments The **Spc** function can be used only inside **Print** statement.

When the **Print** statement is used, the **Spc** function will use the following rules for determining the number of spaces to output:

1. If *n* is less than the total line width, **Spc** outputs *n* spaces.
2. If *n* is greater than the total line width, **Spc** outputs *n* **Mod** *width* spaces.
3. If the difference between the current print position and the output line width (call this difference x) is less than *n* or *n* **Mod** *width*, then **Spc** skips to the next line and outputs *n* - x spaces.

To set the width of a print line, use the **Width** statement.

SQLClose Function

See Also **Example**

Disconnects from an ODBC data source connection that was established by **SQLOpen**.

Syntax **SQLClose (** *connection&* **)**

where:	**is:**
connection&	A named argument that must be a long integer, returned by **SQLOpen**.

Comments The return is a variant. Success returns 0 and the connection is subsequently invalid. If the connection is not valid, -1 is returned.

SQLError Function

See Also **Example**

Can be used to retrieve more detailed information about errors that might have occurred when making an ODBC function call. Returns errors for the last ODBC function and the last connection.

Syntax **SQLError (** *destination()* **)**

where:	**is:**
destination	A two dimensional array in which each row contains one error. A named argument that is required, must be an array of variants.

Comments There is no return value. The fields are: 1) character string indicating the ODBC error class/subclass, 2) numeric value indicating the data source native error code, 3) text message describing the error.

If there are no errors from a previous ODBC function call, then a 0 is returned in the callers array at (1,1). If the array is not two dimensional or does not provide for the return of the three fields above, then an error message is returned in the callers array at (1,1).

SQLExecQuery Function

See Also **Example**

Executes an SQL statement on a connection established by **SQLOpen**.

Syntax **SQLExecQuery (** *connection& , query$* **)**

where:	**is:**
connection&	A named argument, required. A long integer, returned by **SQLOpen**.
query$	A string containing a valid SQL statement. The return is a variant.

Comments It returns the number of columns in the result set for SQL SELECT statements; for UPDATE, INSERT, or DELETE it returns the number of rows affected by the statement. Any other SQL statement returns 0. If the function is unable to execute the query on the specified data source, or if the connection is invalid, a negative error code is returned.

If **SQLExecQuery** is called and there are any pending results on that connection, the pending results are replaced by the new results.

SQLGetSchema Function

See Also **Example**

Returns a variety of information, including information on the data sources available, current user ID, names of tables, names and types of table columns, and other data source/database related information.

Syntax **SQLGetSchema (***connection , action% , qualifier$, ref()* **)**

where:	**is:**
connection	A long integer returned by **SQLOpen.**
action%	Required.
qualifier$	Required.
ref()	A variant array for the results appropriate to the action requested, must be an array even if only one dimension with one element. The return is a variant.

Comments A negative return value indicates an error. A -1 is returned if the requested information cannot be found or if the connection is not valid. The destination array must be properly dimensioned to support the action or an error will be returned. Actions 2 and 3 are not currently supported. Action 4 returns all tables and does not support the use of the *qualifier*. Not all database products and ODBC drivers support all actions.

Action	**Meaning**
1	List of available datasources (dimension of *ref()* is one)
2	List of databases on the current connection (not supported)
3	List of owners in a database on the current connection (not supported)
4	List of tables on the specified connection
5	List of columns in a the table specified by *qualifier. (ref()* must be two dimensions). Returns column name and SQL data type.
6	The user ID of the current connection user.
7	The name of the current database.
8	The name of the data source for the current connection.
9	The name of the DBMS the data source users (e.g., Oracle).
10	The server name for the data source.
11	The terminology used by the data source to refer to owners.
12	The terminology used by the data source to refer to a table.
13	The terminology used by the data source to refer to a qualifier.
14	The terminology used by the data source to refer to a procedure.

SQLOpen Function

See Also **Example**

Establishes a connection to an ODBC data source specified in *connectStr* and returns a connection ID in the return, and the completed connection string in *outputStr.* If the connection cannot be established, then a negative number ODBC error is returned.

Syntax **SQLOpen (** *connectStr$, outputStr$, prompt%* **)**

where:	**is:**
connectStr	A named argument, a required parameter.
outputStr	Optional
prompt	Optional.

Comments The content of *connectStr* is described in the Microsoft Programmers Reference Guide for ODBC. An example string might be DSN=datasourcename; UID=myid; PWD=mypassword. The return must be a long.

prompt specifies when the driver dialog box is displayed. When *prompt* is omitted, **SQLOpen** uses 2 as the default.

Prompt Value	**Meaning**
1	Driver dialog is always displayed.
2	Driver dialog is displayed only when the specification is not sufficient to make the connection.
3	The same as 2, except that dialogs that are not required are grayed and cannot be modified.
4	Driver dialog is not displayed. If the connection is not successful, an error is returned.

SQLRequest Function

See Also **Example**

Establishes a connection to the data source specified in *connectionStr,* executes the SQL statement contained in *query,* returns the results of the request in the *ref()* array, and closes the connection.

Syntax **SQLRequest(** *connectionStr$, query$, outputStr$, prompt% , columnNames% , ref()* **)**

where:	**is:**
connectionStr$	A required argument.
query$	A required argument.
outputStr$	Contains the completed connection string.
prompt%	An integer that specifies when driver dialog boxes are displayed (see **SQLOpen**).
columnNames%	An integer with a value of 0 or nonzero. When *columnNames* is nonzero, column names are returned as the first row of the *ref()* array. If *columnNames* is omitted, the default is 0.
ref()	A required argument that is a two dimensional variant array.

Comments In the event that the connection cannot be made, the query is invalid, or other error condition, a negative number error is returned. In the event the request is successful, the positive number of results returned or rows affected is returned. Other SQL statements return 0.

The arguments are named arguments. The return is a variant.

SQLRetrieve Function

See Also **Example**

Fetches the results of a pending query on the connection specified by *connection* and returns the results in the *destination()* array.

Syntax **SQLRetrieve(** *connection& , destination() , maxColumns% , maxRows% , columnNames% , rowNumbers% , fetchFirst%* **)**

where:	**is:**
connection&	A long.
destination()	A two dimensional variant array.
maxColumns%	An integer and an optional parameter, used to specify the number of columns to be retrieved in the request.
maxRows%	An integer and an optional parameter, used to specify the number of rows to be retrieved in the request.
columnNames%	An integer and an optional parameter, defaults to 0.
rowNumbers%	An integer and an optional parameter, defaults to 0.
fetchFirst%	An integer and an optional parameter, defaults to 0.

Comments The return value is the number of rows in the result set or the *maxRows* requested. If the function is unable to retrieve the results on the specified connection, or if there are not results pending, 1 is returned. If no data is found, the function returns 0.

The arguments are named arguments. The return is a variant.

If *maxColumns* or *maxRows* are omitted, the array size is used to determine the maximum number of columns and rows retrieved, and an attempt is made to return the entire result set. Extra rows can be retrieved by using **SQLRetrieve** again and by setting *fetchFirst* to 0. If *maxColumns* specifies fewer columns than are available in the result, **SQLRetrieve** discards the rightmost result columns until the results fit the specified size.

When *columnNames* is nonzero, the first row of the array will be set to the column names as specified by the database schema. When r*owNumbers* is nonzero, row numbers are returned in the first column of *destination().* **SQLRetrieve** will clear the users array prior to fetching the results.

When *fetchFirst* is nonzero, it causes the result set to be repositioned to the first row if the database supports the function. If the database does not support repositioning, the result set 1 error will be returned.

If there are more rows in the result set than can be contained in the *destination()* array or than have been requested using *maxRows*, the user can make repeated calls to **SQLRetrieve** until the return value is 0.

SQLRetrieveToFile Function

See Also **Example**

Fetches the results of a pending query on the connection specified by *connection* and stores them in the file specified by *destination.*

Syntax **SQLRetrieveToFile(** *connection& , destination$, columnNames% , columnDelimiter$* **)**

where:	**is:**
connection&	A required argument. A long integer
destination$	A required argument. A string containing the file and path to be used for storing the results.
columnNames%	An integer; when nonzero, the first row of the file will be set to the column names as specified by the database schema. If *columnNames* is omitted, the default is 0.
columnDelimiter$	Specifies the string to be used to delimit the fields within each row. If *columnDelimiter* is omitted, a horizontal tab is used to delimit fields.

Comments Upon successful completion of the operation, the return value is the number of rows in the result set. If the function is unable to retrieve the results on the specified connection, or if there are not results pending, -1 is returned.

The arguments are named arguments. The return is a variant.

Sqr Function

See Also **Example**

Returns the square root of a number.

Syntax **Sqr(** *number* **)**

where:	**is:**
number	An expression containing the number to use.

Comments The return value is single-precision for an integer, currency or single-precision numeric expression, double precision for a long, Variant or double-precision numeric expression.

Static Statement

See Also **Example**

Declares variables and allocate storage space.

Syntax **Static** *variableName* [**As** *type*] [,*variableName* [**As** *type*]] **...**

where:	**is:**
variableName	The name of the variable to declare.
type	The data type of the variable.

Comments Variables declared with the **Static** statement retain their value as long as the program is running. The syntax of **Static** is exactly the same as the syntax of the **Dim** statement.

All variables of a procedure can be made static by using the **Static** keyword in a definition of that procedure See **Function** or **Sub** for more information.

StaticComboBox Statement

See Also **Example**

Creates a combination of a list of choices and a text box.

Syntax A **StaticComboBox** *x , y , dx , dy , text$, .field*

Syntax B **StaticComboBox** *x , y , dx , dy , stringarray$() , .field*

where:	**is:**
x , y	The upper left corner coordinates of the list box, relative to the upper left corner of the dialog box.
dx , dy	The width and height of the combo box in which the user enters or selects text.
text$	A string containing the selections for the combo box.
stringarray$	An array of dynamic strings for the selections in the combo box.
.field	The name of the dialog-record field that will hold the text string entered in the text box or chosen from the list box.

Comments The **StaticComboBox** statement is equivalent to the **ComboBox** or **DropComboBox** statement, but the list box of **StaticComboBox** always stays visible. All dialog functions and statements that apply to the **ComboBox** apply to the **StaticComboBox** as well.

The *x* argument is measured in 1/4 system-font character-width units. The *y* argument is measured in 1/8 system-font character-width units. (See **Begin Dialog** for more information.)

The *text$* argument must be defined, using a **Dim** Statement, before the **Begin Dialog** statement is executed. The arguments in the *text$* string are entered as shown in the following example:

dimname = "*listchoice*"+Chr$(9)+"*listchoice*"+Chr$(9)+"*listchoice*"...

The string in the text box will be recorded in the field designated by the *.field* argument when the OK button (or any pushbutton other than Cancel) is pushed. The *field* argument is also used by the dialog statements that act on this control.

Use the **StaticComboBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Stop Statement

Example

Halts program execution.

Syntax **Stop**

Comments **Stop** statements can be placed anywhere in a program to suspend its execution. Although the **Stop** statement halts program execution, it does not close files or clear variables.

Str Function

See Also **Example**

Returns a string representation of a number.

Syntax **Str**[**$**]**(** *number* **)**

where:	**is:**
number	The number to represent as a string.

Comments The precision in the returned string is single-precision for an integer or single-precision numeric expression, double precision for a long or double-precision numeric expression, and currency precision for currency. Variants return the precision of their underlying vartype.

The dollar sign, "$", in the function name is optional. If specified the return type is string. If omitted, the function will return a **Variant** of vartype 8 (String).

StrComp Function

See Also **Example**

Compares two strings and returns an integer specifying the result of the comparison.

Syntax **StrComp(** *string1$, string2$* [*, compare%*] **)**

where:	**is:**
string1$	Any expression containing the first string to compare.
string2$	The second string to compare.
compare%	An integer for the method of comparison (0=case-sensitive, 1=case-insensitive).

Comments **StrComp** returns one of the following values:

Value	**Meaning**
-1	*string1$* < *string2$*
0	*string1$* = *string2$*
>1	*string1$* > *string2$*
Null	*string1$* = Null or *string2$* = Null

If *compare%* is 0, a case sensitive comparison based on the ANSI character set sequence is performed. If *compare%* is 1, a case insensitive comparison is done based upon the relative order of characters as determined by the country code setting for your system. If omitted, the module level default, as specified with **Option Compare** is used.

The *string1* and *string2* arguments are both passed as Variants. Therefore, any type of expression is supported. Numbers will be automatically converted to strings.

String Function

See Also **Example**

Returns a string consisting of a repeated character.

Syntax A **String**[**$**]**(** *number , Character%* **)**

Syntax B **String**[**$**] **(** *number , string-expression$* **)**

where:	**is:**
number	The length of the string to be returned.
Character%	A numeric expression that contains an integer for the decimal ANSI code of the character to use.
string-expression$	A string argument, the first character of which becomes the repeated character.

Comments *number* must be between 0 and 32,767.

Character% must evaluate to an integer between 0 and 255.

The dollar sign, "$", in the function name is optional. If specified the return type is string. If omitted, the function returns a **Variant** of vartype 8 (String).

Sub ... End Sub Statement

See Also **Example**

Defines a subprogram procedure.

Syntax **[Static] [Private] Sub** *name* **[(** [**Optional**] *parameter* **[As** *type***] , ...)]**

End Sub

where:	**is:**
name	The name of the subprogram.
parameter	A comma-separated list of parameter names.
type	A data type for *parameter*

Comments A call to a subprogram stands alone as a separate statement. (See the **Call** statement). Recursion is supported.

The data type of a parameter can be specified by using a type character or by using the **As** clause. Record parameters are declared by using an **As** clause and a *type* that has previously been defined using the **Type** statement. Array parameters are indicated by using empty parentheses after the *parameter*. The array dimensions are not specified in the **Sub** statement. All references to an array within the body of the subprogram must have a consistent number of dimensions.

If a *parameter* is declared as **Optional**, its value can be omitted when the function is called. Only Variant parameters can be declared as optional, and all optional parameters must appear after all required parameters in the **Sub** statement. The function **IsMissing** must be used to check whether an optional parameter was omitted by the user or not. See the **Call** statement for more information on using named parameters.

The procedure returns to the caller when the **End Sub** statement is reached or when an **Exit Sub** statement is executed.

The **Static** keyword specifies that all the variables declared within the subprogram will retain their values as long as the program is running, regardless of the way the variables are declared.

The **Private** keyword specifies that the procedures will not be accessible to functions and subprograms from other modules. Only procedures defined in the same module will have access to a **Private** subprogram.

Basic procedures use the call by reference convention. This means that if a procedure assigns a value to a parameter, it will modify the variable passed by the caller.

The MAIN subprogram has a special meaning. In many implementations of Basic, MAIN will be called when the module is "run". The MAIN subprogram is not allowed to take arguments.

Use **Function** to define a procedure that has a return value.

Tab Function

See Also **Example**

Moves the current print position to the column specified.

Syntax **Tab (** *n* **)**

where:	**is:**
n	The new print position to use.

Comments The **Tab** function can be used only inside **Print** statement. The leftmost print position is position number 1.

When the **Print** statement is used, the **Tab** function will use the following rules for determining the next print position:

1. If *n* is less than the total line width, the new print position is *n*.
2. If *n* is greater than the total line width, the new print position is *n* **Mod** *width* .
3. If the current print position is greater than *n* or *n* **Mod** *width*, **Tab** skips to the next line and sets the print position to *n* or *n* **Mod** *width*.

To set the width of a print line, use the **Width** statement.

Tan Function

See Also **Example**

Returns the tangent of an angle in radians.

Syntax **Tan**(*number*)

where:	**is:**
number	An expression containing the angle in radians.

Comments *number* is specified in radians, and can be either positive or negative.

The return value is single-precision if the angle is an integer, currency or single-precision value, double precision for a long, Variant or double-precision value.

To convert degrees to radians, multiply by PI/180. The value of PI is 3.14159.

Text Statement

See Also **Example**

Places line(s) of text in a dialog box.

Syntax **Text** *x , y , dx , dy , text$* [, **.***id*]

where:	**is:**
x , y	The upper left corner coordinates of the text area, relative to the upper left corner of the dialog box.
dx , dy	The width and height of the text area.
text$	A string containing the text to appear in the text area defined by *x , y*.
.id	An optional identifier used by the dialog statements that act on this control.

Comments If the width of *text$* is greater than *dx*, the spillover characters wrap to the next line. This will continue as long as the height of the text area established by *dy* is not exceeded. Excess characters are truncated.

By preceding an underlined character in *text$* with an ampersand (&), you enable a user to press the underlined character on the keyboard and position the cursor in the combo or text box defined in the statement immediately following the **Text** statement.

Use the **Text** statement only between a **Begin Dialog** and an **End Dialog** statement.

TextBox Statement

See Also **Example**

Creates a text box in a dialog box.

Syntax **TextBox [NoEcho]** *x , y , dx , dy , .field*

where:	**is:**
x , y	The upper left corner coordinates of the text box, relative to the upper left corner of the dialog box.
dx , dy	The width and height of the text box area.
.field	The name of the dialog record field to hold the text string.

Comments A *dy* value of 12 will usually accommodate text in the system font.

When the user selects the OK button, or any pushbutton other than cancel, the text string entered in the text box will be recorded in *.field*.

The **NoEcho** keyword is often used for passwords; it displays all characters entered as asterisks (*).

Use the **TextBox** statement only between a **Begin Dialog** and an **End Dialog** statement.

Time Function

See Also **Example**

Returns a string representing the current time.

Syntax **Time**[**$**]

Comments The **Time** function returns an eight character string. The format of the string is "*hh:mm:ss*" where *hh* is the hour, *mm* is the minutes and *ss* is the seconds. The hour is specified in military style, and ranges from 0 to 23.

The dollar sign, "$", in the function name is optional. If specified, the return type is String. If omitted, the function will return a **Variant** of vartype 8 (String).

Time Statement

See Also **Example**

Sets the system time.

Syntax Time[**$**] **=** *expression*

where:	**is:**
expression	An expression that evaluates to a valid time.

Comments When **Time** (with the dollar sign $) is used, the *expression* must evaluate to a string of one of the following forms:

hh	Set the time to *hh* hours 0 minutes and 0 seconds
hh:mm	Set the time to *hh* hours *mm* minutes and 0 seconds.
hh:mm:ss	Set the time to *hh* hours *mm* minutes and *ss* seconds

Time uses a 24-hour clock. Thus, 6:00 P.M. must be entered as 18:00:00.

If the dollar sign '$' is omitted, *expression* can be a string containing a valid date, a **Variant** of vartype 7 (date) or 8 (string).

If *expression* is not already a Variant of vartype 7 (date), **Time** attempts to convert it to a valid time. It recognizes time separator characters defined in the International section of the Windows Control Panel. **Time** (without the **$**) accepts both 12 and 24 hour clocks.

Timer Function

See Also **Example**

Returns the number of seconds that have elapsed since midnight.

Syntax **Timer**

Comments The Timer function can be used in conjunction with the **Randomize** statement to seed the random number generator.

TimeSerial Function

See Also **Example**

Returns a time as a Variant of type 7 (date/time) for a specific hour, minute, and second.

Syntax **TimeSerial(** *hour%*, *minute%*, *second%* **)**

where:	**is:**
hour%	A numeric expression for an hour (0-23).
minute%	A numeric expression for a minute (0-59).
second%	A numeric expression for a second (0-59).

Comments You also can specify relative times for each argument by using a numeric expression representing the number of hours, minutes, or seconds before or after a certain time.

TimeValue Function

See Also **Example**

Returns a time value for a specified string.

Syntax **TimeValue(** *time$* **)**

where:	**is:**
time$	A string representing a valid date time value.

Comments The **TimeValue** function returns a **Variant** of vartype 7 (date/time) that represents a time between 0:00:00 and 23:59:59, or 12:00:00 A.M. and 11:59:59 P.M., inclusive.

Trim Function

See Also **Example**

Returns a copy of a string after removing all leading and trailing spaces.

Syntax **Trim**[**$**]**(** *string* **)**

where:	**is:**
string	An expression containing the string to trim.

Comments **Trim** accepts expressions of type String. **Trim** accepts any type of *string* including numeric values and will convert the input value to a string.

The dollar sign, "$", in the function name is optional. If specified, the return type is String. If omitted, the function typically returns a **Variant** of vartype 8 (String). If the value of *string* is NULL, a Variant of vartype 1 (Null) is returned.

Type Statement

See Also **Example**

Declares a user-defined type.

Syntax

Type *userType*
field1 **As** *type1*
field2 **As** *type2*
...
End Type

where:	**is:**
userType	A string expression for the name of the user-defined type.
field1 , field2	The name of a field in the user-defined type.
type1 , type2	A data type: Integer, Long, Single, Double, Currency, String, String**length*, Variant, or another user-defined type.

Comments

The user-defined type declared by **Type** can then be used in the **Dim** statement to declare a record variable. A user-defined type is sometimes referred to as a *record type* or a *structure type*.

field cannot be an array. However, arrays of records are allowed.

The **Type** statement is not valid inside of a procedure definition. To access the fields of a record, use notation of the form:

```
recordName.fieldName
```

To access the fields of an array of records, use notation of the form:

```
arrayName( index ).fieldName
```

Typeof Function

See Also

Returns a value indicating whether an object is of a given class (-1=TRUE, 0=FALSE).

Syntax **If Typeof** *objectVariable* **Is** *className* **then. . .**

where:	**is:**
objectVariable	The object to test.
className	The class to compare the object to.

Comments **Typeof** can only be used in an **If** statement and cannot be combined with other Boolean operators. That is, **Typeof** can only be used exactly as shown in the syntax above.

To test if an object does *not* belong to a class, use the following code structure:

```
If Typeof objectVariable Is className Then
Else
        Rem Perform some action.
End If
```

UBound Function

See Also **Example**

Returns the upper bound of the subscript range for the specified array.

Syntax **UBound(** *arrayname* [, *dimension*] **)**

where:	**is:**
arrayname	The name of the array to use.
dimension	The dimension to use.

Comments The dimensions of an array are numbered starting with 1. If the *dimension* is not specified, 1 is used as a default.

LBound can be used with **UBound** to determine the length of an array.

UCase Function

See Also **Example**

Returns a copy of a string after converting all lower case letters to upper case.

Syntax **UCase**[**$**]**(** *string* **)**

where:	**is:**
string	An expression that evaluates to a string.

Comments The translation is based on the country specified in the Windows Control Panel.

UCase accepts expressions of type string. **UCase** accepts any type of argument and will convert the input value to a string.

The dollar sign, "$", in the function name is optional. If specified, the return type is string. If omitted, the function typically returns a **Variant** of vartype 8 (String). If the value of *string* is Null, a Variant of vartype 1 (Null) is returned.

Unlock Statement

See Also **Example**

Controls access to an open file.

Syntax **Unlock** [#]*filenumber%* [, { *record&* | [*start&*] **To** *end&* }]

where:	**is:**
filenumber%	An integer expression identifying the open file.
record&	Number of the starting record to unlock.
start&	Number of the first record or byte offset to lock/unlock.
end&	Number of the last record or byte offset to lock/unlock.

Comments The *filenumber%* is the number used in the **Open** statement of the file.

For **Binary** mode, *start&*, and *end&* are byte offsets. For **Random** mode, *start&*, and *end&* are record numbers. If *start&* is specified without *end&*, then only the record or byte at *start&* is locked. If **To** *end&* is specified without *start&*, then all records or bytes from record number or offset 1 to *end&* are locked.

For **Input**, **Output** and **Append** modes, *start&,* and *end&* are ignored and the whole file is locked.

Lock and **Unlock** always occur in pairs with identical parameters. All locks on open files must be removed before closing the file, or unpredictable results will occur.

Val Function

See Also **Example**

Returns the numeric value of the first number found in the specified string.

Syntax **Val(** *string$* **)**

where:	**is:**
string$	A string expression containing a number.

Comments Spaces in the source string are ignored. If no number is found, **Val** returns 0.

VarType Function

See Also **Example** **Overview**

Returns the Variant type of the specified Variant variable (0-9).

Syntax **VarType(** *varname* **)**

where:	**is:**
varname	The **Variant** variable to use.

Comments The value returned by **VarType** is one of the following:

Ordinal	**Representation**
0	(Empty)
1	Null
2	Integer
3	Long
4	Single
5	Double
6	Currency
7	Date
8	String
9	Object

Weekday Function

See Also **Example**

Returns the day of the week for the specified date-time value.

Syntax **Weekday(** *date* **)**

where:	**is:**
date	An expression containing a date time value.

Comments The **Weekday** function returns an integer between 1 and 7, inclusive (1=Sunday, 7=Saturday).

Weekday accepts any expression, including strings, and attempts to convert the input value to a date value.

The return value is a **Variant** of vartype 2 (Integer). If the value of *date* is NULL, a Variant of vartype 1 (Null) is returned.

While ... Wend

See Also **Example**

Controls a repetitive action.

Syntax **While** *condition*
statementblock
Wend

where:	**is:**
condition	An expression that evaluates to TRUE (non-zero) or FALSE (zero).
statementblock	A series of statements to execute if *condition* is TRUE.

Comments The *statementblock* statements are until *condition* becomes 0 (FALSE).

The **While** statement is included in OPEN Script for compatibility with older versions of Basic. The **Do** statement is a more general and powerful flow control statement.

Width Statement

See Also **Example**

Sets the output line width for an open file.

Syntax **Width** [#]*filenumber% , width%*

where:	**is:**
filenumber%	An integer expression for the open file to use.
width%	An integer expression for the width of the line (0 to 255).

Comments *Filenumber%* is the number assigned to the file when it is opened. See the **Open** statement for more information.

A value of zero (0) for *width%* indicates there is no line length limit. The default *width%* for a file is zero (0).

With Statement

See Also **Example**

Executes a series of statements on a specified variable.

Syntax

With *variable*
statement_block
End With

where:	**is:**
variable	The variable to be changed by the statements in *statement_block*.
statement_block	The statements to execute.

Comments *Variable* can be an object or a user-defined type. The **With** statements can be nested.

Write Statement

See Also **Example**

Writes data to an open sequential file.

Syntax **Write** [**#**] *filenumber%* [,*expressionlist*]

where:	**is:**
filenumber%	An integer expression for the open file to use.
expressionlist	One or more values to write to the file.

Comments The file must be opened in **Output** or **Append** mode. *Filenumber%* is the number assigned to the file when it is opened. See the **Open** statement for more information.

If *expressionlist* is omitted, the **Write** statement writes a blank line to the file. (See **Input** for more information.)

Year Function

See Also **Example**

Returns the year component (1-12) of a date-time value.

Syntax **Year(** *date* **)**

where:	**is:**
date	An expression that can evaluate to a date time value.

Comments The **Year** function returns an integer between 100 and 9999, inclusive.

Year accepts any type of *date*, including strings, and will attempt to convert the input value to a date value.

The return value is a **Variant** of vartype 2 (Integer). If the value of *date* is NULL, a Variant of vartype 1 (Null) is returned.

OPEN Script Language Reference

Overview Topics
A list of topics that describe how to use features in OPEN Script; also includes a comparison of OPEN Script to other Basic languages

Alphabetical List
All statements and functions in OPEN Script in an alphabetical list

Functional List
All statements and functions in OPEN Script organized by functional group, such as Dialog Boxes, Arrays, or Math Functions

Program Examples
A list of program examples that illustrate OPEN Script language statements and functions

For information on how to use Help, press F1.

Close | Copy All | Copy | Print

Alphabetical List

A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

Close | Copy All | Copy | Print

Functional List

A

Abs	Return the absolute value of a number
AppActivate	Activate another application
Asc	Return an integer corresponding to a character code
Atn	Return the arc tangent of a number

B

Beep	Produce a short beeping tone through the speaker
Begin Dialog	Begin a dialog box definition
Button	Define a button dialog box control
ButtonGroup	Begin definition of a group of button dialog box controls

C

Call	Transfer control to a subprogram
CancelButton	Define a cancel-button dialog box control
Caption	Define the title of a dialog box
CCur	Convert a value to currency
CDbl	Convert a value to double-precision floating point
ChDir	Change the default directory for a drive
ChDrive	Change the default drive
CheckBox	Define a checkbox dialog box control
Chr	Convert a character code to a string
CInt	Convert a value to an integer by rounding
Clipboard	Access the Windows Clipboard
CLng	Convert a value to a long by rounding
Close	Close a file
ComboBox	Define a combobox dialog box control
Command	Return the command line specified when the MAIN sub was run
Const	Declare a symbolic constant
Cos	Return the cosine of an angle
CreateObject	Create an OLE Automation object
CSng	Convert a value to single-precision floating point
CStr	Convert a value to a string
$CStrings	Treat backslash in string as an escape character as in 'C'
CurDir	Return the current directory for a drive
CVar	Convert an number or string to a variant
CVDate	Convert a value to a variant date

D

Date Function	Return the current date
Date Statement	Set the current date

DateSerial	Return the date value for year, month, and day specified
DateValue	Return the date value for string specified
Day	Return the day of month component of a date-time value
DDEAppReturnCode	Return a code from an application on a DDE channel
DDEExecute	Send one or more commands to an application on a DDE channel
DDEInitiate	Open a dynamic data exchange (DDE) channel
DDEPoke	Send data to an application on a DDE channel
DDERequest	Return data from an application on a DDE channel
DDETerminate	Close a DDE channel
Declare	Forward declare a procedure in the same module or in a dynamic link library
Def*type*	Declare the default data type for variables
Derived Functions	List of computed trigonometric and logarithmic functions
Dialog Function	Display a dialog box and return the command button pressed
Dialog Statement	Display a dialog box
Dim	Declare variables
Dir	Return a filename that matches a pattern
DlgControlId	Return numeric ID of a dialog control
DlgEnable Function	Determine whether a dialog control is enabled or disabled
DlgEnable Statement	Enable or disable a dialog control
DlgEnd	Closes the active dialog box
DlgFocus Function	Return ID of the dialog control having input focus
DlgFocus Statement	Set focus to a dialog control
DlgListBoxArray Function	Return contents of a list box or combo box
DlgListBoxArray Statement	Set contents of a list box or combo box
DlgSetPicture	Change the picture in the Picture control
DlgText Function	Return the text associated with a dialog control
DlgText Statement	Set the text associated with a dialog control
DlgValue Function	Return the value associated with a dialog control
DlgValue Statement	Set the value associated with a dialog control
DlgVisible Function	Determine whether a control is visible or hidden
DlgVisible Statement	Show or hide a dialog control
Do...Loop	Control repetitive actions
DoEvents	Let operating system process messages
DropComboBox	Define a drop combobox dialog box control
DropListBox	Define a drop list box dialog box control

E

Environ	Return a string from the operating system's environment
Eof	Check for end of file
Erase	Reinitialize contents of an array
Erl	Return the line number where a run-time error occurred
Err Function	Return a run-time error code

Err Statement	Set the run-time error code
Error Function	Return a string representing an error
Error Statement	Generate an error condition
Exit	Cause the current procedure or loop structure to return
Exp	Return the value of *e* raised to a power

F

FileAttr	Return information about an open file
FileCopy	Copy a file
FileDateTime	Return modification date and time of a specified file
FileLen	Return the length of specified file in bytes
Fix	Return the integer part of a number
For...Next	Loop a fixed number of times
Format	Convert a value to a string using a picture format
FreeFile	Return the next unused file number
Function	Define a function
FV	Return the future value for a stream of periodic cash flows

G

Get	Read bytes from a file
GetAttr	Return attributes of specified file, directory of volume label
GetField	Return a substring from a delimited source string
GetObject	Return the name of an OLE Automation object
Global	Declare a global variable
Goto	Send control to a line label
GroupBox	Define a groupbox in a dialog box

H

Hex	Return the hexadecimal representation of a number, as a string
Hour	Return the hour of day component of a date-time value

I

If ... Then ... Else	Branch on a conditional value
$Include	Tell the compiler to include statements from another file
Input Function	Return a string of characters from a file
Input Statement	Read data from a file or from the keyboard
InputBox	Display a dialog box that prompts for input
InStr	Return the position of one string within another
Int	Return the integer part of a number
IPmt	Return the interest portion of a loan or annuity payment
IRR	Return the internal rate of return
Is	Determine whether two object variables refer to the same object
IsDate	Determine whether a value is a legal date
IsEmpty	Determine whether a variant has been initialized
IsMissing	Determine whether an optional parameter was supplied to a

OKButton	Define an OK button dialog box control
On...Goto	Branch to one of several labels depending upon value
On Error	Control run-time error handling
Open	Open a disk file or device for I/O
OptionButton	Define an OptionButton dialog box control
OptionGroup	Begin definition of a group of OptionButton dialog box controls
Option Base	Declare the default lower bound for array dimensions
Option Compare	Declare the default case sensitivity for string comparisons
Option Explicit	Force all variables to be explicitly declared

P

PasswordBox	Display a dialog box that prompts for input. Don't echo input.
Picture	Include a bitmap picture (.BMP file) in a dialog box
Pmt	Return the periodic payment for a loan or annuity
PPmt	Return the principal paid on a loan or annuity
Print	Print data to a file or to the screen
PushButton	Define a push button dialog box control
Put	Write data to an open file
PV	Return the present value for a stream of cash flows

R

Randomize	Initialize the random-number generator
Rate	Return the interest rate for a loan or annuity
ReDim	Declare dynamic arrays and reallocate memory
Rem	Treat the remainder of the line as a comment
Reset	Close all open disk files
Resume	End an error-handling routine
Right	Return the right portion of a string
RmDir	Remove a directory from a disk
Rnd	Return a random number
Rset	Right-align one string within another
RTrim	Remove trailing spaces from a string

S

Second	Return the second component of a date-time value
Seek Function	Return the current position for a file
Seek Statement	Set the current position for a file
Select Case	Execute one of a series of statement blocks
SendKeys	Send keystrokes to another application
Set	Set an object variable to a value
SetAttr	Set attribute information for a file
SetField	Replace a substring within a delimited target string
Sgn	Return a value indicating the sign of a number
Shell	Run an executable program
Sin	Return the sine of an angle

V

Val	Convert a string to a number
VarType	Return the type of data stored in a variant

W

Weekday	Return the day of the week for the specified date-time value
While ... Wend	Control repetitive actions
Width	Set output-line width for an open file
With	Execute statements on an object or a user-defined type
Write	Write data to a sequential file

Y

Year	Return the year component of a date-time value

Close | Copy All | Copy | Print

Functional List

Close | Copy All | Copy | Print **Alphabetical List**

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Arrays

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Arrays

Erase	Reinitialize contents of an array
LBound	Return the lower bound of an array's dimension
ReDim	Declare dynamic arrays and reallocate memory
UBound	Return the upper bound of an array's dimension

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

$CStrings	Treat backslash in string as an escape character as in 'C'
$Include	Tell the compiler to include statements from another file
$NoCStrings	Tell the compiler to treat a backslash as a normal character
Line Continuation	Continuing a long statement across multiple lines
Rem	Treat the remainder of the line as a comment

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Call	Transfer control to a subprogram
Do...Loop	Control repetitive actions
Exit	Cause the current procedure or loop structure to return
For...Next	Loop a fixed number of times
Goto	Send control to a line label
If ... Then ... Else	Branch on a conditional value
Let	Assign a value to a variable
Lset	Left-align one string or a user-defined variable within another
On...Goto	Branch to one of several labels depending upon value
Rset	Right-align one string within another
Select Case	Execute one of a series of statement blocks
Set	Set an object variable to a value
Stop	Stop program execution
While ... Wend	Control repetitive actions
With	Execute a series of statements on a specified variable

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Date Function	Return the current date
Date Statement	Set the system date
DateSerial	Return the date value for year, month, and day specified
DateValue	Return the date value for string specified
Day	Return the day of month component of a date-time value
Hour	Return the hour of day component of a date-time value
IsDate	Determine whether a value is a legal date.
Minute	Return the minute component of a date-time value
Month	Return the month component of a date-time value
Now	Return the current date and time
Second	Return the second component of a date-time value
Time Function	Return the current time
Time Statement	Set the current time
Timer	Return the number of seconds since midnight
TimeSerial	Return the time value for hour, minute, and second specified
TimeValue	Return the time value for string specified
Weekday	Return the day of the week for the specified date-time value
Year	Return the year component of a date-time value

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Const	Declare a symbolic constant
Declare	Forward declare a procedure in the same module or in a dynamic link library
Def*type*	Declare the default data type for variables
Dim	Declare variables
Function ... End Function	Define a function
Global	Declare a global variable
Option Base	Declare the default lower bound for array dimensions
Option Compare	Declare the default case sensitivity for string comparisons
Option Explicit	Force all variables to be explicitly declared
ReDim	Declare dynamic arrays and reallocate memory
Static	Define a static variable or subprogram
Sub ... End Sub	Define a subprogram
Type	Declare a user-defined data type

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print Defining Dialog Boxes

Close | Copy All | Copy | Print Running Dialog Boxes

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print Defining Dialog Boxes

Begin Dialog	Begin a dialog box definition
Button	Define a button dialog box control
ButtonGroup	Begin definition of a group of button dialog box controls
CancelButton	Define a Cancel button dialog box control
Caption	Define the title of a dialog box
CheckBox	Define a checkbox dialog box control
ComboBox	Define a ComboBox dialog box control
DropComboBox	Define a drop-down combo box dialog box control
DropListBox	Define a drop-down list box dialog box control
GroupBox	Define a group box in a dialog box
ListBox	Define a list box dialog box control
OKButton	Define an OK button dialog box control
OptionButton	Define an OptionButton dialog box control
OptionGroup	Begin definition of a group of OptionButton dialog box controls
Picture	Define a Picture control
PushButton	Define a pushbutton dialog box control
StaticComboBox	Define a static combo box dialog box control
Text	Define a line of text in a dialog box
TextBox	Define a text box in a dialog box

Close | Copy All | Copy | Print Running Dialog Boxes

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Screen Input/Output

Strings

Variants

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print Defining Dialog Boxes

Close | Copy All | Copy | Print Running Dialog Boxes

Dialog Function	Display a dialog box and return the button pressed
Dialog Statement	Display a dialog box
DlgControlId	Return numeric ID of a dialog control
DlgEnable Function	Tell whether a dialog control is enabled or disabled
DlgEnable Statement	Enable or disable a dialog control
DlgEnd	Close the active dialog box
DlgFocus Function	Return ID of the dialog control having input focus
DlgFocus Statement	Set focus to a dialog control
DlgListBoxArray Function	Return contents of a list box or combo box
DlgListBoxArray Statement	Set contents of a list box or combo box
DlgSetPicture	Change the picture in the Picture control
DlgText Function	Return the text associated with a dialog control
DlgText Statement	Set the text associated with a dialog control
DlgValue Function	Return the value associated with a dialog control
DlgValue Statement	Set the value associated with a dialog control
DlgVisible Function	Tell whether a control is visible or hidden
DlgVisible Statement	Show or hide a dialog control

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Strings

Variants

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

DDEAppReturnCode	Return a code from an application on a DDE channel
DDEExecute	Send commands to an application on a DDE channel
DDEInitiate	Open a dynamic data exchange (DDE) channel
DDEPoke	Send data to an application on a DDE channel
DDERequest	Return data from an application on a DDE channel
DDETerminate	Close a DDE channel

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

AppActivate	Activate another application
Command	Return the command line specified when the MAIN sub was run
Date Statement	Set the current date
DoEvents	Let operating system process messages
Environ	Return a string from the operating system's environment
Randomize	Initialize the random-number generator
SendKeys	Send keystrokes to another application
Shell	Run an executable program

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Erl	Return the line number where a run-time error occurred
Err Function	Return a run-time error code
Err Statement	Set the run-time error code
Error	Generate an error condition
Error Function	Return a string representing an error
On Error	Control run-time error handling
Resume	End an error-handling routine
Trappable Errors	Errors that can be trapped by OPEN Script code

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print Disk and Directory Control

Close | Copy All | Copy | Print File Control

Close | Copy All | Copy | Print File Input/Output

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print Disk and Directory Control

ChDir	Change the default directory for a drive
ChDrive	Change the default drive
CurDir	Return the current directory for a drive
Dir	Return a filename that matches a pattern
MkDir	Make a directory on a disk
RmDir	Remove a directory from a disk

Close | Copy All | Copy | Print File Control

Close | Copy All | Copy | Print File Input/Output

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print Disk and Directory Control

Close | Copy All | Copy | Print File Control

FileAttr	Return information about an open file
FileCopy	Copy a file
FileDateTime	Return modification date and time of a specified file
FileLen	Return the length of specified file in bytes
GetAttr	Return attributes of specified file, directory of volume label
Kill	Delete files from a disk
Name	Rename a disk file
SetAttr	Set attribute information for a file

Close | Copy All | Copy | Print File Input/Output

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print Disk and Directory Control

Close | Copy All | Copy | Print File Control

Close | Copy All | Copy | Print File Input/Output

Close	Close a file
Eof	Check for end of file
FreeFile	Return the next unused file number
Get	Read bytes from a file
Input Function	Return a string of characters from a file
Input Statement	Read data from a file or from the keyboard
Line Input	Read a line from a sequential file
Loc	Return current position of an open file
Lock	Control access to some or all of an open file by other processes
Lof	Return the length of an open file
Open	Open a disk file or device for I/O
Print	Print data to a file or to the screen
Put	Write data to an open file
Reset	Close all open disk files
Seek Function	Return the current position for a file
Seek Statement	Set the current position for a file
Spc	Output given number of spaces
Tab	Move print position to the given column
Unlock	Control access to some or all of an open file by other processes
Width	Set output-line width for an open file
Write	Write data to a sequential file

Close Copy All Copy Print **Math Functions**

Close Copy All Copy Print **Objects**

Close Copy All Copy Print **ODBC**

Close Copy All Copy Print **Screen Input/Output**

Close Copy All Copy Print **Strings**

Close Copy All Copy Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print Financial Functions

Close | Copy All | Copy | Print Numeric Functions

Close | Copy All | Copy | Print Trigonometric Functions

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print Financial Functions

FV	Return future value of a cash flow stream
IPmt	Return interest payment for a given period
IRR	Return internal rate of return for a cash flow stream
NPV	Return net present value of a cash flow stream
Pmt	Return a constant payment per period for an annuity
PPmt	Return principal payment for a given period
PV	Return present value of a future stream of cash flows
Rate	Return interest rate per period

Close | Copy All | Copy | Print Numeric Functions

Close | Copy All | Copy | Print Trigonometric Functions

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print Financial Functions

Close | Copy All | Copy | Print Numeric Functions

Abs	Return the absolute value of a number
Exp	Return the value of *e* raised to a power
Fix	Return the integer part of a number
Int	Return the integer part of a number
IsNumeric	Determine whether a value is a legal number
Log	Return the natural logarithm of a value
Rnd	Return a random number
Sgn	Return a value indicating the sign of a number
Sqr	Return the square root of a number
Derived Functions	How to compute other numeric functions

Close | Copy All | Copy | Print Trigonometric Functions

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print Financial Functions

Close | Copy All | Copy | Print Numeric Functions

Close | Copy All | Copy | Print Trigonometric Functions

Atn	Return the arc tangent of a number
Cos	Return the cosine of an angle.
Sin	Return the sine of an angle
Tan	Return the tangent of an angle
Derived Functions	How to compute other trigonometric functions

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Clipboard	Access the Windows Clipboard
CreateObject	Create an OLE Automation object
GetObject	Retrieve an OLE Automation object from a file or get the active OLE object for an OLE class
Is	Determine whether two object variables refer to the same object
Me	Get the current object
New	Allocate and initialize a new OLE Automation object
Nothing	Set an object variable to not refer to an object
Object	Declare an OLE Automation object
Typeof	Check the class of an object
With	Execute statements on an object or a user-defined type

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

SQLClose	Close a data source connection
SQLError	Return a detailed error message ODBC functions
SQLExecQuery	Execute an SQL statement
SQLGetSchema	Obtain information about data sources, databases, terminology, users, owners, tables, and columns
SQLOpen	Establish a connection to a data source for use by other functions
SQLRequest	Make a connection to a data source, execute an SQL statement, return the results
SQLRetrieve	Return the results of a select that was executed by SQLExecQuery into a user-provided array
SQLRetrieveToFile	Return the results of a select that was executed by SQLExecQuery into a user-specified file

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Beep	Produce a short beeping tone through the speaker
Input Function	Return a string of characters from a file
Input Statement	Read data from a file or from the keyboard
InputBox	Display a dialog box that prompts for input
MsgBox Function	Display a Windows message box
MsgBox Statement	Display a Windows message box
PasswordBox	Display a dialog box that prompts for input. Don't echo input.
Print	Print data to a file or to the screen

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print String Functions

Close | Copy All | Copy | Print String Conversions

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print String Functions

GetField	Return a substring from a delimited source string
Hex	Return the hexadecimal representation of a number, as a string
InStr	Return the position of one string within another
LCase	Convert a string to lower case
Left	Return the left portion of a string
Len	Return the length of a string or size of a variable
Like Operator	Compare a string against a pattern
LTrim	Remove leading spaces from a string
Mid Function	Return a portion of a string
Mid Statement	Replace a portion of a string with another string
Oct	Return the octal representation of a number, as a string
Right	Return the right portion of a string
RTrim	Remove trailing spaces from a string
SetField	Replace a substring within a delimited target string
Space	Return a string of spaces
Str	Return the string representation of a number
StrComp	Compare two strings.
String	Return a string consisting of a repeated character

Trim	Remove leading and trailing spaces from a string
UCase	Convert a string to upper case

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print String Functions

Close | Copy All | Copy | Print String Conversions

Asc	Return an integer corresponding to a character code
CCur	Convert a value to currency
CDbl	Convert a value to double-precision floating point
Chr	Convert a character code to a string
CInt	Convert a value to an integer by rounding
CLng	Convert a value to a long by rounding
CSng	Convert a value to single-precision floating point
CStr	Convert a value to a string
CVar	Convert an number or string to a variant
CVDate	Convert a value to a variant date
Format	Convert a value to a string using a picture format
Val	Convert a string to a number

Close | Copy All | Copy | Print **Variants**

Close | Copy All | Copy | Print

Close | Copy All | Copy | Print **Alphabetical List**

Functional List

Below is an expandable list of OPEN Script statements and functions. To see a list of the functions and statements in each group, click the button on the left.

Close | Copy All | Copy | Print **Arrays**

Close | Copy All | Copy | Print **Compiler Directives**

Close | Copy All | Copy | Print **Control Flow**

Close | Copy All | Copy | Print **Dates & Times**

Close | Copy All | Copy | Print **Declarations**

Close | Copy All | Copy | Print **Dialog Boxes**

Close | Copy All | Copy | Print **Dynamic Data Exchange (DDE)**

Close | Copy All | Copy | Print **Environment Control**

Close | Copy All | Copy | Print **Errors**

Close | Copy All | Copy | Print **Files**

Close | Copy All | Copy | Print **Math Functions**

Close | Copy All | Copy | Print **Objects**

Close | Copy All | Copy | Print **ODBC**

Close | Copy All | Copy | Print **Strings**

Close | Copy All | Copy | Print **Screen Input/Output**

Close | Copy All | Copy | Print **Variants**

IsEmpty	Determine whether a variant has been initialized
IsNull	Determine whether a variant contains a NULL value
Null	Return a null variant
VarType	Return the type of data stored in a variant

Close | Copy All | Copy | Print

Program Examples

The list below indicates several small programs that demonstrate the use of OPEN Script functions and statements. To view the example program, click the button to the left of the title.

 Hello World

Simple Basic program that demonstrates calls to subroutines and functions

 Bitmap Viewer

Displays a series of bitmap files (.bmp) in a dialog box

Close | Copy All | Copy | Print Find Files

Finds a test file containing a specified string

Close | Copy All | Copy | Print Greatest Common Factor

Updates a dialog box dynamically, based on user input

Close | Copy All | Copy | Print Quicksort

Basic implementation of recursive version of quicksort

OPEN Script Glossary Close

call by reference
call by value
comment
control ID
dialog control
function
label
metacommands
name
precedence order
procedure
subprogram
type character
vartype

call by reference

Arguments passed by reference to a procedure can be modified by the procedure. Procedures written in Basic are defined to receive their arguments by reference. If you call such a procedure and pass it a variable, and if the procedure modifies its corresponding formal parameter, it will modify the variable. Passing an expression by reference is legal in Basic; if the called procedure modifies its corresponding parameter, a temporary value will be modified, with no apparent effect on the caller.

call by value

When an argument is passed by value to a procedure, the called procedure receives a copy of the argument. If the called procedure modifies its corresponding formal parameter, it will have no effect on the caller. Procedures written in other languages such as C can receive their arguments by value.

comment

A comment is text that documents a program. Comments have no effect on the program (except for metacommands). In Basic, a comment begins with a single quote, and continues to the end of the line. If the first character in a comment is a dollar sign ($), the comment is interpreted as a metacommand. Lines beginning with the keyword **Rem** are also interpreted as comments.

control ID

This can be either a text string, in which case it is the name of the control, or it can be a numeric ID. Note that control IDs are case-sensitive and do not include the dot that appears before the ID. Numeric IDs depend on the order in which dialog controls are defined. You can find the numeric ID using the **DlgControlID** function.

dialog control
An item in a dialog box, such as a list box, combo box, or command button.

function

A procedure that returns a value. In Basic, the return value is specified by assigning a value to the name of the function, as if the function were a variable.

label

A label identifies a position in the program at which to continue execution, usually as a result of executing a **GoTo** statement. To be recognized as a label, a name must begin in the first column, and must be immediately followed by a colon (":"). Reserved words are not valid labels.

metacommand

A metacommand is a command that gives the compiler instructions on how to build the program. In Basic, metacommands are specified in comments that begin with a dollar sign ($).

name

A Basic name must start with a letter (A through Z). The remaining part of a name can also contain numeric digits (0 through 9) or an underscore character (_). A name cannot be more than 40 characters in length. Type characters are not considered part of a name.

precedence order

The system OPEN Script uses to determine which operators in an expression to evaluate first, second, and so on. Operators with a higher precedence are evaluated before those with lower precedence. Operators with equal precedence are evaluated from left to right. The default precedence order (from high to low) is: numeric, string, comparison, logical.

procedure

A series of OPEN Script statements and functions executed as a unit. Both subprograms (**Sub**) and functions (**Function**) are called procedures.

subprogram
A procedure that does not return a value.

type character

A special character used as a suffix to a name of a function, variable, or constant. The character defines the data type of the variable or function. The characters are:

Dynamic String	$
Integer	%
Long integer	&
Single precision floating point	!
Double precision floating point	#
Currency exact fixed point	@

vartype

The internal tag used to identify the type of value currently assigned to a variant. One of the following:

Empty	0
Null	1
Integer	2
Long	3
Single	4
Double	5
Currency	6
Date	7
String	8
Object	9

Close | Copy All | Copy | Print

Overview Topics

Using This Help System

Conventions

Describes program and typographic conventions

Using the Examples

How to use the example provided with each function and statement

How OPEN Script Compares to Other Versions of Basic

Describes differences between OPEN Script and earlier versions of Basic, Visual Basic, and Word Basic

Using OPEN Script

Data Types

Defines data types and their use

Dialog Boxes

How to create and run a custom dialog box

Dynamic Data Exchange

How to use DDE to talk with other applications

Error Handling

How to trap errors

Expressions

How to use operators to form string and numeric expressions

Object Handling

How to create and use objects

Conventions

See Also **Typographic Conventions**

OPEN Script uses the following programming conventions:

Arguments

Arguments to subroutines and functions you write are listed after the subroutine or function and might or might not be enclosed in parentheses. Whether you use parentheses depends on how you want to pass the argument to the subroutine or function: either by value or by reference.

If an argument is passed by value, it means that the variable used for that argument retains its value when the subroutine or function returns to the caller. If an argument is passed by reference, it means that the variable's value might be (and probably will be) changed for the calling procedure. For example, suppose you set the value of a variable, x, to 5 and pass x as an argument to a subroutine, named mysub. If you pass x by value to mysub, the value of x will always be 5 after mysub returns. If you pass x by reference to mysub, however, x could be 5 or any other value resulting from the actions of mysub.

To pass an argument by value, use one of the following syntax options:

```
Call mysub((x))
mysub(x)
y=myfunction((x))
Call myfunction((x))
```

To pass an argument by reference, use one of the following options:

```
Call mysub(x)
mysub x
y=myfunction(x)
Call myfunction(x)
```

Externally declared subroutines and functions (such as DLL functions) can be declared to take byVal arguments in their declaration. In that case, those arguments are always passed byVal.

Named Arguments

When you call a subroutine or function that takes arguments, you usually supply values for those arguments by listing them in the order shown in the syntax for the statement or function. For example, suppose you define a function this way:

myfunction(*id*, *action*, *value*)

From the above syntax, you know that the function called **myfunction** requires three arguments: *id*, *action*, and *value*. When you call this function, you supply those arguments in the order shown. If the function contains just a few arguments, it is fairly easy to remember the order of each of the arguments. However, if a function has several arguments, and you want to be sure the values you supply are assigned to the correct arguments, use named arguments.

Named arguments are arguments identified by name rather than by position in the syntax. To use a named argument, use the following syntax:

namedarg := *value*

Using this syntax for myfunction, you get:

myfunction id:=1, action:="get", value:=0

The advantage of named arguments, though, is that you do not need to remember the original order as they were listed in the syntax, so the following function call is also correct:

myfunction action:="get", value:=0, id:=1

With named arguments, order is not important.

The other significant advantage to named arguments is when you call functions or subroutines that have a mix of required and optional arguments. Ordinarily, you need to use commas as placeholders in the syntax for the optional arguments that you do not use. With named arguments, however, you can specify just the arguments you want to use and their values and forget about their order in the syntax. For example, if myfunction is defined as:

```
myfunction(id, action, value, Optional counter)
```

you can use named arguments as follows:

```
myfunction id:="1", action:="get", value:="0"
```

or,

```
myfunction value:="0", counter:="10", action:="get", id:="1"
```

Note: Although you can shift the order of named arguments, you cannot omit required arguments.

All OPEN Script functions and statements accept named arguments. The argument names are listed in their syntax for the statement and function.

Arrays

Array dimensions are enclosed in parentheses after the array name:

```
arrayname(a,b,c)
```

Comments

Comments are preceded by an apostrophe and can appear on their own line in a procedure or directly after a statement or function on the same line:

```
'this comment is on its own line
Dim i as Integer        'this comment is on the code line
```

Line Continuation

Long statements can be continued across more than one line by typing a space-underscore at the end of a line and continuing the statement on the next line. (You can add a comment after the underscore.)

```
Dim trMonth As Integer _   'month of transaction
        trYear As Integer    ' year of transaction
```

Records

Elements in a record are identified using the following syntax:

```
record.element
```

where *record* is the previously defined record name and *element* is a member of that record.

Typographic Conventions

OPEN Script Help uses the following typographic conventions:

To represent:	**Help syntax is:**
Statements and functions	Boldface; initial letter uppercase: **Abs** **Len**(*variable*)
Arguments to statements or functions	All lowercase, italicized letters: *variable, rate, prompt$*
Optional arguments and/or characters	Italicized arguments and/or characters in brackets: [,*caption$*], [*type$*], [$]
Required choice for an argument from a list of choices	A list inside braces, with OR operator (|) separating choices: {Goto *label* | Resume Next | Goto 0}

Using the Examples

In addition to the definition of each statement or function, the Help System also offers a small working example of each. You will notice the word **Example** next to the words **See Also** in the upper region of the window (under the topic title).

Clicking **Example** opens a separate window. The Example window contains a small working example of an OPEN Script program that uses the given statement or function. You can simply look at the contents of this window, or you can run the example in OPEN Script to see how it works.

To run the example, follow these steps:

☑ Start the OPEN Script Editor.

☑ From the statement or function Example window, use one of the command buttons to copy the example to the clipboard (you can copy either part of the example or all of it).

Paste the contents into the OPEN Script Editor window. (If you copy the whole example, the lines of description will appear as well; however, since each of these lines is preceded by an apostrophe, they function as comments.)

Run the program.

To run the examples that show ODBC functions (those beginning with SQL):

You need to have Microsoft Access installed on your machine.

To run the examples that show Object functions:

You need to have VISIO installed on your machine. A few of the Object functions do not have examples associated with them.

See Also

- Data Type Conversion
- Dynamic Arrays
- Variants

Data Types

See Also

Basic is a strongly-typed language. Variables can be declared implicitly on first reference by using a type character; if no type character is present, the default type of **Variant** is assumed. Alternatively, the type of a variable can be declared explicitly with the **Dim** statement. In either case, the variable can only contain data of the declared type. Variables of user-defined type must be explicitly declared. OPEN Script supports standard Basic numeric, string, record and array data. OPEN Script also supports Dialog Box Records and Objects (which are defined by the application).

Arrays

Arrays are created by specifying one or more subscripts at declaration or **Redim** time. Subscripts specify the beginning and ending index for each dimension. If only an ending index is specified, the beginning index depends on the **Option Base** setting. Array elements are referenced by enclosing the proper number of index values in parentheses after the array name, e.g., *arrayname(i,j,k)*. See the **Dim** statement for more information.

Numbers

The five numeric types are:

Type	From	To
Integer	-32,768	32,767
Long	-2,147,483,648	2,147,483,647
Single	-3.402823e+38	-1.401298e-45,
	0.0,	
	1.401298e-45	3.402823466e+38
Double	-1.797693134862315d+308	-4.94065645841247d-308,
	0.0,	
	2.2250738585072014d-308	1.797693134862315d+308
Currency	-922,337,203,685,477.5808	922,337,203,685,477.5807

Numeric values are always signed.

Basic has no true Boolean variables. Basic considers 0 to be FALSE and any other numeric value to be TRUE. Only numeric values can be used as Booleans. Comparison operator expressions always return 0 for FALSE and -1 for TRUE.

Integer constants can be expressed in decimal, octal, or hexadecimal notation. Decimal constants are expressed by simply using the decimal representation. To represent an octal value, precede the constant with "&O" or "&o" (e.g., &o177). To represent a hexadecimal value, precede the constant with "&H" or "&h" (e.g., &H8001).

Records

A record, or record variable, is a data structure containing one or more elements, each of which has a value. Before declaring a record variable, a **Type** must be defined. Once the **Type** is defined, the variable can be declared to be of that type. The variable name should not have a type character suffix. Record elements are referenced using dot notation, e.g.,

varname.elementname. Records can contain elements that are themselves records.

Dialog box records look like any other user-defined data type. Elements are referenced using the same *recname.elementname* syntax. The difference is that each element is tied to an element of a dialog box. Some dialog boxes are defined by the application, others by the user. See the **Begin Dialog** statement for more information.

Strings

Basic strings can be either fixed or dynamic. Fixed strings have a length specified when they are defined, and the length cannot be changed. Fixed strings cannot be of 0 length. Dynamic strings have no specified length. Any string can vary in length from 0 to 32,767 characters. There are no restrictions on the characters that can be included in a string. For example, the character whose ANSI value is 0 can be embedded in strings.

Data Type Conversions

Basic will automatically convert data between any two numeric types. When converting from a larger type to a smaller type (for example **Long** to **Integer**), a runtime numeric overflow might occur. This indicates that the number of the larger type is too large for the target data type. Loss of precision is not a runtime error (e.g., when converting from **Double** to **Single**, or from either float type to either integer type).

Basic will also automatically convert between fixed strings and dynamic strings. When converting a fixed string to dynamic, a dynamic string that has the same length and contents as the fixed string will be created. When converting from a dynamic string to a fixed string, some adjustment might be required. If the dynamic string is shorter than the fixed string, the resulting fixed string will be extended with spaces. If the dynamic string is longer than the fixed string, the resulting fixed string will be a truncated version of the dynamic string. No runtime errors are caused by string conversions.

Basic will automatically convert between any data type and **variants**. Basic will convert variant strings to numbers when required. A type mismatch error will occur if the variant string does not contain a valid representation of the required number.

No other implicit conversions are supported. In particular, Basic will not automatically convert between numeric and string data. Use the functions **Val** and **Str$** for such conversions.

Dynamic Arrays

Dynamic arrays differ from fixed arrays in that you do not specify a subscript range for the array elements when you dimension the array. Instead, the subscript range is set using the **Redim** statement. With dynamic arrays, you can set the size of the array elements based on other conditions in your procedure. For example, you might want to use an array to store a set of values entered by the user, but you do not know in advance how many values the user has. In this case, you dimension the array without specifying a subscript range and then execute a ReDim statement each time the user enters a new value. Or, you might want to prompt for the number of values a user has and execute one ReDim statement to set the size of the array before prompting for the values.

If you use ReDim to change the size of an array and want to preserve the contents of the array at the same time, be sure to include the Preserve argument to the ReDim statement.

If you **Dim** a dynamic array before using it, the maximum number of dimensions it can have is 8. To create dynamic arrays with more dimensions (up to 60), do not Dim the array at all; instead use just the **ReDim** statement inside your procedure.

The following procedure uses a dynamic array, *varray*, to hold cash flow values entered by the user:

```
Sub main
    Dim aprate as Single
    Dim varray() as Double
    Dim cflowper as Integer
    Dim msgtext
    Dim x as Integer
    Dim netpv as Double
    cflowper=InputBox("Enter number of cash flow periods")
    ReDim varray(cflowper)
    For x= 1 to cflowper
        varray(x)=InputBox("Enter cash flow amount for period #" & x & ":")
    Next x
    aprate=InputBox("Enter discount rate: ")
    If aprate>1 then
        aprate=aprate/100
    End If
    netpv=NPV(aprate,varray())
    msgtext="The net present value is: "
    msgtext=msgtext & Format(netpv, "Currency")
    MsgBox msgtext
End Sub
```

Variant Data Type

The variant data type can be used to define variables that contain any type of data. A tag is stored with the variant data to identify the type of data that it currently contains. You can examine the tag by using the **VarType** function.

A variant can contain a value of any of the following types:

Type/ Name		Size of Data	Range
0	(Empty)	0	N/A
1	Null	0	N/A
2	Integer	2 bytes (short)	-32768 to 32767
3	Long	4 bytes (long)	-2.147E9 to 2.147E9
4	Single	4 bytes (float)	-3.402E38 to -1.401E-45 (negative)
			1.401E-45 to 3.402E38 (positive)
5	Double	8 bytes (double)	-1.797E308 to -4.94E-324 (negative)
			4.94E-324 to 1.797E308 (positive)
6	Currency	8 bytes (fixed)	-9.223E14 to 9.223E14
7	Date	8 bytes (double)	Jan 1st, 100 to Dec 31st, 9999
8	String	0 to ~64kbytes	0 to ~64k characters
9	Object	N/A	N/A

Any newly-defined Variant defaults to being of Empty type, to signify that it contains no initialized data. An Empty Variant converts to zero when used in a numeric expression, or an empty string in a string expression. You can test whether a variant is uninitialized (empty) with the **IsEmpty** function.

Null variants have no associated data and serve only to represent invalid or ambiguous results. You can test whether a variant contains a null value with the **IsNull** function. Null is not the same as Empty, which indicates that a variant has not yet been initialized.

See Also

- Begin Dialog...End Dialog
- Dialog Function
- Dialog Statement
- Dialog Functions and Statements

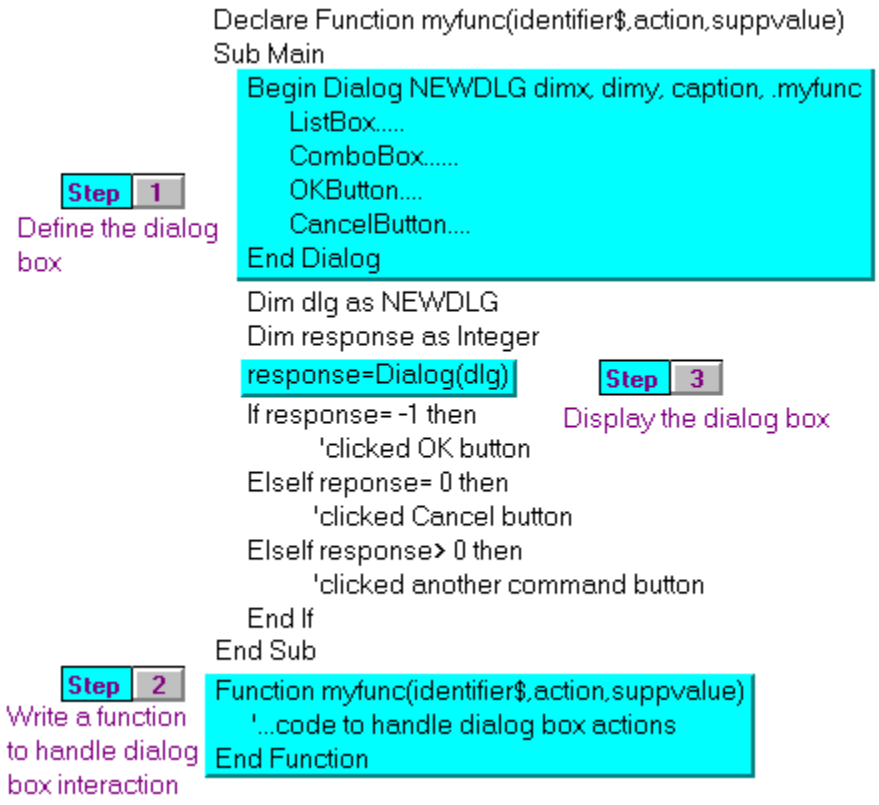
Dialog Boxes

See Also

To create and run a dialog box, follow these three steps:

1. Define a dialog box record using the **Begin Dialog...End Dialog** statements and the dialog box definition statements such as **TextBox**, **OKButton**.
2. Create a function to handle dialog box interactions using the **Dialog Functions and Statements**. (Optional)
3. Display the dialog box using either the **Dialog Function** or **Dialog Statement**.

The example code skeleton below illustrates these steps.



Step 1: Define a dialog box

The **Begin Dialog... End Dialog** statements define a dialog box. The last parameter to the Begin Dialog statement is the name of a function, prefixed by a period (.).This function handles interactions between the dialog box and the user.

The Begin Dialog statement supplies three parameters to your function: an **identifier** (a dialog control ID), the **action** taken on the control, and a **value** with additional action information. Your function should have these three arguments as input parameters. See the Begin Dialog...End Dialog statement for more information.

Step 2: Write a dialog box function

This function defines dialog box behavior. For example, your function could disable a check box, based on a user's action. The body of the function uses the "Dlg"-prefixed OPEN Script statements and functions to define dialog box actions.

Define the function itself using the **Function...End Function** statement or declare it using the **Declare** statement *before* using the **Begin Dialog** statement. Enter the name of the function as the last argument to Begin Dialog. The function receives three parameters from Begin Dialog and returns a value. Return a non-zero value to leave the dialog box open after the user clicks a command button (such as Help).

Step 3: Display the dialog box

You use the **Dialog** function (or statement) to display a dialog box. The argument to Dialog is a variable name that you previously dimensioned as a dialog box record. The name of the dialog box record comes from the **Begin Dialog... End Dialog** statement. The return values for the Dialog function determine which key was pressed: -1 for OK, 0 for Cancel, >0 for a command button. If you use the **Dialog** statement, it returns an error if the user presses Cancel, which you can then trap with the **On Error** statement.

Dialog Functions and Statements

The function you create uses the "Dlg" dialog functions and statements to manipulate the active dialog box. This is the *only* function that can use these functions and statements. The list of the "Dlg" functions and statements is as follows:

DlgControlId	Return numeric ID of a dialog control
DlgEnable Function	Tell whether a control is enabled or disabled
DlgEnable Statement	Enable or disable a dialog control
DlgFocus Function	Return ID of the dialog control having input focus.
DlgFocus Statement	Set focus to a dialog control
DlgListBoxArray Function	Return contents of a list box or combo box
DlgListBoxArray Statement	Set contents of a list box or combo box
DlgText Function	Return the text associated with a dialog control
DlgText Statement	Set the text associated with a dialog control
DlgValue Function	Return the value associated with a dialog control
DlgValue Statement	Set the value associated with a dialog control
DlgVisible Function	Tell whether a control is visible or disabled
DlgVisible Statement	Show or hide a dialog control

Most of these functions and statements take control ID as their first argument. For example, if a checkbox was defined with the following statement:

CheckBox 20, 30, 50, 15, "My check box", **.**Check1

then **DlgEnable "Check1", 1** enables the checkbox, and **DlgValue("Check1")** returns 1 if the checkbox is currently checked, 0 if not. Note that the IDs are case-sensitive and do not include the dot that appears before the ID. Dialog functions and statements can also work with numeric IDs. Numeric IDs depend on the order in which dialog controls are defined.

For example, if the checkbox that we considered was the first control defined in the dialog record, then **DlgValue(0)** would be equivalent to **DlgValue("Check1").** (The control numbering begins from 0, and the **Caption** control does not count.) Find the numeric ID using the **DlgControlID** function.

Note that for some controls (such as buttons and texts) the last argument in the control definition, ID, is optional. If it is not specified, the text of the control becomes its ID. For example, the Cancel button can be referred as "Cancel" if its ID was not specified in the **CancelButton** statement.

See Also

- DDEAppReturnCode
- DDEInitiate
- DDEExecute
- DDEPoke
- DDERequest
- DDETerminate

Dynamic Data Exchange (DDE)

See Also

Dynamic data exchange (DDE) is a process by which two applications communicate and exchange data. One application can be your Basic program. To "talk" to another application and send it data, you need to open a connection, called a DDE channel, using the statement, **DDEInitiate**. The application must already be running before you can open a DDE channel. To start an application, use the **Shell** command.

DDEInitiate requires two arguments: the DDE application name and a topic name. The DDE application name is usually the name of the .EXE file used to start the application, without the .EXE extension. For example, the DDE name for Microsoft Word is "WINWORD". The topic name is usually a filename to get or send data to, although there are some reserved DDE topic names, such as **System**. Refer to the documentation for the application, to get a list of the available topic names.

After you have opened a channel to an application, you can get text and numbers (**DDERequest**), send text and numbers (**DDEPoke**) or send commands (**DDEExecute**). When you have finished communicating with the application, you should close the DDE channel using **DDETerminate**. Because you have a limited number of channels available at once (depending on the operating system in use and the amount of memory you have available), it is a good idea to close a channel as soon as you finish using it.

The other DDE command available in OPEN Script is **DDEAppReturnCode**, which you use for error checking purposes. After getting or sending text, or executing a command, you might want to use **DDEAppReturnCode** to make sure the application performed the task as expected. If an error did occur, your program can notify the user of the error.

Expressions

An expression is a collection of two or more terms that perform a mathematical or logical operation. The terms are usually either variables or functions that are combined with an operator to evaluate to a string or numeric result. You use expressions to perform calculations, manipulate variables, or concatenate strings.

Expressions are evaluated according to precedence order. Use parentheses to override the default precedence order.

The precedence order (from high to low) for the operators is:

Numeric Operators

String Operators

Comparison Operators

Logical Operators

Numeric Operators

^	Exponentiation
-,+	Unary minus and plus
*, /	Numeric multiplication or division. For division, the result is a **Double**.
\	Integer division. The operands can be **Integer** or **Long**.
Mod	Modulus or Remainder. The operands can be **Integer** or **Long**.
-, +	Numeric addition and subtraction. The + operator can also be used for string concatenation.

String Operators

&	String concatenation
+	String concatenation

Comparison Operators (Numeric and String)

>	Greater than
<	Less than
=	Equal to
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

For numbers, the operands are widened to the least common type (**Integer** is preferred over **Long**, which is preferred over **Single**, which is preferred over **Double**). For **Strings**, the comparison is case-sensitive, and based on the collating sequence used by the language specified by the user using the Windows Control Panel. The result is 0 for FALSE and -1 for TRUE.

Logical Operators

Not	Unary Not - operand can be **Integer** or **Long**. The operation is performed bitwise (one's complement).
And	And - operands can be **Integer** or **Long**. The operation is performed bitwise.
Or	Inclusive Or - operands can be **Integer** or **Long**. The operation is performed bitwise.
Xor	Exclusive Or - operands can be **Integer** or **Long**. The operation is performed bitwise.
Eqv	Equivalence - operands can be **Integer** or **Long**. The operation is performed bitwise. (A **Eqv** B) is the same as (**Not** (A **Xor** B)).
Imp	Implication - operands can be **Integer** or **Long**. The operation is performed bitwise. (A **Imp** B) is the same as ((**Not** A) OR B).

See Also

- CreateObject
- GetObject
- Is
- Nothing
- Set

Object Handling

See Also

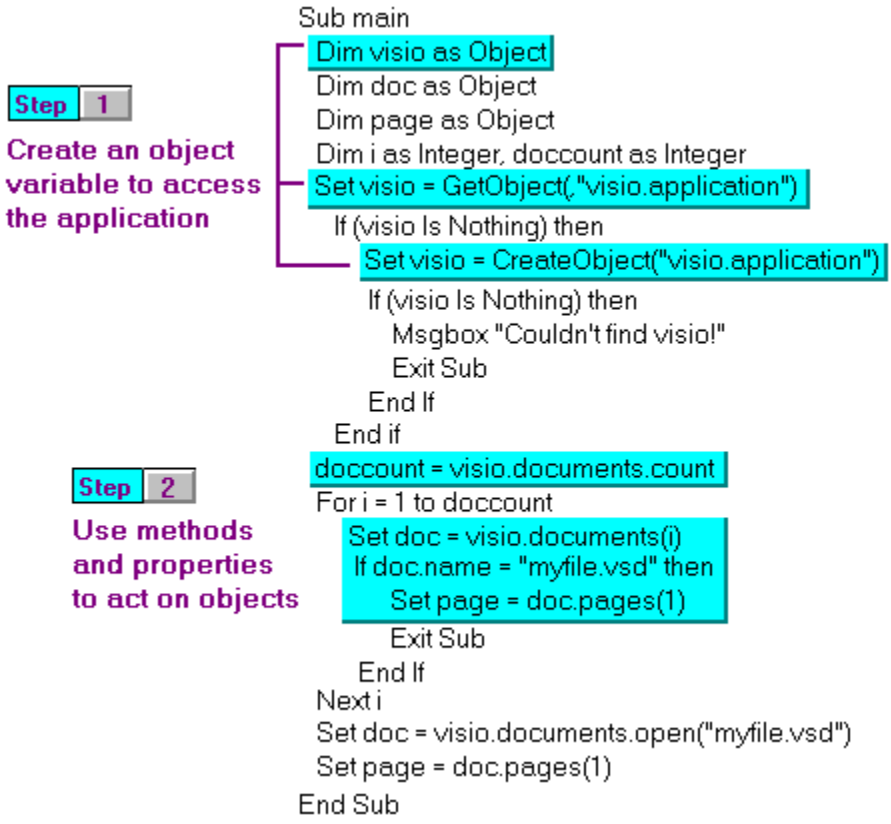
Objects can be the end products of a software application, such as a spreadsheet, graph, or document. An OLE Automation object provides access to the functions of an application through the methods and properties of the object. Each software application that implements OLE Automation has its own set of properties and methods that change the characteristics of an object or use the features of the application.

For some applications, you can link and embed objects from some applications into others. Also, some applications allow use of OLE Automation objects in scripts that automate the tasks of the application.

Properties affect the characteristics of an object. For example, width is a property of a range of cells in a spreadsheet, colors are a property of graphs, margins are a property of word processing documents, and usernames are a property of network applications.

Methods cause the application to do something to an object. Examples are Calculate for a spreadsheet, Snap to Grid for a graph, AutoSave for a document, and SendFile and ReceiveFile for some network applications.

In OPEN Script, you have the ability to access an object and use the originating software application to change properties and methods of that object. Before you can use an object in a procedure, however, you must access the software application associated with the object by assigning it to an object variable. Then you attach an object name (with or without properties and methods) to the variable to manipulate the object. The syntax for doing this is shown in the following example code.



Note: The examples shown here are specific to the VISIO software application. Object, property, and method names vary from one application to another. For the applicable names

to use, see the software documentation for the application you want to access.

Step 1: Create an object variable to access the application

The **Dim** statement creates an object variable called "visio" and assigns the application, VISIO, to it. The **Set** statement assigns the VISIO application to the variable visio using either **GetObject** or **CreateObject**. You use GetObject if the application is already open on the Windows desktop. Use CreateObject if the application is not open.

Step 2: Use methods and properties to act on objects.

To access an object, property or method, you use this syntax:

appvariable.object.property

appvariable.object.method

For example, **visio.document.count** is a value returned by the Count method of the Document object for the VISIO application, which is assigned to the Integer variable doccount.

Alternatively, you can create a second object variable and assign the Document object to it using VISIO's Document method, as the Set statement shows.

See Also

- Err Function
- Err Statement
- Error$ Function
- Error Statement
- Resume
- On Error
- Trappable Errors

Error Handling

See Also

OPEN Script contains three error handling statements and functions for trapping errors in your program: **Err**, **Error**, and **On Error**. OPEN Script returns a code for many of the possible runtime errors you might encounter. See **Trappable Errors** for a complete list of codes.

In addition to the errors trapped by OPEN Script, you might want to create your own set of codes for trapping errors specific to your program. You would do this if, for example, your program establishes rules for file input and the user does not follow the rules. You can trigger an error and respond appropriately using the same statements and functions you would use for OPEN Script-returned error codes.

Regardless of the error trapped, you have one of two methods to handle errors; one is to put error-handling code directly before a line of code where an error might occur (such as after a File Open statement), and the other is to label a separate section of the procedure just for error handling, and force a jump to that label if any error occurs. The On Error statement handles both options.

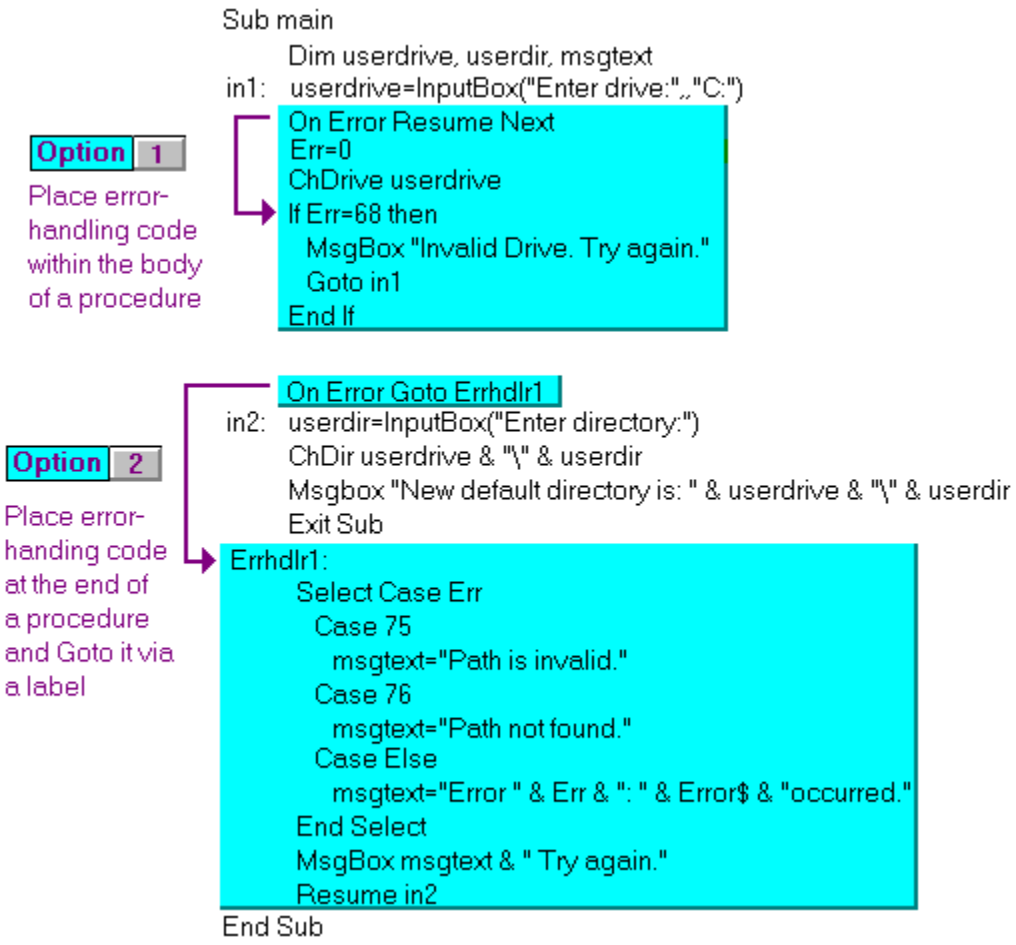
For more information, refer to one of the topics below:

Trapping Errors Returned by OPEN Script

Trapping User-defined (Non-OPEN Script) Errors

Trapping Errors Returned by OPEN Script

This code example shows the two ways to trap errors. Option 1 places error-handling code directly before the line of code that could cause an error. Option 2 contains a labeled section of code that handles any error.



Option 1: Trap error within body of code

The **On Error** statement identifies the line of code to go to in case of an error. In this case, the Resume Next parameter means execution continues with the next line of code after the error. In this example, the line of code to handle errors is the **If** statement. It uses the **Err** statement to determine which error code is returned.

Option 2: Trap error using error handler

The **On Error** statement used here specifies a label to jump to in case of errors. The code segment is part of the main procedure and uses the **Err** statement to determine which error code is returned. To make sure your code doesn't accidentally fall through to the error handler, precede it with an **Exit** statement.

Trapping User-Defined (Non-OPEN Script) Errors

These code examples show the two ways to set and trap user-defined errors. Both options use the **Error** statement to set the user-defined error to the value 30000. To trap the error, option 1 places error-handling code directly before the line of code that could cause an error. Option 2 contains a labeled section of code that handles any user-defined errors.

Option 1

Place error-handling code within the body of a procedure

```
Sub Main
    Dim custname as String
    On Error Resume Next
in1:  Err=0
    custname=InputBox$("Enter customer name:")
    If custname="" then
      Error 30000
      Select Case Err
        Case 30000
          MsgBox "You must enter a customer name."
          Goto in1
        Case Else
          MsgBox "Undetermined error. Try again."
          Goto in1
      End Select
    End If
    Msgbox "The name is: " & custname
End Sub
```

Option 2

Place error-handing code at the end of a procedure and Goto it via a label

```
Sub Main
    Dim custname as String
    On Error Goto Errhandler
in1:  Err=0
    custname=InputBox$("Enter customer name:")
    If custname="" then
      Error 30000
    End If
    Msgbox "The name is: " & custname
    Exit Sub
Errhandler:
    Select Case Err
      Case 30000
        MsgBox "You must enter a customer name."
      Case Else
        MsgBox "Undetermined error. Try again."
    End Select
    Resume in1
End Sub
```

How OPEN Script Compares to Other Versions of Basic

See also: **How OPEN Script Compares to Visual Basic and Word Basic**

Differences Between OPEN Script and Earlier Versions of Basic

If you are familiar with older versions of Basic (those that predate Windows), you will notice that OPEN Script includes many new features and changes from the language you have learned. OPEN Script more closely resembles other higher level languages popular today, such as C and Pascal.

The topics below describe some of the differences you will notice between the older Basics and OPEN Script.

Line Numbers and Labels

Older versions of Basic require numbers at the beginning of every line. More recent versions do not support these line numbers; in fact, they will generate error messages.

If you want to reference a line of code, you can use a label. A label can be any combination of text and numbers. Usually, it is a single word followed by a colon, which is placed at the beginning of a line of code. These labels are used by the **Goto** statement.

Subroutines and Modularity of the Language

OPEN Script is a modular language; code is divided into subroutines and functions. The subroutines and functions you write use the OPEN Script statements and functions to perform actions. The first subroutine in a script must be called main.

Global Variables

The placement of variable declarations determines their scope:

Scope	Definition
Local	Dimensioned inside a subroutine or function. The variable is accessible only to the subroutine or function that dimensioned it.
Module	Dimensioned outside any subroutine or function. The variable is accessible to any subroutine or function in the same file.
Global	Dimensioned outside any subroutine or function using the **Global** statement. The variable is accessible to any subroutine or function in any module (file).

Data Types

Modern Basic is now a typed language. In addition to the standard data types -- numeric, string, array, and record -- OPEN Script includes variants and objects.

Variables that are defined as variants can store any type of data. For example, the same variable can hold integers one time, and then, later in a procedure, it can hold strings.

OLE Automation objects give you the ability to manipulate complex data supplied by an application, such as windows or forms, or to use the functions of the application, such as, for a networking application, sending or receiving data over a network.

Dialog Box Handling

OPEN Script contains extensive dialog box support to give you great flexibility in creating and running your own custom dialog boxes. You define a dialog box with dialog control statements between the **Begin Dialog...End Dialog** statements, and then display it using the **Dialog** statement (or function).

OPEN Script stores information about the selections the user makes in the dialog box. When the dialog box is closed, your program can access this information.

OPEN Script also includes statements and functions to display other types of boxes:

message boxes notify the user of an event;

password boxes do not echo the user's keystrokes on the screen; and

input boxes prompt for a single line of input.

Financial Functions

OPEN Script includes a list of financial functions, for calculating such things as loan payments, internal rates of return, or future values based on a company's cash flows.

Date and Time Functions

The date and time functions have been expanded to make it easier to compare a file's date to today's date, set the current date and time, time events, and perform scheduling-type functions (such as finding the date for next Tuesday).

Object Handling

Windows includes OLE Automation object handling. An OLE Automation object can be the end product of a software application, such as a document from a word processing application.You can use such OLE Automation objects to link and embed objects from one application into another.

An OLE Automation object can also provide access to the functions of an application through the methods and properties of the object. You can use OLE Automation objects in scripts to automate the tasks of the application.

The **Object** data type permits your OPEN Script code to access another software application through its objects and use the features of the application or change its properties.

Environment Control

OPEN Script includes the ability to call another software application (**AppActivate**), and send the application keystrokes (**SendKeys**). Other environment control features include the ability to run an executable program (**Shell**), temporarily suspend processing to allow the operating system to process messages (**DoEvents**), and return values in the operating system environment table (**Environ$**).

How OPEN Script Compares to Visual Basic and Word Basic

There are several versions of Basic with which you might be familiar, the most common being Visual Basic and Word Basic. OPEN Script shares a substantial common core of functions and statements with these versions; however, each one has unique capabilities.

Differences Between OPEN Script and Visual Basic

OPEN Script is very similar to Microsoft Visual Basic; however, there are some differences.

Functions and Statements Unique to OPEN Script

OPEN Script offers a few statements and functions not found in Visual Basic:

- $CStrings
- $Include
- $NoCStrings
- GetField$
- SetField$

Control-Based Objects

OPEN Script does not predefine or include any Visual Basic object, such as a Button Control. As a result, a VB property such as BorderStyle is not an intrinsic part of OPEN Script. This does not mean that as an integrator, you cannot define an OPEN Script object that has BorderStyle as a property. You will probably define many objects that are intrinsic to your application in the process of integration.

Dialog Box Capabilities and VBA

VB does not have a syntax to create or run dialog boxes. In contrast, OPEN Script has a set of functions and statements to enable the use of dialog boxes (they are similar to those in Word).

Microsoft offers a modified version of VB in some of its products, such as Excel. Called Visual Basic for Applications (VBA), this version does provide dialog box handling statements and functions.

Differences Between OPEN Script and Word Basic

Word Basic is a precursor to Visual Basic that is included in Microsoft Word. Word Basic supports dialog boxes, but it does not support objects.

Dialog Box Capabilities

The dialog box capabilities in OPEN Script and Word are very similar. Word does offer some statements and functions that OPEN Script does not, such as DlgFilePreview. As well, OPEN Script offers some features that Word does not.

In response to the need for certain types of dialog box support, OPEN Script offered some dialog box options before Word Basic did. Later, Word Basic came out with their own syntax for these options. As a result, there are minor differences in the way the two languages handle dialog boxes.

Button vs. PushButton

Button is the original OPEN Script syntax; PushButton is the Word Basic syntax. The two are interchangeable, and OPEN Script supports both.

PushButton is preferred, and is used throughout the Examples.

Dialog Box Units

The measurement units used in the two dialog box syntaxes are different. OPEN Script supports both, and you can choose to use either.

The Examples use OPEN Script units. As a result, if you use Word units, some of the dialog

boxes created in the Examples might look odd.

User Input Mechanisms

There are slight differences in some of the mechanisms for user input:

OPEN Script	Word Basic
StaticComboBox or ComboBox (in OPEN Script, these are interchangeable)	ComboBox (Word Basic supports only this syntax)
DropComboBox	N/A

Trappable Errors

The following table lists the runtime errors that OPEN Script returns. These errors can be trapped by **On Error**. The **Err** function can be used to query the error code, and the **Error** function can be used to query the error text.

Error code	Error Text
5	Illegal function call
6	Overflow
7	Out of memory
9	Subscript out of range
10	Duplicate definition
11	Division by zero
13	Type Mismatch
14	Out of string space
19	No Resume
20	Resume without error
28	Out of stack space
35	Sub or Function not defined
48	Error in loading DLL
52	Bad filename or number
53	File not found
54	Bad file mode
55	File already open
58	File already exists
61	Disk full
62	Input past end of file
63	Bad record number
64	Bad filename
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 set to Nothing
93	Invalid pattern
94	Illegal use of NULL
102	Command failed
429	Object creation failed
438	No such property or method
439	Argument type mismatch
440	Object error
901	Input buffer would be larger than 64K

902	Operating system error
903	External procedure not found
904	Global variable type mismatch
905	User-defined type mismatch
906	External procedure interface mismatch
907	Pushbutton required
908	Module has no MAIN
910	Dialog box not declared

Derived Trigonometric Functions

A number of trigonometric functions can be written in Basic using the built-in functions. The following table lists several of these functions:

Function	Computed By:
Secant	Sec(x) = 1/Cos(x)
CoSecant	CoSec(x) = 1/Sin(x)
CoTangent	CoTan(x) = 1/Tan(x)
ArcSine	ArcSin(x) = Atn(x/Sqr(-x*x+1))
ArcCosine	ArcCos(x) = Atn(-x/Sqr(-x*x+1))+1.5708
ArcSecant	ArcSec(x) = Atn(x/Sqr(x*x-1))+Sgn(x-1)*1.5708
ArcCoSecant	ArcCoSec(x) = Atn(x/Sqr(x*x-1))+(Sgn(x)-1)*1.5708
ArcCoTangent	ArcTan(x) = Atn(x)+1.5708
Hyperbolic Sine	HSin(x) = (Exp(x)-Exp(-x))/2
Hyperbolic Cosine	HCos(x) = (Exp(x)+Exp(-x))/2
Hyperbolic Tangent	HTan(x) = (Exp(x)-Exp(-x))/(Exp(x)+Exp(-x))
Hyperbolic Secant	HSec(x) = 2/(Exp(x)+Exp(-x))
Hyperbolic CoSecant	HCoSec(x) = 2/(Exp(x)-Exp(-x))
Hyperbolic Cotangent	HCotan(x) = (Exp(x)+Exp(-x))/(Exp(x)-Exp(-x))
Hyperbolic ArcSine	HArcSin(x) = Log(x+Sqr(x*x+1))
Hyperbolic ArcCosine	HArcCos(x) = Log(x+Sqr(x*x-1))
Hyperbolic ArcTangent	HArcTan(x) = Log((1+x)/(1-x))/2
Hyperbolic ArcSecant	HArcSec(x) = Log((Sqr(-x*x+1)+1)/x)
Hyperbolic ArcCoSecant	HArcCoSec(x) = Log((Sgn(x)*Sqr(x*x+1)+1)/x)
Hyperbolic ArcCoTangent	HArcCoTan(x) = Log((x+1)/(x-1))/2