

•  
•  
•  
•  
•  
•  
•

SCSI Toolbox LLC  
PO Box 620520  
Littleton, CO 80162  
(303) 972-2072  
[www.scsitoolbox.com](http://www.scsitoolbox.com)



• • • • • • • • • •

**SCSI Toolbox, LLC**  
**SCRIPTWRITER PRO**  
**Language Reference Manual**  
**Version 5.0 – March 2004**





# Table of Contents

**SCRIPTWRITER PRO SCRIPTING LANGUAGE ELEMENTS ..... 9**

COMMENTS ..... 9

NUMBERS ..... 10

VARIABLE AND CONSTANT NAMES ..... 10

VARIABLE TYPES ..... 10

OTHER DATA TYPES ..... 12

CONTROL STRUCTURES ..... 13

SUBROUTINES AND FUNCTIONS ..... 15

BYREF AND BYVAL ..... 15

CALLING PROCEDURES IN DLLS ..... 17

FILE INPUT/OUTPUT ..... 18

ARRAYS ..... 19

USER DEFINED TYPES ..... 22

DIALOG SUPPORT ..... 23

STATEMENTS AND FUNCTIONS USED IN DIALOG FUNCTIONS ..... 30

*DlgControlId Function* ..... 31

*DlgFocus Statement, DlgFocus() Function* ..... 31

*DlgListBoxArray, DlgListBoxArray()* ..... 32

*DlgSetPicture* ..... 32

*DlgValue, DlgValue()* ..... 32

OLE AUTOMATION ..... 33

WHAT IS OLE AUTOMATION? ..... 33

ACCESSING AN OBJECT ..... 34

WHAT IS AN OLE OBJECT? ..... 35

OLE FUNDAMENTALS ..... 37

OLE AUTOMATION AND MICROSOFT WORD EXAMPLE: ..... 37

MAKING APPLICATIONS WORK TOGETHER ..... 38

THE REGISTRATION DATABASE ..... 39

**SCRIPTING LANGUAGE OVERVIEW ..... 41**

    QUICK REFERENCE OF THE FUNCTIONS AND STATEMENTS AVAILABLE ..... 41

**LANGUAGE REFERENCE A - Z ..... 46**

ABS FUNCTION ..... 46

APPACTIVATE STATEMENT ..... 47

ASC FUNCTION ..... 48

ATN FUNCTION ..... 48

BEEP STATEMENT ..... 48

CALL STATEMENT ..... 49

CBOOL FUNCTION ..... 50

CDATE FUNCTION ..... 50

CDBL FUNCTION ..... 51

ChDIR STATEMENT .....	51
ChDRIVE STATEMENT .....	52
CHECKBOX .....	52
CHOOSE FUNCTION .....	53
CHR FUNCTION .....	53
CINT FUNCTION .....	54
CLNG FUNCTION .....	54
CLOSE STATEMENT .....	55
CONST STATEMENT .....	56
COS FUNCTION .....	57
CREATEOBJECT FUNCTION .....	57
CSNG FUNCTION .....	58
CSTR FUNCTION .....	59
CURDIR FUNCTION .....	59
CVAR FUNCTION .....	60
DATE FUNCTION .....	60
DATE SERIAL FUNCTION .....	61
DATEVALUE FUNCTION .....	62
DAY FUNCTION .....	62
DECLARE STATEMENT .....	63
DIALOG, DIALOG FUNCTION .....	64
DIM STATEMENT .....	65
DIR FUNCTION .....	66
DLGENABLE STATEMENT .....	67
DLGTEXT STATEMENT .....	68
DLGVISIBLE STATEMENT .....	68
Do... LOOP STATEMENT .....	69
END STATEMENT .....	70
EOF FUNCTION .....	70
ERASE STATEMENT .....	71
EXIT STATEMENT .....	72
EXP .....	72
FILECOPY FUNCTION .....	73
FILELEN FUNCTION .....	73
FIX FUNCTION .....	73
FOR EACH ... NEXT STATEMENT .....	74
FOR...NEXT STATEMENT .....	74
FORMAT FUNCTION .....	75
FREEFILE FUNCTION .....	85
FUNCTION STATEMENT .....	85
GET OBJECT FUNCTION .....	86
GLOBAL STATEMENT .....	87
GoTo STATEMENT .....	87
HEX .....	88
HOUR FUNCTION .....	88
IF... THEN... ELSE STATEMENT .....	89
INPUT FUNCTION .....	91
INPUTBOX FUNCTION .....	91
INSTR .....	92
INT FUNCTION .....	92
ISARRAY FUNCTION .....	93
ISDATE .....	93
ISEMPTY .....	94
ISNULL .....	94
ISNUMERIC .....	94

ISOBJECT FUNCTION .....	95
KILL STATEMENT .....	96
LBOUND FUNCTION.....	96
LCASE, FUNCTION .....	97
LEFT .....	97
LEN .....	98
LET STATEMENT.....	98
LINE INPUT # STATEMENT.....	99
LOF .....	99
LOG .....	100
MID FUNCTION .....	100
MINUTE FUNCTION.....	101
MKDIR .....	102
MONTH FUNCTION .....	103
MSGBOX FUNCTION MSGBOX STATEMENT.....	103
NAME STATEMENT .....	105
NOW FUNCTION.....	105
OCT FUNCTION.....	106
OKBUTTON.....	106
ON ERROR .....	107
OPEN STATEMENT .....	109
OPTION BASE STATEMENT .....	111
OPTION EXPLICIT STATEMENT .....	112
PRINT METHOD .....	112
PRINT # STATEMENT .....	113
REDIM STATEMENT.....	115
REM STATEMENT.....	115
RIGHT FUNCTION.....	116
RMDIR STATEMENT .....	116
RND FUNCTION.....	117
SECOND FUNCTION.....	117
SEEK FUNCTION .....	118
SEEK STATEMENT.....	119
SELECT CASE STATEMENT .....	119
SENDKEYS FUNCTION .....	120
SET STATEMENT .....	121
SHELL FUNCTION .....	122
SIN FUNCTION .....	123
SPACE FUNCTION .....	123
SQR FUNCTION .....	123
STATIC STATEMENT .....	124
STOP STATEMENT.....	125
STR FUNCTION .....	125
STRCOMP FUNCTION.....	126
STRING FUNCTION.....	126
SUB STATEMENT .....	127
TAN FUNCTION.....	127
TEXT STATEMENT .....	128
TEXTBOX STATEMENT .....	128
TIME FUNCTION.....	129
TIMER EVENT .....	129
TIMESERIAL - FUNCTION.....	130
TIMEVALUE - FUNCTION.....	130
TRIM, LTRIM, RTRIM FUNCTIONS.....	131
TYPE STATEMENT.....	131

UBOUND FUNCTION .....	133
UCASE FUNCTION .....	133
VAL .....	134
VARTYPE .....	134
WEEKDAY FUNCTION .....	135
WHILE... WEND STATEMENT .....	135
WITH STATEMENT .....	136
WRITE # - STATEMENT .....	137
YEAR FUNCTION.....	138
<b>DEVELOPER TOOLBOX SCSI EXTENSION FUNCTIONS .....</b>	<b>139</b>
MULTITHREADED SCSI COMMANDS .....	139
<i>IssueThreadedCDB</i> .....	139
<i>GetThreadedCDBStatus</i> .....	141
<i>GetThreadedCDBStatusWData</i> .....	142
<i>ReleaseThreadID</i> .....	143
<i>MultiThreaded CDB example</i> .....	143
<b>MULTITHREADED DISK TEST FUNCTIONS.....</b>	<b>146</b>
INTRODUCTION TO DTB DISK TESTS .....	146
<i>DTB Disk Test functions are broken into four types of operations:</i> .....	146
<i>DTB Disk Test Functions</i> .....	146
<i>SCSIPrepareForNewDiskTestSequence</i> .....	146
<i>SCSIStartDiskTestSequence</i> .....	147
<i>This function starts the test sequence running on each device specified in the test device list. Once the test sequence has been started you will use the various SCSIGetDiskTestStatus() functions to check on the progress of each device being tested.</i> .....	147
<i>SCSIAddDiskDeviceToBeTested</i> .....	147
<i>SCSIGetDiskTestStatus</i> .....	147
<i>SCSIGetDiskTestStatusWData</i> .....	147
<i>The Following Functions add various tests into the test sequence list. These functions will define the test steps as write, read, seek, start/stop, and will specify if the test is run for a period of time or for a number of blocks transferred, etc. Data patterns are also specified here.</i> .....	148
<i>SCSIAddDiskWriteTest_Time</i> .....	148
<i>SCSIAddDiskWriteTest_Blocks</i> .....	148
<i>SCSIAddDiskReadTest_Time</i> .....	148
<i>SCSIAddDiskReadTest_Blocks</i> .....	148
<i>SCSIAddDiskWriteReadTest_Time</i> .....	148
<i>SCSIAddDiskWriteReadTest_Blocks</i> .....	149
<i>SCSIAddDiskSeekTest_Time</i> .....	149
<i>SCSIAddDiskSeekTest_Blocks</i> .....	149
<i>SCSIAddDiskVerifyTest_Time</i> .....	149
<i>SCSIAddDiskVerifyTest_Blocks</i> .....	149
<i>SCSIAddDiskWriteVerifyTest_Time</i> .....	149
<i>SCSIAddDiskWriteVerifyTest_Blocks</i> .....	149
<i>SCSIAddDiskFormatTest_SameSize</i> .....	150
<i>SCSIAddDiskFormatTest_NewSize</i> .....	150
<i>SCSIAddDiskFWDownloadTest</i> .....	150
<i>SCSIAddDiskSpinTest</i> .....	150
<i>SCSIPauseDiskTest</i> .....	150
<i>SCSIResumeDiskTest</i> .....	150
<i>SCSIStopDiskTest</i> .....	151
<i>SCSIStopAllDiskTest</i> .....	151
<b>MULTITHREADED TAPE TEST FUNCTIONS.....</b>	<b>151</b>

INTRODUCTION TO DTB TAPE TESTS .....	151
<i>DTB Tape Test functions are broken into four types of operations:</i> .....	151
<i>DTB Tape Test Functions</i> .....	152
<i>SCSIPrepareForNewTapeTestSequence</i> .....	152
<i>SCSIStartTapeTestSequence</i> .....	152
<i>SCSIAddTapeDeviceToBeTested</i> .....	152
<i>SCSIGetTapeTestStatus</i> .....	152
<i>SCSIGetTapeTestStatusWData</i> .....	153
<i>Returns detailed test progress data (including Sense data) on the specified device.</i> .....	153
<i>The Following Functions add various tests into the test sequence list. These functions will define the test steps as write, read, space, rewind, and will specify if the test is run for a period of time or for a number of blocks transferred, etc. Data patterns are also specified here.</i> .....	153
<i>SCSIAddTapeWriteTest_Time</i> .....	153
<i>SCSIAddTapeWriteTest_MBytes</i> .....	153
<i>SCSIAddTapeReadTest_Time</i> .....	153
<i>SCSIAddTapeReadTest_</i> .....	154
<i>SCSIAddTapeWriteFMTest</i> .....	154
<i>SCSIAddTapeReadFMTest</i> .....	154
<i>SCSIAddTapeRewindTest</i> .....	154
<i>SCSIAddTapeSpaceTest</i> .....	154
<i>SCSIAddTapeCompressionTest</i> .....	154
<i>SCSIAddTapeLogPageTest</i> .....	154
<i>SCSIAddTapeSynchronizeTest</i> .....	154
<i>SCSIAddTapeExternalProgram</i> .....	155
<i>SCSIPauseTapeTest</i> .....	155
<i>SCSIResumeTapeTest</i> .....	155
<i>SCSIStopTapeTest</i> .....	155
<i>SCSIStopAllTapeTest</i> .....	155
LSI SCSI PROTOCOL CONTROL FUNCTIONS .....	156
<i>SCSIHostAdapterCount</i> .....	156
<i>SCSIValidTarget</i> .....	156
<i>SCSIPSetSpeed5</i> .....	156
<i>SCSIPSetSpeed10</i> .....	156
<i>SCSIPSetSpeed20</i> .....	157
<i>SCSIPSetSpeed40</i> .....	157
<i>SCSIPSetSpeed80</i> .....	157
<i>SCSIPSetSpeed160</i> .....	157
<i>SCSIPSetSpeed320</i> .....	157
<i>SCSIPSetWide</i> .....	158
<i>SCSIPSetNarrow</i> .....	158
<i>SCSIPSetNegotiated</i> .....	158
<i>SCSIPResetBus</i> .....	158
<i>SCSIPGetNegotiated</i> .....	158
SCSIAND .....	159
SCSIBUFFER2FILE .....	159
SCSIBUFFERSIZE .....	159
SCSICHECKRANDOMBLOCK .....	159
SCSICOMPAREBUFFERS .....	160
SCSICMQ .....	161
SCSIDEC2HEX .....	161
SCSIDISKCORRUPTBLOCK .....	162
SCSIDISKGETECCSPAN .....	162
SCSIGETREADLONGSIZE .....	162
SCSIDISKREAD .....	162
SCSIDISKREADFUA .....	163

SCSIDISKREADLONG .....	163
SCSIDISKSTARTSTOP .....	164
SCSIDISKUNLOAD .....	164
SCSIDISKVERIFY .....	165
SCSIDISKWRITE .....	165
SCSIDISKWRITEFUA .....	167
SCSIDISKWRITELONG .....	167
SCSIDLT_FWDL .....	167
SCSIFILE2BUFFER .....	167
SCSIFILEOFFSET2BUFFER .....	168
SCSIFILLBLOCKNUM .....	168
SCSIFILLBUFFER .....	169
SCSIFILLPATTERN .....	169
SCSIFILLRANDOM .....	169
SCSIGETBUFFER .....	171
SCSIGETBUFFER MODE .....	172
SCSIGETDEVICETYPE .....	172
SCSIGETDLLVERSION .....	173
SCSIERRORDETAILS .....	173
SCSIGETPRODUCT .....	173
SCSIGETRANDOMERRORS .....	174
SCSIGETAPECAPACITY .....	174
SCSIGETVENDOR .....	175
SCSIGETVERSION .....	175
SCSIHEX2DEC .....	176
SCSIHOSTADAPTERCOUNT .....	176
SCSIHPLTO_FWDL .....	176
SCSIIBMLTO_FWDL .....	176
SCSIINITIALIZEELEMENTSTATUS .....	176
SCSIINITIALIZEELEMENTSTATUSRANGE .....	177
SCSIINQUIRY .....	178
SCSILOADBUFFER .....	178
SCSILOGSENSE .....	179
SCSIMODESENSE .....	181
SCSIMODESELECT .....	181
SCSIMODESELECTFULL .....	181
SCSIMODESENSEFULL .....	182
SCSIMOVEMEDIUM .....	182
SCSIOR .....	183
SCSIPOSITIONTOELEMENT .....	183
SCSIREADCAPACITY .....	184
SCSIREADELEMENT STATUS .....	184
SCSIRESETHBA .....	190
SCSIROLLPATTERN .....	190
SCSISEAGATELTO_FWDL .....	190
SCSISONYAIT_FWDL .....	191
SCSISEARCHBUFFER .....	191
SCSISDLT_FWDL .....	191
SCSISEGMENTED_FWDL .....	192
SCSISetBUFFER MODE .....	193
SCSISetTIMEOUT .....	193
SCSITAPEBLOCKSIZE .....	194
SCSITAPEREWIND .....	194
SCSITAPEUNLOAD .....	194
SCSITAPEWFM .....	195

SCSITAPERADF.....	195
SCSITAPEWRITEF.....	197
SCSITAPERADV.....	197
SCSITAPEWRITEV.....	197
SCSITAPEFSF.....	197
SCSITAPEFSR.....	198
SCSITAPESPACEEOD.....	198
SCSITARGETCOUNT.....	199
SCSITUR.....	199
SCSIUSERCDB.....	199
SCSIVIEWSENSE.....	200
SCSIXOR.....	201



# ScriptWriter Pro Scripting Language Elements

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

---

## Comments

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

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

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

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

Everything after the apostrophe is a comment.

## Statements:

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

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

## Which is equivalent to:

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

## Line Continuation Character:

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

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

---

## Numbers

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

---

## Variable and Constant Names

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

---

## Variable Types

### Variant

As is the case with Visual Basic, when a variable is introduced in ScriptWriter Pro, it is not necessary to declare it first (see option

explicit for an exception to this rule). When a variable is used but not declared then it is implicitly declared as a **variant** data type. Variants can also be declared explicitly using "As Variant" as in Dim x As Variant. The variant data type is capable of storing numbers, strings, dates, and times. When using a variant you do not have to explicitly convert a variable from one data type to another. This data type conversion is handled automatically.

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



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

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

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

## Variants and Concatenation

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

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

---

## Other Data Types

The twelve data types available in ScriptWriter Pro are shown below:

### Data Types

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

### Scope of Variables

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

### Declaration of Variables

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

```
Sub Main
```

```

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

End Sub

```

---

## Control Structures

ScriptWriter Pro has complete process control functionality. The control structures available are **Do** loops, **While** loops, **For** loops, **Select Case**, **If Then**, and **If Then Else**. In addition, ScriptWriter Pro has two branching statements: **GoTo** and **GoSub**. The **Goto** Statement branches to the label specified in the **Goto** Statement.

```

Goto label1
.
.
.

label1:

```

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

The **GoSub** statement also requires a label to be specified like the **GoTo** statement. The difference between the **Goto** and the **GoSub** is that the **GoSub** statement returns to its original position in the code after executing a return statement.

```

GoSub Label2
.
.
.

Label2:
.
.
.

Return

```

## Loop Structures

### Do Loops

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

```

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

```

```
Do Until condition
Statement(s)...
Loop
```

```
Do
Statements...
Loop While condition
```

```
Do
statements...
Loop Until condition
```

**Do While** and **Do Until** check the condition before entering the loop, thus the block of statements inside the loop are only executed when those conditions are met. **Do Loop While** and **Do Loop Until** check the condition after having executed the block of statements thereby guaranteeing that the block of statements is executed at least once.

### While Loop

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

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

### For ... Next Loop

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

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

### If and Select Statements

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

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

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

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

cumbersome, at which time you should consider using the **Select** statement.

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

The **Select Case** statement tests the same variable for many different values. This statement tends to be easier to read, understand and follow and should be used in place of a complicated **If...Then...ElseIf** statement.

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

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

---

## Subroutines and Functions

### Naming conventions

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

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

---

## ByRef and ByVal

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

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

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

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

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

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

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

### Scalar Variables

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

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

### Passing User Defined Types by Ref to DLL's and ScriptWriter Pro functions

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



```

        szPathName As String * 128
    End Type

    ' OpenFile() Flags
    Global Const OF_READ = &H0
    Global Const OF_WRITE = &H1
    Global Const OF_READWRITE = &H2
    Global Const OF_SHARE_COMPAT = &H0
    Global Const OF_SHARE_EXCLUSIVE = &H10
    Global Const OF_SHARE_DENY_WRITE = &H20
    Global Const OF_SHARE_DENY_READ = &H30
    Global Const OF_SHARE_DENY_NONE = &H40
    Global Const OF_PARSE = &H100
    Global Const OF_DELETE = &H200
    Global Const OF_VERIFY = &H400
    Global Const OF_CANCEL = &H800
    Global Const OF_CREATE = &H1000
    Global Const OF_PROMPT = &H2000
    Global Const OF_EXIST = &H4000
    Global Const OF_REOPEN = &H8000

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

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

```

---

## Calling Procedures in DLLs

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

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

```

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

```

```

Declare Sub InvertRect Lib "User" (ByVal hDC AS Integer, aRect As Rectangle)

```

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

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

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

### **Passing and Returning Strings**

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

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

#### OLE API

SysAllocString/SysAllocStringLength

SysAllocString/SysAllocStringLength

SysFreeString

SysStringLength

SysReAllocStringLength

SysReAllocString

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

---

## **File Input/Output**

ScriptWriter Pro supports full sequential file I/O.

Functions and Statements that apply to sequential file access:

## Dir, EOF, FileCopy, FileLen, Seek, Open, Close, Input, Line Input, Print and Write

```
' File I/O Examples

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

End Sub

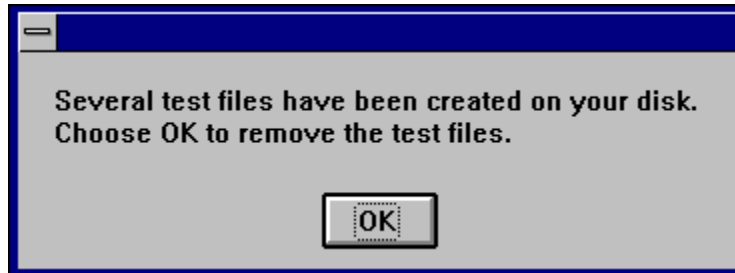
Sub test

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

End Sub

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

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



---

## Arrays

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

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

Ways to declare a fixed-size array:

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

ScriptWriter Pro supports Dynamic arrays.

### Declaring an Array

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

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

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

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

Sub Main () ...
```

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

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

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

```
Option Base 1
```

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

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

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

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

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

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

### MultiDimensional Arrays

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

```
Static Mat(20, 20) As Double
```

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

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

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

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

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

```
Dim ArrTest(5, 3, 2)
```

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

---

## User Defined Types

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

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

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

Type type2
  a As Integer
  o As type1
End Type

Dim type2a As type2
Dim type1a As type1

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



---

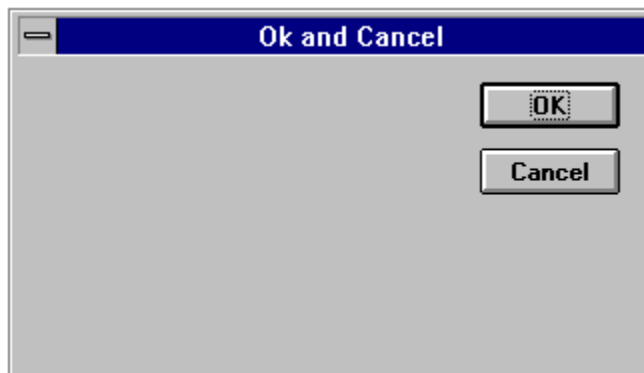
## Dialog Support

ScriptWriter Pro has support for custom dialogs. The syntax is similar to the syntax used in Microsoft Word Basic. The dialog syntax is not part of Microsoft Visual Basic or Microsoft Visual Basic For Applications (VBA). ScriptWriter Pro has complete support for dialogs. The type of dialogs supported are outlined below.

### Dialog Box controls

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

### OK and Cancel Buttons



```
Sub Main
  Begin Dialog ButtonSample 16,32,180,96,"OK and Cancel"
```

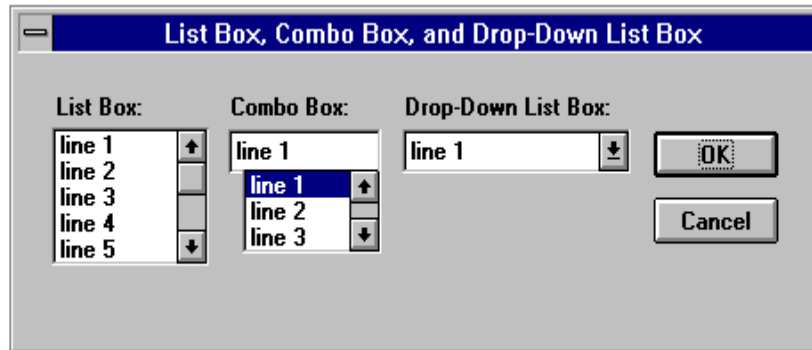
```

        OKButton 132,8,40,14
        CancelButton 132,28,40,14
    End Dialog
        Dim Dlg1 As ButtonSample
        Button = Dialog (Dlg1)
    End Sub

```

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

## List Boxes, Combo Boxes and Drop-down List Boxes



```

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

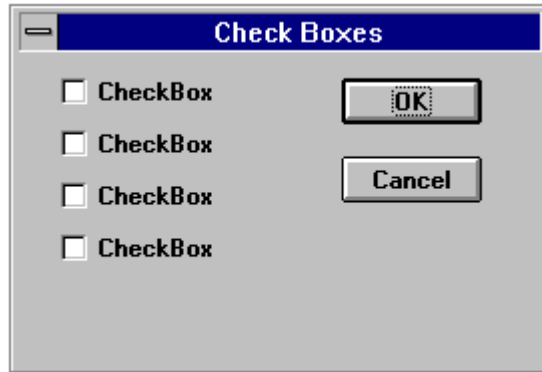
    Begin Dialog BoxSample 16,35,256,89,"List Box, Combo Box, and Drop-Down List Box"
        OKButton 204,24,40,14
        CancelButton 204,44,40,14
        ListBox 12,24,48,40, MyList$ ( ),.Lstbox
        DropListBox 124,24,72,40, MyList$ ( ),.DrpList
        ComboBox 68,24,48,40, MyList$ ( ),.CmboBox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

```

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



## Check Boxes



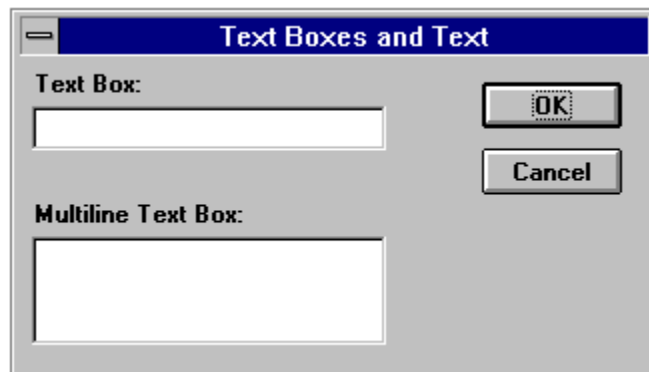
```

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

```

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

## Text Boxes and Text



```

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

```

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

```

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

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

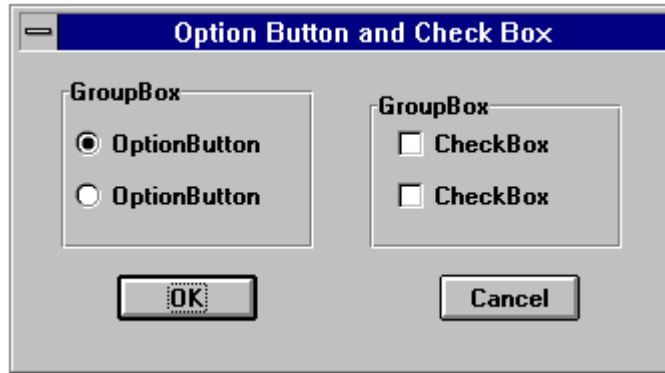
    MsgBox "TextBox: "& Dlg1.joe
End Sub

```

## Option Buttons and Group Boxes

You can have option buttons to allow the user to choose one option from several. Typically, you would use a group box to surround a group of option

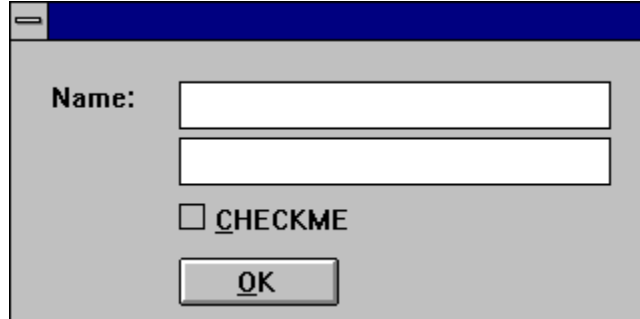
buttons, but you can also use a group box to set off a group of check boxes or any related group of controls.



```

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

```



```

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

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

```

## The Dialog Function

ScriptWriter Pro supports the dialog function. This function is a user-defined function that can be called while a custom dialog box is displayed. The dialog function makes nested dialog boxes possible and receives messages from the dialog box while it is still active.

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

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

```
Begin Dialog UserDialog1 60,60, 260, 188, "3", .ScriptWriter Pro
```

## The Dialog Box Controls

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

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

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

## The Dialog Function Syntax

The syntax for the dialog function is as follows:

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

All parameters in the dialog function are required.

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

**ControlID\$** Receives the identifier of the dialog box control

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

**Action 1** The value passed before the dialog becomes visible

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

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

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

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

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

---

## Statements and Functions Used in Dialog Functions

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

	control.
DlgText, DlgText	The <b>DlgText</b> statement is used to set the text or text label for a dialog box control. <b>TheDlgText()</b> function returns the label of a control.
DlgValue, DlgValue()	The <b>DlgValue</b> statement is used to select or clear a dialog box control. Then <b>DlgValue()</b> function returns the setting of a control.
DlgVisible, DlgVisible()	The <b>DlgVisible</b> statement is used to hide or show a dialog box control. The <b>DlgVisible()</b> function is used to determine whether a control is visible or hidden.

## DlgControlId Function

DlgControlId(*Identifier*)

Used within a dialog function to return the numeric identifier for the dialog box control specified by *Identifier*, the string identifier of the dialog box control. Numeric identifiers are numbers, starting at 0 (zero), that correspond to the positions of the dialog box control instructions within a dialog box definition. For example, consider the following instruction in a dialog box definition:

```
CheckBox 90, 50, 30, 12, "&Update", .MyCheckBox
```

The instruction DlgControlId("MyCheckBox") returns 0 (zero) if the CheckBox instruction is the first instruction in the dialog box definition, 1 if it is the second, and so on.

In most cases, your dialog functions will perform actions based on the string identifier of the control that was selected.

## DlgFocus Statement, DlgFocus() Function

DlgFocus Identifier

DlgFocus()

The DlgFocus statement is used within a dialog function to set the focus on the dialog box control identified by Identifier while the dialog box is displayed. When a dialog box control has the focus, it is active and responds to keyboard input. For example, if a text box has the focus, any text you type appears in that text box.

The DlgFocus() function returns the string identifier for the dialog box control that currently has the focus.

### Example:

This example sets the focus on the control "MyControl1" when the dialog box is initially displayed. (The main subroutine that contains the dialog box definition is not shown.)

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

End Function

### **DlgListBoxArray, DlgListBoxArray()**

DlgListBoxArray Identifier, ArrayVariable()

DlgListBoxArray(Identifier, ArrayVariable())

The DlgListBoxArray statement is used within a dialog function to fill a ListBox, DropListBox, or ComboBox with the contents of ArrayVariable() while the dialog box is displayed.

The DlgListBoxArray() function fills ArrayVariable() with the contents of the ListBox, DropListBox, or ComboBox specified by Identifier and returns the number of entries in the ListBox, DropListBox, or ComboBox. The ArrayVariable() parameter is optional (and currently not implemented) with the DlgListBoxArray() function; if ArrayVariable() is omitted, DlgListBoxArray() returns the number of entries in the specified control.

### **DlgSetPicture**

DlgSetPicture Identifier, PictureName

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

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

### **DlgValue, DlgValue()**

DlgValue Identifier, Value

DlgValue(Identifier)

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

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

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

```
Sub Main
```

```
Begin Dialog UserDialog1 60,60, 260, 188, "3", .ScriptWriter Pro
Text 8,10,73,13, "Text Label:"
```



```

        TextBox 8, 26, 160, 18, .FText
        CheckBox 8, 56, 203, 16, "Check to display controls",. Chk1
        GroupBox 8, 79, 230, 70, "This is a group box:", .Group
        CheckBox 18,100,189,16, "Check to change button text", .Chk2
        PushButton 18, 118, 159, 16, "File History", .History
        OKButton 177, 8, 58, 21
        CancelButton 177, 32, 58, 21
    End Dialog

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

Function ScriptWriter Pro( ControlID$, Action%, SuppValue%)

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

Select Case Action%

Case 1
    DlgScriptWriter Pro "Group", 0
    DlgVisible "Chk2", 0
    DlgVisible "History", 0

Case 2
    If ControlID$ = "Chk1" Then
        DlgScriptWriter Pro "Group"
        DlgVisible "Chk2"
        DlgVisible "History"
    End If

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

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

Case Else

End Select
ScriptWriter Pro =1

End Function

```

---

## OLE Automation

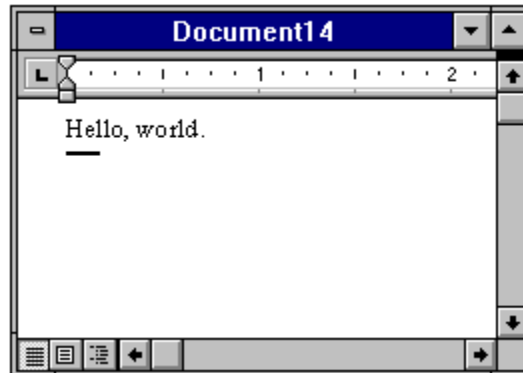
### What is OLE Automation?

OLE Automation is a standard, promoted by Microsoft, that applications use to expose their OLE objects to development tools, ScriptWriter Pro Basic, and containers that support OLE Automation. A spreadsheet application may

expose a worksheet, chart, cell, or range of cells all as different types of objects. A word processor might expose objects such as application, paragraph, sentence, bookmark, or selection.

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

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



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

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

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

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

---

## Accessing an object

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

Name	Description
------	-------------

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

## What is an OLE Object?

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

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

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

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

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

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

		StatusBar
		Top
		Value
		Visible
		Width

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

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

Collection object	Methods	Properties
Documents	Add	Application
	Close	Count
	Item	Parent

	Open	

---

## OLE Fundamentals

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

The following terms and concepts are fundamental to understanding OLE.

### OLE Object

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

### OLE Automation

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

### Class

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

---

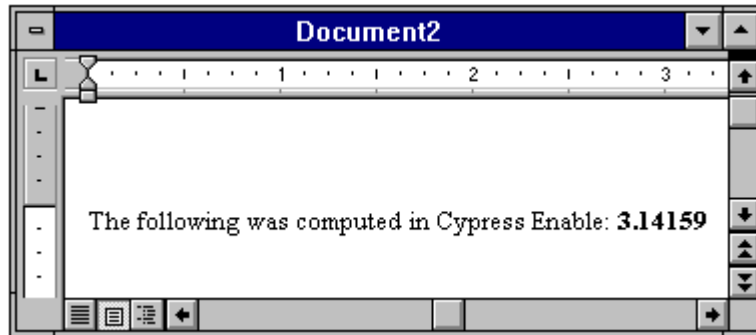
## OLE Automation and Microsoft Word example:

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

```
Dim myData As String

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

MsgBox "Done"
End Sub
```



---

## Making Applications Work Together

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

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

### WIN.INI

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

### The Registration Database

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

## The Registration database

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

### Associations.

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

### Shell Operations.

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

### OLE Object Servers.

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

### DDE/OLE Automation.

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

It is useful to appreciate the difference in structure between the **WIN.INI** file and the **REG.DAT** database. **WIN.INI** is simply a text document. There are no special structures other than headings (simply titles enclosed in brackets) that organize the information. If you want to locate an item in the **WIN.INI** file, you must search through the file for the specific item you want to locate. The registration database is a tree-like, structured database used for storing

information relating to program and file operations, in particular, those that involve the use of DDE or OLE. The tree structure makes it easier to keep the complex set of instructions, needed to implement DDE and OLE operations, organized and accessible by the applications that need to use them. This is not possible when you are working with a text document like **WIN.INI**. The **WIN.INI** file records all sorts of information about the Windows system in a simple sequential listing.



# Scripting Language Overview

---

## Quick reference of the Functions and Statements available

### Type/Functions/Statements

---

#### Flow of Control

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

---

#### Converting

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

---

#### Dialog

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

---

#### File I/O

FileCopy, ChDir, ChDrive, CurDir, CurDir, Mkdir,Rmdir, Open, Close, Print #, Kill, FreeFile, LOF, FileLen, Seek, EOF, Write #, Input, Line Input, Dir, Name, GetAttr, SetAttr, Dir,

---

**Math**

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

---

**Procedures**

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

---

**Strings**

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

---

**Variables and Constants**

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

---

**Error Trapping**

On Error, Resume

---

**Date/Time**

Date, Now, Time, Timer

---

**DDE**

DDEInitiate, DDEExecute, DDETerminate

---

**Arrays**

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

---

**Miscellaneous**

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

**Data Types**

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

## Operators

### Arithmetic Operators

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

\*Arithmetic operators follow mathematical rules of precedence

\* '+' or '&' can be used for string concatenation.

### Operator Precedence

Operator	Description	Order
()	parenthesis	highest
^	exponentiation	

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

## Relational Operators

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

## Logical Operators

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

## Functions, Statements, Reserved words - Quick Reference

---

Abs, Access, Alias, And Any

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

---

# Language Reference A - Z

---

## Abs Function

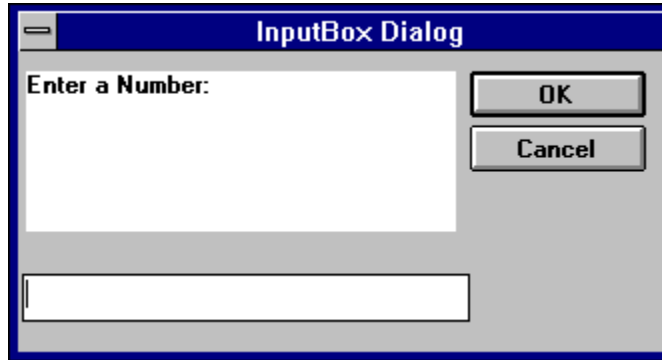
*Abs (number)*

Returns the absolute value of a number.

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

### Example:

```
Sub Main
    Dim Msg, X, Y
    X = InputBox("Enter a Number:")
    Y = Abs(X)
    Msg = "The number you entered is " & X
    Msg = Msg + ". The Absolute value of " & X & " is " & Y
    MsgBox Msg 'Display Message.
End Sub
```



---

## AppActivate Statement

AppActivate "*app*"

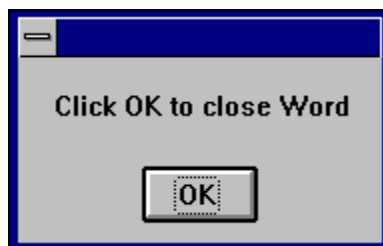
Activates an application.

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

Related Topics: Shell, SendKeys

### Example:

```
Sub Main ()
    AppActivate "Microsoft Word"
    SendKeys "%F,%N, ScriptWriter Pro",True
    Msg = "Click OK to close Word"
    MsgBox Msg
    AppActivate "Microsoft Word" ' Focus back to Word
    SendKeys "%F,%C,N", True      'Close Word
End Sub,, ScriptWriter Pro
```



## Asc Function

Asc (*str*)

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

### Example:

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

---

## Atn Function

Atn (*rad*)

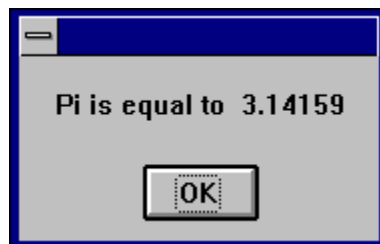
Returns the arc tangent of a number

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

Related Topics: Cos, Tan, Sin

### Example:

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



---

## Beep Statement

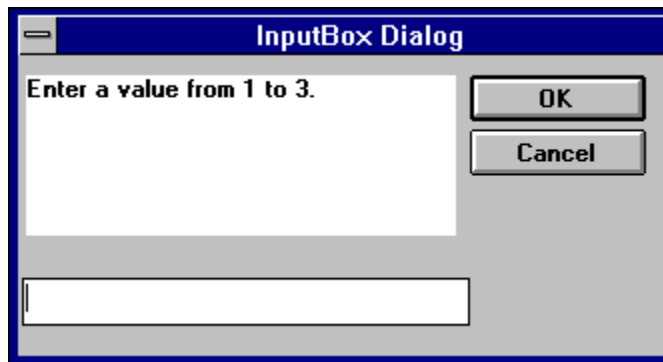


## Beep

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

### Example:

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



---

## Call Statement

Call funcname [(parameter(s))]

or

[parameter(s)]

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

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

### Example:

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



---

## CBool Function

CBool (*expression*)

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

### Example:

```
Sub Main
    Dim A, B, Check
    A = 5: B = 5
    Check = CBool(A = B)
    Print Check
    A = 0
    Check = CBool(A)
    Print Check
End Sub
```

---

## CDate Function

CVDate (*expression*)

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

### Example:

```
Sub Main
    Dim MyDate, MDate, MTime, MStime
    MybDate = "May 29, 1959" ' Define date.
```

```
MDate = CDate(MybDate) ' Convert to Date data type.
MTime = "10:32:27 PM" ' Define time.
MSTime = CDate(MTime) ' Convert to Date data type.

Print MDate
Print MSTime

End Sub
```

---

## CDbl Function

CDbl (*expression*)

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

### Example:

```
Sub Main ()
  Dim y As Integer

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

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

---

## ChDir Statement

ChDir *pathname*

Changes the default directory

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

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

Related Topics: CurDir, CurDir\$, ChDrive, Dir, Dir\$, Mkdir, Rmdir

### Example:

```

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

```

---

## ChDrive Statement

ChDrive *drivename*

Changes the default drive

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

### Example:

```

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

```

Related Topics: ChDir, CurDir, CurDir\$, MkDir, Rmdir

---

## CheckBox

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

For selecting one or more in a series of choices

### Example:

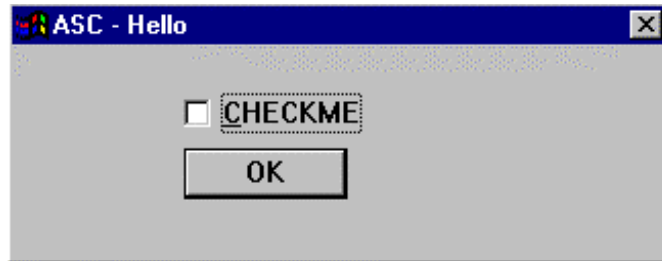
```

Sub Main ()
    Begin Dialog DialogName1 60, 70, 160, 50, "ASC - Hello"
        CHECKBOX 42, 10, 48, 12, "&CHECKME", .checkInt
    End Dialog

```

```
        OKBUTTON 42, 24, 40, 12
    End Dialog
    Dim Dlg1 As DialogName1
    Dialog Dlg1
    If Dlg1.checkInt = 0 Then
        Q = "didn't check the box."
    Else
        Q = "checked the box."
    End If
    MsgBox "You " & Q

End Sub
```



---

## Choose Function

`Choose(number, choice1, [choice2,] [choice3,]... )`

Returns a value from a list of arguments

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

### Example:

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

---

## Chr Function

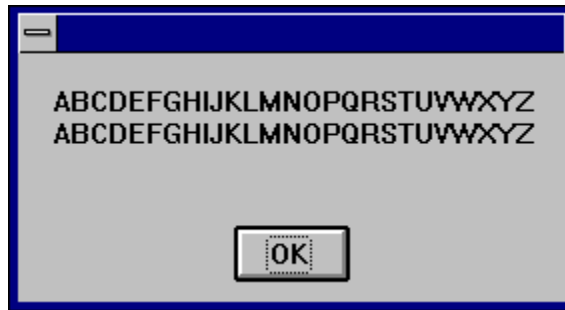
`Chr(int )`

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

Chr returns a String

### Example:

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



---

## CInt Function

*CInt (expression)*

Converts any valid expression to an integer.

### Example:

```
Sub Main ()
    Dim y As Long

    y = 25
    If VarType(y) = 2 Then
        Print y
        x = CInt(y) 'Converts the long value of y to an integer value in x
        Print x
    End If
End Sub
```

---

## CLng Function

*CLng (expression)*

Converts any valid expression into a long.

### Example:

```
Sub Main ()
    Dim y As Integer
```

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

End Sub
```

---

## Close Statement

Close [[#*filename*] [, [#]*filename*],,,

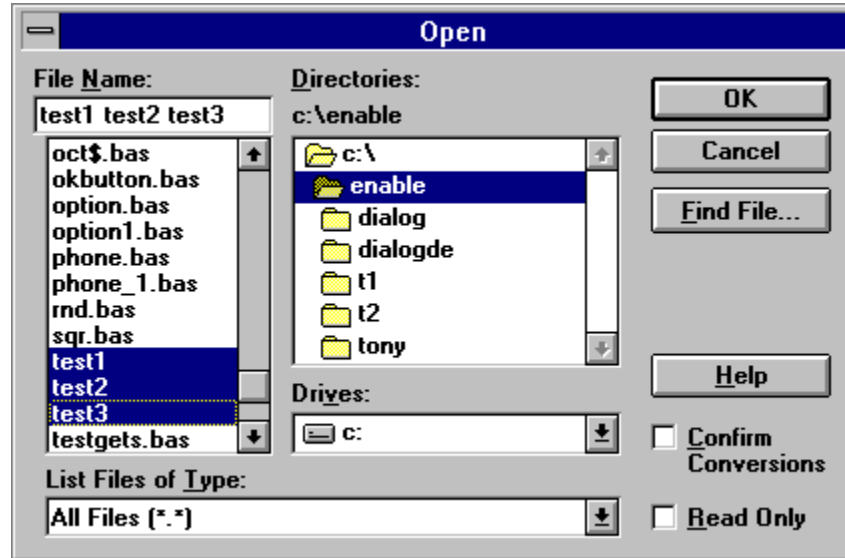
The Close Statement takes one argument *filename*. *Filename* is the number used with the Open Statement to open the file. If the Close Statement is used without any arguments it closes all open files.

### Example:

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

End Sub

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



## Const Statement

*Const name = expression*

Assigns a symbolic name to a constant value.

A constant must be defined before it is used.

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

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

### Example:

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



## Cos Function

Cos (*rad*)

Returns the cosine of an angle

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

### Example:

```
Sub Main()
  Dim J As Double
  Dim I As Single
  Dim K As Integer
  For I =1 To 10
    Msg = Msg & Cos(I) & ", "
    J=Cos(I)
    Print J
    K=Cos(I)
    Print K
  Next I
  MsgBox Msg
  MsgBox Msg1
End Sub
```

' Declare variables.  
' Cos function call  
' Display results.

---

## CreateObject Function

CreateObject (*class*)

Creates an OLE automation object.

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

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

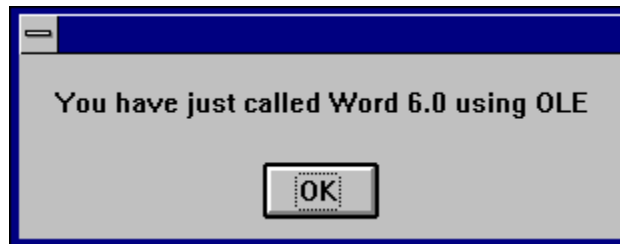
word6.InsertPara
MsgBox "You have just called Word 6.0 using OLE"
End Sub
```

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

Dear Vender:

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

Product Name: ScriptWriter Pro  
Company Name: Software Inc.



---

## CSng Function

CSng (*expression*)

Converts any valid expression to a Single.

**Example:**

```
Sub Main ()
  Dim y As Integer

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

---

## CStr Function

CStr(*expression*)

Converts any valid expression to a String.

### Example:

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

---

## CurDir Function

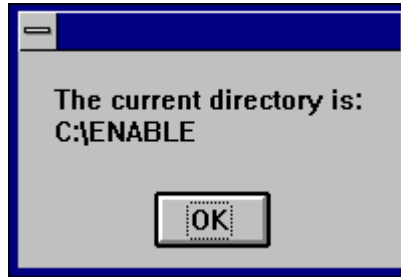
CurDir (*drive*)

Returns the current path for the specified drive

CurDir returns a Variant; CurDir\$ returns a String.

### Example:

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



---

## CVar Function

*CVar (expression)*

Converts any valid expression to a Variant.

### Example:

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

---

## Date Function

Date, Date()

Returns the current system date

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

### Example:

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

```

' current system short time and short date settings.

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

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

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

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

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

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

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

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

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

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

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

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

End Sub

```

---

## DateSerial Function

DateSerial (*year, month, day*)

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

Related Topics: DateValue. TimeSerial, TimeValue

### Example:

```
Sub Main
    Dim MDate
    MDate = DateSerial(1959, 5, 29)
    Print MDate
End Sub
```

---

## DateValue Function

DateValue(*dateexpression*)

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

Related Topics: DateSerial, TimeSerial, TimeValue

### Example:

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

---

## Day Function

Day(*dateexpression*)

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

Related Topics: Month, Weekday, Hour, Second

### Example:

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

## Declare Statement

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

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

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

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

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

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

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

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

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

```
Declare Sub Ontime Lib “Check” ()
```

**ScriptWriter Pro extensions to the declare statement. The following syntax is not supported by Microsoft Visual Basic.**

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

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

Related Topics: Call

### Example:

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

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

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

End Sub
```



---

## Dialog, Dialog Function

*Dialog(DialogRecord)*

Returns a value corresponding to the button the user chooses.

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

The return value or button:

-1 = OK button

0 = Cancel button

> 0 A command button where 1 is the first PushButton in the definition of the dialog and 2 is the second and so on.

### Example:



```

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

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

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

```

---

## Dim Statement

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

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

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

### Example:

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

---

## Dir Function

Dir[(*path,attributes*)]

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

### Example:

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

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

  Dim frame As BitmapDlg

  ' Show the bitmap dialog
  Dialog frame
End Sub

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

  DlgFunc = 1          ' Keep dialog active

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

```

        For i = 1 To count
            x(i) = dir
        Next i
        DlgListBoxArray "List1", x()
    Case 2 ' Click
        fileName = "c:\windows\" & DlgText("List1")
        DlgSetPicture "Picture1", fileName
    End Select
End Function

```

---

## DlgEnable Statement

DlgEnable "*ControlName*", *Value*

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

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

Related Topics: DlgVisible, DlgText

### Example:

```

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

Dim Dlg2 As UserDialog2
Dlg2.FText = "Your default string goes here"
Select Case Action%
Case 1
    DlgEnable "Group", 0
    DlgVisible "Chk2", 0
    DlgVisible "History", 0
Case 2
    If ControlID$ = "Chk1" Then
        DlgEnable "Group", On
        DlgVisible "Chk2"
        DlgVisible "History"
    End If

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

```

```
End If

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

If ControlID$ = "but1" Then

End If

Case Else

End Select
Enable =1

End Function
```

---

## DlgText Statement

DlgText "*ControlName*", *String*

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

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

Related Topics: DlgEnable, DlgVisible

### Example:

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

---

## DlgVisible Statement

DlgVisible "*ControlName*", *Value*

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

The parameter *ControlName* is the name of the control on the dialog box. The parameter *Value* is the value to set it to. 1 = Visible, 0 = Hidden. On is equal to 1. If the second parameter is omitted the status

of the control toggles. The entire example below can be found in the dialog section of this manual and in the example .bas files that ship with ScriptWriter Pro.

Related Topics: DlgEnable, DlgText

**Example:**

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

---

## Do...Loop Statement

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

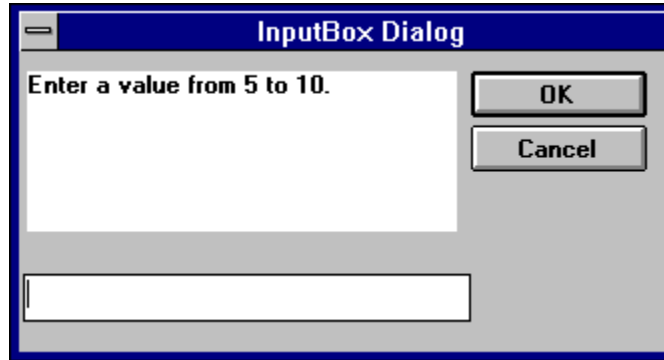
```
Do
    [statements]
    [Exit Do]
    [statements]
Loop [{While|Until} condition]
```

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

Related Topics: While, Wend

**Example:**

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



---

## End Statement

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

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

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

### Example:

```
Sub Main()  
  
    Dim Var1 as String  
  
    Var1 = "hello"  
    MsgBox " Calling Test"  
    Test Var1  
    MsgBox Var1  
  
End Sub  
  
Sub Test(wvar1 as string)  
  
    wvar1 = "goodbye"  
    MsgBox "Use of End Statement"  
End  
  
End Sub
```

---

## EOF Function

EOF(*Filenumber*)

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

Related Topics: Open Statement

**Example:**

```
' Input Function Example

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

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

---

## Erase Statement

Erase *arrayname*[,*arrayname* ]

Reinitializes the elements of a fixed array.

Related Topics: Dim

**Example:**

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

Option Base 1

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

End Sub
```

## Exit Statement

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

Exits a loop or procedure

Related Topics End Statement, Stop Statement

### Example:

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

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

---

## Exp

Exp(*num*)

Returns the base of the natural log raised to a power ( $e^{\text{num}}$ ).

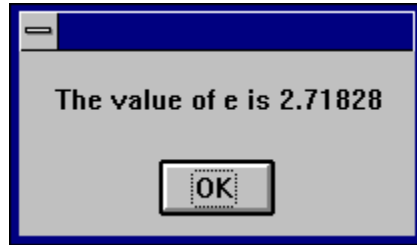
The value of the constant  $e$  is approximately 2.71828.

Related Topics: Log

### Example:

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





---

## FileCopy Function

FileCopy( *sourcefile*, *destinationfile* )

Copies a file from source to destination.

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

### Example:

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

---

## FileLen Function

FileLen( *filename* )

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

Related Topics: LOF Function

### Example:

Sub Main

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

End Sub

---

## Fix Function

Fix(*number* )

Returns the integer portion of a number

Related Topics: Int

**Example:**

```
Sub Main
    Dim MySize
    MySize = Fix(4.345)
    Print MySize
End Sub
```

---

## For each ... Next Statement

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

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

**Example:**

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

---

## For...Next Statement

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

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

**Example:**

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



---

## Format Function

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

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

Format returns returns a string

<b>Part</b>	<b>Description</b>
<i>expression</i>	Expression to be formatted.
<i>fmt</i>	A string of characters that specify how the expression is to displayed, or the name of a commonly-used format that has been predefined in ScriptWriter Pro Basic. Do not mix different type format expressions in a single <i>fmt</i> parameter.

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

To format numbers, you can use the commonly-used formats that have been predefined in ScriptWriter Pro Basic, or you can create user-defined

formats with standard characters that have special meaning when used in a format expression.

Predefined numeric format names:

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

---

<b>Format Name</b>	<b>Description</b>
--------------------	--------------------

Scientific	Use standard scientific notation.
------------	-----------------------------------

True/False	Display False if number is 0, otherwise display True.
------------	---

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

<b>Character</b>	<b>Meaning</b>
------------------	----------------

Null string	Display the number with no formatting.
-------------	--

0	Digit placeholder.
---	--------------------

Display a digit or a zero

If the number being formatted has fewer digits than there are zeros (on either side of the decimal) in the format expression, leading or trailing zeros are displayed. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, the number is rounded to as many decimal places

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

#                   Digit placeholder.

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

.                    Decimal placeholder.

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

---

<b>Character</b>	<b>Meaning</b>
------------------	----------------

---

%                   Percentage placeholder.

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

,                    Thousand separator.

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

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

E-E+e-e+        Scientific format.

If the format expression contains at least one digit placeholder (0 or #) to the right of E-,E+,e- or e+, the number is displayed in scientific formatted E or e inserted between the number and its exponent. The number of digit placeholders to the right determines the number of digits in the exponent. Use E- or e- to place a minus sign next to negative exponents. Use E+ or e+ to place a plus sign next to positive exponents.

:                   Time separator.

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

/ Date separator.

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

<b>Character</b>	<b>Meaning</b>
- + \$ ( ) space	Display a literal character.  To display a character other than one of those listed, precede it with a backslash (\).
\	Display the next character in the format string.  The backslash itself isn't displayed. To display a backslash, use two backslashes (\\).  Examples of characters that can't be displayed as literal characters are the date- and time- formatting characters (a,c,d,h,m,n,p,q,s,t,w,y, and /:), the numeric -formatting characters(#,0,%E,e,comma, and period), and the string-formatting characters (@,&,<,>, and !).
“String”	Display the string inside the double quotation marks.  To include a string in <i>fmt</i> from within ScriptWriter Pro, you must use the ANSI code for a double quotation mark Chr(34) to enclose the text.
*	Display the next character as the fill character.  Any empty space in a field is filled with the character following the asterisk.  Unless the <i>fmt</i> argument contains one of the predefined formats, a format expression for numbers can have from one to four sections separated by semicolons.
<b>If you use</b>	<b>The result is</b>
One section	The format expression applies to all values.

only	
Two	The first section applies to positive values, the second to negative sections values.
Three	The first section applies to positive values, the second to negative sections values, and the third to zeros.
Four	The first section applies to positive values, the second to negative section values, the third to zeros, and the fourth to <b>Null</b> values.

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

```
“$#,##0; ($#,##0)”
```

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

```
“$#,##0; \Z\e\r\o”
```

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

<b>Format (<i>fmt</i>)</b>	<b>Positive 3</b>	<b>Negative 3</b>	<b>Decimal .3</b>	<b>Null</b>
Null string	3	-3	0.3	
0	3	-3	1	
0.00	3.00	-3.00	0.30	
#,##0	3	-3	1	
#,##0.00;;;Nil	3.00	-3.00	0.30	Nil
\$#,##0;(\$#,##0)	\$3	(\$3)	\$1	
\$#,##0.00;(\$#,##0.00)\$3.00	\$3.00	(\$3.00)	\$0.30	

0%	300%	-300%	30%
0.00%	300.00%	-300.00%	30.00%
0.00E+00	3.00E+00	-3.00E+00	3.00E-01
0.00E-00	3.00E00	-3.00E00	3.00E-01

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

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

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

<b>Format Name</b>	<b>Description</b>
General	Display a date and/or time. for real numbers, display a date and time.(e.g. 4/3/93 03:34 PM); If there is no fractional part, display only a date (e.g. 4/3/93); if there is no integer part, display time only (e.g. 03:34 PM).
Long Date	Display a Long Date, as defined in the International section of the Control Panel.
Medium	Display a date in the same form as the Short Date, as defined in the international section of the Control Panel, except spell out the month abbreviation.
Short Date	Display a Short Date, as defined in the International section of the Control Panel.
Long Time	Display a Long Time, as defined in the International section of the Control panel. Long Time includes hours, minutes, seconds.
Medium Time	Display time in 12-hour format using hours and minuets and the Time AM/PM designator.



Short Time    Display a time using the 24-hour format (e.g. 17:45)

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

<b>Character</b>	<b>Meaning</b>
c	Display the date as dddd and display the time as tttt. in the order.
d	Display the day as a number without a leading zero (1-31).
dd	Display the day as a number with a leading zero (01-31).
ddd	Display the day as an abbreviation (Sun-Sat).
dddd	Display a date serial number as a complete date (including day , month, and year).
<b>Character</b>	<b>Meaning</b>
w	Display the day of the week as a number (1- 7 ).
ww	Display the week of the year as a number (1-53).
m	Display the month as a number without a leading zero (1-12). If m immediately follows h or hh, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01-12). If mm immediately follows h or hh, the minute rather than the month is displayed.
mmm	Display the month as an abbreviation (Jan-Dec).
mmmm	Display the month as a full month name (January-December).
q	display the quarter of the year as a number (1-4).
y	Display the day of the year as a number (1-366).
yy	Display the day of the year as a two-digit number (00-99)

yyyy	Display the day of the year as a four-digit number (100-9999).
h	Display the hour as a number without leading zeros (0-23).
hh	Display the hour as a number with leading zeros (00-23).
n	Display the minute as a number without leading zeros (0-59).
nn	Display the minute as a number with leading zeros (00-59).
s	Display the second as a number without leading zeros (0-59).
ss	Display the second as a number with leading zeros (00-59).
tttt	Display a time serial number as a complete time (including hour, minute, and second) formatted using the time separator defined by the Time Format in the International section of the Control Panel. A leading zero is displayed if the Leading Zero option is selected and the time is before 10:00 A.M. or P.M. The default time format is h:mm:ss.
AM/PM	Use the 12-hour clock and display an uppercase AM/PM
am/pm	Use the 12-hour clock display a lowercase am/pm

---

<b>Character</b>	<b>Meaning</b>
------------------	----------------

---

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

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

---

<b>Format</b>	<b>Display</b>
---------------	----------------

---

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

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

---

<b>If you use</b>	<b>The result is</b>
-------------------	----------------------

---

One section only	The format applies to all string data.
Two sections <b>Null</b>	The first section applies to string data, the second to values and zero-length strings.

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

**@** Character placeholder.

---

<b>Character</b>	<b>Meaning</b>
------------------	----------------

---

<b>@</b>	Character placeholder.  Displays a character or a space. Placeholders are filled from right to left unless there is an ! character in the format string.
<b>&amp;</b>	Character placeholder. Display a character or nothing.
<b>&lt;</b>	Force lowercase.

- > Force uppercase.
- ! Force placeholders to fill from left to right instead of right to left.

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

### Example:

```
' Format Function Example

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

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

Sub Main

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

MsgBox Now
MsgBox MyTime

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

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

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

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

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

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

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

End Sub

---

## FreeFile Function

FreeFile

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

Related Topics: Open, Close, Write

### Example:

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

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

End Sub
```

---

## Function Statement

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

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

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

```
([ByVal] variable [As type] [,ByVal] variable [As type] [...])
```

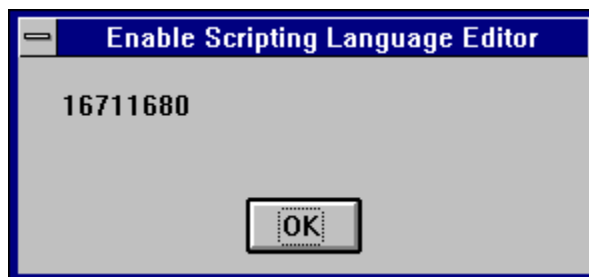
The optional ByVal parameter specifies that the variable is [passed by value instead of by reference (see “ByRef and ByVal” in this manual)].

The optional As type parameter is used to specify the data type. Valid types are String, Integer, Double, Long, and Variant (see “Variable Types” in this manual).

Related Topics: Dim, End, Exit, Sub

### Example:

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



---

## Get Object Function

`GetObject(filename[,class])`

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

Related Topics: CreateObject

## Global Statement

Global Const *constant*

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

Related Topics: Dim, Const and Type Statements

### Example:

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

---

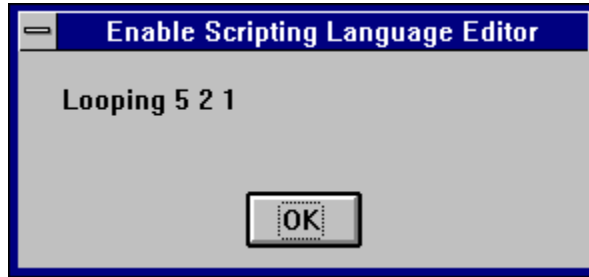
## GoTo Statement

GoTo *label*

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

### Example:

```
Sub main ()
    Dim x,y,z
    For x = 1 to 5
        For y = 1 to 5
            For z = 1 to 5
                Print "Looping" ,z,y,x
                If y > 3 Then
                    GoTo Label1
                End If
            Next z
        Next y
    Next x
Label1:
End Sub
```



---

## Hex

Hex (*num*)

Returns the hexadecimal value of a decimal parameter.

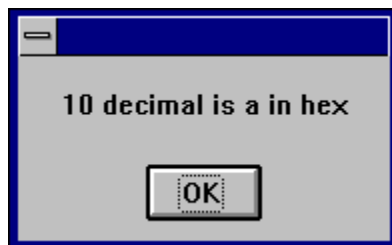
Hex returns a string

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

Related Topics: Oct, Oct\$

### Example:

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



---

## Hour Function

Hour(*string*)

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



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

### Example:

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

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

Sub Main

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

MsgBox Now
MsgBox MyTime

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

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

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

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

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

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

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

End Sub
```

---

## If...Then...Else Statement

If condition Then

.

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

## Syntax 2

If conditional Then statement

Allows conditional statements to be executed in the code.

Related Topics: Select Case

### Example:

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

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

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

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

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

## Input Function

`Input(n , [ #]filename )`

Input returns characters from a sequential file.

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

### Example:

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

---

## InputBox Function

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

InputBox returns a String.

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

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

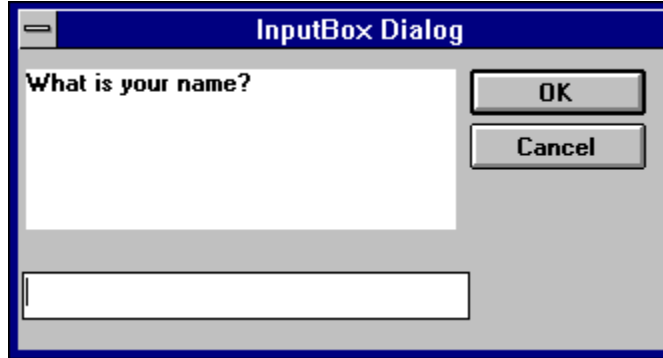
Default is a string that is displayed in the text box as the default entry.

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

### Example:

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

End Sub



---

## InStr

`InStr(numbegin, string1, string2)`

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

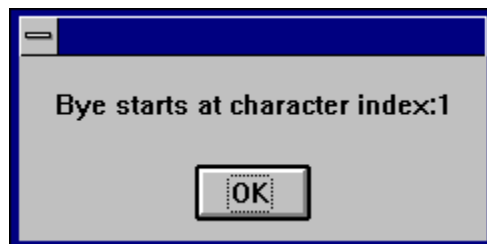
The *numbegin* parameter is not optional and sets the starting point of the search. *numbegin* must be a valid positive integer no greater than 65,535.

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

Related Topics: Mid Function

### Example:

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



---

## Int Function

`Int(number)`

Returns the integer portion of a number

Related Topics: Fix

---

## IsArray Function

`IsArray(variablename )`

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

Related Topics: IsEmpty, IsNumeric, VarType, IsObject

### Example:

```
Sub Main
    Dim MArray(1 To 5) As Integer, MCheck
    MCheck = IsArray(MArray)
    Print MCheck
End Sub
```

---

## IsDate

`IsDate(variant )`

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

Related Topics: IsEmpty, IsNumeric, VarType

### Example:

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

---

## IsEmpty

IsEmpty(*variant* )

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

Related Topics: IsDate, IsNull, IsNumeric, VarType

### Example:

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

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

---

## IsNull

IsNull(*v*)

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

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

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

Related Topics: IsDate, IsEmpty, IsNumeric, VarType

---

## IsNumeric

IsNumeric(*v*)

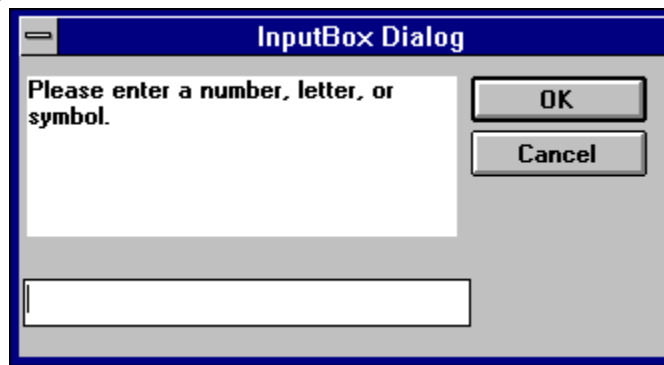
Returns a TRUE or FALSE indicating if the *v* parameter can be converted to a numeric data type.

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

Related topics: IsDate, IsEmpty, IsNull, VarType

### Example:

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



---

## IsObject Function

IsObject(*objectname* )

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

Related Topics: IsEmpty, IsNumeric, VarType, IsObject

### Example:

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

    Set YourObject = MyObject
    MyCheck = IsObject(YourObject)

    Print MyCheck
End Sub
```

---

## Kill Statement

Kill *filename*

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

Related Topics: Rmdir

### Example:

```
Const NumberOfFiles = 3

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

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

---

## LBound Function

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

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

Related Topics: UBound Function

### Example:

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

```
Option Base 1

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



```
Print a(0),a(1),a(2), a(3)
End Sub
```

---

## LCase, Function

LCase[\$](*string*)

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

Related Topics: UCase Function

### Example:

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

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

---

## Left

Left(*string*, *num*)

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

Left returns a Variant, Left\$ returns a String

### Example:

```
Sub Main ()
  Dim LWord, Msg, RWord, SpcPos, UsrInp ' Declare variables.
  Msg = "Enter two words separated by a space."
  UsrInp = InputBox(Msg) ' Get user input.
  print UsrInp
  SpcPos = InStr(1, UsrInp, " ") ' Find space.
  If SpcPos Then
```

```
LWord = Left(UsrInp, SpcPos - 1)      ' Get left word.
print "LWord: "; LWord
RWord = Right(UsrInp, Len(UsrInp) - SpcPos) ' Get right word.
Msg = "The first word you entered is " & LWord
Msg = Msg & "." & " The second word is "
Msg = "The first word you entered is <" & LWord & ">"
Msg = Msg & RWord & "."
Else
    Msg = "You didn't enter two words."
End If
MsgBox Msg      ' Display message.
MidTest = Mid("Mid Word Test", 4, 5)
Print MidTest
End Sub
```

---

## Len

Len(*string*)

Returns the number of characters in a string.

Related Topics: InStr

### Example:

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



---

## Let Statement

[Let] *variablename* = *expression*

Let assigns a value to a variable.

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

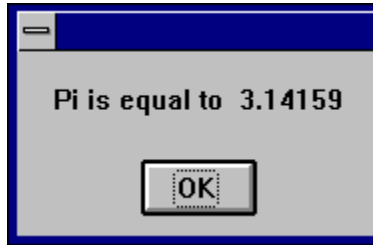
### Example:

```
Sub Form_Click ()
    Dim Msg, Pi      ' Declare variables.
    Let Pi = 4 * Atn(1) ' Calculate Pi.
```

```

Msg = "Pi is equal to " & Str(Pi)
MsgBox Msg           ' Display results.
End Sub

```




---

## Line Input # Statement

Line Input # *filenumber* and *name*

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

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

Related Topics: Open

### Example:

```

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

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

```

---

## LOF

LOF(*filenumber*)

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

Related Topics: FileLen

### Example:

```

Sub Main

```

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

```
End Sub
```

---

## Log

`Log(num)`

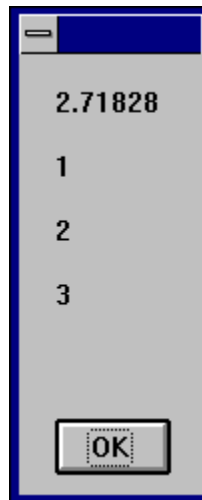
Returns the natural log of a number

The parameter *num* must be greater than zero and be a valid number.

Related Topics: Exp, Sin, Cos

### Example:

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



---

## Mid Function

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

Returns a substring within a string.

**Example:**

```

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

```

---

## Minute Function

*Minute(string)*

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

```

' Format Function Example

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

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

Sub Main

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

MsgBox Now
MsgBox MyTime

```

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

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

End Sub
```

---

## MkDir

MkDir *path*

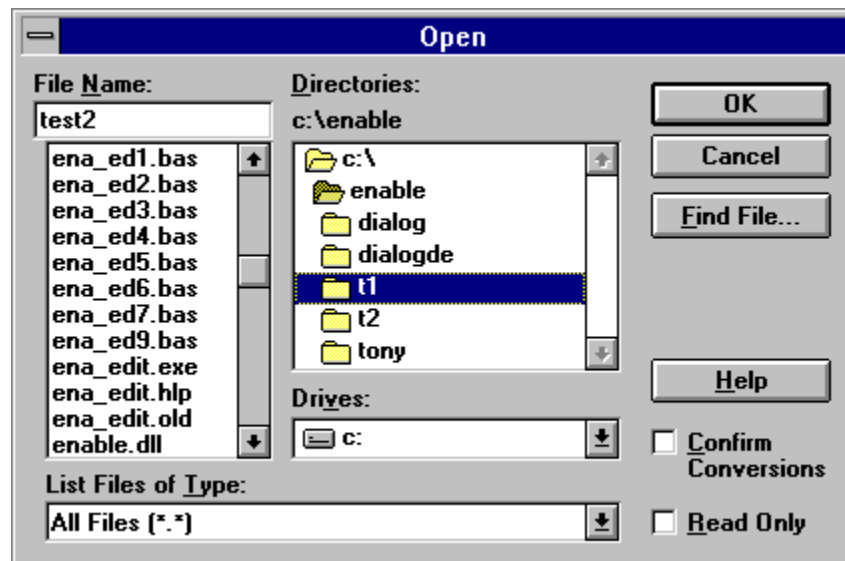
Creates a new directory.

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

### Example:

```
Sub Main
  Dim DST As String

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



---

## Month Function

Month(*number*)

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

Related Topics: Day, Hour, Weekday, Year

### Example:

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

---

## MsgBox Function MsgBox Statement

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

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

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

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

Value	Meaning
0	Display OK button only.
1	Display OK and Cancel buttons.
2	Display Abort, Retry, and Ignore buttons.
3	Display Yes, No, and Cancel buttons.
4	Display Yes and No buttons.
5	Display Retry and Cancel buttons.
16	Stop Icon
32	Question Icon

48	Exclamation Icon
64	Information Icon
0	First button is default.
256	Second button is default.
512	Third button is default.
768	Fourth button is default.
0	Application modal.
4096	System modal

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

title:

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

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

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

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

### MsgBox Function, MsgBox Statement Example

The example uses MsgBox to display a close without saving message in a dialog box with a Yes button a No button and a Cancel button. The Cancel



button is the default response. The MsgBox function returns a value based on the button chosen by the user. The MsgBox statement uses that value to display a message that indicates which button was chosen.

### Related Topics: InputBox, InputBox\$ Function

#### **Example:**

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

---

## **Name Statement**

Name *oldname* As *newname*

Changes the name of a directory or a file.

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

Related Topics: Kill, ChDir

---

## **Now Function**

Now

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

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

Related Topics:

**Example:**

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

---

## Oct Function

Oct (*num*)

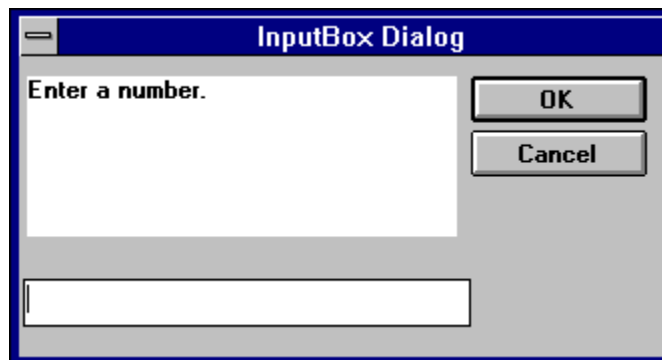
Returns the octal value of the decimal parameter

Oct returns a string

Related Topics: Hex

**Example:**

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



---

## OKButton

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

For selecting options and closing dialog boxes

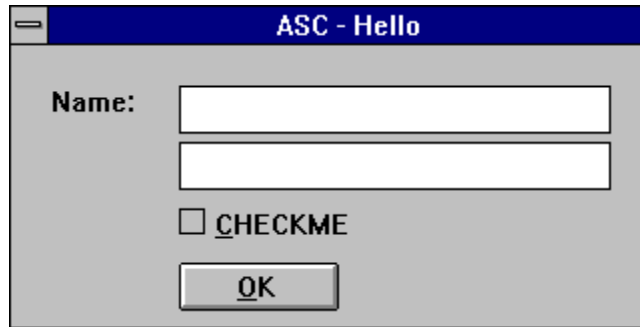
```
Sub Main ()
    Begin Dialog DialogName1 60, 60, 160, 70, "ASC - Hello"
        TEXT 10, 10, 28, 12, "Name:"
```

```

        TEXTBOX 42, 10, 108, 12, .nameStr
        TEXTBOX 42, 24, 108, 12, .descStr
        CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
        OKBUTTON 42, 54, 40, 12
    End Dialog
    Dim Dlg1 As DialogName1
    Dialog Dlg1

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

```




---

## On Error

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

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

Related Topics: Resume

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

### Example:

```

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

```

Errors can be raised with the syntax:

Err.Raise x

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

3: "Return without GoSub";  
5: "Invalid procedure call";  
6: "Overflow";  
7: "Out of memory";  
9: "Subscript out of range";  
10: "Array is fixed or temporarily locked";  
11: "Division by zero";  
13: "Type mismatch";  
14: "Out of string space";  
16: "Expression too complex";  
17: "Can't perform requested operation";  
18: "User interrupt occurred";  
20: "Resume without error";  
28: "Out of stack space";  
35: "Sub, Function, or Property not defined";  
47: "Too many DLL application clients";  
48: "Error in loading DLL";  
49: "Bad DLL calling convention";  
51: "Internal error";  
52: "Bad file name or number";  
53: "File not found";  
54: "Bad file mode";  
55: "File already open";  
57: "Device I/O error";  
58: "File already exists";  
59: "Bad record length";  
60: "Disk full";  
62: "Input past end of file";  
63: "Bad record number";  
67: "Too many files";  
68: "Device unavailable";  
70: "Permission denied";  
71: "Disk not ready";  
74: "Can't rename with different drive";  
75: "Path/File access error";  
76: "Path not found";  
91: "Object variable or With block variable not set";  
92: "For loop not initialized";  
93: "Invalid pattern string";

```

94: "Invalid use of Null";
// OLE Automation Messages
429: "OLE Automation server cannot create object";
430: "Class doesn't support OLE Automation";
432: "File name or class name not found during OLE Automation operation";
438: "Object doesn't support this property or method";
440: "OLE Automation error";
443: "OLE Automation object does not have a default value";
445: "Object doesn't support this action";
446: "Object doesn't support named arguments";
447: "Object doesn't support current local setting";
448: "Named argument not found";
449: "Argument not optional";
450: "Wrong number of arguments";
451: "Object not a collection";
// Miscellaneous Messages
444: "Method not applicable in this context";
452: "Invalid ordinal";
453: "Specified DLL function not found";
457: "Duplicate Key";
460: "Invalid Clipboard format";
461: "Specified format doesn't match format of data";
480: "Can't create AutoRedraw image";
481: "Invalid picture";
482: "Printer error";
483: "Printer driver does not supported specified property";
484: "Problem getting printer information from from the system.";
    // Make sure the printer is setp up correctly.
485: "invalid picture type";
520: "Can't empty Clipboard";
521: "Can't open Clipboard";

```

---

## Open Statement

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

Opens a file for input and output operations.

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

Part	Description

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

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

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

<b>Mode</b>	<b>Description</b>
<i>Input</i>	Sequential input mode.
<i>Output</i>	Sequential output mode.

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

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

<b>Access type</b>	<b>Description</b>
<i>Read</i>	Opens the file for reading only.
<i>Write</i>	Opens the file for writing only.

<i>Read Write</i>	Opens the file for both reading and riting. This mode is valid only for Random and Binary files and files opened for Append mode.
-------------------	---

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

### Example :

```
Sub Main ()

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

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

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

---

## Option Base Statement

Option Base *number*

Declares the default lower bound for array subscripts.

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

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

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

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

Related Topics: Dim, Global and Lbound Statements

### Example :

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

---

## Option Explicit Statement

Option Explicit

Forces explicit declaration of all variables.

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

Related Topics: Const and Global Statements

### Example :

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

---

## Print Method

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

Related Topics:

### Example:

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



```

Print Pi           'Prints the results in the
                  ' compiler messages window
End Sub

```




---

## Print # Statement

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

Writes data to a sequential file.

Print statement Description:

*filename*:

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

Spc(*n*):

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

Tab(*n*):

Name of the Basic function optionally used to tab to the *n*th column before printing *expressionlist*. Multiple use is permitted.

*expressionlist* :

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

{;|,}

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

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

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

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

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

If the data being written is a Variant of VarType 7 (Date), Print # writes the date to the file using the Short Date format defined in the WIN.INI file. When either the date or the time component is missing or zero, Print # writes only the part provided to the file.

The following example writes data to a test file.

### Example :

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

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

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

---

## ReDim Statement

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

Used to declare dynamic arrays and reallocate storage space.

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

### Example:

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

---

## Rem Statement

Rem *remark* 'remark

Used to include explanatory remarks in a program.

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

### Example:

```
Rem This is a remark
```

```
Sub Main()
Dim Answer, Msg                                ' Declare variables.
Do
    Answer = InputBox("Enter a value from 1 to 3.")
    Answer = 2
    If Answer >= 1 And Answer <= 3 Then        ' Check range.
        Exit Do                                ' Exit Do...Loop.
    Else
        Beep                                    ' Beep if not in range.
    End If
Loop
```

```
MsgBox "You entered a value in the proper range."  
End Sub
```

---

## Right Function

Right (*stringexpression*, *n* )

Returns the right most *n* characters of the string parameter.

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

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

Related Topics: Len, Left, Mid Functions.

### Example:

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

---

## Rmdir Statement

Rmdir *path*

Removes an existing directory.

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

Related Topics: ChDir, CurDir

**Example:**

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

Sub Main
  Dim dirName As String

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

---

## Rnd Function

Rnd (*number*)

Returns a random number.

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

**Example:**

'Rnd Function Example

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

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

---

## Second Function

Second (*number*)

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

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

Related Topics: Day, Hour, Minute, Now.

### Example:

```
' Format Function Example

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

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

Sub Main

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

MsgBox Now
MsgBox MyTime

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

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

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

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

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

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

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

End Sub
```

---

## Seek Function

Seek (*filename*)

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

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

Related Topics: Open

### Example:

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

---

## Seek Statement

Seek *filenumber*, *position*

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

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

Related Topics: Open

### Example:

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

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

---

## Select Case Statement

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

```
Select Case testvar
  Case var1
    Statement Block
```

```

        Case var2
            Statement Block
        Case Else
            Statement Block
    End Select

```

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

**Select Case** Number ' Evaluate Number.

**Case 1 To 5** ' Number between 1 and 5, inclusive.

...

' The following is the only Case clause that evaluates to True.

**Case 6, 7, 8** ' Number between 6 and 8.

...

**Case 9 To 10** ' Number is 9 or 10.

...

**Case Else** ' Other values.

...

**End Select**

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

### Example:

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

```

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

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

```

---

## SendKeys Function



### SendKeys (*Keys*, [*wait*])

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

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

#### Example:

```

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

```

---

## Set Statement

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

Assigns an object to an object variable.

Related Topics: Dim, Global, Static

#### Example:

```

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

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

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

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

    visio.Quit

```

End Sub




---

## Shell Function

Shell ( *app* [, *style*] )

Runs an executable program.

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

Window styles:

Normal with focus 1,5,9

Minimized with focus (default) 2

Maximized with focus 3

normal without focus 4,8

minimized without focus 6,7

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

### Example:

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

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

## Sin Function

Sin (*rad*)

Returns the sine of an angle that is expressed in radians

### Example:

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

---

## Space Function

Space[\$] (*number*)

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

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

### Example:

```
' This sample shows the space function
Sub Main
    MsgBox "Hello" & Space(20) & "There"
End Sub
```

---

## Sqr Function

Sqr(*num*)

Returns the square root of a number.

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

### Example:

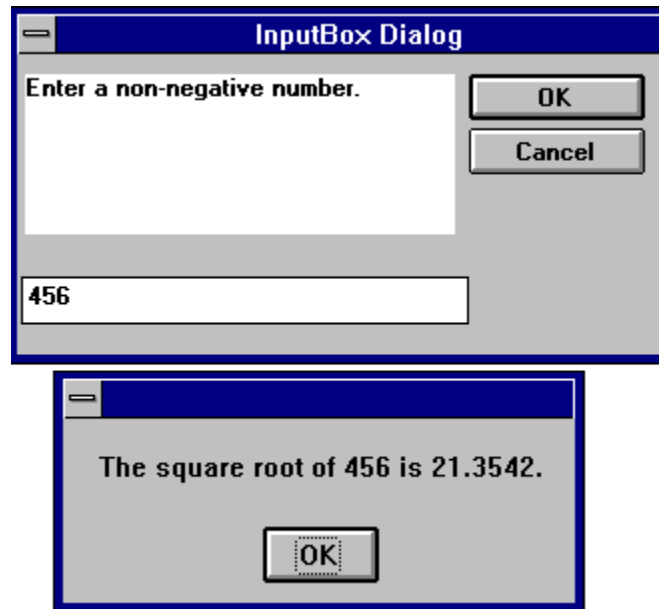
```
Sub Form_Click ()
    Dim Msg, Number ' Declare variables.
    Msg = "Enter a non-negative number."
    Number = InputBox(Msg) ' Get user input.
    If Number < 0 Then
```

```

Msg = "Cannot determine the square root of a negative number."
Else
Msg = "The square root of " & Number & " is "
Msg = Msg & Sqr(Number) & "."
End If
MsgBox Msg      ' Display results.

End Sub

```




---

## Static Statement

### *Static variable*

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

Related Topics: Dim, Function, Sub

### **Example:**

```

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

```

```

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

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

```

---

## Stop Statement

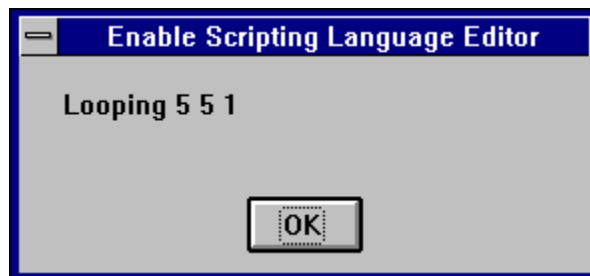
Stop

Ends execution of the program

The Stop statement can be placed anywhere in your code.

### Example:

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



---

## Str Function

*Str(numericexpr)*

Returns the value of a numeric expression.

Str returns a String.

Related topics: Format, Val

### Example:

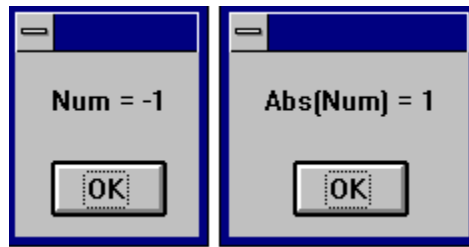
```
Sub main ()
    Dim msg
    a = -1
```

```

msgBox "Num = " & Str(a)
MsgBox "Abs(Num) =" & Str(Abs(a))

```

```
End Sub
```




---

## StrComp Function

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

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

### Example:

```

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

```

---

## String Function

String ( *numeric*, *charcode* )

String returns a string.

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

Related topics: Space Function

### Example:

```

Sub Main
Dim MString
MString = String(5, "*") ' Returns "*****".
MString = String(5, 42) ' Returns "44444".

```

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

---

## Sub Statement

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

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

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

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

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

Related Topics: Call, Dim, Function

### Example:

```
Sub Main  
    Dim DST As String  
  
    DST = "t1"  
    mkdir DST  
    mkdir "t2"  
End Sub
```

---

## Tan Function

Tan(*angle*)

Returns the tangent of an angle as a double.

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

Related Topic: Atn, Cos, Sin

**Example:**

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

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

---

**Text Statement**

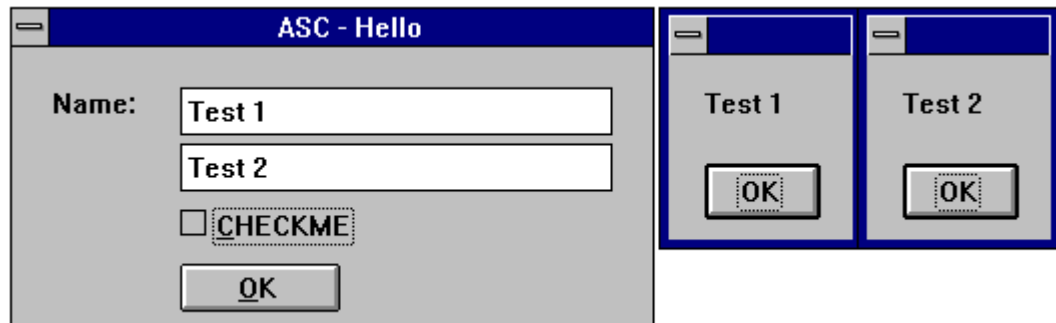
Text Starting X position, Starting Y position, Width, Height, Label

Creates a text field for titles and labels.

**Example:**

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

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



---

**TextBox Statement**

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

Creates a Text Box for typing in numbers and text



**Example:**

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

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

---

## Time Function

Time[()]

Returns the current system time.

Related topics: To set the time use the TIMES\$ statement.

**Example:**

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

---

## Timer Event

Timer

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

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

**Example:**

```
Sub Main

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

```
TS = Timer
MsgBox "Starting Timer"
TE = Timer
TT = TE - TS
Print TT

End Sub
```

---

## TimeSerial - Function

TimeSerial ( *hour, minute, second* )

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

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

### Example:

```
Sub Main
    Dim MTime
    MTime = TimeSerial(12, 25, 27)
    Print MTime
End Sub
```

---

## TimeValue - Function

TimeValue ( *TimeString* )

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

Midnight = TimeValue("23:59:59")

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

### Example:

```
Sub Main
```

```

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

End Sub

```

---

## Trim, LTrim, RTrim Functions

[L| R] Trim (*String* )

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

Ltrim, Rtrim and Trim all return a string

Ltrim removes leading spaces.

Rtrim removes trailing spaces.

Trim removes leading and trailing spaces.

### Example:

```

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

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

```

---

## Type Statement

```

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

```

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

The **Type** statement has these parts:

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

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

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

### Example:

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

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

Type type2
  a As String
  o As type1
End Type

Type type3
  b As Integer
  c As type2
End Type

Dim type2a As type2
Dim type2b As type2
```

```
Dim type1a As type1
Dim type3a as type3

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

---

## UBound Function

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

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

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

### Example:

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

```
Option Base 1

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

---

## UCase Function

Ucase (*String* )

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

Related Topics: Lcase, Lcase\$ Function

### Example:

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

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

---

## Val

Val(*string*)

Returns the numeric value of a string of characters.

### Example:

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

---

## VarType

VarType(*varname*)

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

The parameter *varname* is a variant data type.

VarType	return values:
Empty	0
Null	1
Integer	2
Long	3

Single	4
Double	5
Currency	6 (not available at this time)
Date/Time	7
String	8

Related Topics: IsNull, IsNumeric

**Example:**

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

---

## Weekday Function

*Weekday(date,firstdayof week)*

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

Related Topics: Hour, Second, Minute, Day

**Example:**

```
Sub Main  
  
x = Weekday(#5/29/1959#)  
Print x  
  
End Sub
```

---

## While...Wend Statement

While condition

```
.  
. .  
[StatementBlock]
```

```
.  
. .  
Wend
```

While begins the while...Wend flow of control structure. Condition is any numeric or expression that evaluates to true or false. If the condition is true

the statements are executed. The statements can be any number of valid ScriptWriter Pro Basic statements. Wend ends the While...Wend flow of control structure.

Related Topics: Do...Loop Statement

### Example:

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

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

---

## With Statement

```
With object
    [STATEMENTS]
End With
```

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

Related Topics: While Statement and Do Loop

### Example:

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

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

Type type2
    a As String
    o As type1
End Type

Dim typela As type1
```



```

Dim type2a As type2

Sub Main ()

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

End Sub

```

---

## Write # - Statement

Write #*filename* [*parameterlist* ]

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

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

Related Topics: Open and Print# Statements

### Example:

```

Sub Main ()

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

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

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

End Sub

```

## Year Function

`Year(serial# )`

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

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

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

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

### Example:

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

# Developer Toolbox SCSI Extension Functions

---

## MultiThreaded SCSI Commands

### IssueThreadedCDB

(ByVal ha As Long, ByVal target As Long, ByVal lun As Long, ByRef CDBBytes() As Byte, ByVal nCDBLen As Long, ByRef InOutBuffer() As Byte, ByVal InOutBufferLen As Long, ByVal Direction As Long, ByVal Timeout As Long) As Integer

Issues a User-defined MultiThreaded CDB to the specified ha/target/lun address.

The CDB is defined as follows:

- The CDB is defined in the array CDBBytes()
- The CDB size (6,10,12, or 16 bytes) is defined in nCDBLen
- The data buffer used by the cdb is defined by the byte array InOutBuffer()
- The data buffer length is defined by the long InOutBufferLen
- The data direction (1 = in from target, 0 = out to target) is defined by Direction
- The CDB timeout (in seconds) is defined by Timeout

Returns: Zero on failure, or a ThreadID on success

Example:

See the MultiThreaded CDB example at the end of this section

## GetThreadedCDBStatus

(ByVal nThrdID As Long) As Integer

Checks the status of the CDB associated with the Thread ID *nThrdID*

Returns:

one of the following:

0 = eTestInProgress

1 = eCompleteOnSuccess

2 = eCompleteOnFailure

3 = eTestNotStartedYet

4 = eTestIsPaused

5 = eTestStopped

6 = eErrorOnParamsPassed

7 = eMisCompare

8 = eInvalidBlock

9 = ePendingIOOutstanding

10 = eUnknownStatus

## GetThreadedCDBStatusWData

(ByVal nThrdID As Long, ByRef SenseBytes() As Byte, ByVal nSenseBufLen As Long, ByRef TimeToDoCmd As Double, ByRef InOutBuffer() As Byte, ByVal InOutBufferLen As Long) As Integer

Use this function to check the thread status and to retrieve performance metrics and data if your CDB had a data in phase.

The array SenseBytes() – array length specified by nSenseBufLen – will return sense data if there is any from a CDB that caused a check condition.

TimeToDoCmd will return the time that it took the CDB to execute

InOutBuffer() will contain InOutBufferLen bytes of data if your CDB had a data-in phase

Returns:

one of the following on failure:

0 = eTestInProgress

1 = eCompleteOnSuccess

2 = eCompleteOnFailure

3 = eTestNotStartedYet

4 = eTestIsPaused

5 = eTestStopped

6 = eErrorOnParamsPassed

7 = eMisCompare

8 = eInvalidBlock

9 = ePendingIOOutstanding

10 = eUnknownStatus

Example: See the MultiThreaded CDB example at the end of this section

## ReleaseThreadID

(ByVal nThrdID As Long) As Boolean

It is very important to call this function, passing it the ThreadID, when your command is complete.

Returns:

1 on success, non-1 on failure

Example: see example below

### *MultiThreaded CDB example*

```
Dim ha, tid, lun As Integer
Dim cdbInquiry(6) As Byte
Dim ThreadID As Integer
Dim ThreadStatus As Integer
Dim DataBuffer(256) As Byte
Dim SenseBuffer(16) As Byte
Dim eTestInProgress As Integer
Dim eCompleteOnSuccess As Integer
Dim eCompleteOnFailure As Integer
Dim eTestNotStartedYet As Integer
Dim eErrorOnParamsPassed As Integer
Dim eOutstandingIO As Integer
Dim eUnknownStatus As Integer

Dim retval As Integer
Dim dTimeToDoCmd As Double

Dim results As String
Dim i As Integer

cdbInquiry(0) = &h12
cdbInquiry(1) = 0
cdbInquiry(2) = 0
```

```
cdblnquiry(3) = 0
cdblnquiry(4) = &h20
cdblnquiry(5) = 0
```

```
eTestInProgress = 0
eCompleteOnSuccess = 1
eCompleteOnFailure = 2
eTestNotStartedYet = 3
eErrorOnParamsPassed = 6
eOutstandingIO = 9
eUnknownStatus = 10
```

```
ha = 2
tid = 1
lun = 0
```

```
ThreadID = SCSIIssueThreadedCDB(ha, tid, lun, cdblInquiry(), 6, DataBuffer(), 32, 1, 10)
```

```
If ThreadID = 0 Then ' issuethreadedcdb failed
```

```
    MsgBox "Failure issueing threadedCDB"
```

```
    Stop
```

```
Else
```

```
statusloop:
```

```
    ThreadStatus = SCSIGetThreadedCDBStatus(ThreadID)
```

```
    If ThreadStatus = eCompleteOnSuccess Then
```

```
        ThreadStatus = SCSIGetThreadedCDBStatusWData(ThreadID,SenseBuffer(),16,dTimeToDoCmd,DataBuffer(),256)
        results = "ThreadID " & ThreadID & " Completed - INQUIRY data = "
```

```
        For i = 0 To 31
```

```
            results = results & DataBuffer(i) & " "
```

```
        Next
```

```
        MsgBox results
```

```
        MsgBox "Releasing ThreadID " & ThreadID
```

```
        retval = SCSIReleaseThreadID(ThreadID)
```

```
        Stop
```

```
    Else
```

```
        MsgBox "ThreadID " & ThreadID & ", ThreadStatus = " & ThreadStatus
```

```
        GoTo statusloop
```

```
    End If
```

```
End If
```





# MultiThreaded Disk Test Functions

---

## Introduction to DTB Disk Tests

DTB Disk Test functions are broken into four types of operations:

1. Building a list of which devices should be tested
2. Building a Test Sequence, which is a list of tests to be run on each device
3. Executing the Test Sequence
4. Checking the status of each device in the test list

### DTB Disk Test Functions

#### SCSIPrepareForNewDiskTestSequence

Declare Function SCSIPrepareForNewDiskTestSequence Lib () As Long

Cleans up all structures used to define test and devices lists. Call this function at the beginning of any program that will do disk testing.

## SCSIStartDiskTestSequence

Declare Function SCSIStartDiskTestSequence Lib () As Long

This function starts the test sequence running on each device specified in the test device list. Once the test sequence has been started you will use the various SCSIGetDiskTestStatus() functions to check on the progress of each device being tested.

## SCSIAddDiskDeviceToBeTested

Declare Function SCSIAddDiskDeviceToBeTested Lib (ByVal nHA As Long,ByVal nTid As Long,ByVal nLun As Long) As Long

Use this function to add a device specified by Host Adapter, Target, & LUN to the device test list.

Example: bSuccess = SCSIAddDiskDeviceToBeTested(2, 1, 0) adds the device at HBA 2, Target 1, LUN 0 to the test list.

## SCSIGetDiskTestStatus

Declare Function SCSIGetDiskTestStatus Lib (ByVal nHA As Long,ByVal nTid As Long,ByVal nLun As Long,ByVal nTestNumber As Long) As eTEST\_STATUS

This function will return information on the test progress of the specified device. No error information will be returned with this function – if you want to retrieve error information use the function SCSIGetDiskTestStatusWData() function.

## SCSIGetDiskTestStatusWData

Declare Function SCSIGetDiskTestStatusWData Lib (ByVal nHA As Long,ByVal nTid As Long,ByVal nLun As Long,ByVal nTestNumber As Long, ByRef SenseBytes() As Byte,ByVal nSenseBufLen As Long) As eTEST\_STATUS

This function will return detailed information about the test progress of the specified drive, including Sense Data if there is a failure.

The Following Functions add various tests into the test sequence list. These functions will define the test steps as write, read, seek, start/stop, and will specify if the test is run for a period of time or for a number of blocks transferred, etc. Data patterns are also specified here.

### SCSIAddDiskWriteTest\_Time

Declare Function SCSIAddDiskWriteTest\_Time Lib (ByVal nNumberOfMinutes As Long,ByVal nRandomOrSeq As Long,ByVal nStartLBA As Long,ByVal ePatType As ePATTERN\_TYPE,ByVal pstrPatternFile As Variant,ByVal bOverlayLBA As Long) As Long

### SCSIAddDiskWriteTest\_Blocks

Declare Function SCSIAddDiskWriteTest\_Blocks Lib (ByVal lNumBlocksToWrite As Long,ByVal nRandomOrSeq As Long,ByVal nStartLBA As Long,ByVal ePatType As ePATTERN\_TYPE,ByVal pstrPatternFile As Variant,ByVal bOverlayLBA As Long) As Long

### SCSIAddDiskReadTest\_Time

Declare Function SCSIAddDiskReadTest\_Time Lib (ByVal nNumberOfMinutes As Long,ByVal nRandomOrSeq As Long,ByVal nStartLBA As Long,ByVal ePatType As ePATTERN\_TYPE,ByVal pstrPatternFile As Variant,ByVal bOverlayLBA As Long,ByVal bCompareData As Long) As Long

### SCSIAddDiskReadTest\_Blocks

Declare Function SCSIAddDiskReadTest\_Blocks Lib (ByVal lNumBlocksToWrite As Long,ByVal nRandomOrSeq As Long,ByVal nStartLBA As Long,ByVal ePatType As ePATTERN\_TYPE,ByVal pstrPatternFile As Variant,ByVal bOverlayLBA As Long,ByVal bCompareData As Long) As Long

### SCSIAddDiskWriteReadTest\_Time

Declare Function SCSIAddDiskWriteReadTest\_Time Lib (ByVal nNumberOfMinutes As Long,ByVal nRandomOrSeq As Long,ByVal nStartLBA As Long,ByVal ePatType As ePATTERN\_TYPE,ByVal pstrPatternFile As Variant,ByVal bOverlayLBA As Long,ByVal bCompareData As Long) As Long

### SCSIAddDiskWriteReadTest\_Blocks

Declare Function SCSIAddDiskWriteReadTest\_Blocks Lib (ByVal INumBlocksToWrite As Long,ByVal nRandomOrSeq As Long,ByVal nStartLBA As Long,ByVal ePatType As ePATTERN\_TYPE,ByVal pstrPatternFile As Variant,ByVal bOverlayLBA As Long,ByVal bCompareData As Long) As Long

### SCSIAddDiskSeekTest\_Time

Declare Function SCSIAddDiskSeekTest\_Time Lib (ByVal nNumberOfMinutes As Long,ByVal nRandomOrSeq As Long,ByVal nStartLBA As Long) As Long

### SCSIAddDiskSeekTest\_Blocks

Declare Function SCSIAddDiskSeekTest\_Blocks Lib (ByVal INumBlocksToWrite As Long,ByVal nRandomOrSeq As Long,ByVal nStartLBA As Long) As Long

### SCSIAddDiskVerifyTest\_Time

Declare Function SCSIAddDiskVerifyTest\_Time Lib (ByVal nNumberOfMinutes As Long,ByVal nRandomOrSeq As Long,ByVal nStartLBA As Long) As Long

### SCSIAddDiskVerifyTest\_Blocks

Declare Function SCSIAddDiskVerifyTest\_Blocks Lib (ByVal INumBlocksToVerify As Long,ByVal nRandomOrSeq As Long,ByVal nStartLBA As Long) As Long

### SCSIAddDiskWriteVerifyTest\_Time

Declare Function SCSIAddDiskWriteVerifyTest\_Time Lib (ByVal nNumberOfMinutes As Long,ByVal nRandomOrSeq As Long,ByVal nStartLBA As Long,ByVal ePatType As ePATTERN\_TYPE,ByVal pstrPatternFile As Variant,ByVal bOverlayLBA As Long,ByVal bByteCheck As Long) As Long

### SCSIAddDiskWriteVerifyTest\_Blocks

Declare Function SCSIAddDiskWriteVerifyTest\_Blocks Lib (ByVal INumBlocksToVerify As Long,ByVal nRandomOrSeq As Long,ByVal nStartLBA As Long,ByVal ePatType As ePATTERN\_TYPE,ByVal pstrPatternFile As Variant,ByVal bOverlayLBA As Long,ByVal bByteCheck As Long) As Long

### SCSIAddDiskFormatTest\_SameSize

Declare Function SCSIAddDiskFormatTest\_SameSize Lib () As Long

### SCSIAddDiskFormatTest\_NewSize

Declare Function SCSIAddDiskFormatTest\_NewSize Lib (ByVal nNewBlockSize As Long) As Long

### SCSIAddDiskFWDownloadTest

Declare Function SCSIAddDiskFWDownloadTest Lib (ByVal varFirmwareFileName As Variant) As Long

### SCSIAddDiskSpinTest

Declare Function SCSIAddDiskSpinTest Lib (ByVal nNumberOfMinutes As Long) As Long

### SCSIPauseDiskTest

Declare Function SCSIPauseDiskTest Lib (ByVal nHA As Long,ByVal nTid As Long,ByVal nLun As Long,ByVal nTestNumber As Long) As Long

This function will pause the currently executing test running on the specified device. To resume testing use the SCSIResumeDiskTest() function.

### SCSIResumeDiskTest

Declare Function SCSIResumeDiskTest Lib (ByVal nHA As Long,ByVal nTid As Long,ByVal nLun As Long,ByVal nTestNumber As Long) As Long

This functions resumes a previously paused test on the specified device.

## SCSIStopDiskTest

Declare Function SCSIStopDiskTest Lib (ByVal nHA As Long,ByVal nTid As Long,ByVal nLun As Long,ByVal nTestNumber As Long) As Long

This function will stop the currently running test on the specified device. If there are subsequent tests in the test sequence they will begin immediately.

## SCSIStopAllDiskTest

Declare Function SCSIStopAllDiskTest Lib () As Long

This function will stop the currently running test on the specified device, and if there are subsequent tests in the test sequence they will NOT be executed.

# MultiThreaded Tape Test Functions

---

## Introduction to DTB Tape Tests

DTB Tape Test functions are broken into four types of operations:

1. Building a list of which devices should be tested
2. Building a Test Sequence, which is a list of tests to be run on each device
3. Executing the Test Sequence
4. Checking the status of each device in the test list

## DTB Tape Test Functions

### SCSIPrepareForNewTapeTestSequence

Declare Function SCSIPrepareForNewTapeTestSequence Lib () As Long

Cleans up all structures used to define test and devices lists. Call this function at the beginning of any program that will do tape testing.

### SCSIStartTapeTestSequence

Declare Function SCSIStartTapeTestSequence Lib () As Long

Starts executing the test sequence on each device in the selected device list

### SCSIAddTapeDeviceToBeTested

Declare Function SCSIAddTapeDeviceToBeTested Lib (ByVal nHA As Long,ByVal nTid As Long,ByVal nLun As Long) As Long

Lets you add a device specified by it's HBA, Target, and LUN into the device test list

### SCSIGetTapeTestStatus

Declare Function SCSIGetTapeTestStatus Lib (ByVal nHA As Long,ByVal nTid As Long,ByVal nLun As Long,ByVal nTestNumber As Long) As eTEST\_STATUS

Returns test progress data on the specified device.



## SCSIGetTapeTestStatusWData

Declare Function SCSIGetTapeTestStatusWData Lib (ByVal nHA As Long,ByVal nTid As Long,ByVal nLun As Long,ByVal nTestNumber As Long, ByRef SenseBytes() As Byte,ByVal nSenseBufLen As Long) As eTEST\_STATUS

Returns detailed test progress data (including Sense data) on the specified device.

The Following Functions add various tests into the test sequence list. These functions will define the test steps as write, read, space, rewind, and will specify if the test is run for a period of time or for a number of blocks transferred, etc. Data patterns are also specified here.

See the example VB code illustrating DTB TMM testing project to see how each test step in a sequence can be specified.

## SCSIAddTapeWriteTest\_Time

Declare Function SCSIAddTapeWriteTest\_Time Lib (ByVal nNumberOfMinutes As Long,ByVal ePatType As ePATTERN\_TYPE,ByVal varPatternFile As Variant) As Long

## SCSIAddTapeWriteTest\_MBytes

Declare Function SCSIAddTapeWriteTest\_MBytes Lib (ByVal nNumMBytesToWrite As Long,ByVal ePatType As ePATTERN\_TYPE,ByVal varPatternFile As Variant) As Long

## SCSIAddTapeReadTest\_Time

Declare Function SCSIAddTapeReadTest\_Time Lib (ByVal nNumberOfMinutes As Long,ByVal ePatType As ePATTERN\_TYPE,ByVal varPatternFile As Variant,ByVal bCompareData As Long) As Long

### SCSIAddTapeReadTest\_

Declare Function SCSIAddTapeReadTest\_MBytes Lib (ByVal nNumMBytesToRead As Long,ByVal ePatType As ePATTERN\_TYPE,ByVal varPatternFile As Variant,ByVal bCompareData As Long) As Long

### SCSIAddTapeWriteFMTest

Declare Function SCSIAddTapeWriteFMTest Lib () As Long

### SCSIAddTapeReadFMTest

Declare Function SCSIAddTapeReadFMTest Lib () As Long

### SCSIAddTapeRewindTest

Declare Function SCSIAddTapeRewindTest Lib () As Long

### SCSIAddTapeSpaceTest

Declare Function SCSIAddTapeSpaceTest Lib (ByVal eSpaceType As eSPACE\_TYPE\_TAPE) As Long

### SCSIAddTapeCompressionTest

Declare Function SCSIAddTapeCompressionTest Lib (ByVal eCompType As eCOMPRESSION\_TYPE\_TAPE) As Long

### SCSIAddTapeLogPageTest

Declare Function SCSIAddTapeLogPageTest Lib (ByVal eLogType As eLOGPAGE\_TYPE\_TAPE) As Long

### SCSIAddTapeSynchronizeTest

Declare Function SCSIAddTapeSynchronizeTest Lib () As Long

This function will pause testing until all of the devices in the test list are at this same point in the test sequence, then testing will proceed.

### SCSIAddTapeExternalProgram

Declare Function SCSIAddTapeExternalProgram Lib (ByVal varProgramName As Variant) As Long

This function allows you to specify an external program to be executed as a test step.

### SCSIPauseTapeTest

Declare Function SCSIPauseTapeTest Lib (ByVal nHA As Long,ByVal nTid As Long,ByVal nLun As Long,ByVal nTestNumber As Long) As Long

This function will pause testing on the specified device until the VBSCSIResumeTapeTest() function is called to resume testing.

### SCSIResumeTapeTest

Declare Function SCSIResumeTapeTest Lib (ByVal nHA As Long,ByVal nTid As Long,ByVal nLun As Long,ByVal nTestNumber As Long) As Long

Resumes the current test on a previously paused device.

### SCSIStopTapeTest

Declare Function SCSIStopTapeTest Lib (ByVal nHA As Long,ByVal nTid As Long,ByVal nLun As Long,ByVal nTestNumber As Long) As Long

This function will Stop the currently executing test on the specified device. If there are more tests in the test sequence the next test step will be started.

### SCSIStopAllTapeTest

Declare Function SCSIStopAllTapeTest Lib () As Long

This function will stop the current test on the specified device and any subsequent tests in the test sequence will NOT be executed.

---

## LSI SCSI Protocol Control Functions

### SCSIPHostAdapterCount

Declare Function SCSIPHostAdapterCount Lib () As Integer

Returns the number of LSI-based U320 HBA's in the system

### SCSIPValidTarget

Declare Function SCSIPValidTarget Lib (ByVal ha as Long, ByVal target as Long) As Integer

Pass this function the target number of a device on an LSI-based U320 HBA to see if the target is recognized.

Returns 0 if the target is recognized, non-zero if not.

### SCSIPSetSpeed5

Declare Function SCSIPSetSpeed5 Lib (ByVal ha as long, ByVal target as long ) as Integer

Sets the negotiated speed between the HBA and the specified target to 5MB/second

Returns 0 on success, non-zero on failure

### SCSIPSetSpeed10

Declare Function SCSIPSetSpeed10 Lib (ByVal ha as long, ByVal target as long ) as Integer

Sets the negotiated speed between the HBA and the specified target to 10MB/second

Returns 0 on success, non-zero on failure

### SCSIPSetSpeed20

Declare Function SCSIPSetSpeed20 Lib (ByVal ha as long, ByVal target as long ) as Integer  
Sets the negotiated speed between the HBA and the specified target to 20MB/second

Returns 0 on success, non-zero on failure

### SCSIPSetSpeed40

Declare Function SCSIPSetSpeed40 Lib (ByVal ha as long, ByVal target as long ) as Integer  
Sets the negotiated speed between the HBA and the specified target to 40MB/second

Returns 0 on success, non-zero on failure

### SCSIPSetSpeed80

Declare Function SCSIPSetSpeed80 Lib (ByVal ha as long, ByVal target as long ) as Integer  
Sets the negotiated speed between the HBA and the specified target to 80MB/second

Returns 0 on success, non-zero on failure

### SCSIPSetSpeed160

Declare Function SCSIPSetSpeed160 Lib (ByVal ha as long, ByVal target as long ) as Integer  
Sets the negotiated speed between the HBA and the specified target to 160MB/second

Returns 0 on success, non-zero on failure

### SCSIPSetSpeed320

Declare Function SCSIPSetSpeed320 Lib (ByVal ha as long, ByVal target as long ) as Integer  
Sets the negotiated speed between the HBA and the specified target to 320MB/second

Returns 0 on success, non-zero on failure

### SCSIPSetWide

Declare Function SCSIPSetWide Lib (ByVal ha as long, ByVal target as long ) as Integer

Sets the transfer width to wide between the HBA and the specified Target

Returns 0 on success, non-zero on failure

### SCSIPSetNarrow

Declare Function SCSIPSetNarrow Lib (ByVal ha as long, ByVal target as long ) as Integer

Sets the transfer width to narrow between the HBA and the specified Target

Returns 0 on success, non-zero on failure

### SCSIPSetNegotiated

Declare Function SCSIPSetNegotiated Lib (ByVal ha as long, ByVal target as long, ByVal offset as integer, ByVal period as integer, ByVal iu as integer, ByVal dt as integer, ByVal qas as integer, ByVal wide as integer) as integer

Allows any of the negotiated parameters to be set to the specified values

Returns 0 on success, non-zero on failure

### SCSIPResetBus

Declare Function SCSIPResetBus Lib (ByVal ha as long) as integer

Issues a SCSI Bus Reset

Returns 0 on success, non-zero on failure

### SCSIPGetNegotiated

Declare Function SCSIPGetNegotiated Lib (ByVal ha as long, ByVal target as long, offset as integer, period as integer, iu as integer, dt as integer, qas as integer, wide as integer) as integer

Returns the negotiated parameters from the specified device.

---

## SCSIAnd

(ByVal number1 As int, ByVal number 2 As int) As Integer

Logically AND's number1 with number2

Returns: results of AND operation

Example:

```
Dim retval as integer
retval = SCSIAnd(&H82, &H7f)
` AND hex 82 with hex 7f - returns &H02
```

---

## SCSIBuffer2File

(ByVal buffnum As int, ByVal datalength As int, filename as String) As Integer

Writes *datalength* bytes of buffer # *buffnum* into file *filename*.  
The file filename is opened for append, and is created if it does not exist.

Returns: returns 0 on success, non-zero on failure

Example:

```
Dim retval as integer
retval = SCSIBuffer2File(0,512,"myfile.dat")
` write 512 bytes from buffer 0 to the file "myfile.dat"
```

---

## SCSIBufferSize

() as long

Returns the maximum size of buffer 0.

---

## SCSICheckRandomBlock

(ByVal buffer As Integer, ByVal blocksize as Integer, ByVal blocknum as long, ByVal offset as Long, ByVal numberofblocks as integer) As Integer

Checks the data in *buffer* to make sure that the block numbers and random data are correct. *Blocksize* specifies the size of the expected data blocks, *blocknum* specifies the first block number in the series of blocks, *numberofblocks* specifies how many blocks are expected to be in the buffer. *Offset* allows you to index into the buffer to a specific block.

Return value is  $-1$  if the check is successful. If the return value does not equal  $-1$  you should call `SCSIGetRandomErrors` to determine the cause of failure.

### Example:

```
Dim retval as Integer

retval = SCSICheckRandomBlock(0,512,123,0,64)

If retval <> -1 Then
msgbox "check random Retval = " & retval
retval =SCSIGetRandomErrors(e_block,a_block,block,offset,e_data,a_data)

' the error parameters that are not -1 explain what went wrong
msgbox "e_block = " & e_block & ", a_block = " & a_block & ", e_data = " & e_data & ",
a_data = " & a_data & ", block = " & block & ", offset = " & offset & ", g_error = " &
retval

End If
```

---

## SCSICompareBuffers

(ByVal startbyte As Long, ByVal numbyte As long) As Integer

Compares the contents of two buffers buffer 0 and buffer 1), starting from *startbyte*, for length *numbyte*.

Returns: returns 0 on success, byte number of miscompare on failure

### Example:

```
Dim bufferdata(512) As Integer
Dim retval As Integer
Dim results As String
Dim NL As String

NL = Chr(10)

retval = SCSIFillPattern(0,4) 'fill buffer 0 with random data
retval = SCSIFillPattern(1,4) 'fill buffer 1 with random data

' compare buffer 0 with 1, starting with byte 1. compare 512 bytes.
retval = SCSICompareBuffers(1,512)
If retval <> 0 then
results = "Compare failed at byte " & retval
results = results & NL & "buffer 0 = "
retval = SCSIGetBuffer(0,512,bufferdata)
For i = 0 To 32
results = results & Format(Hex(bufferdata(i)),"@e") & " "
Next
```



```
results = results & NL &"buffer 1 = "  
retval = SCSIGetBuffer(1,512,bufferdata)  
For i = 0 To 32  
    results = results & Format(Hex(bufferdata(i)),"@@" ) & " "  
Next  
MsgBox results  
Stop  
End If
```

---

## SCSICMQ

As Integer

Clears the Windows message queue so messages from button pushes, etc, can be processed. Call this function before checking status.

Returns: returns 0 on success, non-zero on failure

---

## SCSIDec2Hex

(ByVal decimal as integer, hexdata() as Byte) as integer

This function accepts a decimal number, then converts it into hexadecimal and fills in a array of four bytes with the hex values.

Returns: returns 0 on success, non-zero on failure

Example:

```
Dim retval as integer  
Dim hexbytes(4) as integer  
Dim results as string  
Dim loop as integer  
  
RetVal = SCSIDec2Hex(12345,hexdata)  
Results = ""  
For loop = 0 To 3  
    results = results & Format(Hex(hexdata(loop)),"@@" ) & " "  
Next  
MsgBox results
```

---

## SCSIDiskCorruptBlock

(ByVal ha as integer, ByVal target as integer, ByVal lun as integer, ByVal blocknum as long, ByVal eccspan as integer) as integer

This function corrupts the block specified by blocknum, on the drive specified by ha/target/lun, with an error that is eccspan bits long.

Return value is the status of the WRITE LONG cdb – 0 = command complete (success), 2 = check condition (failed)

---

## SCSIDiskGetECCSpan

(ByVal ha as integer, ByVal target as integer, ByVal lun as integer) as long

This function issues a MODE SENSE to read Mode Page 1 from the disk specified by ha/target/lun.

The return value is the Correction Span value reported in the Error Correction mode page.

---

## SCSIGetReadLongSize

(ByVal ha as integer, ByVal target as integer, ByVal lun as integer) as long

Returns the number of bytes that should be used as the blocksize parameter in the SCSIReadLong and SCSIWriteLong functions.

---

## SCSIDiskRead

(ByVal ha As Integer, ByVal target as Integer, ByVal lun as Integer, ByVal count As Integer, ByVal highblock As Long, ByVal blocksize As Long, ByVal buffer As Integer) As Integer

Reads *count blocksize* size blocks of data, starting at *highblock*, from drive specified by *ha/target/lun* into *buffer*.

Returns: returns 0 on success, non-zero on failure

Example:

```
Dim bufferdata(512) As Integer
Dim sensedata(16) As Integer
Dim highblock As Long
```

```

Dim blocksize As Long
Dim count As Integer
Dim ha As Integer
Dim target As Integer
Dim lun As Integer
Dim retval As Integer
Dim i As Integer
Dim testloop As Integer
Dim results As String
Dim NL As String

NL = Chr(10)

ha = 1
target = 6
lun = 0
startblock = 100

count = 1

retval = SCSIRead(ha,target,lun,count,startblock,512,1)

If retval <> 0 Then
    results = "Read command failed - Status = " & retval
    retval = SCSIViewSense(sensedata)
    results = results & NL & "Sense Key = " & Format(Hex(sensedata(2)), "@@")
    results = results & NL & "Sense Code = " & Format(Hex(sensedata(12)), "@@")
    results = results & NL & "ASQ = " & Format(Hex(sensedata(13)), "@@")
    MsgBox results
    Stop
End If

```

---

## SCSIDiskReadFUA

(ByVal ha As Integer, ByVal target As Integer, ByVal lun As Integer, ByVal count As Integer, ByVal highblock As Long, ByVal blocksize As Long, ByVal buffer As Integer ) As Integer

This function is the same as SCSIRead, but it sets the Force Unit Access bit of the CDB.

See documentation on SCSIRead

---

## SCSIDiskReadLong

(ByVal ha As Integer, ByVal target As Integer, ByVal lun As Integer, ByVal correct As Integer, ByVal highblock As Long, ByVal blocksize As Integer, ByVal buffer As Integer ) As Integer

Issues a SCSI READ LONG cdb to read **blocksize** bytes from block **highblock** of the drive specified by **ha/target/lun**. The correct bit of the cdb can be set with the **correct** parameter. The data is read into the buffer specified by **buffer**.

Use the SCSIGetReadLongSize function to determine the correct blocksize to specify.

Return value will be 0 for COMMAND COMPLETE, 2 for CHECK CONDITION

## SCSIDiskStartStop

(ByVal ha As Integer, ByVal target As Integer, ByVal lun As Integer, ByVal start As Integer) As Integer

Starts (*start* = 1) or stops (*start* = 0) drive specified by *ha/target/lun*.

Returns: returns 0 on success, non-zero on failure

Example:

```
Dim retval as integer
Dim ha As Integer
Dim target As Integer
Dim lun As Integer
Dim retval As Integer

` change ha, target, lun to match your system
ha = 1
target = 6
lun = 0

`stop the drive
retval = SCSIDiskStartStop(ha,target,lun,0)

`start the drive
retval = SCSIDiskStartStop(ha,target,lun,1)
```

---

## SCSIDiskUnload

(ByVal ha As Integer, ByVal target As Integer, ByVal lun As Integer) As Integer

Ejects/Unloads media from drive specified by *ha/target/lun*.

Returns: returns 0 on success, non-zero on failure

Example:

```
Dim retval as integer
Dim ha As Integer
Dim target As Integer
Dim lun As Integer
Dim retval As Integer

` change ha, target, lun to match your system
Ha = 1
Target = 6
Lun = 0

`unload the drive
retval = SCSIDiskUnload(ha,target,lun)
```

---

## SCSIDiskVerify

(ByVal ha As Integer, ByVal target as Integer, ByVal lun as Integer, ByVal count As Integer, ByVal highblock As Long, ByVal blocksize As Long, ByVal buffer As Integer) As Integer

Issues a SCSI VERIFY command to the drive specified by *ha/target/lun*. Verifies *count blocksize* blocks of data, starting at *highblock*.

Returns: returns 0 on success, non-zero on failure

Example:

```
Dim bufferdata(512) As Integer
Dim sensedata(16) As Integer
Dim highblock As Long
Dim blocksize As Long
Dim count As Integer
Dim ha As Integer
Dim target As Integer
Dim lun As Integer
Dim retval As Integer
Dim i As Integer
Dim testloop As Integer
Dim results As String
Dim NL As String

NL = Chr(10)

ha = 1
target = 6
lun = 0
startblock = 100
count = 1

retval = SCSIDiskVerify(ha,target,lun,count,startblock,512,1)

If retval <> 0 Then
    results = "Verify command failed - Status = " & retval
    retval = SCSIViewSense(sensedata)
    results = results & NL & "Sense Key = " & Format(Hex(sensedata(2)), "@@")
    results = results & NL & "Sense Code = " & Format(Hex(sensedata(12)), "@@")
    results = results & NL & "ASQ = " & Format(Hex(sensedata(13)), "@@")
    MsgBox results
    Stop
End If
```

---

## SCSIDiskWrite

(ByVal ha As Integer, ByVal target as Integer, ByVal lun as Integer, ByVal count As Integer, ByVal highblock As Long, ByVal blocksize As Long, ByVal buffer As Integer) As Integer

Writes *count blocksize* size blocks of data, starting at *highblock*, to drive specified by *ha/target/lun* into *buffer*.

Returns: returns 0 on success, non-zero on failure

Example: uses disk write and read

```
Dim bufferdata(512) As Integer
Dim sensedata(16) As Integer
Dim highblock As Long
Dim blocksize As Long
Dim count As Integer
Dim ha As Integer
Dim target As Integer
Dim lun As Integer
Dim retval As Integer
Dim i As Integer
Dim testloop As Integer
Dim results As String
Dim NL As String

NL = Chr(10)

ha = 1
target = 6
lun = 0
startblock = 100
count = 1

For i = 0 To 64
    bufferdata(i) = i
Next

For testloop = 100 To 105
startblock = testloop

retval = SCSIFillBlockNum(0, startblock, 1, 512)
retval = SCSIDiskWrite(ha, target, lun, count, startblock, 512, 0)

If retval <> 0 Then
    results = "Disk Write failed - Status = " & retval
    retval = SCSIViewSense(sensedata)
    results = results & NL & "Sense Key = " & Format(Hex(sensedata(2)), "@@")
    results = results & NL & "Sense Code = " & Format(Hex(sensedata(12)), "@@")
    results = results & NL & "ASQ = " & Format(Hex(sensedata(13)), "@@")
    MsgBox results
    Stop
End If

retval = SCSIDiskRead(ha, target, lun, count, startblock, 512, 1)

If retval <> 0 Then
    results = "Read command failed - Status = " & retval
    retval = SCSIViewSense(sensedata)
    results = results & NL & "Sense Key = " & Format(Hex(sensedata(2)), "@@")
    results = results & NL & "Sense Code = " & Format(Hex(sensedata(12)), "@@")
    results = results & NL & "ASQ = " & Format(Hex(sensedata(13)), "@@")
    MsgBox results
    Stop
End If

retval = SCSICompareBuffers(1, 511)
If retval <> 0 Then
    results = "Compare failed at byte " & retval
    results = results & NL & "Write buffer = "
    retval = SCSIGetBuffer(0, 512, bufferdata)
    For i = 0 To 32
        results = results & Format(Hex(bufferdata(i)), "@@") & " "
    Next

    results = results & NL & "Read buffer = "
    retval = SCSIGetBuffer(1, 512, bufferdata)
    For i = 0 To 32
        results = results & Format(Hex(bufferdata(i)), "@@") & " "
    Next
    MsgBox results
    Stop
End If
```

```
MsgBox "Test Passed"  
Stop
```

---

## SCSIDiskWriteFUA

(ByVal ha As Integer, ByVal target As Integer, ByVal lun As Integer, ByVal count As Integer, ByVal highblock As Long, ByVal blocksize As Long, ByVal buffer As Integer ) As Integer

This function is the same as SCSIWrite, but it sets the Force Unit Access bit of the CDB.

See documentation on SCSIWrite

---

## SCSIDiskWriteLong

(ByVal ha As Integer, ByVal target As Integer, ByVal lun As Integer, ByVal highblock As Long, ByVal blocksize As Integer, ByVal buffer As Integer ) As Integer

Issues a SCSI WRITE LONG cdb to write *blocksize* bytes from block *highblock* of the drive specified by *ha/target/lun*. The data is written from the buffer specified by *buffer*.

Use the SCSIGetReadLongSize function to determine the correct blocksize to specify.

Return value will be 0 for COMMAND COMPLETE, 2 for CHECK CONDITION

---

## SCSIDLT\_FWDL

(ByVal ha as integer, ByVal target as integer, ByVal lun as integer, ByVal FileName as String) as Integer

Download firmware to a Quantum DLT drive

Returns: 0 on success, -2 if firmware file not found, -1 if download fails

---

## SCSIFile2Buffer

(ByVal buffnum As int, ByVal datalength As int, filename as String) As Integer

copies *datalength* bytes from file *filename* into buffer # *buffnum* .

Returns: returns 0 on success, non-zero on failure

---

## SCSIFileOffset2Buffer

(ByVal buffnum As int, ByVal datalength As int, filename as String, ByVal offset as long)  
As Integer

copies *datalength* bytes from file *filename* into buffer # *buffnum* .  
The data is copied from *offset* bytes from the beginning of the file.

This function is useful if you need to copy the contents of a file to a SCSI device in “chunks”, for example, use this function to download firmware to a device using segmented WRITE BUFFER commands.

Returns: returns 0 on success, non-zero on failure

---

## SCSIFillBlockNum

(ByVal buffer as Integer, ByVal sblock as Long, ByVal count as Integer, ByVal blocksize as Integer) As Integer

Fills buffer # *buffer* with disk *blocknumber* data, starting at byte 0 of buffer, continuing for *count X blocksize* bytes (*count* blocks of data)

Returns: returns 1 on success, -1 on failure

Example:

See the SCSIFillBlockNum function used in the example for the SCSIDiskWrite function



---

## SCSIFillBuffer

(ByVal buffer as Integer, ByVal count as Long, ByVal pattsize as Integer, bufferdata() as Byte) as Integer

Fills buffer # *buffer* with *count* bytes of data. The buffer is filled with the pattern specified in the *bufferdata()*. The number of bytes in the data pattern is specified by *pattsize*.

Example:

For example, you may specify a data pattern that is:

- 4 bytes long
- consists of the hex data 01, 02, 03, 04

Then you could repeat this pattern into the first 1,024 bytes of buffer 0

With the command:

```
Ret = SCSIFillBuffer(0,1024,4,databuf)
```

Returns: returns 1 on success, -1 on failure

---

## SCSIFillPattern

(ByVal buffer as Integer, ByVal pattern as Integer) as Integer

Fills buffer # *buffer* with one of the following patterns, based on *pattern*:

- 0 – all zeros – 0
- 1 – all ones – 1
- 2 – alternating 0/1 – 0xA5
- 3 – alternating 1/0 – 0x5A
- 4 – random data

Returns: returns 1 on success, -1 on failure

Example:

```
Retval = SCSIFillPattern(0,4)  
' fills buffer 0 with random data
```

---

## SCSIFillRandom

(ByVal buffer As Integer, ByVal blocksize As Integer, ByVal startingblock as Long, ByVal numberofblocks as integer) As Integer

Fills the specified buffer with `numberofblocks` `blocksize` blocks of random data. The first four bytes of each block of data contain the block number, the next four bytes contain the seed used to generate the random data for that block.

Return value is the number of blocks generated.

Example:

```
Dim retval as Integer
```

```
retval = SCSIFillRandom(0,512,123,64)
```

```
`Fills buffer 0 with 64 512 byte blocks of random data with block numbers 123 - 187.
```

---

## SCSIGetBuffer

(ByVal buffer as Integer, ByVal count as Long, bufferdata() as Byte) as Integer

Retrieves *count* bytes of data from buffer # *buffer*, returns data in byte array *bufferdata*.

Returns: returns 1 on success, -1 on failure

Example:

See the example for the SCSIUserDefinedCDB function

---

## SCSIGetBufferMode

(ByVal ha As Integer, ByVal target As Integer, ByVal lun As Integer,) as Integer

Retrieves the current buffer setting of the selected tape drive.

Returns: returns the current buffer mode setting

Example:

```
Dim target As Integer
```

```
Dim retval As Integer
```

```
Dim buffmode As Integer
```

```
retval = SCSIGetDIIVersion()  
MsgBox "version = " & retval
```

```
buffmode = SCSIGetBufferMode(2,4,0)  
MsgBox "buffer mode = " & buffmode
```

```
retval = SCSISetBufferMode(2,4,0,0)
```

```
buffmode = SCSIGetBufferMode(2,4,0)  
MsgBox "buffer mode = " & buffmode
```

---

## SCSIGetDeviceType

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer ) as Integer

Issues a SCSI INQUIRY command to drive specified by *ha/target/lun* and returns PERIPHERAL DEVICE TYPE value (INQUIRY byte 0).

Returns: PERIPHERAL DEVICE TYPE on success, -1 on failure.

Example:

```
Dim devtype integer  
Dim ha as integer  
Dim target as integer  
Dim lun as integer
```

```
ha = 2  
target = 0  
lun = 0
```

```
Devtype = SCSIGetDeviceType(ha,target,lun)
Msgbox "Device type = " & Devtype
```

---

## SCSIGetDllVersion

( ) as Integer

Returns the current version of the PTI pssl dll

Example:

```
dim retval as integer

retval = SCSIGetDllVersion()
msgbox "DLL version " & retval
```

---

## SCSIErrorDetails

(iostat as integer, hbastat as integer, scsistat as integer) as integer

Retrieves the SRB (Driver), HBA, and SCSI Target status from the most recently issued CDB.

Returns 0

Example:

```
Dim SRBstat as integer
Dim Hastat as integer
Dim Targetstat as integer
Dim retval as integer

retval = SCSIGetErrorDetails(SRBstat, Hastat, Targetstat)

Msgbox "SRB Status = " & SRBstat & ", Host Adapter Status = " & Hastat & ",
Target Status = " & Targetstat
```

---

## SCSIGetProduct

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer , product as String) as Integer

Issues a SCSI INQUIRY command to drive specified by *ha/target/lun* and returns PRODUCT ID value (INQUIRY bytes 16-31) in string *product*.

Returns: returns 1 on success, 0 on failure

Example:

```
Dim devtype as integer
Dim ha as integer
Dim target as integer
Dim lun as integer
Dim product as string

Ha = 2
Target = 0
Lun = 0

retval = SCSIGetProduct.Ha,Target,lun,product)

msgbox "Product = " & product
```

---

## SCSIGetRandomErrors

(expected\_blocknum as long, actual\_blocknum as long, block as integer, offset as integer, expected\_data as integer, actual\_Data as integer) As Integer

Returns error information describing a SCSICheckRandomBlock failure.

Return value will be 1 if valid error information is available, -1 if the error information is invalid. The values returned in the passed parameters will be -1 if they are invalid, otherwise they will contain information that will show if the block number did not compare (expected\_blocknum is what the block number should have been, actual\_blocknum will be what blocknumber was read). If a data byte does not compare block will contain the block number the error occurred in, offset will contain the byte offset within that block, and expected\_data and actual\_data will show the data error.

---

## SCSIGetTapeCapacity

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, tbs as Long) as Integer

Issues a SCSI MODE SENSE command to drive specified by *ha/target/lun*, returns Tape Block Length (Mode Page Block Descriptor bytes 5-7) in long tbs.

Returns: returns 1 on success, -1 on failure

---

## SCSIGetVendor

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, vendor as String) as Integer

Issues a SCSI INQUIRY command to drive specified by *ha/target/lun* and returns VENDOR ID value (INQUIRY bytes 8-15) in string *vendor*.

Returns: returns 1 on success, 0 on failure

Example:

```
Dim devtype integer
Dim ha as integer
Dim target as integer
Dim lun as integer
Dim vendor as string

Ha = 2
Target = 0
Lun = 0

retval = SCSIGetProduct.Ha,Target,lun,vendor)

msgbox "Vendor = " & vendor
```

---

## SCSIGetVersion

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer , version as String) as Integer

Issues a SCSI INQUIRY command to drive specified by *ha/target/lun* and returns VERSION ID value (INQUIRY bytes 32-35) in string *version*.

Returns: returns 1 on success, 0 on failure

Example:

```
Dim devtype integer
Dim ha as integer
Dim target as integer
Dim lun as integer
Dim version as string

Ha = 2
Target = 0
Lun = 0

retval = SCSIGetVersion.Ha,Target,lun,version)

msgbox "Version = " & version
```

---

## SCSIHex2Dec

(ByVal hexdata() as Byte) as Integer

This function accepts an array of four bytes, then converts it into decimal number and returns that number

Returns: decimal number

---

## SCSIHostAdapterCount

() as Integer

Returns: number of SCSI host adapters in system on success

---

---

## SCSIHPLTO\_FWDL

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal FileName as String) as Integer

Download firmware to a HP LTO drive

Returns: 0 on success, -2 if firmware file not found, -1 if download fails

---

## SCSIIBMLTO\_FWDL

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal FileName as String) as Integer

Download firmware to a IBM LTO drive

Returns: 0 on success, -2 if firmware file not found, -1 if download fails

---

---

## SCSIInitializeElementStatus

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer) as Integer



Issues an Initialize Element Status command to the addressed jukebox device.

Returns: Zero on success, non-zero on failure

Example:

```
Dim retval as Integer
Dim ha as Integer
Dim target as Integer
Dim lun as Integer

ha = 2
target = 5
lun = 0

retval = SCSIInitializeElementStatus(ha, target, lun)

if (retval = 0) then
    MsgBox "Initialize Element Status successful"
else
    MsgBox "Initialize Element Status FAILED"
```

---

## SCSIInitializeElementStatusRange

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal range as Integer, ByVal address as Integer, ByVal num as Integer) as Integer

Issues an Initialize Element Status with Range command to the addressed jukebox device.

The Range field indicates which elements to initialize. If range = 0 initialize all elements. If range = 1 initialize the range of elements specified by the address and number fields.

Returns: Zero on success, non-zero on failure

Example:

```
Dim retval as Integer
Dim ha as Integer
Dim target as Integer
Dim lun as Integer
Dim range as Integer
Dim start as Integer

ha = 2
target = 5
lun = 0

range = 1
start = 10

retval = SCSIInitializeElementStatusRange(ha, target, lun, range, start)

if (retval = 0) then
    MsgBox "Initialize Element Status with Rangesuccessful"
```

```

else
    MsgBox "Initialize Element Status with Range FAILED"

```

---

## SCSIInquiry

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, inqdata() as Byte) as Integer

Issues a SCSI INQUIRY command to drive specified by *ha/target/lun* and returns “raw” INQUIRY data (INQUIRY bytes 0 –63) in byte array inqdata.

Returns: returns 1 on success, 0 on failure

Example:

```

Dim inqdata(64) As Integer
Dim retval As Integer
Dim ha As Integer
Dim target As Integer
Dim lun As Integer
Dim results As String
Dim inqstr As String
Dim i As Integer
Dim NL
Dim length As Integer
Dim vend As String
Dim vers As String
Dim prod As String

target = 4
ha = 1
lun = 0

retval = SCSIInquiry(ha, target, lun, inqdata)

If retval <> 1 Then
    MsgBox "Inquiry failed - Status = " & retval
    Stop
Else
    retval = SCSIGetVendor(ha, target, lun, vend)
    retval = SCSIGetProduct(ha, target, lun, product)
    retval = SCSIGetVersion(ha, target, lun, vers)

    results = "Vendor = " & vend & " Product = " & prod & " Version = " & vers

    MsgBox results

    results = "Hex INQUIRY data = "
    For i = 0 To 32
        results = results & Format(Hex(inqdata(i)), "@@") & " "
    Next
    MsgBox results
End If

```

---

## SCSILoadBuffer

(ByVal buffer as Integer, ByVal count as Long, bufferdata() as Byte) as Integer

Fills buffer # *buffer* with *count* bytes of *bufferdata*.

Returns: returns 1 on success, -1 on failure

---

## SCSILogSense

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal page as Integer, ByVal pagecode as Integer, logdata() as Byte) as Integer

Issues a LOG SENSE command for log page *page*, page code *pagecode* to the drive specified by *ha/target/lun*. Log Sense data is returned in the byte array *logdata*.

Returns: returns 1 on success, -1 on failure

### Example:

'This example reads LOG PAGE 30 and saves and interprets the log data as 'AIT tape log data.

```
Dim sensedata(16) As Integer

Dim ha As Integer
Dim target As Integer
Dim lun As Integer
Dim retval As Integer

Dim testloop As Integer
Dim innerloop As Integer
Dim results As String
Dim NL As String

Dim inqdata(256) As Integer
Dim inqstr As String
Dim i As Integer
Dim length As Integer
Dim vend As String
Dim vers As String
Dim prod As String

Dim today
NL = Chr(10)

ha = 1
target = 6
lun = 0

Open "aitlogs.txt" For Output As #1

'MsgBox Now
today = Now

Print #1, today & " "
Print #1, " "

retval = SCSIGetDLLVersion()
MsgBox "Dll version = " & retval

retval = SCSIInquiry(ha, target, lun, inqdata())
If retval <> 1 Then
    MsgBox "Inquiry failed - Status = " & retval
    Stop
```

```
Else
    retval = SCSIGetVendor(ha,target,lun,vendor)
    retval = SCSIGetProduct(ha,target,lun,product)
    retval = SCSIGetVersion(ha,target, lun , vers)

    results = "Vendor = " & vend & " Product = " & prod & " Version = " & vers
    MsgBox results

    Print #1, results
    Print #1, ""
    Print #1, "Host adapter = " & ha & " Target = " & target & " LUN = " & lun
    Print #1, " "
End If

getlogs
Close
MsgBox "Test Finished"
Stop

Sub GetLogs
    retval = SCSILogSense(ha,target,lun,&H30, &H40,inqdata())

    If retval <> 1 Then
        MsgBox "Log Sense failed - Status = " & retval
        Stop
    Else
        results = "Tape log = "
        For i = 0 To 128
            results = results & Hex(inqdata(i)) & " "
        Next
        Print #1, results

        results = "Tape Log Page (30h) = "
        Print #1, results

        results = "Current Number of Groups Written = "
        results = results & Hex(inqdata(8)) & Hex(inqdata(9)) & Hex(inqdata(10))
        Print #1, results

        results = "Current Number of RAW Retries = "
        results = results & Hex(inqdata(15)) & Hex(inqdata(16))
        Print #1, results

        results = "Current Number of Groups Read = "
        results = results & Hex(inqdata(21)) & Hex(inqdata(22)) & Hex(inqdata(23))
        Print #1, results

        results = "Current Number of ECC-3 Retries = "
        results = results & Hex(inqdata(28)) & Hex(inqdata(29))
        Print #1, results
        Print #1, " "

        results = "Previous Number of Groups Written = "
        results = results & Hex(inqdata(34)) & Hex(inqdata(35)) & Hex(inqdata(36))
        Print #1, results

        results = "Previous Number of RAW Retries = "
        results = results & Hex(inqdata(41)) & Hex(inqdata(42))
        Print #1, results

        results = "Previous Number of Groups Read = "
        results = results & Hex(inqdata(47)) & Hex(inqdata(48)) & Hex(inqdata(49))
        Print #1, results

        results = "Previous Number of ECC-3 Retries = "
        results = results & Hex(inqdata(54)) & Hex(inqdata(55))
        Print #1, results
        Print #1, " "

        results = "Total Number of Groups Written = "
        results = results & Hex(inqdata(60)) & Hex(inqdata(61)) & Hex(inqdata(62))
        & Hex(inqdata(63))
        Print #1, results

        results = "Total Number of RAW Retries = "
```

```
results = results & Hex(inqdata(68)) & Hex(inqdata(69)) & Hex(inqdata(70))
Print #1, results

results = "Total Number of Groups Read = "
results = results & Hex(inqdata(75)) & Hex(inqdata(76)) & Hex(inqdata(77))
          & Hex(inqdata(78))
Print #1, results

results = "Total Number of ECC-3 Retries = "
results = results & Hex(inqdata(83)) & Hex(inqdata(84))
          & Hex(inqdata(85))
Print #1, results
Print #1, " "

results = "Load Count = "
results = results & Hex(inqdata(90)) & Hex(inqdata(91))
Print #1, results
Print #1, " "

End If
End Sub
```

---

## SCSIModeSense

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal page as Integer, ByVal pagecode as Integer, modedata() as Byte) as Integer

Issues a MODE SENSE command for mode page *page*, page code *pagecode* to the drive specified by *ha/target/lun*. Mode Sense data is returned in the byte array *modedata*.

Returns: returns 1 on success, -1 on failure

---

## SCSIModeSelect

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal sp as Integer, , modedata() as Byte) as Integer

Issues a MODE SELECT command to the drive specified by *ha/target/lun*. MODE PAGE data is transferred from byte array *modedata*, excluding BLOCK DESCRIPTOR DATA.

Returns: returns 1 on success, -1 on failure

---

## SCSIModeSelectFull

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal sp as Integer, modedata() as Byte) as Integer

Issues a MODE SELECT command to the drive specified by *ha/target/lun*. MODE PAGE data is transferred from byte array *modedata*, including header + block descriptor + page data.

Returns: returns 1 on success, -1 on failure

---

## SCSIModeSenseFull

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal page as Integer, ByVal pagecode as Integer, modedata() as Byte) as Integer

Issues a MODE SENSE command for mode page *page*, page code *pagecode* to the drive specified by *ha/target/lun*. MODE SENSE command is issued with DBD (Disable Block Descriptor) bit NOT set, therefore block descriptor data IS transferred. Mode Sense data (header + block descriptor + page data) is returned in the byte array *modedata*.

Returns: returns 1 on success, -1 on failure

---

## SCSIMoveMedium

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal transport as Integer, ByVal source as Integer, ByVal destination as Integer) as Integer

Issues an Move Medium command to the addressed jukebox device.

Transport = media transport (picker) address

Source = source element address

Destination = destination element address

Returns: Zero on success, non-zero on failure

Example:

```
Dim retval as Integer
Dim ha as Integer
Dim target as Integer
Dim lun as Integer
Dim picker as Integer
Dim source as Integer
Dim dest as Integer
```

```
ha = 2
target = 5
lun = 0
```

```
picker = 86
source = 5
destination = 12
```

```
retval = SCSIMoveMedium(ha, target, lun, picker, source, dest)
if (retval = 0) then
    MsgBox "Move Medium OK"
else
    MsgBox "Move Medium FAILED"
```

---

## SCSIOr

(ByVal number1 As int, ByVal number 2 As int) As Integer

Logically OR's number1 with number2

Returns: results of OR operation

Example:

```
Dim retval as integer
retval = SCSIOr(&H82, &H7f)
' OR hex 82 with hex 7f - returns &Hff
```

---

## SCSIPositionToElement

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal transport as Integer, ByVal destination as Integer )as Integer

Issues a Position to Element command to the addressed jukebox device.

Transport = media transport (picker) address

Destination = destination element address

Returns: Zero on success, non-zero on failure

Example:

```
Dim retval as Integer
Dim ha as Integer
Dim target as Integer
Dim lun as Integer
Dim picker as Integer
Dim dest as Integer

ha = 2
target = 5
lun = 0

picker = 86
destination = 12

retval = SCSIPositionToElement(ha, target, lun, picker, dest)
```

```

if (retval = 0) then
    MsgBox "Position to Element OK"
else
    MsgBox "Position to Element FAILED"

```

---

## SCSIReadCapacity

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, highblock as Long, blocksize as Long) as Integer

Issues a SCSI READCAPACITY command to drive specified by *ha/target/lun* and returns the devices HIGHBLOCK # in *highblock* and BLOCKSIZE in *blocksize*.

Returns: returns 0 on success, non-zero on failure

Example:

```

Dim retval As Integer
Dim highblock As Long
Dim blocksize As Long
Dim target As Integer
Dim ha As Integer
Dim lun As Integer
Dim results As String
Dim sensedata(16) As Integer
Dim NL As String

NL = Chr(10)

ha = 1
target = 6
lun = 0

retval = SCSIReadCapacity(ha, target, lun, highblock, blocksize)

If retval <> 1 Then
    results = "READ CAPACITY failed - Status = " & retval
    retval = SCSIViewSense(sensedata)
    results = results & NL & "Sense Key = " & Format(Hex(sensedata(2)), "@@")
    results = results & NL & "Sense Code = " & Format(Hex(sensedata(12)), "@@")
    results = results & NL & "ASQ = " & Format(Hex(sensedata(13)), "@@")
    MsgBox results
    Stop
End If

MsgBox "highblock = " & highblock & " " & "blocksize = " & blocksize

```

---

## SCSIReadElement Status

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal elementtype as Integer, ByVal startelement as Integer, ByVal num as Integer, ByVal length as Long, eldata as Byte) as Integer

Issues a Read Element Status command to the addressed jukebox device.

Elementtype = jukebox element type requested – 0 for all types



Startelement = elements equal to or greater than the starting address are returned

Num = number of element descriptors to return

Length = byte length allocated for returned element descriptors

Eldata = pointer to an array of bytes to hold returned element descriptors

Returns: Zero on success, non-zero on failure

Example:

```

Dim eldata(1024) As Integer
Dim retval As Integer
Dim ha As Integer
Dim target As Integer
Dim lun As Integer
Dim results As String
Dim inqstr As String
Dim i As Integer
Dim NL
Dim length As Integer
Dim vendor As String
Dim vers As String
Dim product As String
Dim source As Integer
Dim picker As Integer
Dim dest As Integer
Dim x As Object
Dim bytecount As Integer
Dim dataoffset As Integer
Dim ellength As Integer
Dim pvol As Integer

Set x = CreateObject("Logger.Application")
x.WriteLine "Jukebox command test program "
x.WriteLine ""

pvol = 0
picker = 86
source = 82
dest = 0
bytecount = 0
dataoffset = 0

target = 6
ha = 2
lun = 0

retval = SCSIInquiry(ha,target,lun,eldata)
If retval <> 1 Then
    MsgBox "Inquiry failed - Status = " & retval
    Stop
Else
    retval = SCSIGetVendor(ha,target,lun,vendor)
    retval = SCSIGetProduct(ha,target,lun,product)
    retval = SCSIGetVersion(ha,target,lun,vers)

    results = "Vendor = " & vendor & " Product = " & product & " Version = "
        & vers

    x.Write "Inquiry data = "
    x.WriteLine results
End If

retval = SCSIReadElementStatus(ha,target,lun,2,0,255,1024,eldata)

If retval <> 0 Then
    MsgBox "Read Element Status failed - Status = " & retval
    Stop
Else
    dataoffset = 8

```

```
bytecount = eldata(5) + eldata(6) + eldata(7)
ellength = eldata(dataoffset + 2) + eldata(dataoffset + 3)

x.WriteLine ""
x.WriteLine ""

x.Write "Element type = " & eldata(8)
If (eldata(8) = 1) Then
  x.WriteLine " - Tape Pickers"
If (eldata(8) = 2) Then
  x.WriteLine " - Cartridge Magazine Slots"
If (eldata(8) = 3) Then
  x.WriteLine " - Tape Drives"

If (eldata(9) And 128) Then
  x.WriteLine "PVolTag = 1"
  pvol = 1
Else
  x.WriteLine "PVolTag = 0"
  pvol = 0
End If

If (eldata(9) And 64) Then
  x.WriteLine "AVolTag = 1"
Else
  x.WriteLine "AVolTag = 0"
End If

x.WriteLine "Element Descriptor Length = " & ellength
x.WriteLine "Byte Count of Descriptor data = " & bytecount
x.WriteLine "First Element Address = " & eldata(0) & eldata(1)
x.WriteLine "Number of Elements = " & eldata(2) & eldata(3)
x.WriteLine "Number of bytes of element status = " & bytecount
x.WriteLine ""

dataoffset = dataoffset + 8
bytecount = bytecount - 8

While (bytecount > 0)
  x.WriteLine ""
  x.WriteLine ""
  x.WriteLine "Element Address = " & eldata(dataoffset)
  & eldata(dataoffset+1)
  If (eldata(dataoffset+2) And 1) Then
    x.WriteLine " - Full, "
  Else
    x.WriteLine " - Empty, "
  End If

  If (eldata(dataoffset+2) And 4) Then
    x.Write " - Except = 1, "
  Else
    x.Write " - Except = 0, "
  End If

  If (eldata(dataoffset+2) And 8) Then
    x.Write "Access = 1, "
  Else
    x.Write "Access = 0, "
  End If

  x.WriteLine ""
  x.Write " - Additional Sense Code = " & eldata(dataoffset + 4)
  x.WriteLine ", Additional Sense Code Qualifier = "
  & eldata(dataoffset + 5)

  If (eldata(dataoffset + 6) And 128) Then
    x.Write " - SValid = 1 "
  Else
    x.Write " - SValid = 0, "
  End If

  If (eldata(dataoffset + 6) And 64) Then
    x.Write "Invert = 1 "
  Else
```

```

    x.Write "Invert = 0"
End If
x.Writeline ""

x.WriteLine " - Source Storage Element Address = "
    & eldata(dataoffset + 10) & eldata(dataoffset + 11)

If (pvol = 1) Then
    x.Write " - Primary Volume Tag Information = "
    For i = 12 To 47
        x.Write eldata(dataoffset + i)
    Next i
End If
bytecount = bytecount - ellength
dataoffset = dataoffset + ellength
Wend

x.WriteLine ""
End If

x.WriteLine "-----"

retval = SCSIReadElementStatus(ha,target,lun,1,0,255,1024,eldata)
If retval <> 0 Then
    MsgBox "Read Element Status failed - Status = " & retval
    Stop
Else
    dataoffset = 8
    bytecount = eldata(5) + eldata(6) + eldata(7)
    ellength = eldata(dataoffset + 2) + eldata(dataoffset + 3)

    x.WriteLine ""

    x.WriteLine ""
    x.Write "Element type = " & eldata(8)
    If (eldata(8) = 1) Then
        x.WriteLine " - Tape Pickers"
    If (eldata(8) = 2) Then
        x.WriteLine " - Cartridge Magazine Slots"
    If (eldata(8) = 3) Then
        x.WriteLine " - Tape Drives"

    x.WriteLine "First Element Address = " & eldata(0) & eldata(1)
    x.WriteLine "Number of Elements = " & eldata(2) & eldata(3)
    x.WriteLine "Number of bytes of element status = " & bytecount
    x.WriteLine "Element Descriptor Length = " & ellength
    x.WriteLine ""

    dataoffset = dataoffset + 8
    bytecount = bytecount - 8

    While (bytecount > 0)
        x.WriteLine ""
        x.WriteLine ""
        x.WriteLine "Element Address = " & eldata(dataoffset)
            & eldata(dataoffset+1)
        If (eldata(dataoffset+2) And 1) Then
            x.WriteLine " - Full, "
        Else
            x.WriteLine " - Empty, "
        End If

        If (eldata(dataoffset+2) And 4) Then
            x.Write " - Except = 1, "
        Else
            x.Write " - Except = 0, "
        End If

        If (eldata(dataoffset+2) And 8) Then
            x.Write "Access = 1, "
        Else
            x.Write "Access = 0, "
        End If

        x.WriteLine ""
    End While

```

```

x.Write " - Additional Sense Code = " & eldata(dataoffset + 4)
x.WriteLine ", Additional Sense Code Qualifier = "
    & eldata(dataoffset + 5)

If (eldata(dataoffset + 6) And 128) Then
    x.Write " - SValid = 1 "
Else
    x.Write " - SValid = 0, "
End If

If (eldata(dataoffset + 6) And 64) Then
    x.Write "Invert = 1 "
Else
    x.Write "Invert = 0"
End If

x.WriteLine ""
x.WriteLine " - Source Storage Element Address = "
    & eldata(dataoffset + 10) & eldata(dataoffset + 11)

If (pvol = 1) Then
    x.Write " - Primary Volume Tag Information = "
    For i = 12 To 47
        x.Write eldata(dataoffset + i)
    Next i
End If

bytecount = bytecount - ellength
dataoffset = dataoffset + ellength
Wend

x.WriteLine ""
End If

x.WriteLine "-----"

retval = SCSIReadElementStatus(ha,target,lun,4,0,255,1024,eldata)
If retval <> 0 Then
    MsgBox "Read Element Status failed - Status = " & retval
    Stop
Else
    dataoffset = 8
    bytecount = eldata(5) + eldata(6) + eldata(7)
    ellength = eldata(dataoffset + 2) + eldata(dataoffset + 3)

    x.WriteLine ""

    x.WriteLine ""
    x.Write "Element type = " & eldata(8)
    If (eldata(8) = 1) Then
        x.WriteLine " - Tape Pickers"
    If (eldata(8) = 2) Then
        x.WriteLine " - Cartridge Magazine Slots"
    If (eldata(8) = 4) Then
        x.WriteLine " - Tape Drives"

    x.WriteLine "First Element Address = " & eldata(0) & eldata(1)
    x.WriteLine "Number of Elements = " & eldata(2) & eldata(3)
    x.WriteLine "Number of bytes of element status = " & bytecount
    x.WriteLine "Element Descriptor Length = " & ellength
    x.WriteLine ""

    dataoffset = dataoffset + 8
    bytecount = bytecount - 8

    While (bytecount > 0)
        x.Write "Element Address = " & eldata(dataoffset) & eldata(dataoffset+1)
        x.WriteLine ""

        If (eldata(dataoffset+2) And 1) Then
            x.WriteLine " - Full, "
        Else
            x.WriteLine " - Empty, "
        End If
    End While

```

```
If (eldata(dataoffset+2) And 4) Then
  x.Write " - Except = 1, "
Else
  x.Write " - Except = 0, "
End If

If (eldata(dataoffset+2) And 8) Then
  x.Write "Access = 1, "
Else
  x.Write "Access = 0, "
End If

x.WriteLine ""

x.Write " - Additional Sense Code = " & eldata(dataoffset + 4)
x.WriteLine ", Additional Sense Code Qualifier = "
  & eldata(dataoffset + 5)

If (eldata(dataoffset + 6) And 128) Then
  x.Write " - Not Bus = 1,"
Else
  x.Write " - Not Bus = 0,"
End If

If (eldata(dataoffset + 6) And 32) Then
  x.Write " ID Valid = 1,"
Else
  x.Write " ID Valid = 0,"
End If

If (eldata(dataoffset + 6) And 16) Then
  x.Write " LU Valid = 1,"
Else
  x.Write " LU Valid = 0,"
End If

x.WriteLine "Logical Unit Number = " & (eldata(dataoffset + 6) And 7)
x.WriteLine " - SCSI Bus Address = " & eldata(dataoffset + 7)

If (eldata(dataoffset + 6) And 128) Then
  x.Write " - SValid = 1 "
Else
  x.Write " - SValid = 0, "
End If

If (eldata(dataoffset + 6) And 64) Then
  x.Write "Invert = 1 "
Else
  x.Write "Invert = 0"
End If

x.Writeline ""
x.WriteLine " - Source Storage Element Address = "
  & eldata(dataoffset + 10) & eldata(dataoffset + 11)

If (pvol = 1) Then
  x.Write " - Primary Volume Tag Information = "
  For i = 12 To 47
    x.Write eldata(dataoffset + i)
  Next i
End If

bytecount = bytecount - ellength
dataoffset = dataoffset + ellength
Wend

x.WriteLine ""
End If

x.WriteLine "-----"
x.Write "The End"
```

## SCSIResetHBA

(ByVal hba as Integer) as Integer

This function causes a bus reset and a bus rescan of the specified HBA

Returns: returns 0 on success, non-zero on failure

Example:

```
Dim retval As Integer

Sub main
  retval = SCSIResetHBA(0)
End Sub
```

---

## SCSIRollPattern

(ByVal buffnum as Integer, ByVal start as Long, ByVal number as Long) as Integer

Writes *number* bytes of data into the buffer specified by *buffnum*. The data written begins with the 4-byte long value specified by *start*. This value is incremented by one each time it is written into the buffer.

Returns: returns 0 on success, non-zero on failure

Example:

```
Dim retval As Integer

Sub main
  retval = SCSIRollPattern(0,0,2048)
End Sub
```

---

## SCSISEAGATELTO\_FWDL

(ByVal ha as integer, ByVal target as integer, ByVal lun as integer, ByVal FileName as String) as Integer

Download firmware to a Seagate LTO drive

Returns: 0 on success, -2 if firmware file not found, -1 if download fails

## SCSISonyAIT\_FWDL

(ByVal ha as integer, ByVal target as integer, ByVal lun as integer, ByVal FileName as String) as Integer

Download firmware to a Sony AIT drive

Returns: 0 on success, -2 if firmware file not found, -1 if download fails

---

## SCSISearchBuffer

(ByVal buffer as Integer, searchdata() as Byte, ByVal searchsize as Integer, ByVal searchlength as Long ) as Integer

Searches buffer # *buffer* for the first occurrence of *searchdata*. *Searchlength* specifies how much of the buffer to search (-1 searches the entire buffer), *searchsize* specifies the number of significant bytes in the pattern *searchdata*.

Returns: returns byte count of the first byte of buffer that matches pattern on success, -1 on failure

Example:

See the example code for the SCSIUserDefinedCDB function

---

## SCSISDLT\_FWDL

(ByVal ha as integer, ByVal target as integer, ByVal lun as integer, ByVal FileName as String) as Integer

Download firmware to a Quantum SuperDLT drive

Returns: 0 on success, -2 if firmware file not found, -1 if download fails

## SCSISegmented\_FWDL

(ByVal ha as integer, ByVal target as integer, ByVal lun as integer, ByVal FileName as String) as Integer

Download firmware to a device

Returns: 0 on success, -2 if firmware file not found, -1 if download fails



---

## SCSISetBuffer Mode

(ByVal ha As Integer, ByVal target As Integer, ByVal lun As Integer, buffermode as Integer) as Integer

Sets the buffer mode of the selected tape drive

Returns: zero on success, non-zero on failure

Example:

Dim target As Integer

Dim retval As Integer

Dim buffmode As Integer

```
retval = SCSIGetDllVersion()  
MsgBox "version = " & retval
```

```
buffmode = SCSIGetBufferMode(2,4,0)  
MsgBox "buffer mode = " & buffmode
```

```
retval = SCSISetBufferMode(2,4,0,0)
```

```
buffmode = SCSIGetBufferMode(2,4,0)  
MsgBox "buffer mode = " & buffmode
```

---

## SCSISetTimeout

(ByVal seconds as Integer) as Integer

Sets the CDB timeout value in seconds. This timeout value will remain in effect until the next SCSISetTimeout function is called.

Specifying a value (seconds) greater than zero will set the timeout to that value in seconds. Specifying a value of zero will set the default timeout to 30 seconds. Specifying a value of less than zero will set the timeout to infinite.

Returns: 1 on success, -1 on failure

## SCSITapeBlockSize

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, tbs as Integer) as Integer

Sets the blocksize of drive specified by *ha/target/lun* to *tbs*.

Returns: 1 on success, -1 on failure

Example:

```
Dim tapeblocksize As Long
Dim retval As Integer
Dim ha As Integer
Dim target As Integer
Dim lun As Integer

ha = 1
target = 5
lun = 0

retval = SCSIGetTapeCapacity(ha, target, lun, tapeblocksize)

MsgBox "TBS = " & tapeblocksize
```

---

## SCSITapeRewind

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal immediate as Integer) as Integer

Rewinds the drive specified by *ha/target/lun*. Returns after rewind completion if *immediate* = 0, or upon acceptance of cdb by device if *immediate* = 1.

Returns: 1 on success, -1 on failure

Example:

```
Dim tapeblocksize As Long
Dim retval As Integer
Dim ha As Integer
Dim target As Integer
Dim lun As Integer

ha = 1
target = 5
lun = 0

retval = SCSITapeRewind(ha, target, lun, 0)
```

---

## SCSITapeUnload

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal immediate as Integer) as Integer

Unloads the drive specified by *ha/target/lun*. Returns after unload completion if *immediate = 0*, or upon acceptance of *cdb* by device if *immediate = 1*.

Returns: 1 on success, -1 on failure

Example:

```
Dim tapeblocksize As Long
Dim retval As Integer
Dim ha As Integer
Dim target As Integer
Dim lun As Integer

ha = 1
target = 5
lun = 0

retval = SCsITapeUnload(ha,target,lun,0)
```

---

## SCSITapeWFM

(ByVal *ha* as Integer, ByVal *target* as Integer, ByVal *lun* as Integer) as Integer

Writes a FILE MARK to drive specified by *ha/target/lun*.

Returns: 1 on success, -1 on failure

Example:

```
Dim tapeblocksize As Long
Dim retval As Integer
Dim ha As Integer
Dim target As Integer
Dim lun As Integer

ha = 1
target = 5
lun = 0

retval = SCsITapeWFM(ha,target,lun)
```

---

## SCSITapeReadF

(ByVal *ha* as Integer, ByVal *target* as Integer, ByVal *lun* as Integer, ByVal *count* as Integer, ByVal *buffer* as Integer) as Integer

Reads *count* fixed blocks from the tape specified by *ha/target/lun* into buffer # *buffer*.

Returns: 1 on success, -1 on failure

### Example:

```
Dim tapeblocksize As Long
Dim retval As Integer
Dim Loopy As Integer
Dim ha As Integer
Dim target As Integer
Dim lun As Integer
Dim NL As String

NL = Chr(10)

ha = 1
target = 5
lun = 0

retval = SCSI_TapeRewind(ha,target,lun,0)
retval = SCSI_GetTapeCapacity(ha,target,lun,tapeblocksize)

If (tapeblocksize <> 32768) Then
    MsgBox "TBS = " & tapeblocksize & " changing to 32768"
    retval = SCSI_SetTapeBlockSize(ha,target,lun,32768)
    retval = SCSI_TapeWFM(ha,target,lun)
    retval = SCSI_TapeRewind(ha,target,lun,0)
    retval = SCSI_GetTapeCapacity(ha,target,lun,tapeblocksize)
End If
    MsgBox "TBS = " & tapeblocksize

retval = SCSI_FillPattern(0,2)

For Loopy = 0 To 100
    retval = SCSI_TapeWriteF(ha,target,lun,1,0)
Next
MsgBox "Wrote 100 Blocks"

retval = SCSI_TapeWFM(ha,target,lun)
MsgBox "WFM"

retval = SCSI_TapeRewind(ha,target,lun,0)
MsgBox "Rewound"

For Loopy = 0 To 100
    retval = SCSI_TapeReadF(ha,target,lun,1,0)
Next
MsgBox "Read 100 blocks"

retval = SCSI_TapeFSR(ha,target,lun)
MsgBox "FSR"

retval = SCSI_TapeFSR(ha,target,lun)
MsgBox "FSR"

retval = SCSI_TapeFSF(ha,target,lun)
MsgBox "FSF"

retval = SCSI_TapeRewind(ha,target,lun,0)
MsgBox "Rewind"

retval = SCSI_TapeSpaceEOD(ha,target,lun)
MsgBox "Space EOD"
```

## SCSITapeWriteF

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal count as Integer, ByVal buffer as Integer) as Integer

Writes *count* fixed blocks to the tape specified by *ha/target/lun* from buffer # *buffer*.

Returns: 1 on success, -1 on failure

Example:

See the example for the SCSITapeReadF function

---

## SCSITapeReadV

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal buffer as Integer) as Integer

Reads one variable block from the tape specified by *ha/target/lun* into buffer # *buffer*.

Returns: 1 on success, -1 on failure

---

## SCSITapeWriteV

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, ByVal count as Long, ByVal buffer as Integer) as Integer

Writes one variable block *count* bytes to the tape specified by *ha/target/lun* from buffer # *buffer*.

Returns: 1 on success, -1 on failure

---

## SCSITapeFSF

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer) as Integer

Spaces forward on file mark on the tape specified by *ha/target/lun*

Returns: 1 on success, -1 on failure

Example:

See the example for the SCSTapeReadF funtion

---

## SCSITapeFSR

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer) as Integer

Spaces reverse on file mark on the tape specified by *ha/target/lun*

Returns: 1 on success, -1 on failure

Example:

See the example for the SCSTapeReadF funtion

---

## SCSITapeSpaceEOD

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer) as Integer

Spaces forward to END OF DATA on the tape specified by *ha/target/lun*

Returns: 1 on success, -1 on failure

Example:

See the example for the SCSTapeReadF funtion

---

## SCSITargetCount

(ByVal ha as Integer) as Integer

Returns: number of targets supported on host adapter *ha*

---

## SCSITUR

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer) as Integer

Issues a TEST UNIT READY command to the drive specified by *ha/target/lun*.

Returns: returns 0 on success, non-zero on failure

Example:

```
Dim retval as integer
Dim ha as integer
Dim target as integer
Dim lun as integer

retval = SCSITUR(ha,target,lun)
If retval <> 0 Then
    MsgBox "This device is off line"
End If
```

---

## SCSIUserCdb

(ByVal ha as Integer, ByVal target as Integer, ByVal lun as Integer, cdb() as Byte, ByVal cdblength as Integer, ByVal datadir as Integer, ByVal datalength as Long, ByVal buffer as Integer) as Integer

Issues the SCSI CDB specified in byte array *cdb* to the device specified in *ha/target/lun*. The length of the CDB is specified in *cdblength*, data direction is specified by *datadir*(0=out from host adapter, 1 = in to host adapter), length of data transferred is specified by *datalength*, and buffer # is specified by *buffer*.

Returns: 1 on success, -1 on failure

Example:

```
Dim userdata(1024) As Integer
```

```

Dim usercdb(12) As Byte
Dim target As Integer
Dim ha As Integer
Dim lun As Integer
Dim retval As Integer
Dim searchdata(10) As Byte

searchdata(0) = Asc("I")
searchdata(1) = Asc("O")
searchdata(2) = Asc("M")
searchdata(3) = Asc("E")

usercdb(0) = &H12
usercdb(1) = 0
usercdb(2) = 0
usercdb(3) = 0
usercdb(4) = &Hff
usercdb(5) = 0

target = 6
ha = 1
lun = 0

retval = SCSIUserCdb(ha,target,lun,usercdb(),6,1,&Hff,0)

If retval = 1 Then
    retval = SCSIGetBuffer(0,32,userdata)
    For i = 0 To 32
        results = results & Format(Hex(userdata(i)),"@@" ) & " "
    Next
    MsgBox results
Else
    MsgBox "command failed"
End If

retval = SCSISearchBuffer(0,searchdata(),4,32)

If (retval >=0) Then
    MsgBox "Match found"
    MsgBox "retval (search) = " & retval
Else
    MsgBox "Match not found"
End If

```

---

## SCSIViewSense

(sensedata() as Byte) as Integer

Returns the latest REQUEST SENSE data in byte array *sensedata*.

Returns: 1 on success, -1 on failure

Example:

```

Dim retval as integer

Dim sensedata(32) As Integer

retval = SCSIViewSense(sensedata)
MsgBox "Key = " & Hex(sensedata(2))& " " & "Code = " & Hex(sensedata(12))
    & " " & "ASQ = " & Hex(sensedata(13))

```



---

## SCSIXor

(ByVal number1 As int, ByVal number 2 As int) As Integer

Logically XOR's number1 with number2

Returns: results of XOR operation

Example:

```
Dim retval as integer
retval = SCSIXor(&H82, &H7f)
` XOR hex 82 with hex 7f - returns &Hfd
```

## INDEX

---

**A**

Abs Function · 43  
Accessing an object · 31  
    CreateObject Function · 31  
    GetObject Function · 31  
Activate · 32  
AppActivate Statement · 44  
Application · 32  
Arrays · 17  
Asc Function · 45  
Atn Function · 45

---

**B**

Beep Statement · 45

---

**C**

Call Statement · 46  
Calling Procedures in DLLs · 14  
CBool Function · 47  
CDate Function · 47  
CDBl Function · 48  
ChDir · 38, 42, 48  
ChDrive · 38  
ChDrive Statement · 49  
Check Boxes · 22  
CheckBox · 49  
Choose Function · 50  
Chr, Function · 50  
Cint Function · 51  
Class · 34  
CLng Function · 51  
Close Statement · 52  
Comments · 6  
Const Statement · 53  
Control Structures · 6, 10  
Cos · 54  
CreateObject · 54  
CSng Function · 55  
CStr Function · 56  
CurDir Function · 56  
CVar Function · 57

---

**D**

Data Types · 39

Date Function · 57  
DateSerial · 58  
DateValue · 59  
Day Function · 59  
Declare Statement · 60  
Dialog Dialog Function · 61  
Dialog Support · 20  
Dim Statement · 63  
Dir\$ Function · 63  
DlgControlId Function · 28  
DlgEnable Statement · 64  
DlgFocus Statement, DlgFocus() Function · 28  
DlgListBoxArray, DlgListBoxArray() · 29  
DlgSetPicture · 29  
DlgText Statement · 65  
DlgValue, DlgValue() · 29  
DlgVisible Statement · 65  
Do...Loop Statement · 66

---

**E**

End Statement · 67  
Eof · 68  
Erase · 68  
Exit Statement · 69  
Exp · 39, 69

---

**F**

File Input/Output · 15  
FileCopy · 38, 70  
FileLen Function · 70  
Fix Function · 71  
For...Next Statement · 71  
Format Statement · 72  
FreeFile Function · 82  
Function Statement · 82

---

**G**

Get Object Function · 83  
Global Statement · 84  
GoTo Statement · 84

---

**H**

Hex, · 85  
Hour Function · 86

---

**I**

If...Then...Else Statement · 11, 87  
 Input, Function · 88  
 InputBox Function · 88  
 Installation · 146  
 InStr · 89  
 Int Function · 90  
 IsArray Function · 90  
 IsDate · 90  
 IsEmpty · 91  
 IsNull · 91  
 IsNumeric · 92  
 IsObject Function · 92

---

**K**

Kill Statement · 93

---

**L**

LBound Function · 93  
 LCase, Function · 94  
 Left · 95  
 Len · 95  
 Let Statement · 96  
 Line Input # Statement · 96  
 List Boxes, Combo Boxes and Drop-down List Boxes · 21  
 LOF · 97  
 Log · 97

---

**M**

Making Applications Work Together · 35  
 Methods · 32  
 Mid Function · 98  
 Minute Function · 98  
 Mkdir · 99  
 Month Function · 100  
 MsgBox · 100

---

**N**

Name Statement · 103  
 Now Function · 103  
 Numbers · 7

---

**O**

Oct Function · 103  
 OK and Cancel Buttons · 20  
 OKButton · 104  
 OLE Automation · 30, 34, 36

What is OLE Automation? · 30, 34, 36  
 OLE Fundamentals · 34  
 OLE Object · 34  
 On Error · 105  
 Open Statement · 107  
 Operators · 40  
 Option Base Statement · 109  
 Option Buttons and Group Boxes · 24  
 Other Data Types · 9  
   Declaration of Variables · 9  
   Scope of Variables · 9

---

**P**

Print # Statement · 110  
 Print Method · 110  
 Properties · 32

---

**R**

Rem Statement · 112, 113  
 Right, Function · 113  
 Rmdir Statement · 114  
 Rnd · 114

---

**S**

ScriptWriter Pro Scripting Language Elements · 6  
 SCSI Functions · 115  
   SCSICompareBuffers · 115  
   SCSIDiskRead · 115  
   SCSIDiskSeekTest · 126  
   SCSIDiskSeqReadTest · 127  
   SCSIDiskSeqWriteReadTest · 128  
   SCSIDiskSeqWriteTest · 128  
   SCSIDiskStartStop · 116  
   SCSIDiskUnload · 116  
   SCSIDiskVerify · 116  
   SCSIDiskVerifyTest · 128  
   SCSIDiskWrite · 116  
   SCSIDiskWriteVerifyTest · 129  
   SCSIFillBlockNum · 117  
   SCSIFillBuffer · 117  
   SCSIFillPattern · 117  
   SCSIGetBuffer · 118  
   SCSIGetDeviceType · 118  
   SCSIGetDevID · 126  
   SCSIGetProduct · 118  
   SCSIGetTapeCapacity · 118  
   SCSIGetVendor · 119  
   SCSIGetVersion · 119  
   SCSIHostAdapterCount · 119  
   SCSIInquiry · 119  
   SCSILoadBuffer · 120  
   SCSILogSense · 120  
   SCSIModeSelect · 120  
   SCSIModeSelectFull · 121

SCSIModeSense · 120  
 SCSIModeSenseFull · 121  
 SCSIReadCapacity · 121  
 SCSISearchBuffer · 122  
 SCSTapeBlockSize · 122  
 SCSTapeFSF · 124  
 SCSTapeFSR · 124  
 SCSTapeReadF · 123  
 SCSTapeReadV · 124  
 SCSTapeRewind · 122  
 SCSTapeSpaceEOD · 125  
 SCSTapeUnload · 122  
 SCSTapeWFM · 123  
 SCSTapeWriteF · 123  
 SCSTapeWriteV · 124  
 SCSITUR · 125  
 SCSIUserCdb · 125  
 SCSIViewSense · 125  
 Second Function · 129  
 Seek Function · 130  
 Seek Statement · 131  
 Select Case Statement · 131  
 SendKeys · 132  
 Set Statement · 133  
 Shell · 36, 134  
 Sin · 134  
 Space · 135  
 Sqr · 135  
 Statements and Functions Used in Dialog Functions · 27  
 Static · 136  
 Stop · 137  
 Str Function · 137  
 StrComp Function · 138  
 String, Function · 138  
 Sub Statement · 139  
 Subroutines and Functions · 12  
     Naming conventions · 12

---

## *T*

Tan Function · 139  
 Text · 140  
 Text Boxes and Text · 22

TextBox · 140  
 The Dialog Function · 25  
 The Dialog Function Syntax · 25  
 Time, Function · 141  
 Timer Event · 141  
 TimeSerial - Function · 142  
 TimeValue - Function · 142  
 Trim, LTrim Rtrim Functions · 143  
 Type Statement · 143  
 Type/Functions/Statements · 38

---

## *U*

UBound Function · 145  
 UCase, Function · 145  
 User Defined Types · 19, 144

---

## *V*

Val · 146  
 Variable and Constant Names · 7  
 Variable Types · 7  
     Variants and Concatenation · 7  
 Variable Types  
     Variant · 7  
 VarType · 146

---

## *W*

Weekday Function · 147  
 What is an OLE Object? · 32  
 While...Wend Statement · 147  
 With Statement · 148  
 Write # - Statement · 149

---

## *Y*

Year · 150