

---

ASPEN

# **PowerScript V15**

## **Reference Manual**



**Advanced Systems for Power Engineering, Inc.**

## NOTICE

*PowerScript*™ is a proprietary scripting language of Advanced Systems for Power Engineering, Inc. (ASPEN).

The information in this document is subject to change without notice. ASPEN assumes no responsibility for any errors that may appear in this document.

Copyright © 1988-2019 Advanced Systems for Power Engineering, Inc. All rights reserved.

## HOW TO ORDER MORE MANUALS

This User's Manual may be duplicated by the Licensee for its own use. You can order a new copy by writing to the address below. Please refer to document OL-PS-2019.

## HOW TO REACH ASPEN

Mailing address: ASPEN  
49 N. San Mateo Drive  
San Mateo, CA 94401  
U.S.A.

Telephone: (650)347-3997  
Fax: (650)347-0233  
eMail (English): support@aspeninc.com  
eMail (Spanish/Portuguese): soporte@aspeninc.com  
Web Site: www.aspeninc.com

Our office hours are from 8:30 a.m. to 5:30 p.m. Pacific time (GMT-8 in the winter, GMT-7 in the summer), Monday through Friday.

---

*ASPEN OneLiner*™, *ASPEN Breaker Rating Module*™, *ASPEN Power Flow*™, *ASPEN DistriView*™, *PowerScript*™, *ASPEN Relay Database*™, *ASPEN Line Constants Program*™, and *ASPEN Line Database*™ are trademarks of Advanced Systems for Power Engineering, Inc.

# Contents

<b>SECTION 1</b>	<b>INTRODUCTION .....</b>	<b>9</b>
1.1	SCRIPTING LANGUAGE.....	9
1.2	LANGUAGE SPECIFICATION.....	10
<b>SECTION 2</b>	<b>TUTORIAL.....</b>	<b>11</b>
2.1	INTRODUCTION .....	11
2.2	EDITING A SCRIPT FILE .....	11
2.3	DEBUGGING A SCRIPT .....	13
2.4	EDITING DIALOG BOXES.....	15
2.5	RUNNING SCRIPTS AS PROGRAM COMMANDS.....	19
2.6	SAMPLE SCRIPTS.....	20
<b>SECTION 3</b>	<b>OVERVIEW.....</b>	<b>21</b>
3.1	OBJECT HANDLES .....	21
3.2	EQUIPMENT TYPE CODE .....	22
3.3	NETWORK DATA ACCESS .....	22
3.4	SHORT CIRCUIT SOLUTION .....	33
3.5	POWER FLOW SOLUTION.....	33
3.6	RESERVED NAMES AND KEYWORDS .....	34
<b>SECTION 4</b>	<b>FUNCTION REFERENCE.....</b>	<b>35</b>
4.1	POWERSCRIPT FUNCTIONS.....	35
	Function AppExit.....	37
	Function BusPicker .....	38
	Function BoundaryEquivalent.....	39
	Function ComputeRelayTime .....	40
	Function DoBreakerRating .....	41
	Function DoFault .....	42
	Function DoArcFlash.....	44
	Function DoSteppedEvent .....	46
	Function DoVS .....	48
	Function DoVSEx.....	49
	Function DoPF, DoPF10 and DoPF11.....	50
	Function ErrorString.....	52
	Function ExportNetwork .....	53
	Function ExportNetworkPSSE.....	54
	Function EquipmentType.....	55
	Function FaultDescription .....	56
	Function FaultSelector .....	57
	Function FileOpenDialog, FileSaveDialog.....	58
	Function FindBusByName.....	59
	Function FindEquipmentByTag.....	60
	Function FolderSelectDialog .....	61
	Function FullBranchName .....	62
	Function FullBusName.....	63
	Function FullRelayName.....	64
	Function GetAreaName, GetZoneName .....	65
	Function GetBusEquipment.....	66
	Function GetData.....	67
	Function GetEquipment.....	68

Function GetFlow.....	69
Function GetObjJournalRecord.....	70
Function GetObjMemo.....	71
Function GetObjTags.....	72
Function GetOlrFileName.....	73
Function GetPFCCurrent.....	74
Function GetPFVoltage.....	75
Function GetProgramVersion.....	76
Function GetPSCVoltage.....	77
Function GetRelay.....	78
Function GetRelayTime.....	79
Function GetLogicScheme.....	80
Function GetSCCurrent.....	81
Function GetSCVoltage.....	83
Function GetSteppedEvent.....	84
Function GetVSVoltage.....	85
Function GetWindowsEnvironmentVariable.....	87
Function LoadDataFile.....	88
Function LocateILObj.....	89
Function MakeOutageList.....	90
Function NextBusByName.....	91
Function NextBusByNumber.....	92
Function PickFault.....	93
Function PostData.....	94
Function ProgressDialog.....	95
Function PrintTTY.....	96
Function PrintObjILPF.....	97
Function ReadChangeFile.....	98
Function RunILPFCommand: ARCFLASHCALCULATOR.....	99
Function RunILPFCommand: ARCFLASHCALCULATOR2018.....	101
Function RunILPFCommand: BUSFAULTSUMMARY.....	103
Function RunILPFCommand: CHECKPRIBACKCOORD.....	104
Function RunILPFCommand: CHECKRELAYOPERATIONSEA.....	106
Function RunILPFCommand: CHECKRELAYOPERATIONPRC023.....	107
Function RunILPFCommand: CHECKRELAYOPERATIONPRC026.....	108
Function RunILPFCommand: CHECKRELAYSETTINGS.....	109
Function RunILPFCommand: EXPORTNETWORK.....	110
Function RunILPFCommand: EXPORTRELAY.....	111
Function RunILPFCommand: INSERTTAPBUS.....	112
Function RunILPFCommand: SETGENREFANGLE.....	115
Function RunILPFCommand: SAVEDATAFILE.....	116
Function SaveDataFile.....	117
Function SetData.....	120
Function SetObjMemo.....	121
Function SetObjTags.....	122
Function ShowFault.....	123
<b>APPENDIX A: CYPRESS ENABLE SCRIPTING LANGUAGE ELEMENTS.....</b>	<b>125</b>
COMMENTS.....	125
STATEMENTS.....	125
LINE CONTINUATION CHARACTER.....	126
NUMBERS.....	126
VARIABLE AND CONSTANT NAMES.....	126
VARIABLE TYPES.....	126
OTHER DATA TYPES.....	127
SCOPE OF VARIABLES.....	128

DECLARATION OF VARIABLES .....	128
CONTROL STRUCTURES .....	128
SUBROUTINES AND FUNCTIONS .....	130
BYREF AND BYVAL .....	130
CALLING PROCEDURES IN DLLS .....	132
FILE INPUT/OUTPUT .....	133
ARRAYS.....	134
USER DEFINED TYPES .....	136
DIALOG SUPPORT .....	137
Control.....	143
SuppValue passed.....	143
ListBox, DropListBox, or ComboBox .....	143
Number of the item selected where 0 (zero) is the first item in the list box, 1 is the second item, and so on. ....	143
CheckBox .....	143
1 if selected, 0 (zero) if cleared.....	143
OptionButton .....	143
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.....	143
TextBox .....	143
Number of characters in the text box. ....	143
ComboBox.....	143
If Action is 3, number of characters in the combo box. ....	143
CommandButton.....	143
A value identifying the button chosen. This value is not often used, since the same information is available from the ControlID\$ value. ....	143
STATEMENTS AND FUNCTIONS USED IN DIALOG FUNCTIONS .....	143
OLE AUTOMATION .....	146
ACCESSING AN OBJECT .....	147
WHAT IS AN OLE OBJECT? .....	147
OLE FUNDAMENTALS .....	149
OLE AUTOMATION AND MICROSOFT WORD EXAMPLE: .....	150
MAKING APPLICATIONS WORK TOGETHER .....	150
THE REGISTRATION DATABASE .....	151
<b>APPENDIX B: CYPRESS ENABLE SCRIPTING LANGUAGE OVERVIEW.....</b>	<b>153</b>
QUICK REFERENCE OF THE FUNCTIONS AND STATEMENTS AVAILABLE .....	153
<b>APPENDIX C: CYPRESS ENABLE LANGUAGE REFERENCE A-Z.....</b>	<b>157</b>
ABS FUNCTION .....	157
APPACTIVATE STATEMENT .....	158
ASC FUNCTION .....	158
ATN FUNCTION.....	159
BEEP STATEMENT.....	159
CALL STATEMENT .....	160
CBOOL FUNCTION .....	160
CDATE FUNCTION .....	161
CDBL FUNCTION .....	161
CHDIR STATEMENT .....	162
CHDRIVE STATEMENT.....	162
CHECKBOX .....	163
CHOOSE FUNCTION .....	164
CHR FUNCTION.....	164
CINT FUNCTION .....	165
CLNG FUNCTION .....	165
CLOSE STATEMENT .....	165
CONST STATEMENT .....	166
COS FUNCTION .....	167

CREATEOBJECT FUNCTION.....	168
CSNG FUNCTION .....	169
CSTR FUNCTION.....	169
CURDIR FUNCTION.....	170
CVAR FUNCTION.....	170
DATE FUNCTION.....	171
DATESERIAL FUNCTION .....	172
DATEVALUE FUNCTION .....	172
DAY FUNCTION .....	173
DECLARE STATEMENT.....	173
DIALOG, DIALOG FUNCTION .....	174
DIM STATEMENT .....	176
DIR FUNCTION.....	176
DLGENABLE STATEMENT.....	177
DLGTEXT STATEMENT .....	178
DLGVISIBLE STATEMENT .....	179
DO...LOOP STATEMENT .....	179
END STATEMENT .....	180
EOF FUNCTION .....	181
ERASE STATEMENT .....	181
EXIT STATEMENT .....	182
EXP .....	182
FILECOPY FUNCTION.....	183
FILELEN FUNCTION .....	183
FIX FUNCTION .....	183
FOR EACH ... NEXT STATEMENT .....	184
FOR...NEXT STATEMENT .....	184
FORMAT FUNCTION .....	185
FREEFILE FUNCTION .....	195
FUNCTION STATEMENT .....	195
GET STATEMENT .....	196
GET OBJECT FUNCTION .....	197
GLOBAL STATEMENT .....	197
GoTo STATEMENT.....	198
HEX.....	198
HOUR FUNCTION .....	199
HTMLDIALOG .....	200
IF...THEN...ELSE STATEMENT.....	200
INPUT # STATEMENT .....	201
INPUT FUNCTION .....	202
INPUTBOX FUNCTION .....	202
INSTR .....	203
INT FUNCTION .....	204
ISARRAY FUNCTION .....	204
ISDATE.....	204
ISEMPTY.....	205
ISNULL .....	205
ISNUMERIC.....	205
ISOBJECT FUNCTION .....	206
KILL STATEMENT .....	207
LBOUND FUNCTION .....	207
LCASE, FUNCTION.....	208
LEFT .....	208
LEN .....	209
LET STATEMENT.....	209
LINE INPUT # STATEMENT .....	210

LOF .....	210
LOG .....	211
MID FUNCTION .....	211
MINUTE FUNCTION.....	212
MkDir .....	213
MONTH FUNCTION .....	213
MsgBox FUNCTION MsgBox STATEMENT .....	214
NAME STATEMENT .....	216
NOW FUNCTION.....	216
OCT FUNCTION .....	217
OKBUTTON .....	217
ON ERROR .....	218
OPEN STATEMENT .....	220
OPTION BASE STATEMENT .....	222
OPTION EXPLICIT STATEMENT .....	223
PRINT METHOD .....	223
PRINT # STATEMENT .....	224
RANDOMIZE STATEMENT .....	226
ReDim STATEMENT.....	226
REM STATEMENT .....	227
RIGHT FUNCTION .....	227
Rmdir STATEMENT .....	228
RND FUNCTION .....	228
SECOND FUNCTION.....	229
SEEK FUNCTION .....	230
SEEK STATEMENT .....	230
SELECT CASE STATEMENT .....	231
SENDKEYS FUNCTION .....	232
SET STATEMENT.....	232
SHELL FUNCTION .....	233
SIN FUNCTION .....	234
SPACE FUNCTION .....	234
SQR FUNCTION .....	234
STATIC STATEMENT .....	235
STOP STATEMENT.....	236
STR FUNCTION .....	236
STRCOMP FUNCTION.....	237
STRING FUNCTION .....	237
SUB STATEMENT .....	238
TAN FUNCTION.....	238
TEXT STATEMENT .....	239
TEXTBOX STATEMENT .....	239
TIME FUNCTION .....	240
TIMER EVENT .....	240
TIME SERIAL - FUNCTION.....	241
TIME VALUE - FUNCTION .....	241
TRIM, LTrim, RTrim FUNCTIONS .....	242
TYPE STATEMENT .....	242
UBOUND FUNCTION .....	244
UCASE FUNCTION .....	244
VAL.....	245
VARType .....	245
WEEKDAY FUNCTION .....	246
WHILE... WEND STATEMENT.....	246
WITH STATEMENT .....	247
WRITE # - STATEMENT .....	248

YEAR FUNCTION .....	248
<b>INDEX .....</b>	<b>251</b>



---

## 1.1 Scripting Language

*ASPEN PowerScript* is a scripting tool embedded in *ASPEN OneLiner* and *ASPEN Power Flow Program* to enable users to “drive” the programs with instructions written in BASIC. You can use *PowerScript* to examine and modify all the system and network parameters and utilize *OneLiner* and *Power Flow* as solution engines.

*PowerScript* has many applications. In the simplest form, a script can be just a few lines of instructions designed to automate a frequently performed task. For instance you can write a script in *OneLiner* to find the highest fault current at a bus with the adjacent branches outaged one at a time. Simple scripts of this type are commonly referred to as “macros.”

With a slightly more complicated script, you can use *PowerScript* to create customized reports of network parameters and solution variables. As an example, you can create your own report in *Power Flow* to list all the PV buses whose output is pegged at the maximum or minimum value.

A script can be a full-fledged computer program with its own decision-making capabilities and computational logic. A *OneLiner* user, for example, can write a script to locate a fault by simulating a number of faults along a line and compare the voltage and current solutions to the values recorded to find the best match. A *Power Flow* user can write a script to perform outage studies.

*PowerScript* has a wide range of input/output capabilities. A script can open disk files for reading in or writing out any kind of text or binary data. It can also interact with the user through dialog boxes and keyboard handling routines. *PowerScript* comes with a built-in dialog box editor that makes designing dialog boxes a snap. The editor has all standard dialog-box controls such as Edit, List, and Combo boxes, and Check and Radio buttons.

*PowerScript* supports OLE Automation, a widely used standard for connecting computer software made by different vendors. This means a script can utilize and manipulate programs, such as MS Word and Excel, which are written as OLE containers. In a sample power-flow script that finds the maximum MW transfer between two areas, the script sends the solution to Excel and directs it to plot the classical voltage-versus-MW curve. The possibility of what you can do with *PowerScript* is limitless.

A text editor is built into *OneLiner* and *Power Flow* to enable you to modify scripts. The build-in script editor has many advanced features including syntax highlighting. A symbolic debugger for *PowerScript* is also built-in. In the debug mode, you can stop the script at any point and examine all the program variables.

Many example scripts are included with *OneLiner* and *Power Flow*. Some of these scripts are useful applications of their own right. We encourage you to take a close look at these scripts.

---

## 1.2 Language Specification

*ASPEN PowerScript* fully supports the Cypress Enable Script programming language syntax, as well as the standard intrinsic functions in BASIC programming language. Comprehensive Cypress Enable Script Language reference is available in Appendices A, B and C of this manual.

*PowerScript* also comes with a collection of subroutines that lets you:

- Examine and modify network data and relay data in *ASPEN OneLiner* and *Power Flow*.
- Perform short circuit and power flow solutions.
- Access the value of solution variables.

These functions are described in Section 3 and Section 4.

If you are new to *PowerScript*, we suggest you follow the tutorial in Section 2.

---

## 2.1 Introduction

In this section you will learn how to edit a script, run it and debug it. You will also see how to turn the script into a function that you can call directly from the menu bar of *OneLiner* or *Power Flow*.

We will use *OneLiner* as the executable program in this tutorial. The same steps can be used in the *Power Flow Program*.

The example script is a very simple per-unit calculator. The script will use the system MVA of the currently open network as the base MVA. It will also set the base kV to the nominal kV of the currently selected object on the one-line diagram. The kV base is set to zero if nothing is selected.

---

## 2.2 Editing a Script File

A BASIC script file is a text file that you can edit with any word processor or text edit. *OneLiner* and *Power Flow* both have built-in editors that are designed for editing script. We encourage you to use the built-in editors.


Please follow these directions to open a script file called `perunit1.bas`.

A 29-bus system is used in most of this tutorial. You will now open its binary data file.

1. **Open the file EXAMPLE30.OLR (EXAMPLE09.OLR for Academic Suite users, and ASPEN9.OLR for Power Flow users.)**

*Note:* It is not necessary to have a network open in order to edit a script file, but in practice, most PowerScript commands require an open data file to run.

2. **Select the Tool | Scripting | Edit / Create Script command**

*Note:* You can click on the button  on the toolbar.

A blank Script Editor window will appear

3. **Open the file `perunit1.pas` in the script editor as follows.**

**Select the File | Open. Click once on the file name 'perunit1.bas' in the list box.** The name will appear in the File Name edit box. **Click on the "Open" button.**

The file will be loaded into the Script Editor

```

PowerScript Editor
File Edit Debug Run Options Help

' ASPEN PowerScript sample program
'
' PERUNIT.BAS
'
' Per-Unit calculator
'
' PowerScript functions called:
'   GetEquipment()
'   GetData()
'
' ===== Dialog box spec (generated by Dialog Editor) =====
Begin Dialog PUDLG 17,16, 139, 111, "PU Calculator "
  Text 28,40,20,12, "Ohm"
  TextBox 16,52,36,12, .Ohm
  PushButton 4,68,56,12, "Ohm  ->  PU", .Convert1
  TextBox 84,52,36,12, .PU
  Text 96,40,12,12, "p.u."
  PushButton 76,68,56,12, "Ohm  <-  PU", .Convert2
  Text 8,8,64,12, "Base MVA (3PH) ="
  TextBox 76,8,36,12, .BaseMVA
  Text 20,24,52,12, "Base kV (L-L) ="
  TextBox 76,24,36,12, .BasekV
  PushButton 48,92,40,12, "Done ", .Done
End Dialog

' ===== main() =====
Sub main()
  Dim dlg As PUDLG
  ' Get system MVA
  If GetData( HND_SYS, SY_dBaseMVA, BaseMVA ) = 0 Then GoTo HasError
  ' Figure out kV base from picked object
  If 0 <> GetEquipment( TC_PICKED, PickedHnd ) Then
    ' Probe to see what's being picked
    Select Case EquipmentType( PickedHnd )
      Case TC_LINE
        If 0 = GetData( PickedHnd, LN_nBus1Hnd, nBusHnd& ) Then GoTo HasError
        If 0 = GetData( nBusHnd&, BUS_dKVNominal, BaseKV ) Then GoTo HasError

```

#### 4. Perform a test run.

**Select the Run | Start command.** The per unit calculator main dialog box will appear. The Base MVA edit box will show the current system MVA base of 100.

**Enter data: Base kV = 132; Ohm = 5;**

**Click on the “Ohm -> PU” button.** The corresponding per unit value of 0.028696 will be displayed in the edit box labeled “p.u”

#### 5. Press Done to close the PU calculator and return to the script editor screen.

## 2.3 Debugging a Script

A nontrivial program has at least one bug – by definition. We purposely introduced one in the example file to give you a chance to fix it.

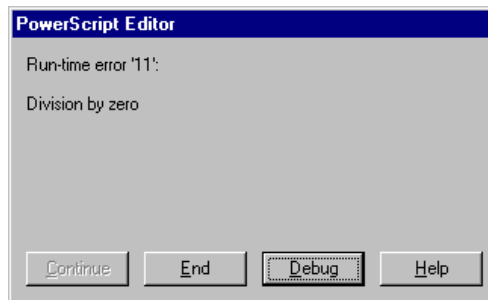
In this part of the tutorial, we assume you have just completed step 3 in section 2.2.

1. **Select Run | Start command.**

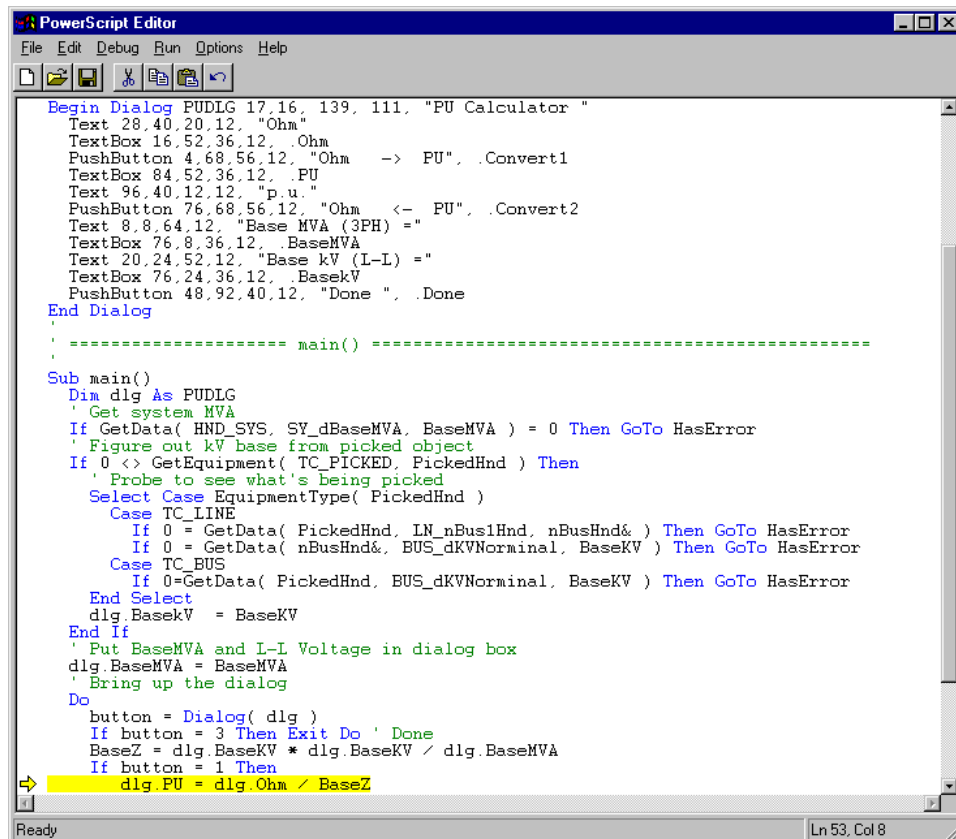
The Per Unit calculator main dialog will appear.

2. **Without entering any data, click on “Ohm->PU” button**

A division-by-zero message box will appear



**Click on Debug button.** The script editor will re-appear in which the line of code where the error occurred is being highlighted in yellow (the last line, at the very bottom).



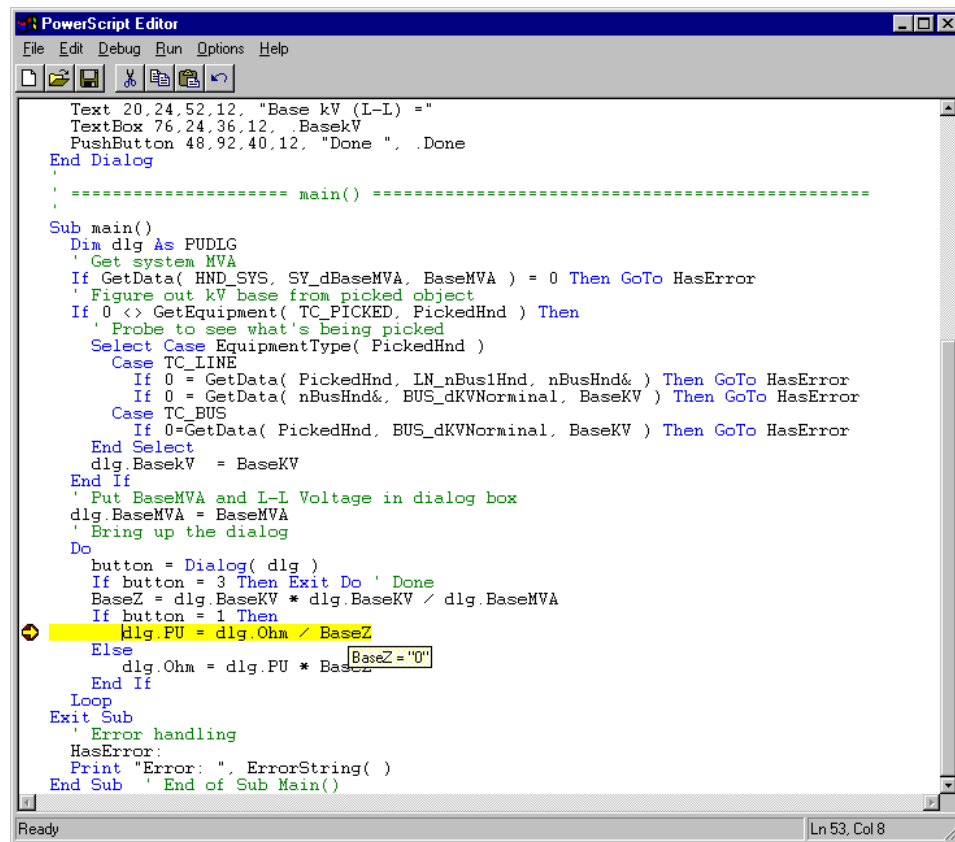
3. Select Run | End to stop the execution of the script.
4. Select the Debug | Toggle Break Point command to set a break point at the error line.

A solid purple dot will appear near the beginning of the line, and the text on the line will be highlighted in purple. This indicates that a break point has been inserted on the line.

5. Select Run | Start.

Leave “Base kV” edit box blank. Click on “Ohm -> PU” button. The script editor will re-appear showing that the program has stopped at the break point position.

Move the mouse pointer over the BaseZ variable on this line. A small window will appear near the mouse pointer showing current value of this variable, which is zero:



```

PowerScript Editor
File Edit Debug Run Options Help
[Icons]

Text 20.24.52.12, "Base kV (L-L) ="
TextBox 76.24.36.12, .BasekV
PushButton 48.92.40.12, "Done ", .Done
End Dialog

' ===== main() =====

Sub main()
Dim dlg As PUDLG
' Get system MVA
If GetData( HND_SYS, SY_dBaseMVA, BaseMVA ) = 0 Then GoTo HasError
' Figure out kV base from picked object
If 0 <> GetEquipment( TC_PICKED, PickedHnd ) Then
' Probe to see what's being picked
Select Case EquipmentType( PickedHnd )
Case TC_LINE
If 0 = GetData( PickedHnd, LN_nBus1Hnd, nBusHnd& ) Then GoTo HasError
If 0 = GetData( nBusHnd&, BUS_dKVNormalinal, BaseKV ) Then GoTo HasError
Case TC_BUS
If 0=GetData( PickedHnd, BUS_dKVNormalinal, BaseKV ) Then GoTo HasError
End Select
dlg.BaseKV = BaseKV
End If
' Put BaseMVA and L-L Voltage in dialog box
dlg.BaseMVA = BaseMVA
' Bring up the dialog
Do
button = Dialog( dlg )
If button = 3 Then Exit Do ' Done
BaseZ = dlg.BaseKV * dlg.BaseKV / dlg.BaseMVA
If button = 1 Then
dlg.PU = dlg.Ohm / BaseZ
Else
dlg.Ohm = dlg.PU * BaseZ
End If
Loop
Exit Sub
' Error handling
HasError:
Print "Error: ", ErrorString( )
End Sub ' End of Sub Main()

```

It's clear that the “divide by zero” error was caused by BaseZ=0 which in turn is caused by the zero value of BaseKV. We will now add some logic to the script program to prevent this from happening.

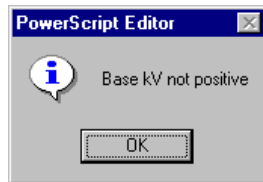
6. Select Run | End to stop the program.

7. **Replace the Do ...Loop code around the break point by the code shown below:**

```
Do
    button = Dialog( dlg )
    If button = 3 Then Exit Do ' Done
    If dlg.BaseKV <= 0 Then
        Print "Base kV not positive"
    ElseIf dlg.baseMVA <= 0 Then
        Print "Base MVA not positive"
    Else
        BaseZ = dlg.BaseKV * dlg.BaseKV / dlg.BaseMVA
        If button = 1 Then
            dlg.PU = dlg.Ohm / BaseZ
        Else
            dlg.Ohm = dlg.PU * BaseZ
        End If
    End If
Loop
```

8. **Select Run | Start to run the new code. Click on “Ohm -> PU” with blank Base kV edit box.**

A message will appear with a warning:



Press OK. The main program dialog will re-appear, ready to accept new input.

9. **Press Done to close the Per Unit calculator program.**

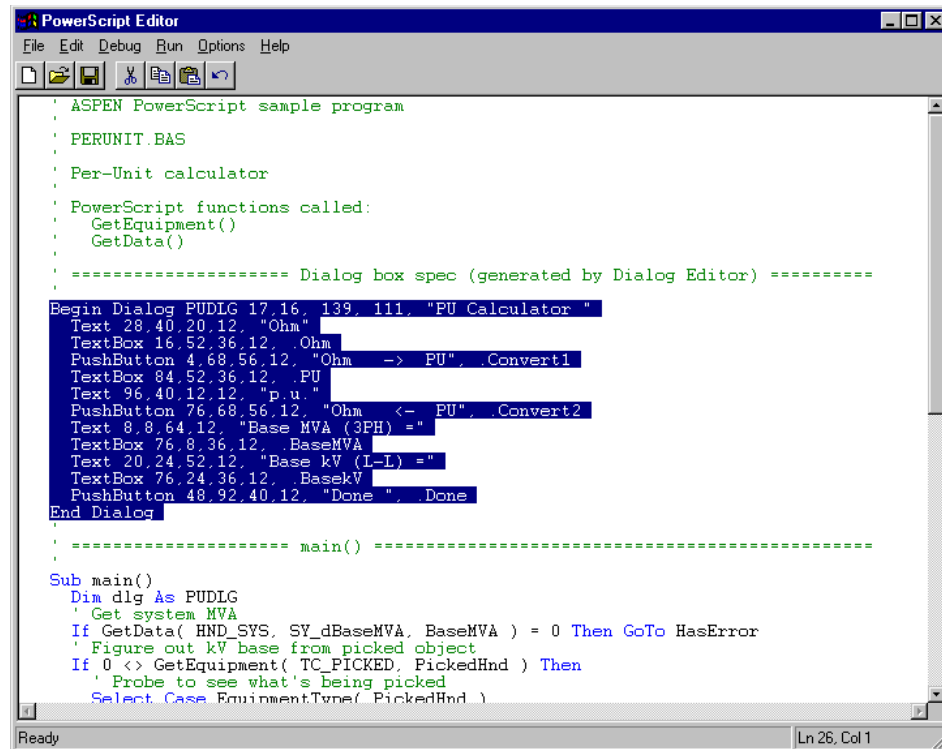
---

## 2.4 Editing Dialog Boxes

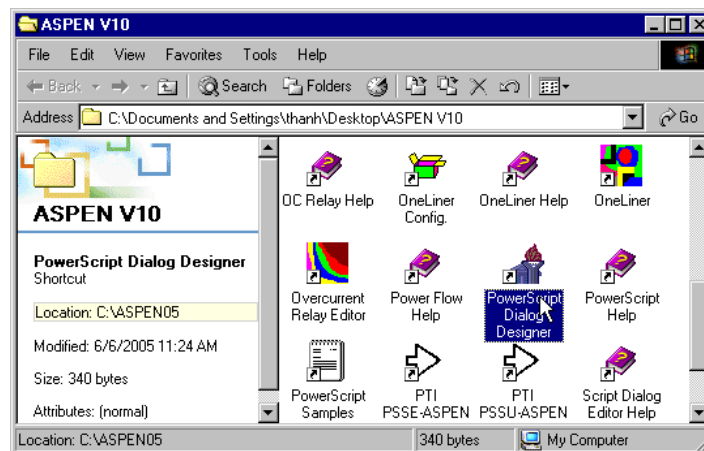
*OneLiner* and *Power Flow* have a dialog-box editor program that lets you create and edit dialog boxes. The editing is done in a graphical setting. You can drag and drop edit boxes, list boxes, combo boxes and other standard dialog-box objects from the toolbar onto your dialog box.

In this part of tutorial we assume that you have just completed step 9 in section 2.3.

1. **Select the text with dialog-box specification, which is surrounded by “Begin Dialog” and “End Dialog” keywords.**
2. **Select command Edit | Copy to transfer the text to Window clipboard.**



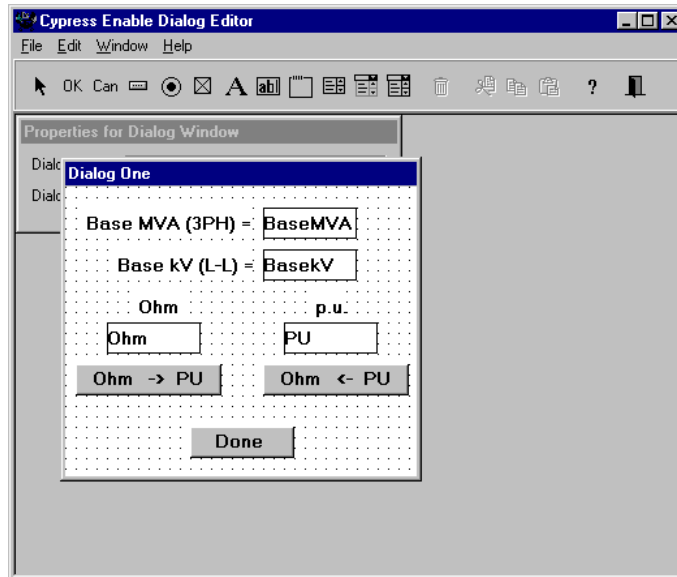
### 3. Open ASPEN V10 program group on the desktop.



Run the dialog designer program by double clicking on its icon. The dialog editor screen will appear.



4. In the Editor, select **File | Load Dialog From Clipboard**. The per unit calculator program main dialog will appear.



5. Click on the Add Text Label button **A** on the Dialog Editor Tool bar. The cursor pointer will become a cross.

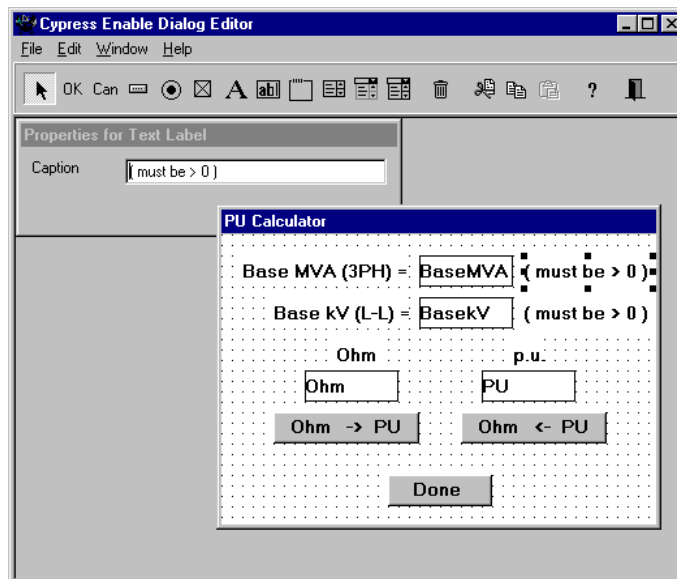
Click once to the right of the BaseMVA edit box.

Select **Window | Properties Window** command. This will bring up Properties for Text Label window.

Enter label “(must be > 0)”

Go back to the Design window. Adjust size and position of various dialog box components to tidy up the dialog box.

Repeat the process above to create another identical label next to BasekV edit box.



6. **When you are satisfied with the new appearance of the dialog box design**  
**Select command** File | Put Dialog on Clipboard. The dialog editor data will be copied to the Windows clipboard.
7. **Switch back to the Script Editor window without closing the dialog editor.**  
*Note: Never attempt to close the Dialog editor by selecting File | Close command or by clicking on the Dialog editor window close button.*  
**Select Edit | Paste to replace the old dialog data in the script program with the newly modified one.**
8. **Select File | Save to save the new code.**
9. **Run the script to check the new code.**
10. **Switch to the Dialog Editor to make any changes to the dialog and repeat step 5-8.**
11. **Select Edit | Dialog Editor to close the dialog editor.**

---

## 2.5 Running Scripts as Program Commands

Once you have created a script, you can execute it directly from the main program window of *OneLiner* or *Power Flow*, via the Tools | Scripting | Run Script command. You can also assign a script file to each of the five customizable menu commands under Tools | User-Defined Commands. This part of the tutorial will introduce you to these features.

**1. Run the script from the Scripting | Run Script command.**

**Select command** Tools | Scripting | Run Script. A file open dialog box will appear.

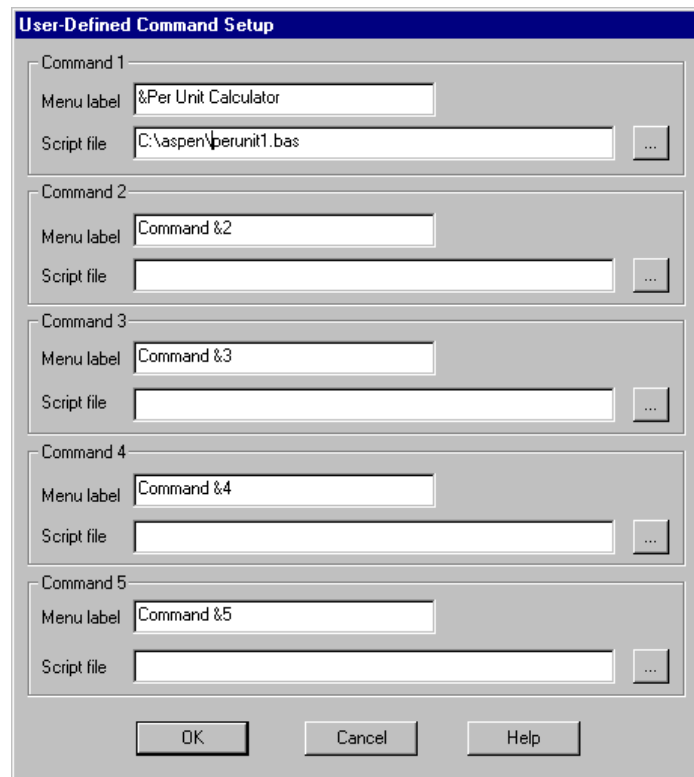
**Note:** You can also use toolbar button  or keyboard short-cut Ctrl-R

**Select file name** perunit1.bas from the list box and click on Open. The main dialog of per unit program will appear.

**Click on Done** when you finish using the per unit calculator.

**2. Customize user-defined commands.**

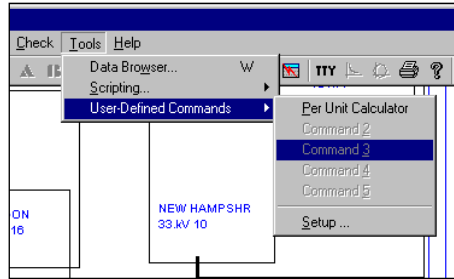
Select command Tools | User-defined Commands | Setup. A dialog box will appear.



The image shows a dialog box titled "User-Defined Command Setup". It contains five sections, each for a command (Command 1 through Command 5). Each section has a "Menu label" text box and a "Script file" text box with a browse button (three dots). Command 1's menu label is "&Per Unit Calculator" and its script file is "C:\aspen\perunit1.bas". Command 2's menu label is "Command &2". Command 3's menu label is "Command &3". Command 4's menu label is "Command &4". Command 5's menu label is "Command &5". At the bottom of the dialog are "OK", "Cancel", and "Help" buttons.

**Enter menu label and full path name to per-unit calculator script file as shown in the picture.**

**Press OK when done.** The Per Unit Calculator command will now be available in the program menu.



3. **Select Tools | User-Defined Commands | Per Unit Calculator.** The script will start running and main dialog of per unit program will appear.

---

## 2.6 Sample Scripts

The per-unit calculator script shown in the previous section is just a simple example designed to show you how to get started with the scripting feature. To explore further, you may want to examine other sample scripts that are shipped with *OneLiner* and *Power Flow Program*.

You can find these sample scripts in the c:\Program Files\ASPEN\1LPFv11\Script directory for *OneLiner* and c:\Program Files\ASPEN\PFv11\Script directory for *Power Flow*. Within each of these directories is a file README.TXT that contains the most current list of script samples and a brief description of what they do.

You can also find script programs submitted by ASPEN users in the Exchange page on ASPEN's web site (<http://www.aspeninc.com/exchange>). We urge you to submit scripts that you want to share with other ASPEN users.

---

## 3.1 Object Handles

*PowerScript* identifies each network object by a unique number, called a “handle.” You must use one of the following functions to obtain a handle before you can read or update the object’s parameters.

<b>GetEquipment</b>	returns the handle of the next piece of network equipment of the given type.
<b>GetRelay</b>	returns the handle of the next protective device in a relay group.
<b>GetBusEquipment</b>	returns the handle of the next piece of equipment of a given type at a bus.
<b>FindBusByName</b>	returns the handle of the bus having a given name and nominal kV.
<b>NextBusByName</b>	returns the handle of the next bus in sorted bus list.
<b>NextBusByNumber</b>	returns the handle of the next bus in sorted bus list.
<b>GetData</b>	returns handle as part of an object data, e.g., the end buses of a line.

A detailed description of these functions is given in Section 4.

The handle is an identification tag. When you provide the object handle as an input to a *PowerScript* function, the function will know which object you are referring to.

In addition to handles returned by the functions listed above, there are several pre-defined handles that give *PowerScript* access to general system data:

**Table 3.1.** List of predefined *PowerScript* handles

Data	Code
System data	HND_SYS
Power flow case solution	HND_PF
Short circuit case solution	HND_SC

---

## 3.2 Equipment Type Code

The functions **GetEquipment**, **GetRelay**, **GetBusEquipment**, plus several other *PowerScript* functions require you to enter an equipment type code to identify the object type of interest. The complete type code list is in the table 3.2.

**NOTE:** Items in table 3.2 and 3.3 that are shown with underlined type are new in *OneLiner Version 14*

**Table 3.2.** List of *PowerScript* equipment type codes

Equipment type	Code
Bus	TC BUS
Load	TC LOAD
Load unit	TC LOADUNIT
Shunt	TC SHUNT
Shunt unit	TC SHUNTUNIT
Generator	TC GEN
Generator unit	TC GENUNIT
Switched shunt	TC SVD
Branch connection	TC BRANCH
Transmission Line	TC LINE
Mutual coupling pair	TC MU
2-winding transformer	TC XFMR
3-winding transformer	TC XFMR3
Phase shifter	TC PS
Switch	TC SWITCH
Series capacitor	TC SCAP
System parameter	TC SYS
Relay group	TC RLYGROUP
Overcurrent ground relay	TC RLYOCG
Overcurrent phase relay	TC RLYOCP
DS ground relay	TC RLYDSG
DS phase relay	TC RLYDSP
<u>Differential relay</u>	TC RLYD
<u>Voltage relay</u>	TC RLYV
Fuse	TC FUSE
<u>Breaker</u>	TC BREAKER
Recloser - ground unit	TC RECLSRG
Recloser - phase unit	TC RECLSRP
<u>Voltage controlled current source</u>	TC CCGEN
<u>Logic scheme</u>	TC SCHEME
Power flow case	TC PF
Short circuit case	TC SC
First selected object on the 1-line diagram	TC PICKED, TC PICKED1
Second and third selected object on the 1-line diagram	TC PICKED2, TC PICKED3

---

## 3.3 Network Data Access

The *PowerScript* function **GetData** copies network and system data from a *OneLiner* or *Power Flow* case to *PowerScript* program variables. The datum to be read is uniquely identified by an object handle and a parameter code. The list of supported parameter codes is given in table 3.3.

Parameters that are listed in table 3.3 with YES in Write Access column can be modified from *PowerScript*. There are two functions that help you modify the parameters of an object: **SetData** and **PostData**.

Suppose you want to change three parameters of an object. In your program you must call the function **SetData** separately for each of the parameters you are changing. *PowerScript* automatically creates a temporary object in memory with the updated parameters. Once you are done modifying the parameters, you must call the function **PostData** once to validate the object data and if the data is valid, copy the temporary object to the *OneLiner* or *Power Flow* case.

**Table 3.3.** List of *PowerScript* Equipment Parameter Code

Parameter	Data type	Write Access	Parameter Code
<u>Breaker contact parting time for group 1 (cycles)</u>	double	YES	BK_dCPT1
<u>Breaker contact parting time for group 2 (cycles)</u>	double	YES	BK_dCPT2
<u>Breaker interrupting time (cycles)</u>	double	YES	BK_dCycles
<u>Breaker kV range factor</u>	double	YES	BK_dK
<u>Breaker no-ac-decay ratio</u>	double	YES	BK_dNACD
<u>Breaker operating kV</u>	double	YES	BK_dOperatingKV
<u>Breaker max design kV</u>	double	YES	BK_dRatedKV
<u>Breaker interrupting rating</u>	double	YES	BK_dRating1
<u>Breaker momentary rating</u>	double	YES	BK_dRating2
<u>Breaker bus handle</u>	Long	NO	BK_nBusHnd
<u>Breaker do not derate in reclosing operation flag: 1-true; 0-false;</u>	Long	YES	BK_nDontDerate
<u>Breaker In-service flag: 1-true; 2-false;</u>	Long	YES	BK_nInService
<u>Breaker group 1 interrupting current: 1-max current; 0-group current</u>	Long	YES	BK_nInterrupt1
<u>Breaker group 1 interrupting current: 1-max current; 0-group current</u>	Long	YES	BK_nInterrupt2
<u>Breaker rating type: 0- symmetrical current basis; 1- total current basis; 2- IEC</u>	Long	YES	BK_nRatingType
<u>Breaker total operations for group 1</u>	Long	YES	BK_nTotalOps1
<u>Breaker total operations for group 2</u>	Long	YES	BK_nTotalOps2
<u>Breaker protected equipment group 1 in string format</u>	String	NO	BK_sEquipGrp1
<u>Breaker protected equipment group 2 in string format</u>	String	NO	BK_sEquipGrp2
<u>Breaker ID</u>	String	YES	BK_sID
<u>Breaker reclosing intervals for group 1 (s)</u>	array(3) of double	YES	BK_vdRecloseInt1
<u>Breaker reclosing intervals for group 2 (s)</u>	array(3) of double	YES	BK_vdRecloseInt2
<u>Breaker protected equipment group 1 list of equipment handles</u>	array(10) of long	NO	BK_vnG1DevHnd
<u>Breaker protected equipment group 1 list of additional outage handles</u>	array(10) of long	NO	BK_vnG1OutageHnd
<u>Breaker protected equipment group 2 list of equipment handles</u>	array(10) of long	NO	BK_vnG2DevHnd
<u>Breaker protected equipment group 2 list of additional outage handles</u>	array(10) of long	NO	BK_vnG2OutageHnd
<u>Branch near bus handle</u>	Long	NO	BR_nBus1Hnd
<u>Branch far bus handle</u>	Long	NO	BR_nBus2Hnd
<u>Branch bus 3 handle</u>	Long	NO	BR_nBus3Hnd
<u>Branch equipment handle</u>	Long	NO	BR_nHandle
<u>Branch in-service flag: 1- active; 2- out-of-service</u>	Long	NO	BR_nInService
<u>Branch near bus relay group handle</u>	Long	NO	BR_nRlyGrp1Hnd
<u>Branch far bus relay group handle</u>	Long	NO	BR_nRlyGrp2Hnd
<u>Branch bus 3 relay group handle</u>	Long	NO	BR_nRlyGrp3Hnd
<u>Branch type</u>	Long	NO	BR_nType
<u>Bus voltage angle (load flow solution)</u>	double	NO	BUS_dAngleP
<u>Bus nominal kV</u>	double	NO	BUS_dKVnominal
<u>Bus voltage magnitude (load flow solution)</u>	double	NO	BUS_dKVP

**Table 3.3 (Cont.)**

Parameter	Data type	Write Access	Parameter Code
Bus state plane coordinate - X	double	YES	BUS_dSPC <sub>x</sub>
Bus state plane coordinate - Y	double	YES	BUS_dSPC <sub>y</sub>
Bus area	Long	NO	BUS_nArea
Bus number	Long	YES	BUS_nNumber
System slack bus flag: 1=yes; 0=no	Long	YES	BUS_nSlack
Bus substation group	Long	YES	BUS_nSubGroup
Tap bus flag: 0=no; 1- tap bus; 3- tap bus of 3-terminal line	Long	YES	BUS_nTapBus
Bus visibility flag: 1-visible; -1-hidden; 0-not yet placed	Long	YES	BUS_nVisible
Bus zone	Long	YES	BUS_nZone
Bus comment	String	YES	BUS_sComment
Bus location	String	YES	BUS_sLocation
Bus name	String	YES	BUS_sName
Voltage controlled current source MVA rating	double	YES	CC_dMVArating
Voltage controlled current source maximum voltage limit	double	YES	CC_dVmax
Voltage controlled current source minimum voltage limit	double	YES	CC_dVmin
Voltage controlled current source in-service flag: 1-true; 2-false	Long	YES	CC_nInService
Voltage controlled current source voltage measurement location	Long	YES	CC_nVloc
Voltage controlled current source out of service date	String	YES	CC_sOffDate
Voltage controlled current source in service date	String	YES	CC_sOnDate
Voltage controlled current source angle	array(10) of double	YES	CC_vdAng
Voltage controlled current source current	array(10) of double	YES	CC_vdI
Voltage controlled current source voltage	array(10) of double	YES	CC_vdV
Recloser-Ground high current trip	double	YES	CG_dHiAmps
Recloser-Ground high current trip delay	double	YES	CG_dHiAmpsDelay
Recloser-Ground fast curve minimum time	double	YES	CG_dMinTF
Recloser-Ground slow curve minimum time	double	YES	CG_dMinTS
Recloser-Ground fast curve pickup	double	YES	CG_dPickupF
Recloser-Ground slow curve pickup	double	YES	CG_dPickupS
Recloser-Ground reclosing interval 1	double	YES	CG_dRecIntvl1
Recloser-Ground reclosing interval 2	double	YES	CG_dRecIntvl2
Recloser-Ground reclosing interval 3	double	YES	CG_dRecIntvl3
Recloser-Ground fast curve time adder	double	YES	CG_dTimeAddF
Recloser-Ground slow curve time adder	double	YES	CG_dTimeAddS
Recloser-Ground fast curve time multiplier	double	YES	CG_dTimeMultF
Recloser-Ground slow curve time multiplier	double	YES	CG_dTimeMultS
Recloser-Ground curve selection flag: 0- slow; 1- fast	Long	YES	CG_nCurveInUse
Recloser-Ground number of fast operations	Long	YES	CG_nFastOps
Recloser-Ground in service flag: 1- active; 2- out-of-service	Long	YES	CG_nInService
Recloser-Ground relay group handle	Long	NO	CG_nRlyGrHnd
Recloser-Ground total operations to locked out	Long	YES	CG_nTotalOps
Recloser-Ground asset ID	String	YES	CG_sAssetID
Recloser-Ground comments	String	YES	CG_sComment
Recloser-Ground ID	String	YES	CG_sID
Recloser-Ground fast curve	String	NO	CG_sTypeFast
Recloser-Ground slow curve	String	NO	CG_sTypeSlow
Recloser-Phase high current trip	double	YES	CP_dHiAmps
Recloser-Phase high current trip delay	double	YES	CP_dHiAmpsDelay
Recloser-Phase fast curve minimum time	double	YES	CP_dMinTF
Recloser-Phase slow curve minimum time	double	YES	CP_dMinTS
Recloser-Phase fast curve pickup	double	YES	CP_dPickupF
Recloser-Phase slow curve pickup	double	YES	CP_dPickupS
Recloser-Phase reclosing interval 1	double	YES	CP_dRecIntvl1
Recloser-Phase reclosing interval 2	double	YES	CP_dRecIntvl2

**Table 3.3 (Cont.)**



Parameter	Data type	Write Access	Parameter Code
Recloser-Phase reclosing interval 3	double	YES	CP_dRecIntvl3
Recloser-Phase fast curve time adder	double	YES	CP_dTimeAddF
Recloser-Phase slow curve time adder	double	YES	CP_dTimeAddS
Recloser-Phase fast curve time multiplier	double	YES	CP_dTimeMultF
Recloser-Phase slow curve time multiplier	double	YES	CP_dTimeMultS
Recloser-Phase curve selection flag: 0- slow; 1- fast	Long	YES	CP_nCurveInUse
Recloser-Phase number of fast operations	Long	YES	CP_nFastOps
Recloser-Phase in service flag: 1- active; 2- out-of-service	Long	YES	CP_nInService
Recloser-Phase relay group handle	Long	NO	CP_nRlyGrHnd
Recloser-Phase total operations to locked out	Long	YES	CP_nTotalOps
Recloser-Phase comments	String	YES	CP_sComment
Recloser-Phase ID	String	YES	CP_sID
Recloser-Phase fast curve	String	NO	CP_sTypeFast
Recloser-Phase slow curve	String	NO	CP_sTypeSlow
DS ground relay CT ratio	double	NO	DG_dCT
Zero sequence compensation factor K - angle	double	NO	DG_dKang
Zero sequence compensation factor K - magnitude	double	NO	DG_dKmag
DS ground relay VT ratio	double	NO	DG_dVT
DS ground relay in-service flag: 1- active; 2- out-of-service	Long	NO	DG_nInService
<u>DS ground relay parameter count</u>	Long	NO	DG_nParamCount
<u>DS ground signal-only zone flag</u>	Long	YES	DG_nSignalOnly
DS ground relay group handle	Long	NO	DG_nRlyGrHnd
<u>DS ground relay asset ID</u>	String	YES	DG_sAssetID
<u>DS ground relay comment</u>	String	NO	DG_sComment
<u>DS ground relay type name</u>	String	NO	DG_sDSType
DS ground relay ID	String	NO	DG_sID
DS ground relay setting <sup>(1)</sup>	String	YES	DG_sParam
DS ground relay ID2	String	YES	DG_sType
<u>DS ground relay zone delay</u>	array (8) of double	NO	DG_vdDelay
<u>DS ground relay parameter</u>	array(DG_nParamCount) of double	NO	DG_vdParams
<u>DS ground relay zone reach</u>	array(8) of double	NO	DG_vdReach
<u>DS ground relay zone reach 1</u>	array(8) of double	NO	DG_vdReach1
DS ground relay setting labels	array(255) of variant	NO	DG_vParamLabels
DS ground relay settings	array(255) of variant	NO	DG_vParams
DS phase relay CT ratio	double	NO	DP_dCT
DS phase relay VT ratio	double	NO	DP_dVT
DS phase relay in-service flag: 1- active; 2- out-of-service	Long	NO	DP_nInService
<u>DS phase relay parameter count</u>	Long	NO	DP_nParamCount
<u>DS phase signal-only zone flag</u>	Long	YES	DP_nSignalOnly
DS phase relay group handle	Long	NO	DP_nRlyGrHnd
<u>DS phase relay asset ID</u>	String	YES	DP_sAssetID
<u>DS phase relay comment</u>	String	NO	DP_sComment
<u>DS phase relay type name</u>	String	NO	DP_sDSType
DS phase relay ID	String	YES	DP_sID
DS phase relay setting <sup>(1)</sup>	String	YES	DP_sParam
DS phase relay ID2	String	NO	DP_sType
<u>DS phase relay zone delay</u>	array(8) of double	NO	DP_vdDelay
<u>DS phase relay parameter</u>	array(DS_nParamCount) of double	NO	DP_vdParams
<u>DS phase relay zone reach</u>	array(8) of double	NO	DP_vdReach
<u>DS phase relay alternat zone reach</u>	array(8) of double	NO	DP_vdReach1

**Table 3.3 (Cont.)**

Parameter	Data type	Write Access	Parameter Code
DS phase relay setting labels	array(255) of variant	NO	DP_vParamLabels
DS phase relay settings	array(255) of variant	NO	DP_vParams
Fuse 'Compute time using' flag: 1- minimum melt; 2- Total clear	Long	YES	FS_nCurve
Fuse in-service flag: 1- active; 2- out-of-service	Long	YES	FS_nInService
Fuse relay group handle	Long	NO	FS_nRlyGrHnd
<u>Fuse asset ID</u>	String	YES	FS_sAssetID
<u>Fuse comment</u>	String	NO	FS_sComment
Fuse ID	String	YES	FS_sID
Fuse type	String	NO	FS_sType
Fault MVA	double	NO	FT_dMVA
Thevenin equivalent negative sequence resistance	double	NO	FT_dRnt
Thevenin equivalent positive sequence resistance	double	NO	FT_dRPt
Thevenin equivalent zero sequence resistance	double	NO	FT_dRZt
Thevenin equivalent negative sequence reactance	double	NO	FT_dXnt
Thevenin equivalent positive sequence reactance	double	NO	FT_dXPt
X/R ratio at fault point	double	NO	FT_dXR
ANSI X/R ratio at fault point	double	NO	FT_dXRANSI
Thevenin equivalent zero sequence reactance	double	NO	FT_dXZt
Number of faults saved in solution buffer	Long	NO	FT_nNOFaults
Generator current limit 1	double	YES	GE_dCurrLimit1
Generator current limit 2	double	YES	GE_dCurrLimit2
Generator reference angle	double	YES	GE_dRefAngle
Generator scheduled P	double	NO	GE_dScheduledP
Generator scheduled Q	double	NO	GE_dScheduledQ
Generator scheduled V	double	YES	GE_dScheduledV
Generator internal voltage source per unit magnitude	double	YES	GE_dVSourcePU
Generator in-service flag: 1- active; 2- out-of-service	Long	NO	GE_nActive
Handle of generator's controlled bus	Long	NO	GE_nCtrlBusHnd
Generator regulation flag: 1- PQ; 0- PV	Long	NO	GE_nFixedPQ
Generator unit rating	double	YES	GU_dMVArating
Generator unit max MW	double	YES	GU_dPmax
Generator unit min MW	double	YES	GU_dPmin
Generator unit max MVAR	double	YES	GU_dQmax
Generator unit min MVAR	double	YES	GU_dQmin
Generator unit grounding resistance	double	YES	GU_dRz
Generator unit grounding reactance	double	YES	GU_dXz
Generator unit online flag	Long	YES	GU_nOnline
Generator unit ID	String	NO	GU_sID
<u>Generator unit out of service date</u>	String	YES	GU_sOffDate
<u>Generator unit in service date</u>	String	YES	GU_sOnDate
Generator unit resistances: subtransient, synchronous, transient, negative sequence, zero sequence	array(5) of double	YES	GU_vdR
Generator unit reactances: subtransient, synchronous, transient, negative sequence, zero sequence	array(5) of double	YES	GU_vdX
Total load MW	double	NO	LD_dPload
Total load MVAR	double	NO	LD_dQload
Load in-service flag: 1- active; 2- out-of-service	Long	NO	LD_nActive
Line B1	double	YES	LN_dB1
Line B10	double	YES	LN_dB10
Line B2	double	YES	LN_dB2
Line B20	double	YES	LN_dB20
Line G1	double	YES	LN_dG1

Table 3.3 (Cont.)

Parameter	Data type	Write Access	Parameter Code
Line G10	double	YES	LN_dG10
Line G2	double	YES	LN_dG2
Line G20	double	YES	LN_dG20
Line length	double	YES	LN_dLength
Line R	double	YES	LN_dR
Line Ro	double	YES	LN_dR0
Line X	double	YES	LN_dX
Line Xo	double	YES	LN_dX0
Line bus 1 handle	long	NO	LN_nBus1Hnd
Line bus 2 handle	long	NO	LN_nBus2Hnd
Line in-service flag: 1- active; 2- out-of-service	long	YES	LN_nInService
Line mutual pair handle	long	NO	LN_nMuPairHnd
Line bus 1 relay group handle	long	NO	LN_nRlyGr1Hnd
Line bus 2 relay group handle	long	NO	LN_nRlyGr2Hnd
Line circuit ID	string	YES	LN_sID
Line length unit	string	YES	LN_sLengthUnit
Line name	string	YES	LN_sName
<u>Line out of service date</u>	string	YES	LN_sOffDate
<u>Line in service date</u>	string	YES	LN_sOnDate
Line table type	string	YES	LN_sType
Line ratings	array(4) of double	YES	LN_vdRating
<u>Logic scheme in service</u>	long	YES	LS_nInService
<u>Logic scheme relay group handle</u>	Long	NO	LS_nRlyGrpHnd
<u>Logic scheme signal only</u>	long	YES	LS_nSignalOnly
<u>Logic scheme asset ID</u>	string	YES	LS_sAssetID
<u>Logic scheme equation</u>	String	NO	LS_sEquation
<u>Logic scheme ID</u>	string	YES	LS_sID
<u>Logic scheme name</u>	string	NO	LS_sScheme
<u>Logic scheme variables details</u> (one variable per line in the format: name=description)	String	NO	LS_sVariables
Load unit MW	double	NO	LU_dPload
Load unit MVAR	double	NO	LU_dQload
Load unit online flag	long	YES	LU_nOnline
Load unit in-service flag: 1-active; 2- out-of-service	long	YES	LU_nOnline
Load unit ID	string	NO	LU_sID
Load unit ID	string	NO	LU_sID
<u>Load unit out of service date</u>	string	YES	LU_sOffDate
<u>Load unit in service date</u>	string	YES	LU_sOnDate
Load unit MVARs: const. P, const. I, const Z	array(3) of double	YES	LU_vdMAR
Load unit MWs: const. P, const. I, const Z	array(3) of double	YES	LU_vdMW
Mutual pair line 1 handle	long	YES	MU_nHndLine1
Mutual pair line 2 handle	long	YES	MU_nHndLine2
<u>Mutual pair line 1 From percent</u>	array(5) double	YES	MU_vdFrom1
<u>Mutual pair line 2 From percent</u>	array(5) double	YES	MU_vdFrom2
<u>Mutual pair: R</u>	array(5) of double	YES	MU_vdR
<u>Mutual pair line1 To percent</u>	array(5) of double	YES	MU_vdTo1
<u>Mutual pair line 2 To percent</u>	array(5) of double	YES	MU_vdTo2
<u>Mutual pair: X</u>	array(5) of double	YES	MU_vdX
OC ground relay CT ratio	Double	YES	OG_dCT
OC ground relay instantaneous setting	double	NO	OG_dInst
OC ground relay instantaneous delay	double	NO	OG_dInstDelay
<u>OC ground relay reset time</u>	double	YES	OG_dResetTime
OC ground relay tap Ampere	double	YES	OG_dTap
OC ground relay time dial	double	YES	OG_dTDial
OC ground relay time adder	double	YES	OG_dTimeAdd
<u>OC ground relay time adder 2</u>	double	YES	OG_dTimeAdd2

**Table 3.3 (Cont.)**

Parameter	Data type	Write Access	Parameter Code
OC ground relay time multiplier	double	YES	OG_dTimeMult
<u>OC ground relay time multiplier 2</u>	double	YES	OG_dTimeMult2
<u>OC ground relay sensitive to DC offset</u>	long	YES	OG_nDCOffset
OC ground relay directional flag: 0=false;	long	NO	OG_nDirectional
<u>OC ground relay flat definite time delay flag: 1=true; 0=false</u>	long	YES	OG_nFlatDelay
OC ground relay Inst. Directional flag: 0=false;	long	NO	OG_nIDirectional
OC ground relay in-service flag: 1- active; 2- out-of-service	long	NO	OG_nInService
<u>OC ground relay polar option</u>	long	YES	OG_nPolar
OC ground relay group handle	long	NO	OG_nRlyGrHnd
<u>OC ground relay signal only</u>	long	YES	OG_nSignalOnly
<u>OC ground relay asset ID</u>	string	YES	OG_sAssetID
<u>OC ground relay comment</u>	string	NO	OG_sComment
OC ground relay ID	string	YES	OG_sID
OC ground relay type	string	NO	OG_sType
<u>OC ground relay direction setting</u>	array(8) of double	YES	OG_vdDirSetting
OC phase relay CT ratio	double	YES	OP_dCT
OC phase relay instantaneous setting	double	NO	OP_dInst
OC phase relay instantaneous delay	double	NO	OP_dInstDelay
OC phase relay tap Ampere	double	YES	OP_dTap
OC phase relay time dial	double	YES	OP_dTDial
OC phase relay time adder	double	YES	OP_dTimeAdd
<u>OC phase relay time adder 2</u>	double	YES	OP_dTimeAdd2
OC phase relay time multiplier	double	YES	OP_dTimeMult
<u>OC phase relay time multiplier 2</u>	double	YES	OP_dTimeMult2
<u>OC phase relay reset time</u>	double	YES	OP_dResetTime
<u>OC phase relay voltage controlled or restrained percentage</u>	double	YES	OP_dVCtrlRestPcnt
<u>OC phase relay CT connection</u>	long	YES	OP_nByCTConnect
<u>OC phase relay sensitive to DC offset</u>	long	YES	OP_nDCOffset
OC phase relay directional flag: 0=false;	long	NO	OP_nDirectional
<u>OC phase relay flat delay</u>	long	YES	OP_nFlatDelay
OC phase relay Inst. Directional flag: 0=false;	long	NO	OP_nIDirectional
OC phase relay in-service flag: 1- active; 2- out-of-service	long	NO	OP_nInService
<u>OC phase relay polar option</u>	long	YES	OP_nPolar
OC phase relay group handle	long	NO	OP_nRlyGrHnd
<u>OC phase relay signal only</u>	long	YES	OP_nSignalOnly
<u>OC phase relay voltage controlled or restrained</u>	long	YES	OP_nVoltControl
<u>OC phase relay asset ID</u>	string	YES	OP_sAssetID
<u>OC phase relay comment</u>	string	NO	OP_sComment
OC phase relay ID	string	YES	OP_sID
OC phase relay type	string	NO	OP_sType
<u>OC phase relay direction setting</u>	array(8) of double	YES	OP_vdDirSetting
Phase shifter shift angle	double	YES	PS_dAngle
Phase shifter shift angle max	double	YES	PS_dAngleMax
Phase shifter shift angle min	double	YES	PS_dAngleMin
Phase shifter B	double	YES	PS_dB
Phase shifter Bo	double	YES	PS_dB0
Phase shifter B2	double	YES	PS_dB2
<u>Phase shifter MVA1</u>	double	YES	PS_dMVA1
<u>Phase shifter MVA2</u>	double	YES	PS_dMVA2
<u>Phase shifter MVA3</u>	double	YES	PS_dMVA3
Phase shifter MW max	double	YES	PS_dMWmax
Phase shifter MW min	double	YES	PS_dMWmin
Phase shifter R	double	YES	PS_dR
Phase shifter Ro	double	YES	PS_dR0

**Table 3.3 (Cont.)**

Parameter	Data type	Write Access	Parameter Code
Phase shifter R2	double	YES	PS_dR2
Phase shifter X	double	YES	PS_dX
Phase shifter Xo	double	YES	PS_dX0
Phase shifter X2	double	YES	PS_dX2
Phase shifter bus 1 handle	long	NO	PS_nBus1Hnd
Phase shifter bus 2 handle	long	NO	PS_nBus2Hnd
Phase shifter control mode	long	NO	PS_nControlMode
Phase shifter in-service flag: 1- active; 2- out-of-service	long	YES	PS_nInService
Phase shifter relay group 1 handle	long	NO	PS_nRlyGr1Hnd
Phase shifter relay group 2 handle	long	NO	PS_nRlyGr2Hnd
Phase shifter circuit ID	string	NO	PS_sID
Phase shifter name	string	NO	PS_sName
Phase shifter out of service date	string	YES	PS_sOffDate
Phase shifter in service date	string	YES	PS_sOnDate
<u>Differential relay minimum enable differential current (3I0)</u>	double	YES	RD_dPickup3I0
<u>Differential relay minimum enable differential current (3I2)</u>	double	YES	RD_dPickup3I2
<u>Differential relay minimum enable differential current (phase)</u>	double	YES	RD_dPickupPh
<u>Differential relay tapped load coordination delay (I0)</u>	double	YES	RD_dTLCTDDelayI0
<u>Differential relay tapped load coordination delay (I2)</u>	double	YES	RD_dTLCTDDelayI2
<u>Differential relay tapped load coordination delay (phase)</u>	double	YES	RD_dTLCTDDelayPh
<u>Differential relay local current input CTR 1</u>	long	YES	RD_nCTR1
<u>Differential relay in service</u>	long	YES	RD_nInService
<u>Differential relay local current input handle 1</u>	long	NO	RD_nLocalCTHnd1
<u>Relay group reclose logic</u>	long	NO	RD_nLogicRecl
<u>Relay group trip logic</u>	long	NO	RD_nLogicTrip
<u>Relay group max operations</u>	long	NO	RD_nOps
<u>Differential relay group handle</u>	long	NO	RD_nRlyGrpHnd
<u>Differential relay remote device handle 1</u>	long	NO	RD_nRmeDevHnd1
<u>Differential relay remote device handle 2</u>	long	NO	RD_nRmeDevHnd2
<u>Differential relay signal only</u>	long	YES	RD_nSignalOnly
<u>Differential relay asset ID</u>	string	YES	RD_sAssetID
<u>Differential relay ID</u>	string	YES	RD_sID
<u>Differential relay tapped load coordination curve (I0)</u>	string	NO	RD_sTLCCurveI0
<u>Differential relay tapped load coordination curve (I2)</u>	string	NO	RD_sTLCCurveI2
<u>Differential relay tapped load coordination curve (phase)</u>	string	NO	RD_sTLCCurvePh
<u>Relay group interrupting time (cycles)</u>	double	YES	RG_dBreakerTime
<u>Relay group back up group handle</u>	long	NO	RG_nBackupHnd
<u>Relay group branch handle</u>	long	NO	RG_nBranchHnd
<u>Relay group in-service flag: 1- active; 2- out-of-service</u>	long	NO	RG_nInService
<u>Relay group primary group handle</u>	long	NO	RG_nPrimaryHnd
<u>Relay group trip logic scheme handle</u>	long	YES	RG_nTripLogicHnd
<u>Relay group reclose logic scheme handle</u>	long	NO	RG_nReclLogicHnd
<u>Relay group total operations</u>	long	NO	RG_nOps
<u>Relay group annotation</u>	string	NO	RG_sNote
<u>Relay group reclosing intervals</u>	array(4) of double	YES	RG_vdRecloseInt
<u>Voltage relay over-voltage instant pickup (V)</u>	double	YES	RV_dOVIPickup
<u>Voltage relay over-voltage delay</u>	double	YES	RV_dOVTDelay
<u>Voltage relay over-voltage pickup (V)</u>	double	YES	RV_dOVTPickup
<u>Voltage relay under-voltage instant pickup (V)</u>	double	YES	RV_dUVIPickup
<u>Voltage relay under-voltage delay</u>	double	YES	RV_dUVTDelay
<u>Voltage relay under-voltage pickup (V)</u>	double	YES	RV_dUVTPickup
<u>Voltage relay PT ratio</u>	double	YES	RV_dCTR
<u>Voltage relay in service</u>	long	YES	RV_nInService

Table 3.3 (Cont.)

Parameter	Data type	Write Access	Parameter Code
<u>Voltage relay signal only</u>	long	YES	RV_nSignalOnly
<u>Voltage relay operate on voltage option</u>	long	YES	RV_nVoltOperate
<u>Voltage relay group handle</u>	long	NO	RV_rlyGrpHnd
<u>Voltage relay asset ID</u>	string	YES	RV_sAssetID
<u>Voltage relay ID</u>	string	YES	RV_sID
<u>Voltage relay over-voltage element curve</u>	string	NO	RV_sOVCurve
<u>Voltage relay under-voltage element curve</u>	string	NO	RV_sUVCurve
Series capacitor protective level current	double	YES	SC_dIpr
Series capacitor/reactor R	double	YES	SC_dR
Series capacitor/reactor X	double	YES	SC_dX
Series capacitor/reactor bus 1 handle	long	NO	SC_nBus1Hnd
Series capacitor/reactor bus 2 handle	long	NO	SC_nBus2Hnd
Series capacitor/reactor in-service flag: 1- active; 2- out-of-service; 3- bypassed	long	YES	SC_nInService
Series capacitor/reactor circuit ID	string	YES	SC_sID
Series capacitor/reactor name	string	YES	SC_sName
<u>Series capacitor out of service date</u>	string	YES	SC_sOffDate
<u>Series capacitor in service date</u>	string	YES	SC_sOnDate
Shunt unit susceptance (positive sequence)	double	YES	SU_dB
Shunt unit susceptance (zero sequence)	double	YES	SU_dB0
Shunt unit conductance (positive sequence)	double	YES	SU_dG
Shunt unit conductance (zero sequence)	double	YES	SU_dG0
Shunt unit 3-winding transformer flag	long	NO	SU_n3WX
Shunt unit online flag	long	YES	SU_nOnline
Shunt unit ID	string	NO	SU_sID
SVD admittance in use	double	NO	SV_dB
SVD max V	double	NO	SV_dVmax
SVD min V	double	NO	SV_dVmin
SVD in-service flag: 1- active; 2- out-of-service	long	NO	SV_nActive
Handle of SVD controlled bus	long	NO	SV_nCtrlBusHnd
SVD control mode	long	NO	SV_nCtrlMode
SVD increment B0	array(8) of double	NO	SV_vdB0inc
SVD increment B	array(8) of double	NO	SV_vdBinc
SVD number of step	array(8) of double	NO	SV_vnNoStep
Switch current rating	double	YES	SW_dRating
Switch bus 1 handle	long	NO	SW_nBus1Hnd
Switch bus 2 handle	long	NO	SW_nBus2Hnd
Switch default position flag: 1- normally open; 2- normally close; 0-Not defined	long	YES	SW_nDefault
Switch in-service flag: 1- active; 2- out-of-service	long	YES	SW_nInService
<u>Switch relay group handle 1</u>	long	NO	SW_nRlyGrHnd1
<u>Switch relay group handle 2</u>	long	NO	SW_nRlyGrHnd2
Switch position flag: 1- close; 0- open	long	YES	SW_nStatus
<u>Switch ID</u>	string	YES	SW_sID
Switch name	string	YES	SW_sName
<u>Switch out of service date</u>	string	YES	SW_sOffDate
<u>Switch in service date</u>	string	YES	SW_sOnDate
System MVA base	double	NO	SY_dBaseMVA
System number of buses	long	NO	SY_nNObus
System number of generators	long	NO	SY_nNOgen
System number of transmission lines	long	NO	SY_nNOline
System number of loads	long	NO	SY_nNOload
System number of phase shifter	long	NO	SY_nNOps
System number of series capacitors	long	NO	SY_nNOseriescap
System number of shunts	long	NO	SY_nNOshunt

Table 3.3 (Cont.)

Parameter	Data type	Write Access	Parameter Code
System number of 2-winding transformers	long	NO	SY_nNOxfmr
System number of 3-winding transformers	long	NO	SY_nNOxfmr3
System data: comment	string	NO	SY_sFComment
3-winding transformer base MVA for per-unit quantities	long	YES	X3_dBaseMVA
3-winding transformer LTC center tap	double	YES	X3_dLTCCenterTap
3-winding transformer B	double	YES	X3_dB
3-winding transformer B0	double	YES	X3_dB0
3-winding transformer base MVA for per-unit quantities	double	YES	X3_dBaseMVA
3-winding transformer LTC step size	double	YES	X3_dLTCstep
3-winding transformer LTC max tap	double	YES	X3_dMaxTap
3-winding transformer LTC min controlled quantity limit	double	YES	X3_dMaxVW
3-winding transformer LTC min tap	double	YES	X3_dMinTap
3-winding transformer LTC controlled quantity limit	double	YES	X3_dMinVW
3-winding transformer LTC controlled quantity limit	double	YES	X3_dMinVW
3-winding transformer MVA rating $n$ , $n=1,2,3$	double	YES	X3_dMVAn
3-winding transformer R0ps	double	YES	X3_dR0ps
3-winding transformer R0pt	double	YES	X3_dR0pt
3-winding transformer R0st	double	YES	X3_dR0st
3-winding transformer RG1	double	YES	X3_dRG1
3-winding transformer RG2	double	YES	X3_dRG2
3-winding transformer RG3	double	YES	X3_dRG3
3-winding transformer RGn	double	YES	X3_dRGN
3-winding transformer Rps	double	YES	X3_dRps
3-winding transformer Rpt	double	YES	X3_dRpt
3-winding transformer Rst	double	YES	X3_dRst
3-winding transformer winding 1 tap kV	double	YES	X3_dTap1
3-winding transformer winding 2 tap kV	double	YES	X3_dTap2
3-winding transformer winding 3 tap kV	double	YES	X3_dTap3
3-winding transformer X0ps	double	YES	X3_dX0ps
3-winding transformer X0pt	double	YES	X3_dX0pt
3-winding transformer X0st	double	YES	X3_dX0st
3-winding transformer XG1	double	YES	X3_dXG1
3-winding transformer XG2	double	YES	X3_dXG2
3-winding transformer XG3	double	YES	X3_dXG3
3-winding transformer XGn	double	YES	X3_dXGN
3-winding transformer Xps	double	YES	X3_dXps
3-winding transformer Xpt	double	YES	X3_dXpt
3-winding transformer Xst	double	YES	X3_dXst
3-winding transformer auto transformer flag: 1-true; 0-false	long	YES	X3_nAuto
3-winding transformer bus 1 handle	long	NO	X3_nBus1Hnd
3-winding transformer bus 2 handle	long	NO	X3_nBus2Hnd
3-winding transformer bus 3 handle	long	NO	X3_nBus3Hnd
3-winding transformer fictitious bus number	long	NO	X3_nFictBusNo
3-winding transformer in-service flag: 1- active; 2- out-of-service	long	YES	X3_nInService
3-winding transformer LTC tag ganged flag: 0-False; 1-True	long	YES	X3_nLTCGanged
3-winding transformer relay group 1 handle	long	NO	X3_nRlyGr1Hnd
3-winding transformer relay group 2 handle	long	NO	X3_nRlyGr2Hnd
3-winding transformer relay group 3 handle	long	NO	X3_nRlyGr3Hnd
3-winding transformer winding 1 config	string	YES	X3_sCfg1
3-winding transformer winding 2 config	string	YES	X3_sCfg2
3-winding transformer winding 2 config in test	string	YES	X3_sCfg2T
3-winding transformer winding 3 config	string	YES	X3_sCfg3
3-winding transformer winding 3 config in test	string	YES	X3_sCfg3T
3-winding transformer circuit ID	string	NO	X3_sID
3-winding transformer name	string	YES	X3_sName

Table 3.3 (Cont.)

Parameter	Data type	Write Access	Parameter Code
3-winding transformer out of service date	string	YES	X3_sOffDate
3-winding transformer in service date	string	YES	X3_sOnDate
2-winding transformer B	double	YES	XR_dB
2-winding transformer Bo	double	YES	XR_dB0
2-winding transformer B1	double	YES	XR_dB1
2-winding transformer B10	double	YES	XR_dB10
2-winding transformer B2	double	YES	XR_dB2
2-winding transformer B20	double	YES	XR_dB20
2-winding transformer base MVA for per-unit quantities	double	YES	XR_dBaseMVA
2-winding transformer G1	double	YES	XR_dG1
2-winding transformer G10	double	YES	XR_dG10
2-winding transformer G2	double	YES	XR_dG2
2-winding transformer G20	double	YES	XR_dG20
2-winding transformer LTC center tap	double	YES	XR_dLTCcenterTap
2-winding transformer LTC step size	double	YES	XR_dLTCstep
2-winding transformer LTC max tap	double	YES	XR_dMaxTap
2-winding transformer LTC min controlled quantity limit	double	YES	XR_dMaxVW
2-winding transformer LTC min tap	double	YES	XR_dMinTap
2-winding transformer LTC max controlled quantity limit	double	YES	XR_dMinVW
2-winding transformer MVA rating $n$ , $n=1,2,3$	double	YES	XR_dMVAn
2-winding transformer R	double	YES	XR_dR
2-winding transformer Ro	double	YES	XR_dR0
2-winding transformer Rg1	double	YES	XR_dRG1
2-winding transformer Rg2	double	YES	XR_dRG2
2-winding transformer Rgn	double	YES	XR_dRGN
2-winding transformer winding 1 tap kV	double	YES	XR_dTap1
2-winding transformer winding 2 tap kV	double	YES	XR_dTap2
2-winding transformer X	double	YES	XR_dX
2-winding transformer Xo	double	YES	XR_dX0
2-winding transformer Xg1	double	YES	XR_dXG1
2-winding transformer Xg2	double	YES	XR_dXG2
2-winding transformer Xgn	double	YES	XR_dXGN
2-winding transformer auto transformer flag: 1-true; 0-false	long	YES	XR_nAuto
2-winding transformer bus1 handle	long	NO	XR_nBus1Hnd
2-winding transformer bus 2 handle	long	NO	XR_nBus2Hnd
2-winding transformer in-service flag: 1- active; 2- out-of-service	long	YES	XR_nInService
2-winding transformer LTC control bus handle	long	NO	XR_nLTCctrlBusHnd
2-winding transformer LTC tag ganged flag: 0-False; 1-True	long	YES	XR_nLTCGanged
2-winding transformer LTC adjustment priority	long	YES	XR_nLTCPriority
2-winding transformer LTC side: 1; 2; 0	long	NO	XR_nLTCside
2-winding transformer LTC type: 1- control voltage; 2- control MVAR	long	NO	XR_nLTCtype
2-winding transformer metered bus handle	long	NO	XR_nMetered
2-winding transformer side 1 relay group handle	long	NO	XR_nRlyGr1Hnd
2-winding transformer side 2 relay group handle	long	NO	XR_nRlyGr2Hnd
2-winding transformer winding 1 config	string	YES	XR_sCfg1
2-winding transformer winding 2 config	string	YES	XR_sCfg2
2-winding transformer winding 2 config in test	string	YES	XR_sCfg2T
2-winding transformer circuit ID	string	NO	XR_sID
2-winding transformer name	string	NO	XR_sName
2-winding transformer out of service date	string	YES	XR_sOffDate
2-winding transformer in service date	string	YES	XR_sOnDate



NOTE:

(1) Call *GetData* function with this code and string parameter label to retrieve the setting value;

Call *SetData* function with this code and tab delimited string with parameter label and value to set the setting value.

---

## 3.4 Short Circuit Solution

You can call the function **DoFault** to simulate short circuits. The options available in this function are the same as those in the Fault | Specify Classical Fault command in *OneLiner*.

After a fault simulation, you can copy the voltage and current results into *PowerScript* program variables for reports or for further processing.

Because a single fault simulation command in *OneLiner* can initiate simulation of multiple faults of different fault types and branch outages, you must call the function **ShowFault** or **PickFault** to specify the fault of interest.

You can use the functions **GetSCVoltage** and **GetSCCurrent** to retrieve post-fault voltage and current at the terminals of a network object. You can get the total fault current by calling **GetSCCurrent** with the pre-defined handle of short circuit solution, HND\_SC.

*PowerScript* function **GetRelayTime** is used to retrieve operating time of a given relay.

<b>DoFault</b>	Simulates one or more faults.
<b>FaultDescription</b>	Returns the fault description in a string.
<b>GetRelayTime</b>	Gets the operating time of a relay.
<b>GetSCCurrent</b>	Gets the current flow for a load, generator, shunt or branch.
<b>GetSCVoltage</b>	Gets the voltage at a bus or at the end buses of a branch.
<b>PickFault</b>	Selects one of the fault results.
<b>ShowFault</b>	Selects one of the fault results and show it on one-line diagram.

---

## 3.5 Power Flow Solution

You can use the function **DoPF** to simulate a power flow case. The options available in this function are the same as those in the Solve | Power Flow command in *Power Flow*. After a simulation in the *Power Flow Program*, you can copy the voltage and current results into *PowerScript* program variables for generating report and for further processing.

The functions **GetPFVoltage** and **GetPFCurrent** retrieves solution voltage and current at terminals of the given network device. The function **GetFlow** calculates the power flow.

<b>DoPF</b>	Simulates a power flow.
<b>GetFlow</b>	Gets the MW+jMVAR flow for a load, generator, shunt or branch.
<b>GetPFCurrent</b>	Gets the current flow for a load, generator, shunt or branch.
<b>GetPFVoltage</b>	Gets the voltage at a bus or at the end buses of a branch.

---

## 3.6 Reserved names and keywords

*PowerScript* programmers must avoid using following names in their programs:

- BASIC language reserved keywords. (see Basic Language Reference Help for full listing)
- *PowerScript* predefined handles in table 3.1
- *PowerScript* equipment type codes in table 3.2
- *PowerScript* parameter codes in table 3.3.
- Other names in the format “??\*\_” where ? represents at most one character and \* represent at least one character.

---

## 4.1 PowerScript Functions

This section lists all the *PowerScript* functions in alphabetical order of the function name. The following is an alphabetical list:

<b>AppExit</b>	Exit OneLiner/Power Flow application.
<b>BusPicker</b>	Bus selection dialog box.
<b>BoundaryEquivalent</b>	Creates a network equivalent.
<b>ComputeRelayTime</b>	Compute operating time of a protective device.
<b>DoArcFlash</b>	Run arc-flash calculation.
<b><u>DoBreakerRating</u></b>	Run breaker rating study.
<b><u>DoFault</u></b>	Simulates one or more faults.
<b><u>DoSteppedEvent</u></b>	Do stepped-event simulation.
<b>DoVS</b>	Voltage sag analysis.
<b>DoVSEx</b>	Voltage sag analysis with fault duration computed using stepped-event analysis.
<b>DoPF</b>	Simulate a power flow.
<b>ErrorString</b>	Returns text description of the last script function error.
<b>ExportNetwork</b>	Export network to ASPEN text data file.
<b>ExportNetworkPSSE</b>	Export network to PSS/E RAW and SEQ text data files.
<b>EquipmentType</b>	Gets the equipment type given a handle.
<b>FaultDescription</b>	Returns the fault description in a string.
<b>FaultSelector</b>	Fault selection dialog box.
<b><u>FileOpenDialog</u></b>	Open File dialog box.
<b><u>FileSaveDialog</u></b>	Save file dialog box.
<b>FindBusByName</b>	Returns the handle of the bus having a given name and nominal kV.
<b><u>FindEquipmentByTag</u></b>	Returns the handle of the object having a given tag in its tag string .
<b><u>FolderSelectDialog</u></b>	Folder selection dialog box.
<b>FullBusName</b>	Composes a string with bus number, bus name and nominal kV given a handle.
<b><u>FullRelayName</u></b>	Composes a string with relay type, name and branch location.
<b>GetBusEquipment</b>	Returns the handle of the next piece of equipment of a given type at a bus.
<b>GetData</b>	Puts a datum of interest into a program variable.
<b>GetEquipment</b>	Returns the handle of the next piece of network equipment of the given type.
<b>GetFlow</b>	Gets the power flow for a load, generator, shunt or branch in load flow simulation.
<b>GetObjMemo</b>	Gets the memo string of an object.
<b>GetObjTags</b>	Gets the tags string of an object.
<b>GetOlrfFileName</b>	Print full path name of current OLR file.
<b>GetPFCurrent</b>	Gets the current flow for a load, generator, shunt or branch in load flow simulation.
<b>GetPFVoltage</b>	Gets the voltage at a bus or at the end buses of a branch in load flow simulation.
<b>GetProgramVersion</b>	Print string with program version and build information.

<b>GetPSCVoltage</b>	Gets the pre-fault voltage at a bus or at the end buses of a branch.
<b>GetRelay</b>	Returns the handle of the next protective device in a relay group.
<b>GetRelayTime</b>	Gets the operating time of a relay in a fault.
<b>GetSCCurrent</b>	Gets the current flow for a load, generator, shunt or branch in a fault.
<b>GetSCVoltage</b>	Gets the post fault voltage at a bus or at the end buses of a branch.
<b>GetSteppedEvent</b>	Gets detailed result of a step in stepped-event simulation.
<b>GetVSVoltage</b>	Gets the voltage sag analysis result.
<b>LoadDataFile</b>	Open a binary or text data file.
<b>Locate1LObj</b>	Locate an object on the 1-line diagram.
<b>MakeOutageList</b>	Make branch outage list for fault simulation.
<b>NextBusByName</b>	Returns the handle of the next bus in the bus list that is sorted by name.
<b>NextBusByNumber</b>	Returns the handle of the next bus in the bus list that is sorted by number.
<b>PickFault</b>	Selects one of the available fault results.
<b>PostData</b>	Validates all the updated parameters of an equipment and updates the database.
<b>ProgressDialog</b>	Progress dialog box.
<b>PrintTTY</b>	Puts a message on the TTY window.
<b>PrintObj1LPF</b>	Return a text description of network database object
<b>ReadChangeFile</b>	Read a change file
<b>SaveDataFile</b>	Save network data to file.
<b>SetData</b>	Changes the value of a parameter in a temporary object.
<b>SetObjMemo</b>	Changes the memo of a object.
<b>SetObjTags</b>	Changes the tags of a object.
<b>ShowFault</b>	Selects one of the available fault results and show it on the one-line diagram.

## Function AppExit

Function AppExit( byVal nFlag ) as Long

**Purpose:** Terminate application

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

nFlag	[in]	Set bit 1 of this flag to save changes in OLR file
		Set bit 2 of this flag to suppress on-screen dialog boxes

**Return value:**

**Remarks:** You must stop script execution immediately after calling this function. Running PowerScript code after AppExit() can lead to undetermined outcomes.

**Example:**

Call AppExit( 1+2 )	`	Save changes and exit OneLiner/Power Flow. Show no dialog box
Stop	`	Stop script

## Function BusPicker

```
Function Function BusPicker( ByRef sWindowText$, ByRef vnBusHnd1() as Long, _  
                             ByRef vnBusHnd2() as Long ) As Long
```

**Purpose:** Display bus selection dialog.

**Works in:** *OneLiner / Power Flow.*

### Parameters:

sWindowText [in] Dialog title.  
vnBusHnd1 [in] Array of handle number of buses that are already selected when the dialog opens. The list must be terminated with zero.  
vnBusHnd2 [out] Array of handle number of buses that user selected. The list must be terminated with zero.

### Return value:

1 User clicked OK  
0 User clicked Cancel

**Remarks:** Dimension of the two arrays must be adequate to store all buses in the network.

### Example:

```
sWindowText$ = "My Bus Picker"  
nPicked& = BusPicker( sWindowText$, vnBusHnd1, vnBusHnd2 ) strTemp$ = ""  
For ii& = 1 to nPicked  
    nBsHnd& = vnBusHnd2(ii)  
    strName$ = FullBusName( nBsHnd& )  
    strTemp$ = strTemp$ + strName$ + Chr(10) + Chr(13)  
Next
```

## Function BoundaryEquivalent

```
Function BoundaryEquivalent( ByRef sEquFileName$, ByRef vnBusList() as Long,  
    ByRef vdFltOpt() as Double ) As Long\
```

**Purpose:** Run voltage sag analysis

**Works in:** *OneLiner* only.

### Parameters:

sEquFileName	[in]	Path name of equivalent OLR file.
vnBusList	[in]	Array of handles of buses to be retained in the equivalent. The list is terminated with value -1
vdFltOpt	[in]	study parameters
vdFltOpt(1)		- Per-unit elimination threshold
vdFltOpt(2)		- Keep existing equipment at retained buses( 1- set; 0- reset)
vdFltOpt(3)		- Keep all existing annotations (1- set; 0-reset)

### Return value:

1	success
0	failure

**Remarks:** .

### Example:

```
Sub main  
    dim BusList(40) As long  
    dim Options(5) As double  
    OLRFile$ = "c:\TestData\PowerScript\Sample30.olr"  
    EqOLRFile$ = " c:\TestData\PowerScript\Equivalent.olr"  
    If 0 = LoadDataFile( OLRFile$ ) Then  
        Print "Error opening OLR file"  
        Stop  
    End If  
    nBusHnd& = 0  
    nCount = 0  
    While NextBusByName( nBusHnd& ) > 0  
        Call GetData( nBusHnd&, BUS_dKVnorminal, dVall# )  
        If dVall# > 100 Then  
            nCount = nCount + 1  
            BusList(nCount) = nBusHnd  
        End If  
    Wend  
    BusList(nCount+1) = -1  
    Options(1) = 99  
    Options(2) = 1  
    Options(3) = 0  
    If BoundaryEquivalent( EqOLRFile, BusList, Options ) Then  
        Print "OK"  
    Else  
        Print "Not OK"  
    End If  
End Sub
```

## Function ComputeRelayTime

```
Function ComputeRelayTime( ByVal nHandle&, ByRef vdCurMag() as double, _  
    ByRef vdCurAng() as double, ByRef vdVMag() as double, _  
    ByRef vdVAng() as double, ByVal dVpreMag#, _  
    ByVal dVpreAng#, ByRef dTime, ByRef sDevice ) As Long
```

**Purpose:** Computes operating time for a fuse, recloser, an overcurrent relay (phase or ground), or a distance relay (phase or ground) at given currents and voltages.

**Works in:** *OneLiner* only.

### Parameters:

nHandle	[in] relay handle
vdCurMag()	[in] array of relay current magnitude in: phase A, B, C and if applicable Io currents in neutral of transformer windings P and S.
vdCurAng()	[in] array of relay current angles
vdVMag()	[in] array of relay voltage magnitude in phase A, B and C
vdVAng()	[in] array of relay voltage angle
dVpreMag	[in] relay pre-fault positive sequence voltage magnitude
dVpreAng	[in] relay pre-fault positive sequence voltage angle
dTime	[out] relay operating time in seconds
sDevice	[out] relay operation code: NOP No operation ZGn Ground distance zone <i>n</i> tripped ZPn Phase distance zone <i>n</i> tripped Ix Overcurrent relay operating quantity: Ia, Ib, Ic, Io, I2, 3Io, 3I2

### Return value:

1	success
0	failure

**Remarks:** All calls to this function must be preceded by a call to ShowFault or PickFault function.

Relay current multiplying factor will be applied to relay current result from simulation before time calculation.

### Example:

```
Call ComputeRelayTime(nRlyHnd, vdImag, vdIang, vdVmag, vdVang, dVpreMag, dVpreAng, _  
    dTime#, sDevice$ )  
sOutput$ = " T=" & Format(dTime,"0.00") & "(" & sDevice & ")" & _  
    ";Ia=" & Format(vdImag(1),"0.0") & "@" & Format(vdIang(1),"0.0") & _  
    ";Ib=" & Format(vdImag(2),"0.0") & "@" & Format(vdIang(2),"0.0") & _  
    ";Ic=" & Format(vdImag(3),"0.0") & "@" & Format(vdIang(3),"0.0") & _  
    ";IN1=" & Format(vdImag(4),"0.0") & "@" & Format(vdIang(4),"0.0") & _  
    ";IN2=" & Format(vdImag(5),"0.0") & "@" & Format(vdIang(5),"0.0") & _  
    ";Va=" & Format(vdVmag(1),"0.0") & "@" & Format(vdVang(1),"0.0") & _  
    ";Vb=" & Format(vdVmag(2),"0.0") & "@" & Format(vdVang(2),"0.0") & _  
    ";Vc=" & Format(vdVmag(3),"0.0") & "@" & Format(vdVang(3),"0.0")
```



## Function DoBreakerRating

```
Function DoBreakerRating( ByRef vnScope() As Long, _  
    ByVal dRatingThreshold as Double, ByVal dOutputOpt as Double, _  
    ByVal nOptionalReportFlag as Long, _  
    ByRef sReportTXT$, ByRef sReportCSV$, ByRef sConfigFile$ ) As Long
```

**Purpose:** Run breaker rating study.

**Works in:** *OneLiner* only.

### Parameters:

vnScope	[in] Study scope
vnScope[1]	Breaker rating standard: 0-ANSI/IEEE; 1-IEC.
vnScope[2]	Bus selection: 0-All buses; 1-in Area; 2-in Zone; 3- selected buses
vnScope[3]	Selected area or zone number.
vnScope[4]	or list of handle number of selected buses. The last element in the list must be -1.
...	must be -1.
dRatingThreshold	[in] Percent rating threshold.
dOutputOpt	[in] Rating output option: 0- Output only overduty cases; 1- Output all cases; OR Floating number S ( $0 < S < 1$ ) - Check only breakers at buses where ratio "Bus fault current / Breaker rating" exceeds S.
OptionalReportFlag	[in] Integer number flag. Enable various bits to enable optional sections in rating report: Bit 1- Detailed fault simulation result; Bit 2- Breaker name plate data; Bit 3- List of connected equipment.
sReportTXT	[in] Full path name of text report file. Set to emty to omit text report.
sReportCSV	[in] Full path name of CSV report file. Set to emty to omit CSV report.
sConfigFile	[in] Full path name of breaker rating configuration file to apply in this study. Set to emty to omit reading configuration file.

### Return value:

1	success
0	failure

### Example:

```
Sub Main  
    dim vnScope(5) As long  
    sWorkDir$ = "e:\data\PowerScriptDoBreakerRating\  
    sOLRFile$ = sWorkDir & "SAMPLE30_noBreaker.olr"  
    sChangeFile$ = sWorkDir & "breaker1.CHF"  
    sConfigFile$ = sWorkDir & "checkoption1.OSF"  
    If 0 = LoadDataFile( sOLRFile$ ) Then  
        Print "Error: LoadDataFile"  
        Stop  
    End If  
    If 1000 <= ReadChangeFile( sChangeFile, 1 ) Then  
        Print "Error: ReadChangeFile"  
        Stop  
    End If  
    vnScope(1) = 0  
    dThreshold# = 80  
    nOptionalReport# = 1 + 2 + 4  
    dOutputOpt# = 1  
    sReportTXT$ = sWorkDir & "rating" & ".txt"  
    sReportCSV$ = sWorkDir & "rating" & ".csv"  
    If 0 = DoBreakerRating(vnScope, dThreshold, dOutputOpt, nOptionalReport, _  
        sReportTXT$, sReportCSV$, sConfigFile$ ) Then  
        Print "Error: " & testNo$ & "-DoBreakerRating"  
    End If  
End Sub
```

## Function DoFault

```
Function DoFault( ByVal nDevHnd&, ByVal vnFltConn() As Long, _  
    ByVal vdFltOpt() as Double, ByVal vnOutageOpt() as Long, _  
    ByVal vnOutageLst() as Long, _  
    ByVal dFltR#, ByVal dFltX#, ByVal nClearPrev& ) As Long
```

**Purpose:** Simulate one or more faults.

**Works in:** *OneLiner* only.

### Parameters:

nDevHnd	[in] handle of a bus, branch or a relay group.
vnFltConn	[in] fault connection flags. 0 - reset vnFltConn(1) - 1=3PH vnFltConn(2) - 1=2LG_BC;2= 2LG_CA; 2LG_AB vnFltConn(3) - 1=1LG_A; 1=1LG_B; 1=1LG_C; vnFltConn(4) - 1=LL_BC; 1=LL_CA; 1=LL_AB;
vdFltOpt	[in] fault options flags. 1 – set; 0 - reset vdFltOpt(1) - Close-in vdFltOpt(2) - Close-in w/ outage vdFltOpt(3) - Close-in with end opened vdFltOpt(4) - Close-in with end opened w/ outage vdFltOpt(5) - Remote bus vdFltOpt(6) - Remote bus w/ outage vdFltOpt(7) - Line end vdFltOpt(8) - Line end w/ outage vdFltOpt(9) - Intermediate % vdFltOpt(10) - Intermediate % w/ outage vdFltOpt(11) - Intermediate % with end opened vdFltOpt(12) - Intermediate % with end opened w/ outage vdFltOpt(13) - Auto seq. Intermediate % from (*) vdFltOpt(14) - Auto seq. Intermediate % to (*) <u>vdFltOpt(15) - Outage line grounding admittance in mho (***)</u> .
vnOutageLst	[in] list of handles of branches to be outaged; 0 terminated
vnOutageOpt	[in] branch outage option flags. 1 – set; 0 - reset vnOutageOpt(1) - one at a time vnOutageOpt(2) - two at a time vnOutageOpt(3) - all at once vnOutageOpt(4) - breaker failure (**)
dFltR	[in] fault resistance, in Ohm
dFltX	[in] fault reactance, in Ohm
nClearPrev	[in] clear previous result flag. 1 – set; 0 - reset

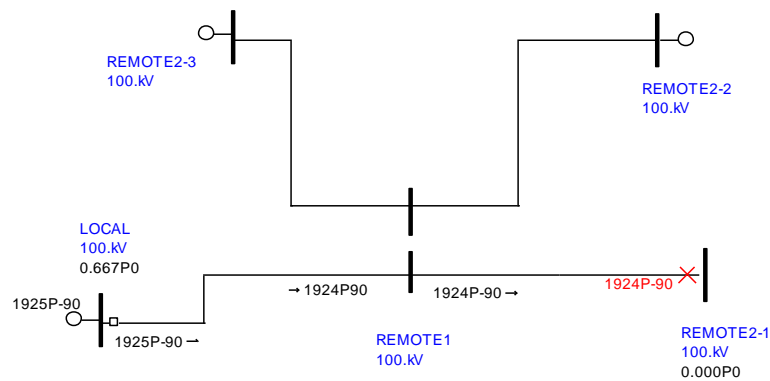
### Return value:

>0	Index number of the last fault simulated by the function
0	failure

### Remarks:

- (\*) To simulate a single intermediate fault without auto-sequencing, set both vdFltOpt(13) and vdFltOpt(14) to zero

(\*\*) Set this flag to 1 to simulate breaker open failure condition that caused two lines that share a common breaker to be separated from the bus while still connected to each other as shown in the picture below. TC\_BRANCH handle of the two lines must be included in the array vnOutageLst.



(\*\*\*)To simulate fault with outaged lines that are solidly grounded at both ends, use 999999.0 for this parameter.

**Example:**

```

' Simulate the faults
If DoFault( nBusHnd, vnFltConn, vdFltOpt, vnOutageOpt, vnOutageLst, dFltR, _
    dFltX, nClearPrev ) = 0 Then GoTo HasError

' Print output
sBusName1 = FullBusName( nBusHnd )
Print #1, "Fault simulation at Bus: ", sBusName1
Print #1, ""
Print #1, "
Phase A
Phase B
Phase C"
Print #1, ""

' Start from the first fault
If PickFault( 1 ) = 0 Then GoTo HasError
Do
    If GetSCCurrent( HND_SC, vdVal1, vdVal2, 4 ) = 0 Then GoTo HasError
    Print #1, FaultDescription(); Chr(10); _
    "
    Format( vdVal1(1), "###0.0"); "@"; Format( vdVal2(1), "#0.0"), Space(5), _
    Format( vdVal1(2), "###0.0"); "@"; Format( vdVal2(2), "#0.0"), Space(5), _
    Format( vdVal1(3), "###0.0"); "@"; Format( vdVal2(3), "#0.0")
Loop While PickFault( SF_NEXT ) > 0
Print "Simulation complete. Report is in " & fileName
Stop
HasError:
Print "Error: ", ErrorString( )
Stop

```

## Function DoArcFlash

```
Function DoAcrFlash( byVal nBusHnd&, byRef vdFltOpt(), byRef vdResult() )  
    as Long
```

**Purpose:** Run arc flash calculator

**Works in:** *OneLiner* only.

### Parameters:

nBusHnd	[in] handle of the study bus.
vdFltOpt	[in] study parameters
vdFltOpt[1]	Equipment category: 0-Switchgear; 1-Cable; 2- open air; 3- MCC's and panelboards 1kV or lower
vdFltOpt[2]	Grounding: 0-Ungrounded;1-Grounded
vdFltOpt[3]	Enclosure: 0-Not enclosed;1-Enclosed
vdFltOpt[4]	Conductor gap in mm
vdFltOpt[5]	Working distance in inches
vdFltOpt[6]	Fault clearing: -1- Auto; -2- manual clearing time; -3- Stepped-event-analysis; >0- clearing device handle
vdFltOpt[7]	Breaker interrupting time in cycles or manual clearing time in seconds
vdFltOpt[8]	Ignore-2-second flag: 0-reset; 1-set
vdFltOpt[9]	Number of tiers to include in protective device list
vdResult	[out] study result
vdResult[1]	Bolted 3PH fault current (kA)
vdResult[2]	Arcing current (kA)
vdResult[3]	Clearing device handle at 100% current
vdResult[4]	Clearing time at 100% current (seconds)
vdResult[5]	Incident energy at 100% current (cal/cm2)
vdResult[6]	Clearing device handle at 85% current
vdResult[7]	Clearing time at 85% current
vdResult[8]	Incident energy at 85% current
vdResult[9]	Required PPE level
vdResult[10]	PPE cat. 1 flash hazard boundary (inches)
vdResult[11]	PPE cat. 2 flash hazard boundary (inches)
vdResult[12]	PPE cat. 3 flash hazard boundary (inches)
vdResult[13]	PPE cat. 4 flash hazard boundary (inches)
vdResult[14]	Above PPE cat. 4 flash hazard boundary (inches)

### Return value:

1	success
0	failure

### Remarks:

### Example:

```
vdOption(1) = 0 '0-Switchgear; 1-Cable; 2- open air  
vdOption(2) = 1 '0-Ungrounded;1-Grounded  
vdOption(3) = 0 '0-No enclosure;1-Enclosed  
vdOption(4) = 153 'Conductor gap in mm  
vdOption(5) = 36 'Working distance in inches  
vdOption(6) = -1 'Fault clearing:-1- Auto;-2- manual clearing Time;>0- clearing device handle  
vdOption(7) = 1.5 'Breaker interrupting time in cycles or manual clearing time in seconds  
vdOption(8) = 1 'Ignore 2 second flag: 0-reset; 1-set;  
If 0 = DoAcrFlash( nBusHnd, vdOption, vdResult ) Then GoTo HasError
```

```
Print  vdResult(1), ",", vdResult(2), _  
      ",", FullRelayName(vdResult(3)), ",", vdResult(4), ",", vdResult(5), _  
      ",", FullRelayName(vdResult(6)), ",", vdResult(7), ",", vdResult(8), _  
      ",", vdResult(9), ",", vdResult(10), ",", vdResult(11), ",", vdResult(12), _  
      ",", vdResult(13)
```

## Function DoSteppedEvent

```
Function DoSteppedEvent (ByVal nDevHnd&, ByRef vdFltOpt() as Double, _  
                        ByRef vnOpt() As Long, ByVal nTiers as Long ) As Long
```

**Purpose:** Simulate one or more faults.

**Works in:** *OneLiner* only.

### Parameters:

nDevHnd	[in] handle of a bus or a relay group.
vdFltOpt	[in] fault simulation options
vdFltOpt (1)	- Fault connection code 1=3LG 2=2LG_BC,3=2LG_CA,4=2LG_AB 5=1LG_A,6=1LG_B,7=1LG_C 8=LL_BC,9=LL_CA,10=LL_AB
vdFltOpt (2)	- Intermediate percent between 0.01-99.99. 0 for a close-in fault. This parameter is ignored if nDevHnd is a bus handle.
vdFltOpt (3)	- Fault resistance, ohm
vdFltOpt (4)	- Fault reactance, ohm
vdFltOpt (4+1)	- Zero or Fault connection code for additional user event
vdFltOpt (4+2)	- Time of additional user event, seconds.
vdFltOpt (4+3)	- Fault resistance in additional user event, ohm
vdFltOpt (4+4)	- Fault reactance in additional user event, ohm
vdFltOpt (4+5)	- Zero or Fault connection code for additional user event
...	
vnOpt	[in] Study options flags. 1 – set; 0 - reset
vnOpt (1)	- Consider OCGnd operations
vnOpt (2)	- Consider OCPH operations
vnOpt (3)	- Consider DSGnd operations
vnOpt (4)	- Consider DSPH operations
vnOpt (5)	- Consider Protection scheme operations
nTiers	[in] Study extent

### Return value:

1	success
0	failure

**Remarks:** After successful completion of DoSteppedEvent() you must call function GetSteppedEvent() to retrieve detailed result of each step in the simulation.

### Example:

```
vdFltOpt(1) = 5 '1LG phase A  
vdFltOpt(2) = 10 'Intermediate percent between 0.01-99.99  
vdFltOpt(3) = 0 'Fault resistance  
vdFltOpt(4) = 0 'Fault reactance  
vdFltOpt(5) = 0 'Zero or nFltconn of additional event  
vnDevOpt(1) = 1 'Consider OCGnd  
vnDevOpt(2) = 1 'Consider OCPH  
vnDevOpt(3) = 1 'Consider DSGnd  
vnDevOpt(4) = 1 'Consider DSPH  
  
nTiers& = 5  
  
If 0 = DoSteppedEvent( nBusHnd, vdFltOpt, vnDevOpt, nTiers ) Then GoTo HasError  
  
' Call GetSteppedEvent with 0 to get total number of events simulated  
nSteps = GetSteppedEvent( 0, dTime#, dCurrent#, nUserEvent&, sEventDesc$, sFaultDest$ )
```

```

Print "Stepped-event simulation completed successfully with ", nSteps-1, " events"

For ii = 1 to nSteps
    Call GetSteppedEvent( ii, dTime#, dCurrent#, nUserEvwent&, sEventDesc$, sFaultDest$ )
    Print "Fault: ", sFaultDest$
    Print sEventDesc$
    Print "Time = ", dTime, " Current= ", dCurrent
Next

Stop
HasError:
Print "Error: ", ErrorString( )
Stop

```

## Function DoVS

```
Function DoVS (ByVal nDevHnd&, ByRef vnFltConn() As Long, _  
              ByRef vdFltOpt() as Double, ByRef sCSVFileName$ ) As Long
```

**Purpose:** Run voltage sag analysis

**Works in:** *OneLiner* only.

### Parameters:

nDevHnd	[in] handle of the monitored bus.
vnFltConn	[in] fault connection flags. 1 – set; 0 - reset
	vnFltConn(1) - 3PH
	vnFltConn(2) - 2LG
	vnFltConn(3) - 1LG
	vnFltConn(4) - LL
vdFltOpt	[in] study parameters
	vdFltOpt(1) - Sag threshold
	vdFltOpt(2) - Line percent divided by 100 (Set to zero for no intermediate faults)
	vdFltOpt(3) - Output All Buses flag (1- set; 0-reset)
	vdFltOpt(4) - Fault impedance - reactance
	vdFltOpt(5) - Fault impedance – resistance
sCSVFileName	[in] CSV output file name

### Return value:

1	success
0	failure

### Remarks:

### Example:

```
Dim vdOption(5) As Double, vdMag(4) As Double  
Dim vnFltConn(4) As Long  
  
' Output file  
CSVFile$ = "c:\0tmp\vs.csv"  
  
' Get picked bus handle  
If GetEquipment( TC_PICKED, nBusHnd& ) = 0 Then  
    Print "Must select a bus"  
    Exit Sub  
End If  
  
vnFltConn(1) = 1 '1LG  
vnFltConn(2) = 0 '2LG  
vnFltConn(3) = 0 '3PH  
vnFltConn(4) = 0 'LL  
  
vdOption(1) = 0.6 'Sag threshold  
vdOption(2) = 0.0 'Line percent  
vdOption(3) = 1 'Ouput all  
vdOption(4) = 0.0 'Zground.imag  
vdOption(5) = 0.0 'Zground.real  
  
If 0 = DoVS( nBusHnd, vnFltConn, vdOption, CSVFile ) Then GoTo HasError  
  
Print "Voltage sag simulation complete. Output is in " + CSVFile
```



## Function DoVSEx

```
Function DoVSEx (ByVal nDevHnd&, ByVal vnFltConn() As Long,  
                ByVal vdFltOpt() As Double, ByVal sCSVFileName$ ) As Long
```

**Purpose:** Run voltage sag analysis with option for computation of fault duration using stepp-event analysis

**Works in:** *OneLiner* only.

### Parameters:

nDevHnd	[in] handle of the monitored bus.
vnFltConn	[in] fault connection flags. 1 – set; 0 - reset
	vnFltConn(1) - 3PH
	vnFltConn(2) - 2LG
	vnFltConn(3) - 1LG
	vnFltConn(4) - LL
vdFltOpt	[in] study parameters
	vdFltOpt(1) - Sag threshold
	vdFltOpt(2) - Line percent divided by 100 (Set to zero for no intermediate faults)
	vdFltOpt(3) - Output All Buses flag (1- set; 0-reset)
	vdFltOpt(4) - Fault impedance - reactance
	vdFltOpt(5) - Fault impedance – resistance
	vdFltOpt(6) - 1.0 to compute fault duration using stepped-event analys. 0 otherwise.
	vdFltOpt(7) - Study extent for stepped-event analysis, in tiers. Must >= 2.
sCSVFileName	[in] CSV output file name

### Return value:

1	success
0	failure

### Remarks:

### Example:

```
Dim vdOption(7) As Double, vdMag(4) As Double  
Dim vnFltConn(4) As Long  
  
' Output file  
CSVFile$ = "c:\tmp\vs.csv"  
  
' Get picked bus handle  
If GetEquipment( TC_PICKED, nBusHnd& ) = 0 Then  
    Print "Must select a bus"  
    Exit Sub  
End If  
  
vnFltConn(1) = 1 '1LG  
vnFltConn(2) = 0 '2LG  
vnFltConn(3) = 0 '3PH  
vnFltConn(4) = 0 'LL  
  
vdOption(1) = 0.6 'Sag threshold  
vdOption(2) = 0.0 'Line percent  
vdOption(3) = 1 'Output all  
vdOption(4) = 0.0 'Zground.imag  
vdOption(5) = 0.0 'Zground.real  
vdOption(6) = 1.0 'Turn on stepped-event analysis  
vdOption(7) = 5.0 'Study extent of 5 tiers from faulted bus.  
  
If 0 = DoVSEx( nBusHnd, vnFltConn, vdOption, CSVFile ) Then GoTo HasError  
  
Print "Voltage sag simulation complete. Output is in " + CSVFile
```

## Function DoPF, DoPF10 and DoPF11

```
Function DoPF( ByVal nSlkBusHnd&, ByRef vdPFSettings() as double, _  
              ByRef vnPFOpt() as double, ByVal nMethod As Long ) As Long
```

```
Function DoPF10( ByVal nSlkBusHnd&, ByRef vdPFSettings() as double, _  
               ByRef vnPFOpt() as double, ByVal sAreas$, _  
               ByVal nMethod As Long ) As Long
```

```
Function DoPF11( ByVal nSlkBusHnd&, ByRef vdPFSettings() as double, _  
               ByRef vnPFOpt() as double, ByVal sAreas$, _  
               ByVal nMethod As Long ) As Long
```

**Purpose:** Perform a power flow study.

**Works in:** *Power Flow* only.

### Parameters:

nSlkBusHnd	[in] Handle of the system slack bus. If nSlkBusHnd = -9999: use slack bus selection from previous power flow run.
vdPFSettings	[in] Power Flow settings vdPFCriteria(1) – Maximum number of iterations vdPFCriteria(2) – MW tolerance vdPFCriteria(3) – MVAR tolerance vdPFCriteria(4) – MW adjustment threshold vdPFCriteria(5) – MVAR adjustment threshold
vnPFOpt	[in] Power Flow options 1 – set; 0 – reset; vnPFOpt(1) – Start from last solution vnPFOpt(2) – Enforce generator VAR limit vnPFOpt(3) – Enforce transformer tap vnPFOpt(4) – Enforce area interchange vnPFOpt(5) – Enable generator remote voltage control vnPFOpt(6) – Enable switched shunt simulation vnPFOpt(7) – Enable phase shifter simulation vnPFOpt(8) – Reset LTC taps vnPFOpt(9) – Enable solution monitor vnPFOpt(10) – ( <i>DoPF11 only</i> ) Start with dc line communication transformer tap at 1.0 vnPFOpt(11) – ( <i>DoPF11 only</i> ) Stagger automatic control adjustments vnPFOpt(12) – ( <i>DoPF11 only</i> ) Move LTC tap to nearest step after convergence
sAreas	[in] List of areas where automatic controls are enforced.
nMethod	[in] Power flow solution method 0 – Newton-Raphson 1 – Fast decoupled

### Return value:

1	success
0	failure

**Remarks:** Use DoPF10 and DoPF11 to take advantage of additional power flow solution options available in Power Flow versions 10 and 11 respectively. DoPF is provided for backward compatibility purposes.

### Example:

```
Dim vnPFOption(10) As Long  
Dim vdPFCriteria(5) As Double  
  
' Various PF settings  
vdPFCriteria(1) = 20 ' Max iterations  
vdPFCriteria(2) = 1 ' MW tolerance
```

```

vdPFCriteria(3) = 1 ' MVAR tolerance
vdPFCriteria(4) = 10 ' MW adj. threshold
vdPFCriteria(5) = 10 ' MVAR adj. threshold
' Set PF options
vnPFOption(1) = 1 ' Use previous result
vnPFOption(2) = 0 ' Ignore Gen var limit
vnPFOption(3) = 1 ' Enforce Xfmr tap
vnPFOption(4) = 0 ' Ignore Area interchange
vnPFOption(5) = 0 ' Ignore Gen remove V control
vnPFOption(6) = 0 ' Ignore SVD
vnPFOption(7) = 1 ' Phase shifter
vnPFOption(8) = 0 ' Reset LTC
vnPFOption(9) = 0 ' Disable solution monitor
' PF method
nMethod = 1 ' Newton-Raphson

' Do the power flow
If DoPF( nSlackBus, vdPFCriteria, vnPFOption, nMethod ) = 0 Then GoTo HasError

```

## Function ErrorString

Function ErrorString() As String

**Purpose:** Return text description of the last script function error.

**Works in:** *OneLiner* and *Power Flow*.

## Function ExportNetwork

```
Function ExportNetwork( ByRef sFileName$, ByRef vnOptions() as long ) As long
```

**Purpose:** Export network data to ASPEN text data file.

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

sFileName	[in] DXT path name
vnOptions	[in] Export options
vnOptions[1]	Area/zone flag: 0- All; 1- Area; 2- Zone
vnOptions[2]	Zone or Area number
vnOptions[3]	Include tie line flag. 0- reset; 1- set

**Return value:**

1	Success
0	Failure

**Remarks:**

**Example:**

## Function ExportNetworkPSSE

```
Function ExportNetworkPSSE( ByRef sFileNameRaw$, ByRef sFileNameSeq$, _  
    ByRef vnOptions() as long ) As long
```

**Purpose:** Export network data to ASPEN text data file.

**Works in:** *OneLiner* and *Power Flow*.

### Parameters:

sFileName	[in] DXT path name
vnOptions	[in] Export options
vnOptions[1]	Area/zone flag: 0- All; 1- Area; 2- Zone
vnOptions[2]	Zone or Area number
vnOptions[3]	Include tie line flag. 0- reset; 1- set
vnOptions[4]	PSS/E version number
vnOptions[5]	First fictitious bus number for 3-winding transformer midpoint
vnOptions[6]	First fictitious bus number for buses with no bus number

### Return value:

1	Success
0	Failure

### Remarks:

### Example:

## Function EquipmentType

```
Function EquipmentType( ByVal nHandle& ) As Long
```

**Purpose:** Gets object type associated with a given handle.

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

nHandle	[in] data object handle
---------	-------------------------

**Return value:**

>0	equipment type code
0	failure

**Remarks:** The input requires a valid equipment handle.

**Example:**

```
If EquipmentType( nBusHnd ) <> TC_BUS Then  
    Print "Must select a bus"  
    Stop  
End If
```

## Function FaultDescription

Function FaultDescription(FltIdx&) As String

**Purpose:** Retrieves description of the fault being displayed. The result is a string.

**Works in:** *OneLiner*.

**Parameters:**

FltIdx [in] Index number of fault in solution buffer

**Return value:** Fault description string

**Remarks:** Call this function with FltIdx = 0 to get description string of the fault currently displayed.

**Example:**

```
If PickFault(1) > 0 then Print FaultDescription(0)
```



## Function FaultSelector

```
Function FaultSelector( ByRef vnFltIndex() as Long,  
                        ByVal sTitle$, ByVal sPrompt$ ) As Long
```

**Purpose:** Display fault selection dialog.

**Works in:** *OneLiner* only.

**Parameters:**

sTitle [in] Dialog title.  
sPrompt [in] Dialog prompt.  
vnFltIndex [out] List of fault index numbers that the user selected. Array index is zero-based.

**Return value:**

Number of fault selected.

**Remarks:**

Array vnFltIndex dimension must be adequate to store user selection.

**Example:**

```
nCount = FaultSelector( nFltIdx, "My Fault Selector", "Please Select One Fault" )
```

## Function FileOpenDialog, FileSaveDialog

```
Function FileOpenDialog( ByRef sInitDir$, ByRef sFilter$, ByVal nOption& ) As String
```

```
Function FileSaveDialog( ByRef sInitDir$, ByRef sFilter$, ByRef sDefaultExt$, ByVal nOption& ) As String
```

**Purpose:** Invoke standard windows dialog box for open and save file

**Works in:** *OneLiner* and *Power Flow*.

### Parameters:

sInitDir	[in] Initial directory selected in the GUI file tree
sFilter	[in] File type single filter or multiple filter group (see remarks)
sDefaultExt	[in] Default file extension
nOption	[in] Dialog options. To use more than one option, add the required values together 1 = File must exist 2 = Path must exist 4 = Allow multiselect 16 = Prompt to overwrite file

**Return value:** Full path name of the file(s) or blank string if user pressed cancel.

### Remarks:

Single filter must be in format *Description/wildcard*// such as "All (\*.\*)|\*.\*)"

Wildcard patterns are separated by semicolon as in example

Multiple groups of filter are separated by pipe character |.

Results for multiple sections are "File1|File2|..."

### Example:

```
sPath$ = FileOpenDialog( "", _  
    "Change files (*.chf)|*.chf|OneLiner files (*.olr;*.dxt)|*.olr;*.dxt|", _  
    4 )  
Print sPath  
sPath$ = FileSaveDialog( "", "", ".txt", 2+16 )  
Print sPath  
sPath$ = FolderSelectDialog( "My Dialog", "c:\" )  
Print sPath
```

## Function FindBusByName

Function FindBusByName( ByVal sName\$, ByVal dNomKV#, ByRef nHandle& ) As Long

**Purpose:** Searches for the bus that has a certain name and nominal kV.

**Works in:** *OneLiner* and *Power Flow*.

### Parameters:

sName	[in] bus name
dNomKV	[in] bus nominal kV
nHandle	[out] bus handle

### Return value:

1	success
0	failure

### Remarks:

### Example:

```
If 0 = FindBusByName( "Reusens", 132, hDev ) Then GoTo HasError
Print "Bus handle found: ", hDev
```

## Function FindEquipmentByTag

```
Function FindEquipmentByTag( ByRef sTag$, ByVal nType&, _  
                           ByRef hHandle ) As Long
```

**Purpose:** Find next object that has the tag in its tag string.

**Works in:** *OneLiner* and *Power Flow*.

### Parameters:

sTag	[in] Tag string
nType	[in] Equipment type. This parameter can be one of the followings: TC_BUS, TC_LOAD, TC_SHUNT, TC_GEN, TC_SVD, TC_LINE, TC_XFMR, TC_XFMR3, TC_PS, TC_SCAP, TC_MU, TC_RLYGROUP, TC_RLYOCG, TC_RLYOCP, TC_RLYDSG, TC_RLYDSP, TC_FUSE, TC_SWITCH, TC_RECLSRP, TC_RECLSRG or zero.
nHandle	[out] Object handle

### Return value:

1	success
0	failure

**Remarks:** To get the first object in the list, call this function with nHandle equal 0. Call this function with nType equal 0 to search all object types.

### Example:

```
Sub main  
  ObjHnd& = 0  
  While FindEquipmentByTag( "Line tag", TC_XFMR, ObjHnd ) > 0  
    If EquipmentType( ObjHnd ) = TC_RLYGROUP Then  
      RelayHnd& = 0  
      While GetRelay( ObjHnd, RelayHnd ) > 0  
        Print "Memo: " + GetObjMemo( RelayHnd ) + _  
          Chr(13) + Chr(10) + "Tags: " + GetObjTags( RelayHnd )  
      Wend  
    Else  
      Print "Memo: " + GetObjMemo( ObjHnd ) + _  
        Chr(13) + Chr(10) + "Tags: " + GetObjTags( ObjHnd )  
    End If  
  Wend  
  Exit Sub  
HasError:  
  Print "Error: ", ErrorString( )  
End Sub
```

## Function FolderSelectDialog

```
Function FolderSelectDialog( ByRef sTitle$, ByRef sInitDir$ ) As String
```

**Purpose:** Invoke dialog box for folder selection

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

sTitle	[in] Dialog box title text
sInitDir	[in] Initial selection

**Return value:** Full path name of the folder or blank string if user pressed cancel.

**Remarks:**

**Example:**

```
sPath$ = FolderSelectDialog( "Data folder", "c:\" )  
Print sPath
```

## Function FullBranchName

```
Function FullBranchName( ByVal nBranchHnd& ) As String
```

**Purpose:** Return a string composed of branch's Bus, Bus2, Circuit ID and type.

**Works in:** *OneLiner and Power Flow.*

**Parameters:**

nBranchHnd [in] handle of a branch object: branch, line, transformer, phase shifter, switch, relay group

**Return value:**

Full branch name if success.

Empty string otherwise.

**Remarks:**

**Example:**

```
' Print branch info
Print FullBranchName (nBranchHnd & )
```

## Function FullBusName

```
Function FullBusName( ByVal nBusHnd& ) As String
```

**Purpose:** Return a string composed of name and kV of the given bus.

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

nBusHnd [in] bus handle

**Return value:**

Full bus name if success.  
Empty string otherwise.

**Remarks:**

**Example:**

```
' Print bus info
  If GetData( nBusHnd&, BUS_sName, sVal1$ ) = 0 Then GoTo HasError
  If GetData( nBusHnd&, BUS_dKVnominal, dVal1# ) = 0 Then GoTo HasError
  If GetData( nBusHnd&, BUS_nNumber, nVal1& ) = 0 Then GoTo HasError
  If GetData( nBusHnd&, BUS_nArea, nVal2& ) = 0 Then GoTo HasError
  If GetData( nBusHnd&, BUS_nZone, nVal3& ) = 0 Then GoTo HasError
  Print "BUS      "; FullBusName( nBusHnd& );" AREA=";nVal2&;" ZONE="; nVal3&
```

## Function FullRelayName

```
Function FullRelayName( ByVal nRelayHnd& ) As String
```

**Purpose:** Return a string composed of relay type, name and branch location.

**Works in:** *OneLiner*.

**Parameters:**

nRelayHnd [in] relay handle

**Return value:**

Full relay name if success.  
Empty string otherwise.

**Remarks:**

**Example:**

```
' Print relay info
Print FullRelayName( nRelayHnd& )
```



## Function GetAreaName, GetZoneName

```
Function GetAreaName( ByVal nNo& ) As String  
Function GetZoneName( ByVal nNo& ) As String
```

**Purpose:** Retrieve name of area or zone number

**Works in:** *OneLiner and Power Flow.*

**Parameters:**

nNo [in] Area or Zone number

**Return value:**

Name of area or zone.

**Remarks:**

**Example:**

```
Sub main  
  Do  
    nNo = Val( InputBox( "Area no" ) )  
    If nNo = 0 Then Stop  
    Print "Area " & nNo & " name=" & GetAreaName( nNo )  
    Print "Zone " & nNo & " name=" & GetZoneName( nNo )  
  Loop  
End Sub
```

## Function GetBusEquipment

```
Function GetBusEquipment( ByVal nBusHnd&, ByVal nType&, _  
                          ByRef nHandle& ) As Long
```

**Purpose:** Retrieves the handle of the next equipment of a given type that is attached to a bus.

**Works in:** *OneLiner* and *Power Flow*.

### Parameters:

nBusHnd	[in] Bus handle
nType	[in] Equipment type. See remark below.
nHandle	[in/out] Equipment handle. Set to zero for the first item. Otherwise, leave it at the value of the previous item.

### Return value:

1	success
-1	already at the end of the list
0	failure

**Remarks:** Set nHandle to zero to get the first equipment handle. You must get the bus handle prior to calling this function. The equipment type can be one of the following:

TC_GEN:	to get the handle for the generator. There can be at most one at a bus.
TC_LOAD:	to get the handle for the load. There can be at most one at a bus.
TC_SHUNT:	to get the handle for the shunt. There can be at most one at a bus.
TC_SVD:	to get the handle for the switched shunt. There can be at most one at a bus.
TC_GENUNIT:	to get the handle for the next generating unit.
TC_LOADUNIT:	to get the handle for the next load unit.
TC_SHUNTUNIT:	to get the handle for the next shunt unit.
TC_BRANCH:	to get the handle for the next branch.

### Example:

```
` Put all bus branches in outage list for fault simulation  
For ii = 1 To 20 ' max 20 outage  
  If GetBusEquipment( nBusHnd, TC_BRANCH, nBrHnd ) > 0 Then  
    vnOutageLst(ii) = nBrHnd  
  Else  
    Exit For  
  End If  
Next  
vnOutageLst(ii) = 0 ' Must always close the list
```

## Function GetData

Function GetData( ByVal deviceHnd&, ByVal paramID&, ByRef outputVal ) as Long

**Purpose:** Reads a network or system parameter into a program variable.

**Works in:** *OneLiner* and *Power Flow*.

### Parameters:

deviceHnd	[in] data object handle
paramID	[in] parameter ID code
outputVal	[out] output variable.

### Return value:

1	success
0	failure

**Remarks:** To read a parameter of a network object, you must first obtain the handle prior to calling this function. To read a system parameter, you can use the pre-defined handle HND\_SYS. Data type of outputVal must agree with that of the parameter being read. The parameter ID code and data type are available in table 3.3.

### Example:

```
If GetData( HND_SYS, SY_dBaseMVA, dVal1 ) = 0 Then GoTo HasError
Print "BASE MVA = ", Format(dVal1,"##0.0")
```

## Function GetEquipment

Function GetEquipment( ByVal nType&, ByRef Handle ) As Long

**Purpose:** Retrieves handle of the next equipment of given type in the system. If nType is set to TC\_PICKED, this function will return the handle of the selected equipment on the one-line diagram. If nType is set to the equipment code for generators, shunts, loads, generating units, shunt units, load units, lines, series capacitors, transformers, phase shifters, switches, relay groups, overcurrent relays, fuses, distance relays, differential relays, voltage relays, reclosers and logic schemes, this function will return the handle of all the objects, one by one, in the order they are stored in the *OneLiner* or *Power Flow* OLR file.

**Works in:** *OneLiner* and *Power Flow*.

### Parameters:

nType	[in] equipment type (table 3.2)
nHandle	[in/out]equipment handle.

### Return value:

1	success
-1	already at the end of the list
0	failure

**Remarks:** Set nHandle to zero to get the first equipment handle. Set nType to TC\_PICKED if you want to see which network element was highlighted on the one-line diagram by the user. (If the user has highlight more than one piece of equipment, only the handle of the first selection is returned.)

### Example:

```
If GetEquipment( TC_PICKED, nBusHnd ) = 0 Then
    Print "Must select a bus"
    Stop
End If
```

## Function GetFlow

```
Function GetFlow( ByVal hHandle&, ByRef vdOut1() as double, _  
                ByRef vdOut2() as double ) As Long
```

**Purpose:** Retrieve power flow for a generator, load, shunt, switched shunt, generating unit, load unit, shunt unit, transmission line, transformer, or phase shifter.

**Works in:** *Power Flow* only.

### Parameters:

nHandle	[in] data object handle
vdOut1	[out] Real power in MW into equipment terminal(s)
vdOut2	[out] Reactive power in MVAR into equipment terminal(s)

### Return value:

1	success
0	failure

**Remarks:** The size of arrays vdOut1 and vdOut2 must be at least equal to the number of buses connected to the equipment: 1 for generator, load, shunt, switched shunt, generating unit, load unit, shunt unit; 2 for Line, 2-winding transformer, phase shifter; switch, switch, 3 for 3-winding transformer. For equipment that has more than one connected bus, the flow results are stored in the following order:

- vdOut1(1), vnOut2(1): flow from equipment's Bus1
- vdOut1(2), vnOut2(2): flow from equipment's Bus2
- vdOut1(3), vnOut2(3): flow from equipment's Bus3

### Example:

```
Dim Parray(3) As Double, Qarray(9) As Double  
' Get end bus names  
If GetData( LineHnd, LN_nBus1Hnd, BusHnd ) = 0 Then GoTo HasError  
Bus1ID = FullBusName( BusHnd )  
If GetData( LineHnd, LN_nBus2Hnd, BusHnd ) = 0 Then GoTo HasError  
Bus2ID = FullBusName( BusHnd )  
' Get current  
If GetFlow( LineHnd, Parray, Qarray, 4 ) = 0 Then GoTo HasError  
' Show them  
Print _  
    "Power on line: "; sVal2$ & "-"; sVal3$ & " ID= "; sVal1$; ": "; Chr(10); _  
    "P1 = "; Format( Parray(1), "#0.0"); " Q1= "; Format( Qarray(1), "#0.0"); _  
    "; P2 = "; Format( Parray(2), "#0.0"); " Q2 = "; Format( Qarray(2), "#0.0")
```

## Function GetObjJournalRecord

Function GetObjJournalRecord( ByVal hHandle& ) As String

**Purpose:** Retrieve journal record details of a data object in the OLR file.

**Works in:** *OneLiner and Power Flow.*

**Parameters:**

nHandle            [in] data object handle

**Return value:**

String of journal record fields, separated by new line character:

- Create date and time
- Created by
- Last modified date and time
- Modified by

**Remarks:**

**Example:**

```
Sub main
    hnd& = 0
    count = 0
    While 1 = GetEquipment(TC_RLYOCG,hnd)
        JRec$ = GetObjJournalRecord(hnd)
        Call parseALine( JRec$, Chr(10), dateCreated$, JRec$ )
        Call parseALine( JRec$, Chr(10), CreatedBy$, JRec$ )
        Call parseALine( JRec$, Chr(10), dateModified$, JRec$ )
        Call parseALine( JRec$, Chr(10), ModifiedBy$, JRec$ )
        Print PrintObj1LPF(hnd) & Chr(10) & _
            " Created: " & dateCreated & " by:" & CreatedBy & _
            Chr(10) & " Modified: " & dateModified & " by: " & ModifiedBy
        If count >= 5 Then Stop
        count = count + 1
    Wend
End Sub

Sub parseALine( ByVal aLine$, ByVal Delim$, ByRef sLeft$, ByRef sRight$ )
    nPos = InStr( 1, aLine$, Delim$ )
    If nPos = 0 Then
        sLeft = aLine$
        sRight = ""
    Else
        sLeft = Left(aLine$, nPos-1)
        sRight = Mid(aLine$, nPos+Len(Delim), 9999 )
    End If
    sLeft = Trim(sLeft)
    sRight = Trim(sRight)
End Sub
```

## Function GetObjMemo

Function GetObjMemo ( ByVal hHandle& ) As String

**Purpose:** Retrieve memo string for a bus, generator, load, shunt, switched shunt, transmission line, transformer, switch, phase shifter, distance relay, overcurrent relay, fuse, recloser, or relay group.

**Works in:** *OneLiner and Power Flow.*

**Parameters:**

nHandle            [in] data object handle

**Return value:**

Object memo field.

**Remarks:**

**Example:**

```
Sub main()
  If GetEquipment( TC_PICKED, ObjHnd& ) = 0 Then
    Print "Please select an object"
    Exit Sub
  End If
  If EquipmentType( ObjHnd ) = TC_RLYGROUP Then
    RelayHnd& = 0
    While GetRelay( ObjHnd, RelayHnd ) > 0
      Print GetObjMemo( RelayHnd )
    Wend
  Else
    Print GetObjMemo( ObjHnd )
  End If
End Sub
```

## Function GetObjTags

Function GetObjTags ( ByVal hHandle& ) As String

**Purpose:** Retrieve tag string for a bus, generator, load, shunt, switched shunt, transmission line, transformer, switch, phase shifter, distance relay, overcurrent relay, fuse, recloser, relay group.

**Works in:** *OneLiner and Power Flow.*

**Parameters:**

nHandle            [in] data object handle

**Return value:**

Object tag string.

**Remarks:**

**Example:**

```
Sub main()  
If GetEquipment( TC_PICKED, ObjHnd& ) = 0 Then  
    Print "Please select an object"  
    Exit Sub  
End If  
If EquipmentType( ObjHnd ) = TC_RLYGROUP Then  
    RelayHnd& = 0  
    While GetRelay( ObjHnd, RelayHnd ) > 0  
        Print GetObjTags( RelayHnd )  
    Wend  
Else  
    Print GetObjTags( ObjHnd )  
End If  
End Sub
```



## Function GetOlrFileName

Function GetOlrFileName() As String

**Purpose:** Print full path name of current OLR file.

**Works in:** *OneLiner and Power Flow.*

**Parameters:**

**Return value:**

Full path name of current OLR file.

## Function GetPFCurrent

```
Function GetPFCurrent( ByVal hHandle&, ByRef vdOut1() as double, _  
    ByRef vdOut2() as double ) As Long
```

**Purpose:** Retrieve current in load flow simulation for a generator, load, shunt, switched shunt, generating unit, load unit, shunt unit, transmission line, transformer, switch, or phase shifter .

**Works in:** *Power Flow* only.

### Parameters:

nHandle	[in] data object handle
vdOut1	[out] current result magnitude into equipment terminals
vdOut2	[out] current result angle in degree, into equipment terminals
nStyle	[in] current result style =0: output current in Amperes =1: output current in per-unit

### Return value:

1	success
0	failure

**Remarks:** The size of arrays vdOut1 and vdOut2 must be at least equal to the number of buses connected to the equipment: 1 for generator, load, shunt, switched shunt, generating unit, load unit, shunt unit; 2 for Line, 2-winding transformer, phase shifter, switch; 3 for 3-winding transformer. For equipment that has more than one connected bus, the current results are stored in the following order:

- vdOut1(1), vnOut2(1): current from equipment's Bus1
- vdOut1(2), vnOut2(2): current from equipment's Bus2
- vdOut1(3), vnOut2(3): current from equipment's Bus3

### Example:

```
Dim MagArray(3) As Double, AngArray(3) As Double  
' Get end bus names  
If GetData( LineHnd, LN_nBus1Hnd, BusHnd ) = 0 Then GoTo HasError  
Bus1ID = FullBusName( BusHnd )  
If GetData( LineHnd, LN_nBus2Hnd, BusHnd ) = 0 Then GoTo HasError  
Bus2ID = FullBusName( BusHnd )  
' Get current  
If GetPFCurrent( LineHnd, MagArray, AngArray, 0 ) = 0 Then GoTo HasError  
' Show them  
Print _  
    "Current on line: "; Bus1ID & "-"; Bus2ID & " ID= "; LineID$; ": "; Chr(10); _  
    "I1 = "; Format( MagArray(1), "#0.0"); "@"; Format( AngArray(1), "#0.0"); Chr(10) _  
    "I2 = "; Format( MagArray(2), "#0.0"); "@"; Format( AngArray(2), "#0.0")
```

## Function GetPFVoltage

```
Function GetPFVoltage( ByVal hHandle&, ByRef vdOut1() as double, _  
    ByRef vdOut2() as double, ByVal nStyle& ) As Long
```

**Purpose:** Retrieve voltage in load flow simulation of a bus, or of connected buses of a line, transformer, switch or phase shifter.

**Works in:** *Power Flow* only.

### Parameters:

nHandle	[in] data object handle
vdOut1	[out] voltage magnitude
vdOut2	[out] voltage angle in degree
nStyle	[in] result style
	=1: output voltage in kV
	=2: output voltage in per-unit

### Return value:

1	success
0	failure

**Remarks:** The size of arrays vdOut1 and vdOut2 must be at least 3. For equipment that is connected to more than one bus, the voltage result is stored in the following order:

- vdOut1(1), vnOut2(1): Voltage at equipment's Bus1
- vdOut1(2), vnOut2(2): Voltage at equipment's Bus2
- vdOut1(3), vnOut2(3): Voltage at equipment's Bus3

### Example:

```
Dim MagArray(3) As Double, AngArray(3) As Double  
' Get end bus names  
If GetData( LineHnd, LN_nBus1Hnd, BusHnd ) = 0 Then GoTo HasError  
Bus1ID = FullBusName( BusHnd )  
If GetData( LineHnd, LN_nBus2Hnd, BusHnd ) = 0 Then GoTo HasError  
Bus2ID = FullBusName( BusHnd )  
' Get current  
If GetPFVoltage( nDevHnd&, vdVal1, vdVal2, 2 ) = 0 Then GoTo HasError  
' Show it  
Print "Voltage on line: "; sVal2$ & "-"; sVal3$ & " ID= "; sVal1$; ": "; Chr(10); _  
    "V1 = "; Format( MagArray(1), "#0.00"); "@"; Format( AngArray(1), "#0.0"); _  
    "; V2 = "; Format( MagArray(2), "#0.00"); "@"; Format( AngArray 2), "#0.0")
```

## Function GetProgramVersion

Function GetProgramVersion () As String

**Purpose:** Print string with program major and minor version.

**Works in:** *OneLiner* and *Power Flow*.

## Function GetPSCVoltage

```
Function GetPSCVoltage( ByVal hHandle&, ByRef vdOut1() as double, _  
    ByRef vdOut2() as double, ByVal nStyle& ) As Long
```

**Purpose:** Retrieve pre-fault voltage of a bus, or of connected buses of a line, transformer, switch or phase shifter.

**Works in:** *OneLiner* only.

### Parameters:

nHandle	[in] data object handle
vdOut1	[out] voltage magnitude
vdOut2	[out] voltage angle in degree
nStyle	[in] result style
	=1: output voltage in kV
	=2: output voltage in per-unit

### Return value:

1	success
0	failure

**Remarks:** The size of arrays vdOut1 and vdOut2 must be at least 3. For equipment that is connected to more than one bus, the voltage result is stored in the following order:

- vdOut1(1), vnOut2(1): Voltage at equipment's Bus1
- vdOut1(2), vnOut2(2): Voltage at equipment's Bus2
- vdOut1(3), vnOut2(3): Voltage at equipment's Bus3

### Example:

```
Dim MagArray(3) As Double, AngArray(3) As Double  
' Get end bus names  
If GetData( LineHnd, LN_nBus1Hnd, BusHnd ) = 0 Then GoTo HasError  
Bus1ID = FullBusName( BusHnd )  
If GetData( LineHnd, LN_nBus2Hnd, BusHnd ) = 0 Then GoTo HasError  
Bus2ID = FullBusName( BusHnd )  
' Show the fault  
If PickFault( 1 ) Then GoTo HasError  
' Get pre-fault  
If GetPSCVoltage( nDevHnd&, vdVal1, vdVal2, 2 ) = 0 Then GoTo HasError  
' Show it  
Print "Pre-fault voltage on line: "; sVal2$ & "-"; sVal3$ & " ID= "; sVal1$; ": "; Chr(10); _  
    "V1 = "; Format( MagArray(1), "#0.00"); "@"; Format( AngArray(1), "#0.0"); _  
    "; V2 = "; Format( MagArray(2), "#0.00"); "@"; Format( AngArray 2), "#0.0")
```

## Function GetRelay

Function GetRelay( ByVal nRlyGrp&, ByRef nRlyHandle& ) As Long

**Purpose:** Get handle of the next relay or fuse in a relay group.

**Works in:** *OneLiner* only.

### Parameters:

nRlyGrp	[in] relay group handle.
nRlyHandle	[int/out] relay handle.

### Return value:

1	success
-1	already at last relay
0	failure

**Remarks:** Set nRlyHandle to zero to get the handle to the first relay in the relay group.

### Example:

```
' Pick the first fault
If PickFault( 1 ) Then GoTo HasError

' Loop through all relays in the database and find their operating times
nRelayCount& = 0
nRelayHnd& = 0
While GetRelay( nPickedHnd, nRelayHnd& ) > 0
    nRelayCount = nRelayCount + 1
    nType = EquipmentType( nRelayHnd )
    If nType = TC_RLYOCG Then nParamID& = OG_sID
    If nType = TC_RLYOCP Then nParamID& = OP_sID
    If nType = TC_RLYDSG Then nParamID& = DG_sID
    If nType = TC_RLYDSP Then nParamID& = DP_sID
    If nType = TC_FUSE Then nParamID& = FS_sID
    If GetData( nRelayHnd, nParamID, sID$ ) = 0 Then GoTo HasError
    If GetRelayTime( nRelayHnd, 1.0, dTime# ) = 0 Then GoTo HasError
    Print "Relay " & sID & ": "; Format( dTime, "#0.#0s" )
Wend
Print "Relays in this group = "; nRelayCount
Stop
' Error handling
HasError:
Print "Error: ", ErrorString( )
Stop
```

## Function GetRelayTime

```
Function GetRelayTime( ByVal nHandle&, ByVal dFactor#, ByRef dTime$ ) As Long
Function GetRelayTimeEx( ByVal nHandle&, ByVal dFactor#, ByRef dTime$,
                        ByRef sDevice$, ByVal nTripOnly& ) As Long
```

**Purpose:** Computes operating time for a fuse, an overcurrent relay (phase or ground), or a distance relay (phase or ground).

**Works in:** *OneLiner* only.

### Parameters:

nHandle	[in] Handle of OC relay, DS relay, Recloser or Fuse
dFactor	[in] relay current multiplying factor
dTime	[out] relay operating time in seconds
sDevice	[out] relay operation code: NOP No operation ZGn Ground distance zone n tripped ZPn Phase distance zone n tripped Ix Overcurrent relay operating quantity: Ia, Ib, Ic, Io, I2, 3Io, 3I2 TOC Time overcurrent element tripped
nTripOnly	[in] Consider relay element signal-only flag 1 - Yes; 0 - No

### Return value:

1	success
0	failure

**Remarks:** All calls to this function must be preceded by a call to ShowFault or PickFault function.

Relay current multiplying factor will be applied to relay current result from simulation before time calculation.

### Example:

```
' Show the fault
If ShowFault( 1, 3, 7, 0, vnShowRelay ) = 0 Then GoTo HasError

' Loop through all relays and find their operating times
nRelayCount& = 0
nRelayHnd& = 0
While GetRelay( nPickedHnd, nRelayHnd& ) > 0
    nRelayCount = nRelayCount + 1
    nType = EquipmentType( nRelayHnd )
    If nType = TC_RLYOCG Then nParamID& = OG_sID
    If nType = TC_RLYOCP Then nParamID& = OP_sID
    If nType = TC_RLYDSG Then nParamID& = DG_sID
    If nType = TC_RLYDSP Then nParamID& = DP_sID
    If nType = TC_FUSE Then nParamID& = FS_sID
    If GetData( nRelayHnd, nParamID, sID$ ) = 0 Then GoTo HasError
    If GetRelayTime( nRelayHnd, 1.0, dTime# ) = 0 Then GoTo HasError
    Print "Relay " & sID & ": "; Format( dTime, "#0.#0s" )
Wend
Print "Relays in this group = "; nRelayCount
Stop
' Error handling
HasError:
Print "Error: ", ErrorString( )
Stop
```

## Function GetLogicScheme

Function GetLogicScheme( ByVal nRlyGrp&, ByRef nSchemeHnd& ) As Long

**Purpose:** Get handle of the next logic scheme in a relay group.

**Works in:** *OneLiner* only.

**Parameters:**

nRlyGrp	[in] relay group handle.
nSchemeHnd	[int/out] logic scheme handle.

**Return value:**

1	success
-1	already at the last scheme in the relay group
0	failure

**Remarks:** Set nSchemeHnd to zero to get the handle to the first scheme in the group.

**Example:**

```
Sub main()

' Get picked object number
If GetEquipment( TC_PICKED, ObjHnd ) = 0 Or EquipmentType( ObjHnd ) <> TC_RLYGROUP Then
    Print "No relay group is selected."
    Exit Sub
End If

schemeCount = 0
schemeHnd = 0
While GetScheme( ObjHnd, schemeHnd ) > 0
    schemeCount = schemeCount + 1
    If GetData( schemeHnd, LS_sID, sID$ ) = 0 Then GoTo hasError
    If GetData( schemeHnd, LS_sAssetID, sAssetID$ ) = 0 Then GoTo hasError
    If GetData( schemeHnd, LS_sScheme, sSchemeType$ ) = 0 Then GoTo hasError
    If GetData( schemeHnd, LS_sEquation, sEquation$ ) = 0 Then GoTo hasError
    If GetData( schemeHnd, LS_sVariables, sVars$ ) = 0 Then GoTo hasError
    If GetData( schemeHnd, LS_nRlyGrpHnd, nRlyGroupHnd& ) = 0 Then GoTo hasError
    Print "Scheme found: " & Chr(13) & _
        sID & "@" & FullBranchName( nRlyGroupHnd ) & Chr(13) & _
        "AssetID = " & sAssetID & Chr(13) & _
        "Type = " & sSchemeType & Chr(13) & _
        "Equation = " & sEquation & Chr(13) & _
        sVars & Chr(13)

Wend
Print "Found total schemes: ", schemeCount
Stop
hasError:
    Print "Error: " + ErrorString()
End Sub
```



## Function GetSCCurrent

```
Function GetSCCurrent( ByVal hHandle&, ByRef vdOut1() as double, _  
    ByRef vdOut2() as double, ByVal nStyle& ) As Long
```

**Purpose:** Retrieve post fault current for a generator, load, shunt, switched shunt, generating unit, load unit, shunt unit, transmission line, transformer, switch or phase shifter. You can get the total fault current by calling this function with the pre-defined handle of short circuit solution, HND\_SC.

**Works in:** *OneLiner* only.

### Parameters:

nHandle	[in] data object handle
vdOut1	[out] current result, real part or magnitude, into equipment terminals
vdOut2	[out] current result, imaginary part or angle in degree, into equipment terminals
nStyle	[in] current result style
	=1: output 012 sequence current in rectangular form
	=2: output 012 sequence current in polar form
	=3: output ABC phase current in rectangular form
	=4: output ABC phase current in polar form

### Return value:

1	success
0	failure

**Remarks:** Size of arrays vdOut1 and vdOut2 must be sufficient to store all current results according to the following table:

Output Arrays	Generator, load, shunt	Line, transformer, phase shifter
vdOut1(1) vdOut2(1) vdOut1(2) vdOut2(2) vdOut1(3) vdOut2(3)	Current Phase A, B, C or Sequence: zero, pos., neg	Current Phase: A, B, C or Seq.: 0, +, - from Bus 1
vdOut1(4) vdOut2(4)	Not used	Transformer only: Neutral current of winding on Bus 1
vdOut1(5) vdOut2(5) vdOut1(6) vdOut2(6) vdOut1(7) vdOut2(7)	Not used	Current Phase: A, B, C or Seq.: 0, +, - from Bus 2
vdOut1(8) vdOut2(8)	Not used	Transformer only: Neutral current of winding on Bus 2
vdOut1(9) vdOut2(5) vdOut1(10) vdOut2(10) vdOut1(11) vdOut2(11)	Not used	3-W Transformer only: Current Phase: A, B, C or Seq. 0, +, - from Bus 3
vdOut1(12) vdOut2(12)	Not used	3-W Transformer only: Delta circulating current on Bus 3

### Example:

```
Dim MagArray(12) As Double, AngArray(12) As Double  
' Get end bus names  
If GetData( LineHnd, LN_nBus1Hnd, BusHnd ) = 0 Then GoTo HasError  
Bus1ID = FullBusName( BusHnd )  
If GetData( LineHnd, LN_nBus2Hnd, BusHnd ) = 0 Then GoTo HasError  
Bus2ID = FullBusName( BusHnd )
```

```

` Must always show fault before retrieving result
If ShowFault( 1, 0, -999, 0, vnShowRelay ) = 0 Then GoTo HasError
` Get post fault current
If GetSCCCurrent( LineHnd, MagArray, AngArray2, 4 ) = 0 Then GoTo HasError
` Show them
Print
  "Current on line: "; Bus1ID & "-"; Bus2ID & " ID= "; LineID$; ": "; Chr(10); _
  "I1a = "; Format( MagArray(1), "#0.0"); "@"; Format( AngArray(1), "#0.0"); _
  "; I1b = "; Format( MagArray(2), "#0.0"); "@"; Format( AngArray(2), "#0.0"); _
  "; I1c = "; Format( MagArray(3), "#0.0"); "@"; Format( AngArray(3), "#0.0"); _
  "; I1g = "; Format( MagArray(4), "#0.0"); "@"; Format( AngArray(4), "#0.0"); Chr(10); _
  "I2a = "; Format( MagArray(5), "#0.0"); "@"; Format( AngArray(5), "#0.0"); _
  "; I2b = "; Format( MagArray(6), "#0.0"); "@"; Format( AngArray(6), "#0.0"); _
  "; I2c = "; Format( MagArray(7), "#0.0"); "@"; Format( AngArray(7), "#0.0"); _
  "; I2g = "; Format( MagArray(8), "#0.0"); "@"; Format( AngArray(8), "#0.0")

```

## Function GetSCVoltage

```
Function GetSCVoltage( ByVal nHandle&, ByRef vdOut1() as double, _  
                    ByRef vdOut2() as double, ByVal nStyle& ) As Long
```

**Purpose:** Retrieves post-fault voltage of a bus, or of connected buses of a line, transformer, switch or phase shifter.

**Works in:** *OneLiner* only.

### Parameters:

nHandle	[in] data object handle
vdOut1	[out] voltage result, real part or magnitude, at equipment terminals
vdOut2	[out] voltage result, imaginary part or angle in degree, at equipment terminals
nStyle	[in] voltage result style

=1: output 012 sequence voltage in rectangular form  
=2: output 012 sequence voltage in polar form  
=3: output ABC phase voltage in rectangular form  
=4: output ABC phase voltage in polar form

### Return value:

1	success
0	failure

**Remarks:** The size of arrays vdOut1 and vdOut2 must be at least equal to 3 times the number of buses connected to the equipment: 1 for a Bus; 2 for a Line, 2-winding transformer, phase shifter, switch; 3 for a 3-winding transformer. For equipment that is connected to more than one bus, voltage result is stored in the arrays in group of 3 phases ABC (or 3 sequences Zero, Positive, Negative) in the following order:

- vdOut1(1...3), vnOut2(1...3): Voltage at equipment's Bus1
- vdOut1(4...6), vnOut2(4...6): Voltage at equipment's Bus2
- vdOut1(7...9), vnOut2(7...9): Voltage at equipment's Bus3

### Example:

```
If GetEquipment( TC_PICKED, nDevHnd ) = 0 Then  
    Print "Must select a line"  
    Stop  
End If  
If EquipmentType( nDevHnd ) <> TC_LINE Then  
    Print "Must select a line"  
    Stop  
End If  
' Get line's end buses  
If GetData( nDevHnd&, LN_sID, sVal1$ ) = 0 Then GoTo HasError  
If GetData( nDevHnd&, LN_nBus1Hnd, nBusHnd& ) = 0 Then GoTo HasError  
sVal2$ = FullBusName( nBusHnd& )  
If GetData( nDevHnd&, LN_nBus2Hnd, nBusHnd& ) = 0 Then GoTo HasError  
sVal3$ = FullBusName( nBusHnd& )  
' Show the fault  
If PickFault( 1 ) Then GoTo HasError  
'Get voltage at the end bus  
If GetSCVoltage( nDevHnd&, vdVal1, vdVal2, 4 ) = 0 Then GoTo HasError  
' Show it  
Print "Voltage on line: "; sVal2$ & "-"; sVal3$ & " ID= "; sVal1$; ": "; Chr(10); _  
    "V1a = "; Format( vdVal1(1), "#0.0"); "@"; Format( vdVal2(1), "#0.0"); _  
    "; V1b = "; Format( vdVal1(2), "#0.0"); "@"; Format( vdVal2(2), "#0.0"); _  
    "; V1c = "; Format( vdVal1(3), "#0.0"); "@"; Format( vdVal2(3), "#0.0"); Chr(10); _  
    "V2a = "; Format( vdVal1(4), "#0.0"); "@"; Format( vdVal2(4), "#0.0"); _  
    "; V2b = "; Format( vdVal1(5), "#0.0"); "@"; Format( vdVal2(5), "#0.0"); _  
    "; V2c = "; Format( vdVal1(6), "#0.0"); "@"; Format( vdVal2(6), "#0.0")
```

## Function GetSteppedEvent

```
Function GetSteppedEvent( ByVal nStep&, ByRef dTime as Double, _  
                        ByRef dCurrent As double, ByRef nUserEvent as Long, _  
                        ByRef sEventDesc$, ByRef sFaultDesc$ ) As Long
```

**Purpose:** Retrieve detailed result of a step in stepped-event simulation .

**Works in:** *OneLiner* only.

### Parameters:

nStep	[in]	Sequential index of the event in the simulation(1 is the initial user-defined event)
dTime	[out]	Event time in seconds
dCurrent	[out]	Highest phase fault current magnitude at this step
nUserEvent	[out]	User defined event flag. 1= true; 0= false
sEventDesc	[out]	Event description string that includes list of all devices that had tripped.
sFaultDesc	[out]	Fault description string of the event.

### Return value:

Number of steps	When called with nStep = 0
1	success
0	failure

**Remarks:** Call this function with nStep = 0 to get total number of events simulated

**Example:** See Function DoSteppedEvent

## Function GetVSVoltage

```
Function GetVSVoltage( ByVal nHandle&, ByRef vdMag() as double ) As Long
```

**Purpose:** Retrieves voltage sag analysis result

**Works in:** *OneLiner* only.

**Parameters:**

nHandle	[in] bus handle
vdMag	[out] Magnitude of voltage sag at monitored bus in fault on this bus

**Return value:**

1	success
0	failure

**Remarks:** The size of arrays vdMag must be at least 4, one each for voltage drop result with fault connections: 1LG, 2LG, 3PH, LL respectively

**Example:**

```
If 0 = GetVSVoltage( nBusHnd, vdMag ) Then GoTo HasError

dSag# = -1.0
For ii=1 to 4
    If dSag < vdMag(ii) then dSag = vdmag(ii)
Next

Print "Voltage sag on fault at: " + FullBusName(nBusHnd) + " = " + Str(dSag)
```

## Function ErrorString

Function ErrorString( ) As String

**Purpose:** Retrieves the description of the last error. The result is a string.

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:** none.

**Remarks:**

**Example:**

```
HasError:
    Print "Error: ", ErrorString( )
    Stop
```

## Function GetWindowsEnvironmentVariable

```
Function GetWindowsEnvironmentVariable( ByRef sName$ ) As String
```

**Purpose:** Retrieves the contents of the specified variable from the Windows environment block. The result is a string.

**Works in:** *OneLiner and Power Flow.*

**Parameters:**

sName [in] The name of the environment variable

**Return value:** If the function succeeds, the return value is the content of the specified variable. If the function fails, the return value is an empty string.

**Example:**

```
Print "The Temp dir is: " & GetWindowsEnvironmentVariable( "TEMP" )
```

## Function LoadDataFile

```
Function LoadDataFile( ByRef sFileName $ ) As Long
```

**Purpose:** Read ASPEN data file.

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

sFileName	[in] Path name of the OLR or DXT file.
-----------	----------------------------------------

**Return value:**

0	Failure
1	Success

**Example:**

```
If 0 = LoadDataFile ( "c:\DataFolder\MyFile.olr" ) then GoTo HasError
```



## Function Locate1LObj

```
Function Locate1LObj( ByVal nHandle&, ByVal nShowNearest& ) As Long
```

**Purpose:** Locate an object on the 1-line diagram (bus, generator, load, shunt, switched shunt, transmission line, transformer, switch, phase shifter, relay group)

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

nHandle	[in]	Handle number of the object.
nShowNearest	[in]	If the object is not visible show nearest visible one 1 – Set; 0 - Reset

**Return value:**

0	Failure
1	Success

## Function MakeOutageList

```
Function MakeOutageList ( ByVal nHandle&, ByVal nMaxTiers&, _  
                        ByVal nWantedTypes, ByRef vnList() as long, _  
                        ByRef nListLen& ) As Long
```

**Purpose:** Return list of neighboring branches that can be used as outage list in the DoFault function on a bus, branch or relay group.

**Works in:** *OneLiner* only.

### Parameters:

nHandle	[in]	Handle number of a bus, relay group or branch.
nMaxTiers	[in]	Number of tiers (must be positive)
nWantedTypes	[in]	Branch type. Sum of one or more following values: 1- Line; 2- 2-winding transformer; 4- Phase shifter; 8- 3-winding transformer; 16- Switch
vnList	[out]	outage list (with zero in the last element).
nListLen	[in]	Length of vnList array
	[out]	Number of outage branches found.

### Return value:

0	Failure
1	Success

**Remarks:** Calling this function with 0 in place of vnList will let you determine the number of outage branches found within the number of tiers specified.

### Example:

```
Sub main  
  If GetEquipment( TC_PICKED, PickedHnd& ) = 0 Or _  
    (EquipmentType( PickedHnd& ) <> TC_BUS And EquipmentType( PickedHnd& ) <> TC_RLYGROUP) Then  
    Print "Please select a bus or a relay group"  
    Stop  
  End If  
  
  Call MakeOutageList(PickedHnd, 0, 1+2+4+8+16, 0, nListLen& )  
  Print nListLen  
  
  dim vnList(20) As long  
  nListLen& = 20  
  Call MakeOutageList(PickedHnd, 0, 1+2+4+8+16, vnList, nListLen& )  
  Print nListLen  
  For ii = 1 to nListLen  
    nHnd& = vnList(ii)  
    Print PrintObj1LPF(nHnd)  
  Next  
  exit Sub  
HasError:  
  Print GetErrorStr()  
End Sub
```

## Function NextBusByName

Function NextBusByName( ByRef nBusHandle& ) As Long

**Purpose:** Get handle of next bus in a bus list which has been sorted by bus name and nominal kV.

**Works in:** *OneLiner* and *Power Flow*.

### Parameters:

nBusHandle [in/out] bus handle

### Return value:

1	success
-1	already at last bus
0	failure

**Remarks:** To get the first bus in the list, call this function with nBusHandle = 0

### Example:

```
` Print bus list shorted by name and kV
nBusHnd& = 0
While NextBusByName( nBusHnd& ) > 0
    ' Print bus info
    If GetData( nBusHnd&, BUS_sName, sVal1$ ) = 0 Then GoTo HasError
    If GetData( nBusHnd&, BUS_dKVnominal, dVal1# ) = 0 Then GoTo HasError
    If GetData( nBusHnd&, BUS_nNumber, nVal1& ) = 0 Then GoTo HasError
    If GetData( nBusHnd&, BUS_nArea, nVal2& ) = 0 Then GoTo HasError
    If GetData( nBusHnd&, BUS_nZone, nVal3& ) = 0 Then GoTo HasError
    Print #1, "BUS    "; FullBusName( nBusHnd& );" AREA=";nVal2&;" ZONE="; nVal3&
Wend
```

## Function NextBusByNumber

Function NextBusByNumber( ByRef hHandle ) As Long

**Purpose:** Gets the handle of next bus in a bus list which has been sorted by bus number.

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

nBusHandle [in/out] bus handle

**Return value:**

1	success
-1	already at last bus
0	failure

**Remarks:** To get the first bus in the list, call this function with nBusHandle = 0

**Example:**

```
` Print bus list shorted by bus number
nBusHnd& = 0
While NextBusByNumber ( nBusHnd& ) > 0
    ' Print bus info
    If GetData( nBusHnd&, BUS_sName, sVal1$ ) = 0 Then GoTo HasError
    If GetData( nBusHnd&, BUS_dKVnormal, dVal1# ) = 0 Then GoTo HasError
    If GetData( nBusHnd&, BUS_nNumber, nVal1& ) = 0 Then GoTo HasError
    If GetData( nBusHnd&, BUS_nArea, nVal2& ) = 0 Then GoTo HasError
    If GetData( nBusHnd&, BUS_nZone, nVal3& ) = 0 Then GoTo HasError
    Print #1, "BUS      "; FullBusName( nBusHnd& );" AREA=";nVal2& " ZONE="; nVal3&
Wend
```

## Function PickFault

Function PickFault( ByVal nFltIndex& ) As Long

**Purpose:** Move output pointer to a specific short circuit simulation case. PickFault() is a simplified version of ShowFault(). All parameters in the previous call to ShowFault() are re-used. Use ShowFault() instead if you wish to control precisely what will be shown on the one-line diagram.

**Works in:** *OneLiner* only.

**Parameters:**

nFltIndex	[in] fault number or
	SF_FIRST: first fault
	SF_NEXT: next fault
	SF_PREV: previous fault
	SF_LAST: last available fault

**Return value:**

1	success
0	failure

**Remarks:** This function must be called before any post fault voltage and current result can be retrieved. All subsequent calls to GetSCVoltage and GetSCCurrent functions will return result from the picked short circuit simulation.

**Example:**

```
' Must always pick a fault before getting V and I results
If PickFault( 1 ) = 0 Then GoTo HasError
Do
  If GetSCCurrent( HND_SC, vdVal1, vdVal2, 4 ) = 0 Then GoTo HasError
  Print #1, FaultDescription(); Chr(10); _
  "
  Format( vdVal1(1), "###0.0"); "@"; Format( vdVal2(1), "#0.0"), Space(5), _
  Format( vdVal1(2), "###0.0"); "@"; Format( vdVal2(2), "#0.0"), Space(5), _
  Format( vdVal1(3), "###0.0"); "@"; Format( vdVal2(3), "#0.0")
Loop While ShowFault( SF_NEXT, 0, -999, 0, vnShowRelay ) > 0
```

## Function PostData

Function PostData( ByVal deviceHnd& ) As Long

**Purpose:** Perform data validation and update data for the given equipment in the network database.

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

deviceHnd      [in] data object handle

**Return value:**

1                success  
0                failure

**Remarks:** Changes to the equipment data made through SetData function will not be committed to the program network database until after this function has been executed with success.

**Example:**

```
If SetData( nSCapHnd, SC_dX, dXc ) = 0 Then GoTo HasError
If SetData( nSCapHnd, SC_dX0, dXc ) = 0 Then GoTo HasError
If SetData( nSCapHnd, SC_dR, dRc ) = 0 Then GoTo HasError
If SetData( nSCapHnd, SC_dR0, dRc ) = 0 Then GoTo HasError
If PostData( nSCapHnd ) = 0 Then GoTo HasError 'Save the whole thing
```

## Function ProgressDialog

```
Function ProgressDialog( ByVal nCommand&, ByRef sDlgTilte$, _  
                        ByRef sDlgText$, ByVal nPercent& ) As Long
```

**Purpose:** Display a progress dialog.

**Works in:** *OneLiner* and *Power Flow*.

### Parameters:

nCommand	[in]	Display mode: 0- Hide the dialog 1- Display the modeless dialog with Cancel button enabled 2- Display the modeless dialog with Cancel button disabled.
sDlgTilte	[in]	Dialog title string
sDlgText	[in]	Progress text string
nPercent	[in]	Percent progress (must be between 0-100)

### Return value:

2	Cancel button pressed
0	No button pressed

**Remarks:** The progress dialog is modeless, which allows the script execution to continue without interruption.

### Example:

```
Sub Main  
  Print "Start"  
  For ii = 1 to 100  
    For jj = 1 to 1000000  
      Next  
      Button = ProgressDialog( 1, "My Dialog", "Progress =" + Str(ii) + "%", ii )  
      If Button = 2 Then  
        Print "Cancel button pressed"  
        GoTo Done  
      End If  
    Next  
  Print "done"  
Done:  
  Call ProgressDialog( 0, "", "", 0 )  
End Sub
```

## Function PrintTTY

Function PrintTTY ( ByVal stringVal& ) As Long

**Purpose:** Output string to the TTY window of *OneLiner* or *Power Flow*.

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

stringVal [in] String to be displayed.

**Return value:**

1	success
0	failure

**Remarks:**

**Example:**

```
stringVal$ = "ANSI X/R = " & AnsiXR  
If PrintTTY(stringVal) = 0 Then GoTo HasError
```



## Function PrintObj1LPF

```
Function PrintObj1LPF ( ByVal nHandle& ) As String
```

**Purpose:** Return a text description of network database object (bus, generator, load, shunt, switched shunt, transmission line, transformer, switch, phase shifter, distance relay, overcurrent relay, fuse, recloser, relay group)

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

nHandle            [in] Object handle number.

**Return value:**

Object description string in format [ObjbType] Bus1-Bus2 ID

## Function ReadChangeFile

Function ReadChangeFile( ByRef sFileName\$, ByVal nFlag& ) As Long

**Purpose:** Read a change file.

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

sFileName	[in] Path name of the change file.
nFlag	[in] Silent mode: 0- false; 1- true; 2- true, save TTY log

**Return value:**

0	No error or warning
N	N = 1000*No or errors + No of warnings

**Important Remarks:** ReadChangeFile function will reset the script engine handle table, which will invalidate all handles that exist up to this point in the script program execution. For this reason user cannot re-use any of them in subsequence script calculation.

**Remarks:** Set nFlag to True to accept all changes in the change file without having to confirm each one separately. If Save TTY log is on, the content of TTY output generated by the read change file operation will be saved to a text file with default name: PowerScriptTTYLog.txt in the Windows %TEMP% folder.

**Example:**

```
Sub main
  CHFPath$ = "h:\data\dd_linemu_s.CHF"
  rCode = ReadChangeFile( CHFPath, 2 )
  Print "RCode = " & Str(rCode) & Chr(13) & Chr(10) _
    & "TTY output had been saved in file: " _
    & GetWindowsEnvironmentVariable( "TEMP" ) & "\PowerScriptTTYLog.txt"
End Sub
```

## Function Run1LPFCommand: ARCFLASHCALCULATOR

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner command: Fault | Arc-flash Hazard Calculator | Arc-flash Hazard Calculator (IEEE 1584-2011).

**Works in:** *OneLiner*.

### Parameters:

sInput           [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

### Return value:

1                Success  
0                Failure

**Remarks:** sInput must include XML node ARCFLASHCALCULATOR attributes in the list below (\* denotes required entries, [] denotes default value).

REPORTPATHNAME	(*) Full pathname of report file.
APPENDREPORT	[0] Append to existing report 0-No;1-Yes
OUTFILETYPE	[2] Output file type 1- TXT; 2- CSV
SELECTEDOBJ	Arcflash bus. Must have one of following values “PICKED “ the highlighted bus on the 1-line diagram “BNAME1',KV1;'BNAME2',KV2;...” Bus name and nominal kV.
TIERS	[0] Number of tiers around selected object. This attribute is ignored if SELECTEDOBJ is not found.
AREAS	[0-9999] Comma delimited list of area numbers and ranges to check relaygroups against backup. This attribute is ignored if SELECTEDOBJ is found.
ZONES	[0-9999] Comma delimited list of zone numbers and ranges to check relaygroups against backup. This attribute is ignored if AREAS or SELECTEDOBJ are found.
KVS	[0-999] Comma delimited list of KV levels and ranges to check relaygroups against backup. This attribute is ignored if SELECTEDOBJ is found.
TAGS	Comma delimited list of bus tags. This attribute is ignored if SELECTEDOBJ is found.
EQUIPMENTCAT	(*) Equipment category: 0-Switch gear; 1- Cable; 2- Open air; 3- MCC's and panelboards 1kV or lower
GROUND	(*) Is the equipment grounded 0-No; 1-Yes
ENCLOS	(*) Is the equipment inside enclosure 0-No; 1-Yes
CONDUCTOR	(*) Conductor gap in mm
WORKDIST	(*) Working distance in inches
ARCDURATION	Arc duration calculation method. Must have one of following values: “FIXED“ Use fixed duration “FUSE “ Use fuse curve “FASTEST“ Use fastest trip time of device in vicinity “DEVICE“ Use trip time of specified device “SEA“ Use stepped-event analysis
ARCTIME	Arc duration in second. Must be present when ARCDURATION=”FIXED”
FUSECURVE	Fuse curve for arc duration calculation. Must be present when ARCDURATION=”FUSECURVE”
BRKINTTIME	Breaker interrupting time in cycle. Must be present when ARCDURATION=”FASTEST” and “DEVICE”
DEVICETIERS	[1] Number of tiers. Must be present when ARCDURATION=”FASTEST” and “SEA”
DEVICE	String with location of the relaygroup and the relay name “BNO1;'BNAME1';KV1;BNO2;'BNAME2';KV2;'CKT';BTYP; RELAY_ID; ”. Format description of these fields are in OneLiner help section 10.2.
ARCTIMELIMIT	[1] Perform no energy calculation when arc duration time is longer than 2 seconds

**Example:**

```
Sub main()
  sInput$ = "< ARCFLASHCALCULATOR " & _
    "REPFILENAME=""c:\\000tmp\\arcflash.csv"" " & _
    "OUTFILETYPE=""2"" " & _
    "SELECTEDOBJ= ""'DOT BUS',13.8"" " & _
    "EQUIPMENTCAT=""0"" " & _
    "GROUNDED=""1"" " & _
    "ENCLOSED=""0"" " & _
    "CONDUCTORGAP=""153"" " & _
    "WORKDIST=""36"" " & _
    "ARCDURATION=""FUSE"" " & _
    "FUSECURVE=""ABB:CLE1-15-030"" " & _
    " />"
  Print sInput
  If Run1LPFCommand( sInput ) Then
    Print "Success"
  Else
    Print ErrorString()
  End If
End Sub
```

## Function Run1LPFCommand: ARCFLASHCALCULATOR2018

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner command: Fault | Arc-flash Hazard Calculator | Arc-flash Hazard Calculator (IEEE 1584-2018).

**Works in:** *OneLiner*.

### Parameters:

sInput           [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

### Return value:

1                Success  
0                Failure

**Remarks:** sInput must include XML node ARCFLASHCALCULATOR attributes in the list below (\* denotes required entries, [] denotes default value).

REPORTPATHNAME	(*) Full pathname of report file.
APPENDREPORT	[0] Append to existing report 0-No;1-Yes
OUTFILETYPE	[2] Output file type 1- TXT; 2- CSV
SELECTEDOBJ	Arcflash bus. Must have one of following values “PICKED “ the highlighted bus on the 1-line diagram “BNAME1',KV1;'BNAME2',KV2;...” Bus name and nominal kV.
TIERS	[0] Number of tiers around selected object. This attribute is ignored if SELECTEDOBJ is not found.
AREAS	[0-9999] Comma delimited list of area numbers and ranges to check relaygroups against backup. This attribute is ignored if SELECTEDOBJ is found.
ZONES	[0-9999] Comma delimited list of zone numbers and ranges to check relaygroups against backup. This attribute is ignored if AREAS or SELECTEDOBJ are found.
KVS	[0-999] Comma delimited list of KV levels and ranges to check relaygroups against backup. This attribute is ignored if SELECTEDOBJ is found.
TAGS	Comma delimited list of bus tags. This attribute is ignored if SELECTEDOBJ is found.
ELECTRODECFG	(*) Electrode configuration: 0-VCB: vertical, inside metal enclosure; 1- VCB: vertical, inside metal enclosure, with insulating barrier; 2- HCB: Horizontal, inside metal enclosure; 3- VOA: Vertical, in open air; 4- HOA: Horizontal, in open air;
BOXH	Enclosure height in inches. Required for electrode configurations VCB, VCB, HCB.
BOXW	Enclosure width in inches. Required for electrode configurations VCB, VCB, HCB.
BOXD	Enclosure depth in inches. Required for electrode configurations VCB, VCB, HCB at voltage level of 600 or lower.
CONDUCTORGAP	(*) Conductor gap in mm
WORKDIST	(*) Working distance in inches
ARCDURATION	Arc duration calculation method. Must have one of following values: “FIXED“ Use fixed duration “FUSE “ Use fuse curve “FASTEST“ Use fastest trip time of device in vicinity “DEVICE“ Use trip time of specified device
ARCTIME	Arc duration in second. Must be present when ARCDURATION=“FIXED”
FUSECURVE	Fuse LibraryName:CurveName for arc duration calculation. Must be present when ARCDURATION=“ FUSECURVE”
BRKINTTIME	Breaker interrupting time in cycle. Must be present when ARCDURATION=“ FASTEST” and “DEVICE”
DEVICETIERS	[1] Number of tiers. Must be present when ARCDURATION=“ FASTEST” and =“SEA”
DEVICE	String with location of the relaygroup and the relay name “BNO1;'BNAME1';KV1;BNO2;'BNAME2';KV2;'CKT';BTYP; RELAY_ID; ”. Format description of these fields are in OneLiner help section 10.2.

**Example:**

```
Sub main()
  sInput$ = "<ARCFLASHCALCULATOR2018 " & _
    "REPFILENAME=""c:\\000tmp\\arcflash.csv"" " & _
    "OUTFILETYPE=""2"" " & _
    "SELECTEDOBJ=""DOT BUS",13.8"" " & _
    "ELECTRODECFG=""0"" " & _
    "BOXH=""36"" " & _
    "BOXW=""24"" " & _
    "CONDUCTORGAP=""153"" " & _
    "WORKDIST=""36"" " & _
    "ARCDURATION=""FUSE"" " & _
    "FUSECURVE=""ABB:CLE1-15-030"" " & _
    " />"
  Print sInput
  If Run1LPFCommand( sInput ) Then
    Print "Success"
  Else
    Print ErrorString()
  End If
End Sub
```

## Function Run1LPFCommand: BUSFAULTSUMMARY

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner command: Faults |Bus fault summary.

**Works in:** *OneLiner*.

### Parameters:

sInput           [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

### Return value:

1                Success  
0                Failure

**Remarks:** sInput must include XML node BUSFAULTSUMMARY with attributes in the list below (\* denotes required entries, [] denotes default value).

REPORTPATHNAME (\*) full valid path to report file

BASELINECASE     pathname of base-line bus fault summary report in CSV format

=== *Only when BASELINECASE is specified*

DIFFBASE         Basis for computing current deviation: [MAX3PH1LG] or MAXPHGND

FLAGPCNT         [15] Current deviation percent threshold.

=== *Only when BASELINECASE is not specified*

BUSLIST           Bus list, one on each row in format 'BusName',kV

BUSNOLIST         Bus number list, coma delimited. This attribute is ignored when BUSLIST is specified

=== *Only when no BUSLIST and BUSNOLIST is specified*

XGND             Fault reactance X

RGND             Fault resistance R

NOTAP            Exclude tap buses: [1]-TRUE; 0-FALSE

PERUNITV         Report voltage in PU

PERUNIT          Report current in PU

AREAS            Area number range

ZONES            Zone number range. This attribute is ignored when AREAS is specified

BUSNOS           Additional bus number range

KVS              Additional bus kV range

TAGS             Additional tag filter

TIERS            check lines in vicinity within this tier number

AREAS            Check all lines in area range

ZONES            Check all lines in zone range

KVS              Additional KV filter

### Example:

```
Sub main
  sInput$ = "<BUSFAULTSUMMARY " & _
    "REPORTPATHNAME=""c:\000tmp\report.csv"" " & _
    "BUSNOLIST=""10,20,60"" " & _
    " />"
  If Run1LPFCommand( sInput ) Then
    Print "Success"
  Else
    Print ErrorString()
  End If
End Sub
```

## Function Run1LPFCommand: CHECKPRIBACKCOORD

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner command: Relay | Check primary/backup relay coordination.

**Works in:** *OneLiner*.

**Parameters:**

sInput [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

**Return value:**

1 Success  
0 Failure

**Remarks:** sInput must include XML node CHECKPRIBACKCOORD with attributes in the list below (\* denotes required entries, [] denotes default value).

REPORTPATHNAME (\*) Full pathname of report file.

OUTFILETYPE [2] Output file type 1- TXT; 2- CSV

SELECTEDOBJ Relay group to check against its backup. Must have one of following values  
"PICKED" the highlighted relaygroup on the 1-line diagram  
"BNO1;'BNAME1';KV1;BNO2;'BNAME2';KV2;'CKT';BTYP;" location string of the relaygroup. Format description is in OneLiner help section 10.2.

PAIRTYPE Coordination pair type to check:  
[0] Check group against its backups  
1 Check group against groups that it backs up  
2 Check all pairs that involve the group

This attribute is ignored if SELECTEDOBJ is not found.

TIERS [0] Number of tiers around selected object. This attribute is ignored if SELECTEDOBJ is not found.

AREAS [0-9999] Comma delimited list of area numbers and ranges to check relaygroups against backup. This attribute is ignored if SELECTEDOBJ is found.

ZONES [0-9999] Comma delimited list of zone numbers and ranges to check relaygroups against backup. This attribute is ignored if AREAS or SELECTEDOBJ are found.

KVS [0-999] Comma delimited list of KV levels and ranges to check relaygroups against backup. This attribute is ignored if SELECTEDOBJ is found.

TAGS Comma delimited list of tags to check relaygroups against backup. This attribute is ignored if SELECTEDOBJ is found.

COORDTYPE Coordination type to check. Must have one of following values  
"0" OC backup/OC primary (Classical)  
"1" OC backup/OC primary (Multi-point)  
"2" DS backup/OC primary  
"3" OC backup/DS primary  
"4" DS backup/DS primary  
"5" OC backup/Recloser primary  
"6" All types/All types

LINEPERCENT Percent interval for sliding intermediate faults. This attribute is ignored if COORDTYPE is 0 or 5.

RUNINTERMEOP 1-true; 0-false. Check intermediate faults with end-opened. This attribute is ignored if COORDTYPE is 0 or 5.

RUNCLOSEIN 1-true; 0-false. Check close-in fault. This attribute is ignored if COORDTYPE is 0 or 5.

RUNCLOSEINEOP 1-true; 0-false. Check close-in fault with end-opened. This attribute is ignored if COORDTYPE is 0 or 5.

RUNLINEEND 1-true; 0-false. Check line-end fault. This attribute is ignored if COORDTYPE is 0 or 5.

RUNREMOTEBUS 1-true; 0-false. Check remote bus fault. This attribute is ignored if COORDTYPE is 0 or 5.



RELAYTYPE	Relay types to check: 1-Ground; 2-Phase; 3-Both.
FAULTTYPE	Fault types to check: 1-3LG; 2-2LG; 4-1LF; 8-LL; or sum of values for desired selection
OUTPUTALL	1- Include all cases in report; 0- Include only flagged cases in report
MINCTI	Lower limit of acceptable CTI range
MAXCTI	Upper limit of acceptable CTI range
OUTRLYPARAMS	Include relay settings in report: 0-None; 1-OC;2-DS;3-Both
OUTAGELINES	Run line outage contingency: 0-False; 1-True
OUTAGEXFMRS	Run transformer outage contingency: 0-False; 1-True
OUTAGEMULINES	Run mutual line outage contingency: 0-False; 1-True. Ignored if OUTAGEMULINES=0
OUTAGEMULINESGND	Run mutual line outage and grounded contingency: 0-False; 1-True. Ignored if OUTAGEMULINES=0
OUTAGE2LINES	Run double line outage contingency: 0-False; 1-True. Ignored if OUTAGEMULINES=0
OUTAGE1LINE1XFMR	Run double line and transformer outage contingency: 0-False; 1-True. Ignored if OUTAGEMULINES=0 or OUTAGEXFMRS =0
OUTAGE2XFMR	Run double and transformer outage contingency: 0-False; 1-True. Ignored if OUTAGEXFMRS =0
OUTAGE3SOURCES	Outage only 3 strongest sources: 0-False; 1-True. Ignored if OUTAGEMULINES=0 and OUTAGEXFMRS =0

#### Example:

```

Sub main()
  sInput$ = "<CHECKPRIBACKCOORD " & _
    "REPFILENAME=""c:\\000tmp\\checkcoord.csv"" " & _
    "OUTFILETYPE=""1"" " & _
    "SELECTEDOBJ=""6; 'NEVADA'; 132.; 8; 'REUSENS'; 132.; '1'; 1;"" " & _
    "COORDTYPE=""6"" " & _
    "OUTPUTALL=""1"" " & _
    "MINCTI=""0.05"" " & _
    "MAXCTI=""99"" " & _
    "LINEPERCENT=""15"" " & _
    "RELAYTYPE=""3"" " & _
    "FAULTTYPE=""5"" " & _
    " />"

  Print sInput
  If Run1LPFCommand( sInput ) Then
    Print "Success"
  Else
    Print ErrorString()
  End If
End Sub

```

## Function Run1LPFCommand: CHECKRELAYOPERATIONSEA

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner command: Relay | Check relay operations using stepped-events.

**Works in:** *OneLiner*.

### Parameters:

sInput [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

### Return value:

1 Success  
0 Failure

**Remarks:** sInput must include XML node CHECKRELAYOPERATIONSEA with attributes in the list below (\* denotes required entries, [] denotes default value).

REPORTPATHNAME	(*) Full pathname of folder for report files.
REPORTCOMMENT	Additional comment string to include in all checking report files
SELECTEDOBJ	Check line with selected relaygroup. Must have one of following values "PICKED " the highlighted relaygroup on the 1-line diagram "BNO1;'BNAME1';KV1;BNO2;'BNAME2';KV2;'CKT';BTYP;" location string of the relaygroup. Format description is in OneLiner help section 10.2.
TIERS	[0] Number of tiers around selected object. This attribute is ignored if SELECTEDOBJ is not found.
AREAS	[0-9999] Comma delimited list of area numbers and ranges.
ZONES	[0-9999] Comma delimited list of zone numbers and ranges. This attribute is ignored if AREAS is found.
KVS	[0-999] Comma delimited list of KV levels and ranges. This attribute is ignored if SELECTEDOBJ is found.
TAGS	Comma delimited list of tags. This attribute is ignored if SELECTEDOBJ is found.
DEVICETYPE	Space delimited list of relay type types to take into consideration in stepped-events: OCG, OCP, DSG, DSP, LOGIC, VOLTAGE, DIFF
FAULTTYPE	Space delimited list of fault types to take into consideration in stepped-events: 1LF, 3LG
OUTAGELINES	Run line outage contingency: 0-False; 1-True
OUTAGEXFMRS	Run transformer outage contingency: 0-False; 1-True
OUTAGEMULINES	Run mutual line outage contingency: 0-False; 1-True. Ignored if OUTAGEMULINES=0
OUTAGEMULINESGND	Run mutual line outage and grounded contingency: 0-False; 1-True. Ignored if OUTAGEMULINES=0
OUTAGE2LINES	Run double line outage contingency: 0-False; 1-True. Ignored if OUTAGEMULINES=0
OUTAGE1LINE1XFMR	Run double line and transformer outage contingency: 0-False; 1-True. Ignored if OUTAGEMULINES=0 or OUTAGEXFMRS =0
OUTAGE2XFMR	Run double and transformer outage contingency: 0-False; 1-True. Ignored if OUTAGEXFMRS =0
OUTAGE3SOURCES	Outage only 3 strongest sources: 0-False; 1-True. Ignored if OUTAGEMULINES=0 and OUTAGEXFMRS =0

### Example:

```
Sub main
  sInput$ = "<CHECKRELAYOPERATIONSEA " & _
    "REPORTPATHNAME=""c:\000tmp\""" & _
    "SELECTEDOBJ=""6; 'NEVADA'; 132.; 8; 'REUSENS'; 132.; '1'; 1;"" " & _
    " />"
  Print sInput
  If Run1LPFCommand( sInput ) Then Print "Success" Else Print ErrorString()
End Sub
```

## Function Run1LPFCommand: CHECKRELAYOPERATIONPRC023

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner command: Relay | Check relay loadability

**Works in:** *OneLiner*.

**Parameters:**

sInput [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

**Return value:**

1 Success  
0 Failure

**Remarks:** sInput must include XML node CHECKRELAYOPERATIONPRC023 with attributes in the list below (\* denotes required entries, [] denotes default value).

REPORTPATHNAME (\*) full valid pathname of report file  
REPORTCOMMENT Report comment string. 255 char or shorter  
SELECTEDOBJ:  
    PICKED Check devices in selected relaygroup  
    BNO1;'BNAME1';KV1;BNO2;'BNAME2';KV2;'CKT';BTYP; location string of branch to check(OneLiner Help section 10.2)  
TIERS check relaygroups in vicinity within this tier number  
AREAS Check all relaygroups in area range  
ZONES Check all relaygroups in zone range  
KVS Additional KV filter  
TAGS Additional tag filter  
USETAGFLAG [0]-AND;[1]-OR  
DEVICETYPE [OCP DSP] Devide type to check. Space delimited  
APPENDREPORT Append report file: 0-False; [1]-True  
LINERATINGTYPE [3] Line rating to use: 0-first; 1-second; 2-Third; 3-Fourth  
XFMRRATINGTYPE [2] Transformer rating to use: 0-MVA1; 1-MVA2; 2-MVA3  
FWRLOADONLY [0] Consider load in forward direction only  
VOLTAGEPU [0.85] Per unit voltage  
LINECURRMULT [1.5] Line load current multiplier  
XFMRCURRMULT [1.5] Transformer load current multiplier  
PFANGLE [30] Power factor angle

**Example:**

```
Sub main
  sReportFile$ = GetOlrFileName() & "_PRC23.CSV"
  sInput$ = "<CHECKRELAYOPERATIONPRC023 " & _
    "REPORTPATHNAME=""" & sReportFile & """" " & _
    "SELECTEDOBJ="""0; 'CLAYTOR'; 132; 0; 'NEVADA'; 132; '1'; 1;"" " & _
    "LOADAMPS="""1000"" " & _
    " />"

  Print sInput
  If Run1LPFCommand( sInput ) Then
    Print "Success Report in " & sReportFile
  Else
    Print "Error: " & ErrorString()
  End If
End Sub
```

## Function Run1LPFCommand: CHECKRELAYOPERATIONPRC026

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner command: Relay | Check relay performance in stable power swing (PRC-026-1).

**Works in:** *OneLiner*.

### Parameters:

sInput           [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

### Return value:

1                Success  
0                Failure

**Remarks:** sInput must include XML node CHECKRELAYOPERATIONPRC026 with attributes in the list below (\* denotes required entries, [] denotes default value).

REPORTPATHNAME	(*) Full pathname of checking report.
REPORTCOMMENT	Additional comment string to include in all checking report files
APPENDREPORT	[1] Append existing report file: 0-False; 1-True
SELECTEDOBJ	Check line with selected relaygroup. Must have one of following values "PICKED " the highlighted relaygroup on the 1-line diagram "BNO1;'BNAME1';KV1;BNO2;'BNAME2';KV2;'CKT';BTYP;" location string of the relaygroup. Format description is in OneLiner help section 10.2.
TIERS	[0] Number of tiers around selected object. This attribute is ignored if SELECTEDOBJ is not found.
AREAS	[0-9999] Comma delimited list of area numbers and ranges.
ZONES	[0-9999] Comma delimited list of zone numbers and ranges. This attribute is ignored if AREAS is found.
KVS	[0-999] Comma delimited list of KV levels and ranges. This attribute is ignored if SELECTEDOBJ attribute is present.
TAGS	Comma delimited list of tags. This attribute is ignored if SELECTEDOBJ attribute is present.
DEVICETYPE	Space delimited list of relay type types to take into consideration in stepped-events: OCP, DSP.
SEPARATIONANGLE	[120] System separation angle for stable power swing calculation.
DELAYLIMIT	[15] Report violation if relay trips in stable power swing with delay faster than this limit (in cycles)
CURRMULT	[1.0] Current multiplier to apply in relay checking performance in stable power swing

### Example:

```
Sub main
  sReportFile = "c:\000tmp\PRC026.xml"
  sInput$ = "<CHECKRELAYOPERATIONPRC026 " & _
    "REPORTPATHNAME=""" & sReportFile & """ " & _
    "KVS="""200-500"" " & _
    " />"

  If Run1LPFCommand( sInput ) Then
    Print "Success Report in " & sReportFile
  Else
    Print "Error: " & ErrorString()
  End If
End Sub
```

## Function Run1LPFCommand: CHECKRELAYSETTINGS

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner command: Relay | Check relay settings.

**Works in:** *OneLiner*.

### Parameters:

sInput [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

### Return value:

1 Success  
0 Failure

**Remarks:** sInput must include XML node CHECKRELAYSETTINGS with attributes in the list below (\* denotes required entries, [] denotes default value).

REPORTPATHNAME	(*) Full pathname of folder for report files.
REPORTCOMMENT	Additional comment string to include in all checking report files
SELECTEDOBJ	Check line with selected relaygroup. Must have one of following values "PICKED " the highlighted relaygroup on the 1-line diagram "BNO1;'BNAME1';KV1;BNO2;'BNAME2';KV2;'CKT';BTYP;" location string of the relaygroup. Format description is in OneLiner help section 10.2.
TIERS	[0] Number of tiers around selected object. This attribute is ignored if SELECTEDOBJ is not found.
AREAS	[0-9999] Comma delimited list of area numbers and ranges.
ZONES	[0-9999] Comma delimited list of zone numbers and ranges. This attribute is ignored if AREAS is found.
KVS	[0-999] Comma delimited list of KV levels and ranges. This attribute is ignored if SELECTEDOBJ attribute is present.
TAGS	Comma delimited list of tags. This attribute is ignored if SELECTEDOBJ attribute is present.
DEVICETYPE	Space delimited list of relay type types to take into consideration in stepped-events: OCG, OCP, DSG, DSP
FAULTTYPE	Space delimited list of fault types to take into consideration in stepped-events: 1LF, 3LG
OUTAGELINES	Run line outage contingency: 0-False; 1-True
OUTAGEXFMRS	Run transformer outage contingency: 0-False; 1-True
OUTAGEMULINES	Run mutual line outage contingency: 0-False; 1-True. Ignored if OUTAGEMULINES=0
OUTAGEMULINESGND	Run mutual line outage and grounded contingency: 0-False; 1-True. Ignored if OUTAGEMULINES=0
OUTAGE2LINES	Run double line outage contingency: 0-False; 1-True. Ignored if OUTAGEMULINES=0
OUTAGE1LINE1XFMR	Run double line and transformer outage contingency: 0-False; 1-True. Ignored if OUTAGEMULINES=0 or OUTAGEXFMRS =0
OUTAGE2XFMR	Run double and transformer outage contingency: 0-False; 1-True. Ignored if OUTAGEXFMRS =0
OUTAGE3SOURCES	Outage only 3 strongest sources: 0-False; 1-True. Ignored if OUTAGEMULINES=0 and OUTAGEXFMRS =0

### Example:

```
Sub main
  sInput$ = "<CHECKRELAYOPERATIONSEA " & _
    "REPORTPATHNAME=""c:\000tmp\" & _
    "SELECTEDOBJ=""6; 'NEVADA'; 132.; 8; 'REUSENS'; 132.; '1'; 1;"" " & _
    " />"
  If Run1LPFCommand( sInput ) Then Print "Success" Else Print ErrorString()
End Sub
```

## Function Run1LPFCommand: EXPORTNETWORK

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner command: File | Export network data command.

**Works in:** *OneLiner*.

**Parameters:**

sInput            [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

**Return value:**

1                Success  
0                Failure

**Remarks:** sInput must include XML node EXPORTNETWORK with attributes in the list below (\* denotes required entries, [] denotes default value).

FORMAT	Output format: [DXT]-ASPEN DXT; PSSE-PSS/E Raw and Seq
SCOPE	Export scope: [0]-Entire network; 1-Area number; 2- Zone number
AREANO	Export area number
ZONENO	Export zone number
INCLUDETIES	Include ties: [0]-False; 1-True
====DXT export only:	
DXTPATHNAME	(*) full valid pathname of ouput DXT file
====PSSE export only:	
RAWPATHNAME	(*) full valid pathname of ouput RAW file
SEQPATHNAME	(*) full valid pathname of ouput SEQ file
PSSEVER	[33] PSS/E version
X3MIDBUSNO	[18000] First fictitious bus number for 3-w transformer mid point
NEWBUSNO	[15000] First bus number for buses with no bus number

**Example:**

```
Sub main
  sExportFile$ = GetOlrFileName() & ".raw"
  sExportFileSEQ$ = GetOlrFileName() & ".seq"
  sInput$ = "<EXPORTNETWORK " & _
    "FORMAT=""PSSE"" " & _
    "RAWPATHNAME="" " & sExportFile & "" " & _
    "SEQPATHNAME="" " & sExportFileSEQ & "" " & _
    "PSSEVER= ""32"" X3MIDBUSNO=""21000"" NEWBUSNO=""31000"" " & _
    "SCOPE= ""1"" AREANO=""1"" INCLUDETIES=""1"" " & _
    " />"

  Print sInput
  If Run1LPFCommand( sInput ) Then
    Print "Success: output in " & sExportFile
  Else
    Print "Error: " & ErrorString()
  End If
End Sub
```

## Function Run1LPFCommand: EXPORTRELAY

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner command: File | Export network data command.

**Works in:** *OneLiner*.

### Parameters:

sInput           [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

### Return value:

1                Success  
0                Failure

**Remarks:** sInput must include XML node EXPORTRELAY with attributes in the list below (\* denotes required entries, [] denotes default value).

FORMAT        Output format: [RAT]-ASPEN RAT;  
SCOPE         Export scope: [0]-Entire network; 1-Area number; 2- Zone number; 3-Invicinity of a bus  
AREANO        Export area number (\*required when SCOPE=1)  
ZONENO        Export zone number (\*required when SCOPE=2)  
SELECTEDOBJ   Selected bus (\*required when SCOPE=3). Must be a string with following content  
              PICKED - Selected bus on the 1-line diagram  
              'BusName' kV - Bus name in single quotes and kV separated by space  
TIERS         [0] Number of tiers (ignored when SCOPE<>3)  
DEVICETYPE    Device type to export. Comma delimited list of the following:  
              OC: Overcurrent  
              DS: Distance  
              RC: Recloser  
              VR: Voltage relay  
              DIFF: Differential relay  
              SCHEME: Logic scheme  
              COORDPAIR: Coordination pair  
              [OC,DS,RC,VR,DIFF,COORDPAIR,SCHEME]  
LASTCHANGEDDATE [01-01-1986] Cutoff last changed date  
RATPATHNAME   (\*) full valid pathname of output RAT file

### Example:

```
Sub main
  sExportFile$ = GetOlrFileName() & ".rat"
  sInput$ = "<EXPORTRELAY " & _
           "RATPATHNAME=""" & sExportFile & """ " & _
           "SCOPE= ""3"" SELECTEDOBJ="""CLAYTOR' 132"" " & _
           "DEVICETYPE= ""OC,DS"" " & _
           " />"
  Print sInput
  If Run1LPFCommand( sInput ) Then
    Print "Success: output in " & sExportFile
  Else
    Print "Error: " & ErrorString()
  End If
End Sub
```

## Function Run1LPFCommand: FAULTSOLUTIONREPORT

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner commands: Faults | Solution Report.

**Works in:** *OneLiner*.

**Parameters:**

sInput [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

**Return value:**

1 Success  
0 Failure

**Remarks:** sInput must include XML node FAULTSOLUTIONREPORT with attributes in the list below (\* denotes required entries, [] denotes default value).

REPORTPATHNAME	(*) full valid path to report file
OUTFILETYPE	[2] Output file type 1- TXT; 2- CSV
CASERANGE	[0] Comma delimited list of case index numbers. For example 1,5-10. 0 - Current case
PERUNITIZ	[0] Unit of current and impedance 1- Per-unit; 0- Amps and ohms
PERUNITV	[1] Unit of voltage 1- Per-unit; 0- kV, L-G
TIER	Report scope: Number of tiers from the faulted bus
BUSLIST	Report scope: List of additional buses. This attribute is ignored if TIER is found
MONITORBRANCH1	Report scope: Monitored branch 1 location string (OneLiner Help section 10.2) BNO1;'BNAME1';KV1;BNO2;'BNAME2';KV2;'CKT';BTYP; This attribute is ignored if TIER or BUSLIST is found.
MONITORBRANCH2	Report scope: Monitored branch 2 location string.
MONITORBRANCH3	Report scope: Monitored branch 3 location string.
HEADING	Additional text to appear on the header of report
LISTEQUIPMENT	List deleted and outagd equipment in the report [0] - NO; 1 - YES

**Example:**

```
Sub main
  sExportFile$ = GetOlrFileName() & ".csv"
  sInput$ = "<FAULTSOLUTIONREPORT " & _
    "REPORTPATHNAME=""" & sExportFile & """ " & _
    "OUTFILETYPE="""2"" " & _
    "CASERANGE="""1,3-5"" " & _
    "PERUNITIZ="""0"" " & _
    "PERUNITV="""0"" " & _
    "TIER="""0"" " & _
    "BUSLIST="""'ALASKA',33;'HAWAII',33;"" " & ">"
  If Run1LPFCommand( sInput ) Then
    Print "Success: output in " & sExportFile
  Else
    Print "Error: " & ErrorString()
  End If
End Sub
```



## Function Run1LPFCommand: FAULTLOCATOR

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner commands: Faults | Fault Locator.

**Works in:** *OneLiner*.

### Parameters:

sInput            [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

### Return value:

1                Success  
0                Failure

**Remarks:** sInput must include XML node FAULTLOCATOR with attributes in the list below (\* denotes required entries, [] denotes default value).

REPORTPATHNAME	(*) full valid path to report file
RELAYGROUP1	(*) Relay group 1 location string (OneLiner Help section 10.2) BNO1;'BNAME1';KV1;BNO2;'BNAME2';KV2;'CKT';BTYP;
PHASORSET1	(*) Recorded fault voltage and current phasor data at relay group 1 (Va_m Va_a Vb_m Vb_a Vc_m Vc_a Ia_m Ia_a Ib_m Ib_a Ic_m Ic_a)
PHASORSET3	Recorded phasor data (pre-fault) at relay group 1 (Va_m Va_a Vb_m Vb_a Vc_m Vc_a Ia_m Ia_a Ib_m Ib_a Ic_m Ic_a)
FLTR	Maximum fault impedance [30]
ENDOPEN	Simulate remote end open: [0] -False; 1 - TRue
RELAYGROUP2	Relay group 2 (Remote) location string
PHASORSET2	Recorded fault voltage and current phasor data at relay group 2 (Va_m Va_a Vb_m Vb_a Vc_m Vc_a Ia_m Ia_a Ib_m Ib_a Ic_m Ic_a)
ERRORMETHOD	Algorithm selection flag: an integer number with corresponding binary bits set or reset accordingly: 1- Auto; 2- Impedance; 3- Reactance; 4- Takagi; 5- Modified Takagi; 6- Novosel; 7- Eriksson. For example if you want to use Auto, Reactance and Modified Takagi, then the flag is $1+2^2+2^4=21$ . If no flag is given, all applicable algorithms will be selected by default.

### Example:

```
Sub main()  
  xml$ = xmlMakeNode ( "FAULTLOCATOR" )  
  Call xmlSetAttribute( xml, "REPORTPATHNAME", sReportFile )  
  Call xmlSetAttribute( xml, "RELAYGROUP1", "1228; '732_MITSUE E'; 240.; 1260; '809_LOU CREE';  
240.; '1'; 1;" )  
  Call xmlSetAttribute( xml, "PHASORSET1", "141.014 142.426 59.2049 0 62.9214 280.56 85.9079  
239.439 2261.95 -50.2205 2216.97 174.78" )  
  Call xmlSetAttribute( xml, "ERRORMETHOD", "21" )  
  If "Y" <> InputBox( xml, "Continue?", "Y" ) Then Stop  
  If 1 = Run1LPFCommand( xml ) Then  
    Print "Success"  
  Else  
    Print "Failure: " & ErrorString()  
  End If  
End Sub
```

## Function Run1LPFCommand: INSERTTAPBUS

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner commands: Network | Insert tap bus

**Works in:** *OneLiner*.

### Parameters:

sInput           [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

### Return value:

1                Success  
0                Failure

**Remarks:** sInput must include XML node INSERTTAPBUS with attributes in the list below (\* denotes required entries, [] denotes default value).

BUSNAME1	(*) Line bus 1 name
BUSNAME2	(*) Line bus 2 name
KV	(*) Line kV
CKTID	(*) Line circuit ID
PERCENT	(*) Percent distance to tap from bus 1 (must be between 0-100)
TAPBUSNAME	(*) Tap bus name

### Example:

```
Sub main
  sCmd$ = "<INSERTTAPBUS " & " ' Command name
  sCmd = sCmd & "BUSNAME1=""NEVADA"" " & " ' Line bus 1 name
  sCmd = sCmd & "BUSNAME2=""CLAYTOR"" " & " ' Line bus 2 name
  sCmd = sCmd & "KV=""132"" " & " ' Line kV
  sCmd = sCmd & "CKTID=""1"" " & " ' Line circuit ID
  sCmd = sCmd & "PERCENT=""30"" " & " ' Percent distance from bus 1
  sCmd = sCmd & "TAPBUSNAME=""NEWTAP"" " & " ' Tap bus name
  sCmd = sCmd & ">"
  If 1 <> Run1LPFCommand( sCmd ) Then GoTo hasError
  If 1 <> Run1LPFCommand( "<SAVEDATAFILE PATHNAME =""NewFile.olr"" />" ) Then GoTo hasError
  Print "Success"
  stop
hasError:
  Print "Error: " + ErrorString()
End Sub
```

## Function Run1LPFCommand: SETGENREFANGLE

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner command: Network | Set generator reference angle command.

**Works in:** *OneLiner*.

**Parameters:**

sInput           [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

**Return value:**

1                Success  
0                Failure

**Remarks:** sInput must include XML node SETGENREFANGLE with attributes in the list below (\* denotes required entries, [] denotes default value).

REPORTPATHNAME   Full pathname of folder for report files.

REFERENCEGEN     Bus name and kV of reference generator in format: 'BNAME', KV.

EQUSSOURCEOPTION   Option for calculating reference angle of equivalent sources. Must have one of the following values

[ROTATE] apply delta angle of existing reference gen

SKIP            Leave unchanged. This option will be in effect automatically when old reference is not valid

ASGEN           Use angle computed for regular generator

**Example:**

```
Sub main
  sInput$ = "<SETGENREFANGLE " & _
    "REPORTPATHNAME=""c:\000tmp\setrefangle.txt"" " & _
    "EQUSSOURCEOPTION=""SKIP"" " & _
    "REFERENCEGEN=""HANCOCK' 13.8"" " & _
    " />"
  Print sInput
  If Run1LPFCommand( sInput ) Then Print "Success" Else Print ErrorString()
End Sub
```

## Function Run1LPFCommand: SAVEDATAFILE

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner commands: File | Save and File | Save as.

**Works in:** *OneLiner*.

**Parameters:**

sInput           [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

**Return value:**

1                Success  
0                Failure

**Remarks:** sInput must include XML node SAVEDATAFILE with attributes in the list below (\* denotes required entries, [] denotes default value).

PATHNAME        Name or full pathname of new OLR file for File | Save as command. If only file name is given, file will be saved in the folder where the current OLR file is located.

If no attribute is specified, the File | Save command will be executed.

**Example:**

```
Sub main
  sInput$ = "< SAVEDATAFILE " & _
           " PATHNAME =""c:\000tmp\newfile.olar""
           " />"
  Print sInput
  If Run1LPFCommand( sInput ) Then Print "Success" Else Print ErrorString()
End Sub
```

## Function Run1LPFCommand: SIMULATEFAULT

Function Function Run1LPFCommand( sInput\$ ) As Long

**Purpose:** Run OneLiner command FAULTS | BATCH COMMAND & FAULT SPEC FILE | EXECUTE COMMAND.

**Works in:** *OneLiner*.

**Parameters:**

sInput           [in] XML string, or full path name to XML file, containing XML node as described in *Remarks* section below.

**Return value:**

1                Success  
0                Failure

**Remarks:** sInput must include XML node SIMULATEFAULT with one or multiple children nodes of <FAULT>. Each <FAULT> node can have up to 40 <FLTSPEC> children nodes, which contain the value of fault specification string data as described in OneLiner's user manual APPENDIX I: FAULT SPECIFICATION FILE. The fault spec data can also be entered as attributes of <FLTSPEC> node (\* denotes required entries, [] denotes default value).

FLTDESC           Fault description String enclosed in quotes. Description can have up to 127 characters.

PHASETYPE        (\*)Which phase(es) are involved in the fault.

1LG: 0="A"; 1="B"; 2="C"

2LG Or LL: 0="A-B"; 1="B-C";2="A-C"

Bus-to-bus: 0="A-A";1="A-B";2="A-C";3="B-B";4="B-C";5="C-C"

FLTCONN        (\*)Fault connection code. Enter 0 For 3LG; 1 For 2LG; 2 For 1LG; 3 For L-L.

FLTAPPL        (\*)Fault application code. Enter 0 For bus fault; 1 For Close-in fault;

2 For bus-to-bus fault; 3 For Line-End fault; 4 For intermediate fault;

5 For branch outage; 6,7,8 For 1-, 2- And 3-phase Open.

FPARAM           Intermediate fault location in percent.

FRA FXA FRB FXB FRC FXC FRGg FXG

Resistance And reactance value of fault impedances Za, Zb, Zc And Zg per figure in section SPECIFY SIMULTANEOUS FAULT COMMAND.

Fault impedance is Not used when FLTAPPL is 5, 6, 7 Or 8, in which case all values must be zero.

BUS1NAME        (\*)Name of first fault bus in quotes. Name can have up to 13 characters.

BUS1KV           (\*)Nominal kV of first fault bus in quotes. Name can have up to 13 characters.

Following data fields are required only If FLTAPPL is other than 0

BUS2NAME        Name of Second fault bus in quotes. Name can have up to 13 characters.

BUS2KV           Nominal kV of Second fault bus in quotes. Name can have up to 13 characters.

BRTYPE           Fault branch Type code. Enter 1 For transmission Line; 2 For 2-winding transformer;

3 For phase shifter; 7 For switch; 10 For 3-winding transformer.

CKTID            Faulted branch circuit ID.

### Example:

Fault specification as <FLTSPEC> node string:

```
Sub main
  sInput$ = "<SIMULATEFAULT>" & _
    "<FAULT>" & _
    "<FLTSPEC>" & _
    ""'Bus Fault on:          28 ARIZONA      132. kV 1LG Type=A' 0 2 0 0 0 0 0 0 0 0
0 'ARIZONA' 132"" " & _
    "</FLTSPEC>" & _
    "</FAULT>" & _
    "</SIMULATEFAULT>"
  Print sInput
  If Run1LPFCommand( sInput ) Then Print "Success" Else Print ErrorString()
End Sub
```

Fault specification as <FLTSPEC> node attributes:

```
Sub main
  sInput$ = "<SIMULATEFAULT>" & _
    "<FAULT>" & _
    "<FLTSPEC " & _
    "FLTDESC=""Bus Fault On:          28 ARIZONA      132. kV 1LG Type=A "" " & _
    "PHASETYPE=""0"" " & _
    "FLTCONN=""2"" " & _
    "FLTAPPL=""0"" " & _
    "FPARAM=""0"" " & _
    "FRA=""0"" " & _
    "FXA=""0"" " & _
    "FRB=""0"" " & _
    "FXB=""0"" " & _
    "FRC=""0"" " & _
    "FXC=""0"" " & _
    "FRG=""0"" " & _
    "FXG=""0"" " & _
    "BUS1NAME=""ARIZONA"" " & _
    "BUS1KV=""132"" " & _
    ">" & _
    "</FAULT>" & _
    "</SIMULATEFAULT>"
  Print sInput
  If Run1LPFCommand( sInput ) Then Print "Success" Else Print ErrorString()
End Sub
```

## Function SaveDataFile

```
Function SaveDataFile( ByRef sFileName$, ByVal nFlag& ) As Long
```

**Purpose:** Save network data to binary file.

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

sFileName	[in]	Full path name of the OLR file.
nFlag	[in]	Bit 1: Overwrite mode: 1- true; 0- false. Bit 2: Silent mode: 1- true; 0- false.

**Return value:**

0	Failure
1	Success

**Remarks:** Call this function with empty sFileName to save the network under current file name.  
Set bit 1 of nFlag to True to attempt to overwrite existing content of the file. Set flat bit 2 to suppress on-screen messges.

**Example:**

## Function SetData

Function SetData( ByVal deviceHnd&, ByVal paramID&, ByVal newValue ) as Long

**Purpose:** Reads the value of a network datum into a program variable.

**Works in:** *OneLiner* and *Power Flow*.

### Parameters:

deviceHnd	[in] data object handle
paramID	[in] parameter ID code (must be labeled with write attribute in table 3.3)
newValue	[in] new value.

### Return value:

1	success
0	failure

**Remarks:** Data type of variable `newValue` must agree with that of the parameter being updated. See Table 3.3 for a full listing of equipment parameters and their type.

### Example:

```
If SetData( nSCapHnd, SC_dX, dXc ) = 0 Then GoTo HasError
If SetData( nSCapHnd, SC_dX0, dXc ) = 0 Then GoTo HasError
If SetData( nSCapHnd, SC_dR, dRc ) = 0 Then GoTo HasError
If SetData( nSCapHnd, SC_dR0, dRc ) = 0 Then GoTo HasError
If PostData( nSCapHnd ) = 0 Then GoTo HasError 'Save modified data to network database
```



## Function SetObjMemo

```
Function SetObjMemo( ByVal deviceHnd&, ByRef sObjMemo$ ) as Long
```

**Purpose:** Set memo field of the object.

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

deviceHnd	[in] data object handle
sObjMemo	[in] object memo string

**Return value:**

1	success
0	failure

**Remarks:**

**Example:**

```
If SetObjMemo( nSCapHnd, "ABCD" ) = 0 Then GoTo HasError      'Set object memo to "ABCD"
```

## Function SetObjTags

```
Function SetObjTags( ByVal deviceHnd&, ByRef sObjTags$ ) as Long
```

**Purpose:** Set tags field of the object.

**Works in:** *OneLiner* and *Power Flow*.

**Parameters:**

deviceHnd	[in] data object handle
sObjTags	[in] object tag string

**Return value:**

1	success
0	failure

**Remarks:**

**Example:**

```
If SetObjTags( nSCapHnd, "ABCD" ) = 0 Then GoTo HasError      'Set object tags to "ABCD"
```

## Function ShowFault

```
Function ShowFault( ByVal nFltIndex&, ByVal nTiers&, ByVal nShowType&, _  
    ByVal nPerUnit&, ByRef vnShowRelay() as Long ) As Long
```

**Purpose:** Show result of a fault simulation case on one-line diagram.

**Works in:** *OneLiner* only.

### Parameters:

nFltIndex	[in] fault number to show or: SF_FIRST: first fault SF_NEXT: next fault SF_PREV: previous fault SF_LAST: last fault
nTiers	[in] number of tiers away from the fault bus
nShowType	[in] output type = 1: zero sequence result = 2: positive sequence result = 3: negative sequence result = 4: phase A result = 5: phase B result = 6: phase C result = 7: relay operating time result
nPerUnit	[in] per unit flag; 1 – set; 0 - reset
vnShowRelay	[in] show relay flag; 1 – set; 0 - reset vnShowRelay(1) - Overcurrent ground relays vnShowRelay(2) - Overcurrent phase relays vnShowRelay(3) - Distance ground relays vnShowRelay(4) - Distance phase relays

### Return value:

1	success
0	failure

### Remarks:

### Example:

```
Dim vnShowRelay(4)  
If ShowFault( 1, 0, 4, 0, vnShowRelay ) = 0 Then GoTo HasError  
Do  
    If GetSCCurrent( HND_SC, vdVal1, vdVal2, 4 ) = 0 Then GoTo HasError  
    Print #1, FaultDescription(); Chr(10); _  
    " "  
    Format( vdVal1(1), "####0.0"); "@"; Format( vdVal2(1), "#0.0"), Space(5), _  
    Format( vdVal1(2), "####0.0"); "@"; Format( vdVal2(2), "#0.0"), Space(5), _  
    Format( vdVal1(3), "####0.0"); "@"; Format( vdVal2(3), "#0.0")  
Loop While ShowFault( SF_NEXT, 0, -999, 0, vnShowRelay ) > 0
```



# APPENDIX A: Cypress Enable Scripting Language Elements

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

---

## Comments

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

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

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

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

Everything after the apostrophe is a comment.

---

## Statements

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

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

Which is equivalent to:

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

---

## Line Continuation Character

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

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

---

## Numbers

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

---

## Variable and Constant Names

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

---

## Variable Types

### Variant

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

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



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

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

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

### Variants and Concatenation

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

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

---

## Other Data Types

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

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

Object	Dim X As Object	4 bytes
Date	Dim D As Date	8 bytes
User Defined Types		size of each element

---

## Scope of Variables

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

## Declaration of Variables

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

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

## Control Structures

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

```
Goto label1
.
.
.

label1:
```

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

### Loop Structures

#### Do Loops

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

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



```

Do Until condition
Statement(s)...
Loop

Do
Statements...
Loop While condition

Do
statements...
Loop Until condition

```

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

### While Loop

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

```

While condition
statements...
Wend

```

### For ... Next Loop

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

```

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

```

### If and Select Statements

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

```

If condition Then Statements...           'single line syntax

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

```

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

```

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

```

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

```

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

```

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

---

## Subroutines and Functions

### Naming conventions

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

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

---

## ByRef and ByVal

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

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

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

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

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

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

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

### Scalar Variables

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

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

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

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

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

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

Sub Main
    Dim ofs As OFSTRUCT
    ` Print OF_READWRITE
    ofs.szPathName = "c:\enable\openfile.bas"
    print ofs.szPathName
    ofs.nErrCode = 5
    print ofs.nErrCode
    OpenFile "t.bas", ofs
    print ofs.szPathName
```

```

        print ofs.nErrCode
    End Sub

```

---

## Calling Procedures in DLLs

DLLs or Dynamic-link libraries are used extensively by Engineers to functions and subroutines located there. There are two main ways that Enable can be extended, one way is to call functions and subroutines in DLLs and the other way is to call functions and subroutines located in the calling application. The mechanisms used for calling procedures in either place are similar. (See the Declare Statement for more 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 Enable are Global to the run and accessible by all subroutines and functions. If the procedure does not return a value, declare it as a subroutine. If the procedure does have a return value declare it as a function.

```

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

```

```

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

```

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

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

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

### Passing and Returning Strings

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

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

### OLE API

```

SysAllocString/SysAllocStringLen
SysAllocString/SysAllocStringLen
SysFreeString

```

```

SysStringLen
SysReAllocStringLen
SysReAllocString

```

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

---

## File Input/Output

Enable supports full sequential and binary file I/O.

Functions and Statements that apply to file access:

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

```

' File I/O Examples

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

End Sub

Sub test

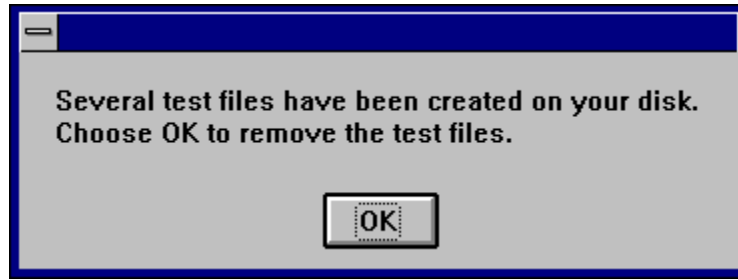
Open "MYFILE" For Input As #1 ' Open file for input.
Do While Not EOF(1) ' Check for end of file.
    Line Input #1, InputData ' Read line of data.
    MsgBox InputData
Loop
Close #1 ' Close file.

End Sub

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

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

```



---

## Arrays

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

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

Ways to declare a fixed-size array:

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

Cypress Enable supports Dynamic arrays.

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

```
Dim ArrayName (10) As 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. Enable Basic does not allow this; you must declare an array before using it.

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

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

## MultiDimensional Arrays

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

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

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

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

You can efficiently process a multidimensional array with the use of for loops. In the following statements the 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. Enable 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 Enable Code. User defined types are global. The variables that are declared as user defined types can be either global or local. User Defined Types in Enable cannot contain arrays at this time

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

Type type2
  a As Integer
  o As type1
End Type

Dim type2a As type2
Dim type1a As type1

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





---

# Dialog Support

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

## Dialog Box controls

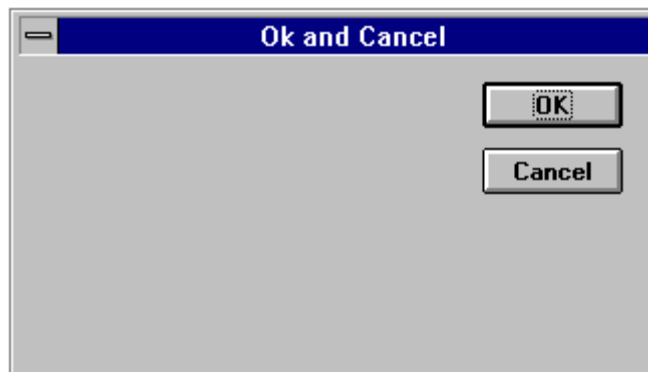
Enable Basic supports the standard Windows dialog box controls. This section introduces the controls available for custom dialog boxes and provides guidelines for using them.

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

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

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

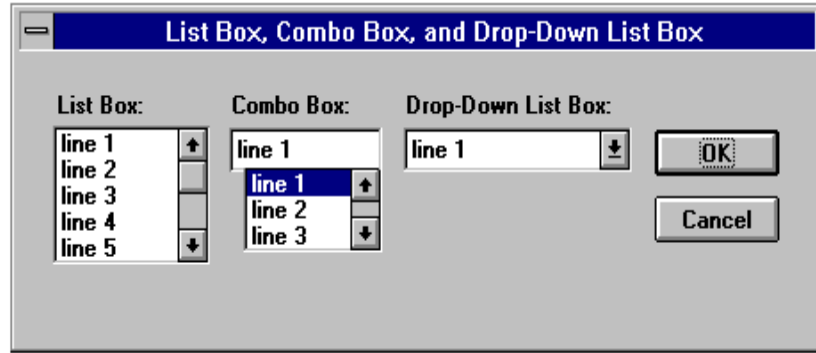
## OK and Cancel Buttons



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

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

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



```

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

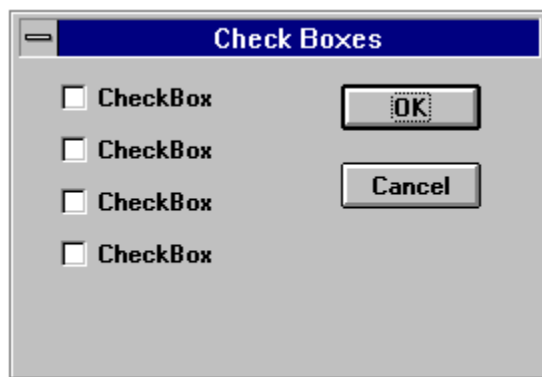
    Begin Dialog BoxSample 16,35,256,89,"List Box, Combo Box, and Drop-Down
List Box"
        OKButton 204,24,40,14
        CancelButton 204,44,40,14
        ListBox 12,24,48,40, MyList$ ( ),.Lstbox
        DropListBox 124,24,72,40, MyList$ ( ),.DrpList
        ComboBox 68,24,48,40, MyList$ ( ),.CmboBox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

End Sub

```

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

### Check Boxes



```

Sub Main

```

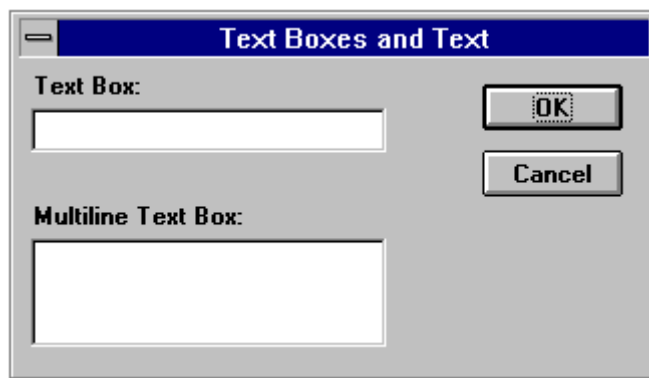
```

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

```

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

## Text Boxes and Text



```

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

End Sub

```

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

```

'=====
' This sample shows how to implement a multiline textbox
'=====
Const ES_LEFT      = &h0000& 'Try these different styles or-ed
together
Const ES_CENTER    = &h0001& ' as the last parameter of Textbox
the change
Const ES_RIGHT     = &h0002& ' the text box style.
Const ES_MULTILINE = &h0004& ' A 1 in the last parameter position
defaults to
Const ES_UPPERCASE = &h0008& ' A multiline, Wantreturn,
AutoVScroll 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&

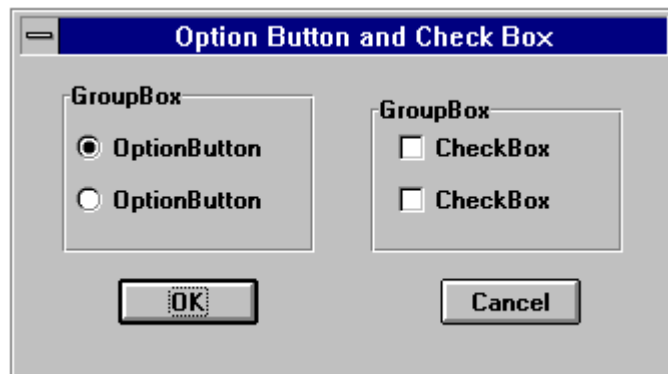
Sub Multiline
    Begin Dialog DialogType 60, 60, 140, 185, "Multiline text Dialog",
    .DlgFunc
        TextBox 10, 10, 120, 150, .joe, ES_MULTILINE Or ES_AUTOVSCROLL Or
ES_WANTRETURN ' Indicates multiline TextBox
        'TextBox 10, 10, 120, 150, .joe, 1 ' indicates multi-line textbox
        CancelButton 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

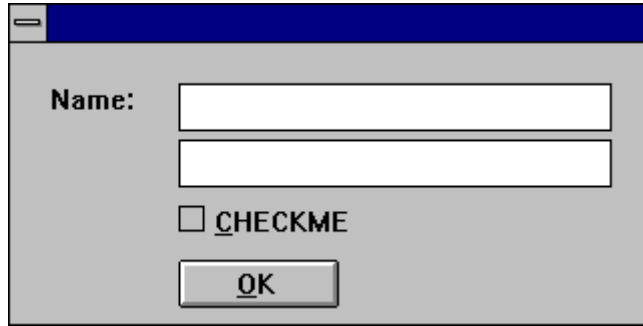
You can have option buttons to allow the user to choose one option from several. Typically, you would use a group box to surround a group of option buttons, but you can also use a group box to set off a group of check boxes or any related group of controls.



```

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, 54, 12
  End Dialog
  Dim Dlg1 As DialogName1
  Dialog Dlg1

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

```

### The Dialog Function

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

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

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

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

### The Dialog Box Controls

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

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

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

## The Dialog Function Syntax

The syntax for the dialog function is as follows:

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

All parameters in the dialog function are required.

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

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

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

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

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

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

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

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

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

---

## Statements and Functions Used in Dialog Functions

Statement or Function	Action or Result
DlgControlId	Returns the numeric equivalent of Identifier\$, the string identifier for a dialog box control.
DlgEnable, DlgEnable()	The <b>DlgEnable</b> statement is used to enable or disable a dialog box control. When a control is disabled, it is visible in the dialog box, but is dimmed and not functional. <b>DlgEnable()</b> is used to determine whether or not the control is enabled.
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", .Enable
        Text 8,10,73,13, "Text Label:"
        TextBox 8, 26, 160, 18, .FText
        CheckBox 8, 56, 203, 16, "Check to display controls",. Chk1
        GroupBox 8, 79, 230, 70, "This is a group box:",.Group
        CheckBox 18,100,189,16, "Check to change button text",.Chk2
        PushButton 18, 118, 159, 16, "File History",.History
        OKButton 177, 8, 58, 21
        CancelButton 177, 32, 58, 21
    End Dialog
```

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

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

```
    Begin Dialog UserDialog2 160,160, 260, 188, "3", .Enable
        Text 8,10,73,13, "New dialog Label:"
        TextBox 8, 26, 160, 18, .FText
        CheckBox 8, 56, 203, 16, "New CheckBox",. ch1
        CheckBox 18,100,189,16, "Additional CheckBox",.ch2
        PushButton 18, 118, 159, 16, "Push Button",.but1
```

```

        OKButton 177, 8, 58, 21
        CancelButton 177, 32, 58, 21
    End Dialog
    Dim Dlg2 As UserDialog2
    Dlg2.FText = "Your default string goes here"

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

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

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

    Case Else

    End Select
    Enable =1

End Function

```

---

## OLE Automation

### What is OLE Automation?

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

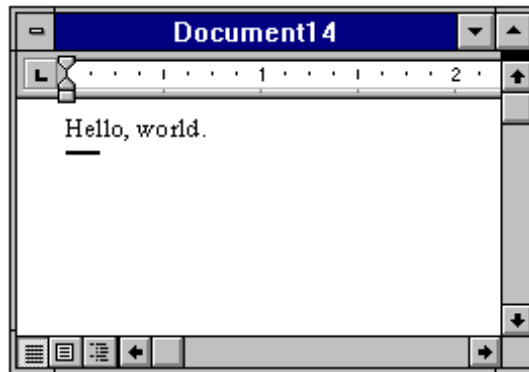
When an application supports OLE Automation, the objects it exposes can be accessed by Enable Basic. You can use Enable Basic to manipulate these objects by invoking methods on the object, or by getting and setting the object's properties, just as you would with the objects in Enable Basic. For example,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

```

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

```

End Sub



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

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

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

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

---

## Accessing an object

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

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

---

## What is an OLE Object?

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

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

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

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

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

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

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

		Subject
		Title
		Value

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

Collection object	Methods	Properties
Documents	Add	Application
	Close	Count
	Item	Parent
	Open	

---

## OLE Fundamentals

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

The following terms and concepts are fundamental to understanding OLE.

### OLE Object

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

### OLE Automation

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

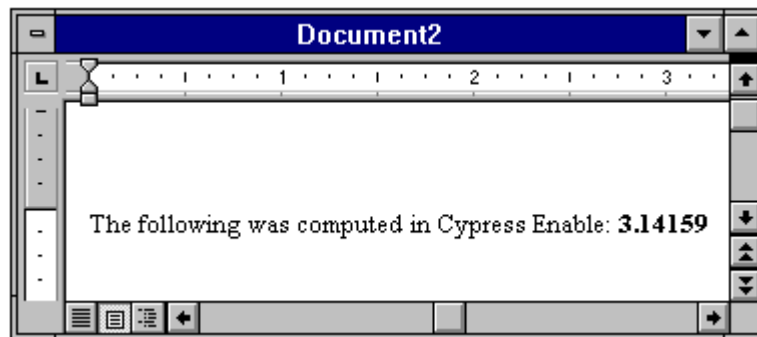
Class

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

---

## OLE Automation and Microsoft Word example:

```
Sub OLEexample()  
    Dim word As Object  
    Dim myData As String  
  
    myData = 4 * Atn(1)      ' Demonstrates Automatic type conversion  
    Set word = CreateObject("Word.Basic")  
    Word.AppShow  
    word.FileNewDefault  
    word.Insert "The following was computed in Cypress Enable: "  
    word.Bold 1              ' Show value in boldface  
    word.Insert myData  
    word.Bold 0  
  
    MsgBox "Done"  
End Sub
```



---

## Making Applications Work Together

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

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

### WIN.INI

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

### The Registration Database.

Starting with Windows 3.1, Each Windows system maintains a *registration database* file that records details about the DDE and OLE functions supported by the installed

applications. The database is stored in file called **REG.DAT** in the \ **WINDOWS** directory.

---

## The Registration database

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

### **Associations.**

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

### **Shell Operations.**

Windows contains two programs that are 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.





# APPENDIX B: Cypress Enable Scripting Language Overview

---

## Quick reference of the Functions and Statements available

### Type/Functions/Statements

---

#### Flow of Control

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

---

#### Converting

Chr, Hex, Oct, Str, CDBl, CInt, 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, Get, Put

---

#### Math

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

---

#### Procedures

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

---

#### Strings

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

---

## Variables and Constants

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

---

## Error Trapping

On Error, Resume

---

## Date/Time

Date, Now, Time, Timer

---

## DDE

DDEInitiate, DDEExecute, DDETerminate

---

## Arrays

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

---

## Miscellaneous

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

## Data Types

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

## Operators

### Arithmetic Operators

Operator	Function	Usage
^	Exponentiation	$x = y^2$
-	Negation	$x = -2$
*	Multiplication	$x\% = 2 * 3$
/	division	$x = 10/2$
Mod	Modulo	$x = y \text{ Mod } z$
+	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	$x \leq Y$
=	Equals	$x = Y$
>=	Greater than or equal to	$x \geq Y$
>	Greater than	$x > Y$
<>	Not equal to	$x \neq Y$

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

# APPENDIX C: Cypress Enable Language Reference A-Z

---

---

## Abs Function

### *Abs (number)*

Returns the absolute value of a number.

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

### **Example:**

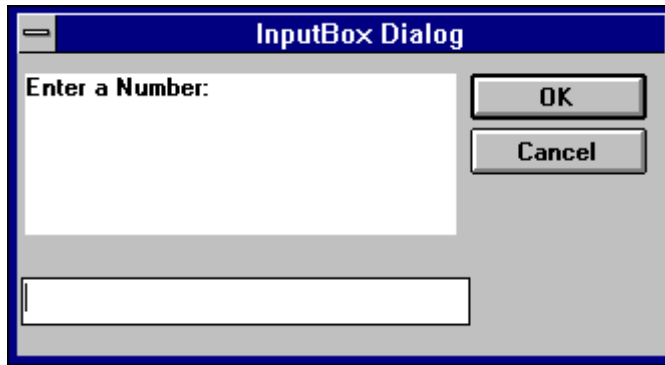
```
Sub Main

    Dim Msg, X, Y

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

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

End Sub
```




---

## AppActivate Statement

*AppActivate "app"*

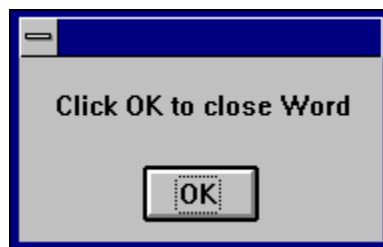
Activates an application.

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

Related Topics: [Shell](#), [SendKeys](#)

### Example:

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




---

## Asc Function

*Asc ( str)*

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

### Example:

```

Sub Main ()
    Dim I, Msg
    For I = Asc("A") To Asc("Z")
        Msg = Msg & Chr(I)
    Next I
    MsgBox Msg
End Sub

```

' Declare variables.  
' From A through Z.  
' Create a string.  
' Display results.

---

## 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

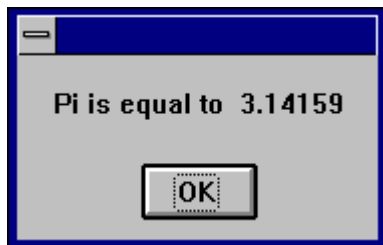
### Example:

```

Sub AtnExample ()
    Dim Msg, Pi
    Pi = 4 * Atn(1)
    Msg = "Pi is equal to " & Str(Pi)
    MsgBox Msg
End Sub

```

' Declare variables.  
' Calculate Pi.  
' Display results.




---

## 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
    Do
        Answer = InputBox("Enter a value from 1 to 3.")
        If Answer >= 1 And Answer <= 3 Then

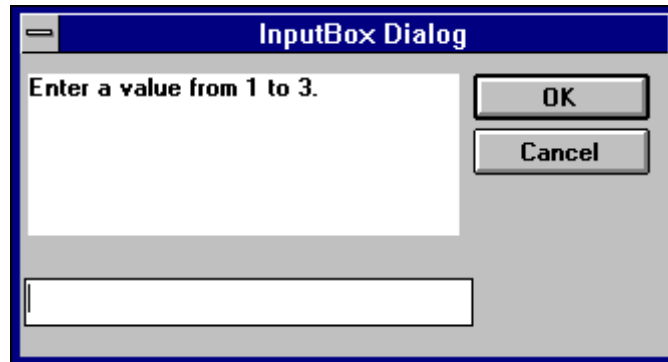
```

' Declare variables.  
' Check range.

```

Exit Do                                ' Exit Do...Loop.
Else
    Beep                               ' Beep if not in range.
End If
Loop
MsgBox "You entered a value in the proper range."
End Sub

```




---

## Call Statement

```

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

```

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

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

### Example:

```

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

```




---

## CBool Function

CBool (*expression*)



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

**Example:**

```
Sub Main

    Dim A, B, Check

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

    A = 0
    Check = CBool(A)
    Print Check

End Sub
```

---

## CDate Function

*CVDate (expression)*

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

**Example:**

```
Sub Main

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

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

    Print MDate
    Print MStime

End Sub
```

---

## CDbl Function

*CDbl (expression)*

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

### Example:

```
Sub Main ()
    Dim y As Integer

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

        x = CDb1(y) 'Converts the integer value of y to a double value in
        x
        x = x * 100000 'y is now 10 digits in the form of x
        Print x
    End If
End Sub
```

---

## ChDir Statement

ChDir *pathname*

Changes the default directory

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

The parameter *pathname* is a string limited to fewer 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, Mkdir, Rmdir

---

## CheckBox

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

For selecting one or more in a series of choices

### Example:

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

        CHECKBOX 42, 10, 48, 12, "&CHECKME", .checkInt
        OKBUTTON 42, 24, 40, 12
    End Dialog
    Dim Dlg1 As DialogName1
    Dialog Dlg1
    If Dlg1.checkInt = 0 Then
        Q = "didn't check the box."
    Else
        Q = "checked the box."
    End If
    MsgBox "You " & Q
End Sub
```



---

## Choose Function

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

Returns a value from a list of arguments

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

### Example:

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

---

## Chr Function

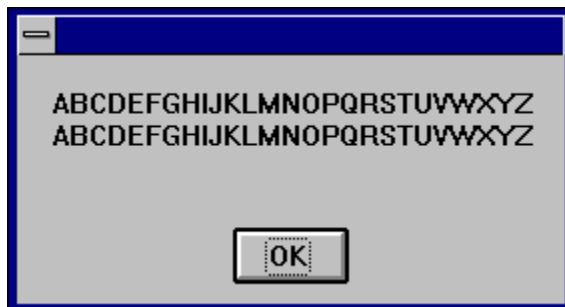
*Chr(int )*

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

Chr returns a String

### Example:

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



---

## CInt Function

**CInt** (*expression*)

Converts any valid expression to an integer.

### Example:

```
Sub Main ()
    Dim y As Long

    y = 25
    Print VarType(y)

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

---

## CLng Function

**CLng** (*expression*)

Converts any valid expression into a long.

### Example:

```
Sub Main ()
    Dim y As Integer

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

---

## Close Statement

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

The Close Statement takes one argument *filename*. *Filename* is the number used with the Open Statement to open the file. If the

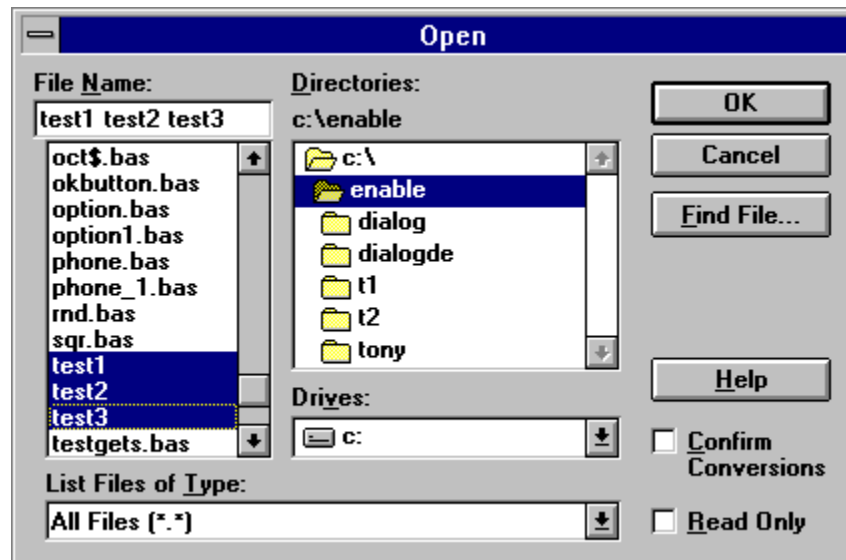
Close Statement is used without any arguments it closes all open files.

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

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

### Example:

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

---

## Cos Function

### Cos (*rad*)

Returns the cosine of an angle

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

### Example:

```
Sub Main()
    Dim J As Double
    Dim I As Single
    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
    ' Declare variables.
    ' Cos function call
    ' Display results.
```

End Sub

---

## CreateObject Function

### CreateObject (*class*)

Creates an OLE automation object.

```
Sub Command1_Click ()
    Dim word6 As object
    Set word6 = CreateObject("Word.Basic")
    word6.FileNewDefault
    word6.InsertPara
    word6.Insert "Attn:"
    word6.InsertPara
    word6.InsertPara
    word6.InsertPara
    word6.Insert "                Vendor Name: "
    word6.Bold 1
    name = "Some Body"
    word6.Insert name
    word6.Bold 0
    word6.InsertPara
    word6.Insert "                Vendor Address:"
    word6.InsertPara
    word6.Insert "                Vendor Product:"
    word6.InsertPara
    word6.InsertPara
    word6.Insert "Dear Vendor:"
    word6.InsertPara
    word6.InsertPara
    word6.Insert "The letter you are reading was created with Cypress Enable."
    word6.Insert " Using OLE Automation Cypress Enable can call any other OLE _ enabled "
    word6.Insert "application. Enable is a Basic Scripting Language for _ applications"
    word6.InsertPara
    word6.InsertPara
    word6.Insert "                Product Name: Cypress Enable"
    word6.InsertPara
    word6.Insert "                Company Name: Cypress Software Inc."
    word6.InsertPara

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

Vendor Name: **Client Name**  
Vendor Address:  
Vendor Product:

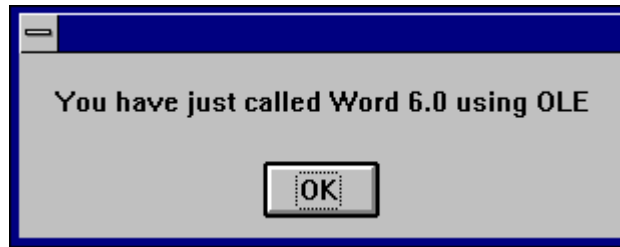
Dear Vendor:

The letter you are reading was created with Cypress Enable.Using  
OLE Automation Cypress Enable can call any other OLE enabled



application. Enable is a Basic Scripting Language for applications

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



---

## CSng Function

*CSng (expression)*

Converts any valid expression to a Single.

**Example:**

```
Sub Main ()
    Dim y As Integer

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

---

## CStr Function

*CStr(expression)*

Converts any valid expression to a String.

**Example:**

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

---

## CurDir Function

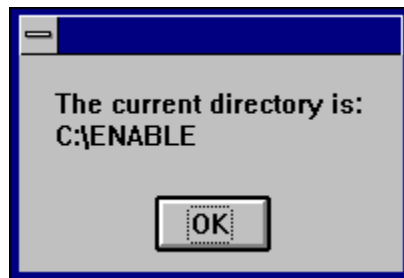
*CurDir (drive)*

Returns the current path for the specified drive

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

### Example:

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



---

## CVar Function

*CVar (expression)*

Converts any valid expression to a Variant.

### Example:

```
Sub Main

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

End Sub
```

---

# Date Function

Date, Date()

Returns the current system date

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

## Example:

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

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

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

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

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

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

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

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

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

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

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

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

```

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

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

End Sub

```

---

## DateSerial Function

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

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

**Related Topics:** DateValue, TimeSerial, TimeValue

### Example:

```

Sub Main

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

End Sub

```

---

## DateValue Function

**DateValue**(*dateexpression*)

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

**Related Topics:** DateSerial, TimeSerial, TimeValue

### Example:

```

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

```

---

## Day Function

*Day(dateexpression)*

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

**Related Topics:** Month, Weekday, Hour, Second

### Example:

```
Sub Main

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

End Sub
```

---

## Declare Statement

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

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

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

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

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

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

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

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

data type. Valid types are String, Integer, Double, Long, and 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" ()
```

**Cypress Enable 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 Enable is embedded.

Related Topics: Call

### Example:

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

    MsgBox "TextBox: " & Dlg1.joe
    MsgBox "ListBox: " & Dlg1.MyList1
    MsgBox Dlg1.Combo1
    MsgBox Dlg1.DropList1
    MsgBox "grp1: " & Dlg1.grp1
    MsgBox "grp2: " & Dlg1.grp2
    Begin Dialog DialogName2 60, 60, 160, 60, "Test Dialog 2"
        Text 10, 10, 28, 12, "Name:"
```

```

        TextBox 42, 10, 108, 12, .fred
        OkButton 42, 44, 40, 12
    End Dialog
    If button = 2 Then
        Dim Dlg2 As DialogName2
        Dialog Dlg2
        MsgBox Dlg2.fred
    ElseIf button = 1 Then
        Dialog Dlg1
        MsgBox Dlg1.Combo1
    End If
End Sub

```

---

## Dim Statement

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

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

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

### Example:

```

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

```

---

## Dir Function

**Dir**[(*path,attributes*)]

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

## Example:

```

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

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

```



```

Dim frame As BitmapDlg

' Show the bitmap dialog
Dialog frame
End Sub

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

    DlgFunc = 1      ' Keep dialog active

    Select Case action
    Case 1 ' Initialize
        temp = Dir( "c:\Windows\*.bmp" )
        count = 0
        While temp <> ""
            count = count + 1
            temp = Dir
        Wend
        Dim x() As String
        ReDim x(count)
        x(0) = Dir( "c:\Windows\*.bmp" )
        For i = 1 To count
            x(i) = dir
        Next i
        DlgListBoxArray "List1", x()
    Case 2 ' Click
        fileName = "c:\windows\" & DlgText("List1")
        DlgSetPicture "Picture1", fileName
    End Select
End Function

```

---

## DlgEnable Statement

### DlgEnable “*ControlName*”, *Value*

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

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

Related Topics: DlgVisible, DlgText

### Example:

```

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

```

```

        OKButton 177, 8, 58, 21
        CancelButton 177, 32, 58, 21
    End Dialog

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

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

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

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

        If ControlID$ = "but1" Then

        End If

    Case Else

    End Select
    Enable =1
End Function

```

---

## DlgText Statement

DlgText “*ControlName*”, *String*

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

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

Related Topics: DlgEnable, DlgVisible

### Example:

```

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

```

---

## DlgVisible Statement

DlgVisible "*ControlName*", *Value*

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

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

Related Topics: DlgEnable, DlgText

### Example:

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

---

## Do...Loop Statement

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

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

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

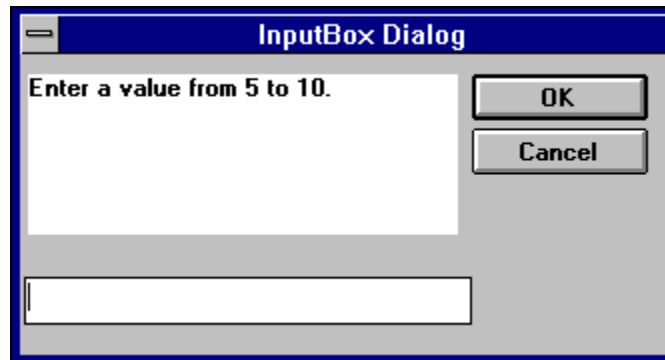
Related Topics: While, Wend

### Example:

```

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

```




---

## End Statement

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

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

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

### Example:

```

Sub Main()

    Dim Var1 as String

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

End Sub

Sub Test(wvar1 as string)

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

End Sub

```

---

# EOF Function

## *EOF(Filenumber)*

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

Related Topics: Open Statement

### Example:

```
' Input Function Example

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

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

---

# Erase Statement

## *Erase arrayname[,arrayname ]*

Reinitializes the elements of a fixed array.

Related Topics: Dim

### Example:

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

Option Base 1

Sub Main
    ' Declare array variables.
    Dim Num(10) As Integer ' Integer array.
    Dim StrVarArray(10) As String ' Variable-string array.
    Dim StrFixArray(10) As String * 10 ' Fixed-string array.
    Dim VarArray(10) As Variant ' Variant array.
    Dim DynamicArray() As Integer ' Dynamic array.
```

```

ReDim DynamicArray(10)    ' Allocate storage space.
Erase Num    ' Each element set to 0.
Erase StrVarArray    ' Each element set to zero-length
    ' string ("").
Erase StrFixArray    ' Each element set to 0.
Erase VarArray    ' Each element set to Empty.
Erase DynamicArray    ' Free memory used by array.

End Sub

```

---

## Exit Statement

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

Exits a loop or procedure

Related Topics End Statement, Stop Statement

### Example:

```

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

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

```

---

## Exp

Exp(*num*)

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

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

Related Topics: Log

### Example:

```

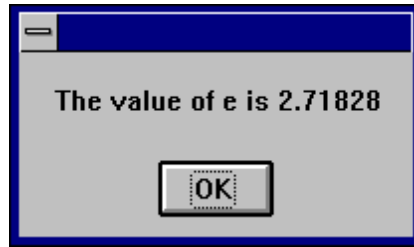
Sub ExpExample ()
    ' Exp(x) is  $e^x$  so Exp(1) is  $e^1$  or  $e$ .
    Dim Msg, ValueOfE    ' Declare variables.
    ValueOfE = Exp(1)    ' Calculate value of  $e$ .

```

```

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 = "SRCFIL" ' 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]
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* [*fnt*] )

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

Format returns returns a string

Part	Description
------	-------------

expression	Expression to be formatted.
------------	-----------------------------

<i>fnt</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 Enable Basic. Do not mix different type format expressions in a single <i>fnt</i> parameter.
------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

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

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

Predefined numeric format names:

<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.

### **Character Meaning**

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

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

	Display a digit or a zero
--	---------------------------

If the number being formatted has fewer digits than there are zeros (on either side of the decimal) in the format expression, leading or trailing zeros are displayed. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, the number is rounded to as many decimal places as there are zeros. If the number has more digits to left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, the extra digits are displayed without modification.

# Digit placeholder.

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

. Decimal placeholder.

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

---

### Character Meaning

% Percentage placeholder.

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

, Thousand separator.

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

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

E-E+e-e+ Scientific format.

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

: Time separator.

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

/ Date separator.

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

---

### Character Meaning

- + \$ ( )    Display a literal character.  
space

To display a character other than one of those listed, precede it with a backslash (\).

\            Display the next character in the format string.

The backslash itself isn't displayed. To display a backslash, use two backslashes (\\).

Examples of characters that can't be displayed as literal characters are the date- and time- formatting characters (a,c,d,h,m,n,p,q,s,t,w,y, and /:), the numeric -formatting characters(#,0,%,E,e,comma, and period), and the string-formatting characters (@,&,<,>, and !).

“String”    Display the string inside the double quotation marks.

To include a string in *fmt* from within Enable, you must use the ANSI code for a double quotation mark Chr(34) to enclose the text.

\*            Display the next character as the fill character.

Any empty space in a field is filled with the character following the asterisk.

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

---

### If you use The result is

One section    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 **Null** values.

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

“\$#,##0; (\$#,##0)”

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

“\$#,##0;;\Z\e\r\o”

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

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

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

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

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

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

<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.

---

**Character Meaning**

---

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).

---

**Character Meaning**

---

w	Display the day of the week as a number (1- 7 ).
ww	Display the week of the year as a number (1-53).
m	Display the month as a number without a leading zero (1-12). If m immediately follows h or hh, the minute rather than the month is displayed.
mm	Display the month as a number with a leading zero (01-12). If mm immediately follows h or hh, the minute rather than the month is displayed.
mmm	Display the month as an abbreviation (Jan-Dec).
mmmm	Display the month as a full month name (January-December).
q	display the quarter of the year as a number (1-4).
y	Display the day of the year as a number (1-366).

yy	Display the day of the year as a two-digit number (00-99)
yyyy	Display the day of the year as a four-digit number (100-9999).
h	Display the hour as a number without leading zeros (0-23).
hh	Display the hour as a number with leading zeros (00-23).
n	Display the minute as a number without leading zeros (0-59).
nn	Display the minute as a number with leading zeros (00-59).
s	Display the second as a number without leading zeros (0-59).
ss	Display the second as a number with leading zeros (00-59).
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

Character	Meaning
-----------	---------

---

A/P	Use the 12-hour clock display a uppercase A/P
a/p	Use the 12-hour clock display a lowercase a/p
AMPM	Use the 12-hour clock and display the contents of the 11:59 string (s1159) in the WIN.INI file with any hour before noon; display the contents of the 2359



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

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

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

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

If you use	The result is
One section only	The format applies to all string data.
Two sections <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.

### Character Meaning

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

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

### Example:

```
' Format Function Example

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

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

Sub Main

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

MsgBox Now
MsgBox MyTime

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

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

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

```

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

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

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

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

End Sub

```

---

## FreeFile Function

FreeFile

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

**Related Topics:** Open, Close, Write

### Example:

```

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

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

End Sub

```

---

## Function Statement

```

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

```

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

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

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

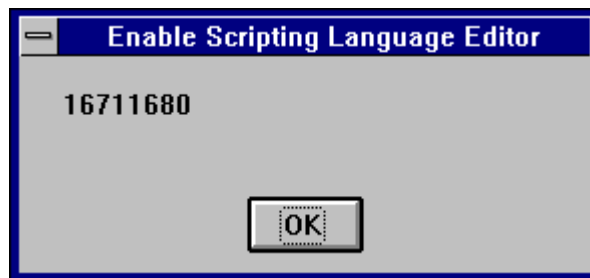
The optional ByVal parameter specifies that the variable is [passed by value instead of by reference (see “ByRef and ByVal” in this manual). The optional As type parameter is used to specify the data type. Valid types are String, Integer, Double, Long, and Variant (see “Variable Types” in this manual).

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 Statement

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

Reads from a disk file into a variable

The Get Statement has these parts:

*Filenumber* The number used to Open the file with.

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

Related Topics: Open, Put

---

## Get Object Function

`GetObject(filename[,class])`

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

Related Topics: CreateObject

---

## Global Statement

Global Const *constant*

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

Related Topics: Dim, Const and Type Statements

### Example:

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

```
End Sub
```

---

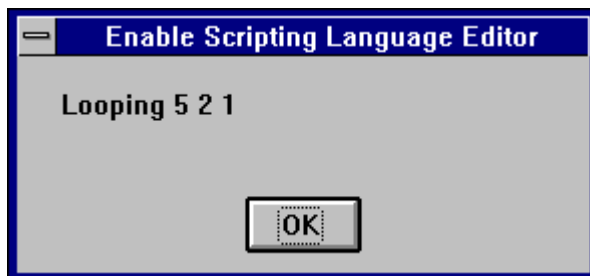
## GoTo Statement

**GoTo** *label*

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

### Example:

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



---

## Hex

**Hex** (*num*)

Returns the hexadecimal value of a decimal parameter.

Hex returns a string

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

Related Topics: Oct, Oct\$

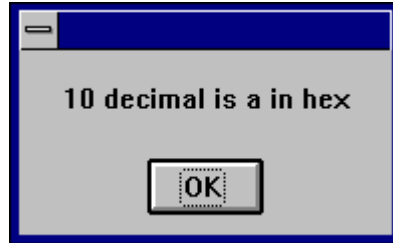
### Example:

```
Sub Main ()
```

```

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

```




---

## Hour Function

*Hour(string )*

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

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

### Example:

```

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

```

```

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

```

```

Sub Main

```

```

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

```

```

MsgBox Now
MsgBox MyTime

```

```

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

```

```

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

```

```

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

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

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

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

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

End Sub

```

---

## HTMLDialog

### HTMLDialog (*path, number*)

Runs a DHTML dialog that is specified in the path.

#### Example:

```

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

'See sample code on the samples disk htmdlglg.bas

```

---

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

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

Syntax 2

```

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

```



```
[statements(s)]  
.  
End If
```

## Syntax 2 If conditional Then statement

Allows conditional statements to be executed in the code.

Related Topics: Select Case

### Example:

```
Sub IfTest  
    ' demo If...Then...Else  
    Dim msg as String  
    Dim nl as String  
    Dim someInt as Integer  
  
    nl = Chr(10)  
    msg = "Less"  
    someInt = 4  
  
    If 5 > someInt Then msg = "Greater" : Beep  
    MsgBox "" & msg  
  
    If 3 > someInt Then  
        msg = "Greater"  
        Beep  
    Else  
        msg = "Less"  
    End If  
    MsgBox "" & msg  
  
    If someInt = 1 Then  
        msg = "Spring"  
    ElseIf someInt = 2 Then  
        msg = "Summer"  
    ElseIf someInt = 3 Then  
        msg = "Fall"  
    ElseIf someInt = 4 Then  
        msg = "Winter"  
    Else  
        msg = "Salt"  
    End If  
    MsgBox "" & msg  
  
End Sub
```

---

## Input # Statement

Input # *filenumber*, *variablelist*

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

The Input # Statement has two parameters *filenumber* and *variablelist*. *filenumber* is the number used in the open statement when the file was

opened and *variablelist* is a Comma-delimited list of the variables that are assigned when read from the file..

### Example:

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

---

## Input Function

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

Input returns characters from a sequential file.

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

### Example:

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

---

## InputBox Function

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

InputBox returns a String.

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

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

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

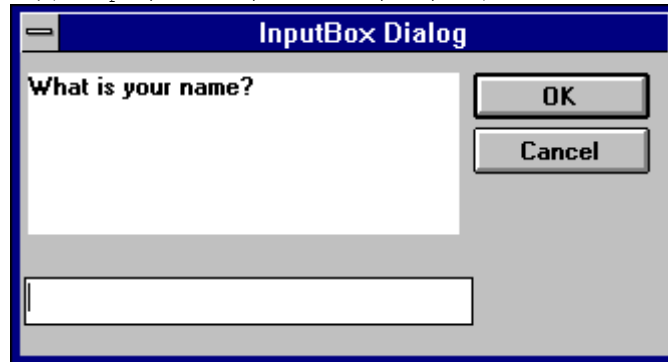
Xpos and Ypos and the x and y coordinates of the relative location of the input dialog box.

### Example:

```

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

```



```

End Sub

```

## 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.

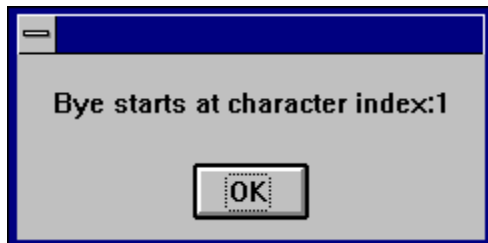
Related Topics: Mid Function

### 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)
MChekk = IsDate(x)
MArray1 = CStr(MArray)
MCheck1 = CStr(MCheck)
Print MArray1 & " is a date " & Chr(10) & MCheck
Print x & " is a date" & Chr(10) & MChekk
End Sub

```

---

## IsEmpty

**IsEmpty(*variant* )**

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

Related Topics: [IsDate](#), [IsNull](#), [IsNumeric](#), [VarType](#)

### Example:

' This sample explores the concept of an empty variant

```

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

```

---

## IsNull

**IsNull(*v*)**

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

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

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

Related Topics: [IsDate](#), [IsEmpty](#), [IsNumeric](#), [VarType](#)

---

## IsNumeric

**IsNumeric(*v*)**

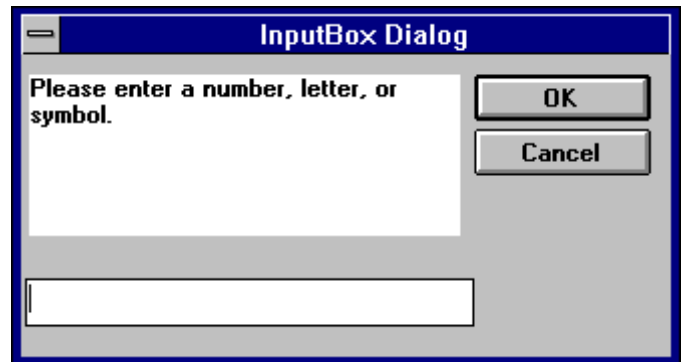
Returns a TRUE or FALSE indicating if the *v* parameter can be converted to a numeric data type.

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

Related topics: `IsDate`, `IsEmpty`, `IsNull`, `VarType`

### Example:

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



End Sub

---

## IsObject Function

`IsObject(objectname )`

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

Related Topics: `IsEmpty`, `IsNumeric`, `VarType`, `IsObject`

### Example:

```
Sub Main

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

    Set YourObject = MyObject
    MyCheck = IsObject(YourObject)
```

```

        Print MyCheck
    End Sub

```

---

## Kill Statement

**Kill** *filename*

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

Related Topics: Rmdir

### Example:

```

Const NumberOfFiles = 3

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

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

```

---

## LBound Function

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

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

Related Topics: UBound Function

### Example:

```

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

```

```

Option Base 1

Sub Main
    Dim a(10) As Double
    MsgBox "LBound: " & LBound(a) & " UBound: " & UBound(a)
    Dim i As Integer

```

```

For i = 0 to 3
    a(i) = 2 + i * 3.1
Next i
Print a(0),a(1),a(2), a(3)
End Sub

```

---

## LCase, Function

**Lcase[\$](*string*)**

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

Related Topics: Ucase Function

### Example:

```

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

```

```

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

```

---

## Left

**Left(*string*, *num*)**

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

Left returns a Variant, Left\$ returns a String

### Example:

```

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

```



```

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

```

---

## Len

### Len(*string*)

Returns the number of characters in a string.

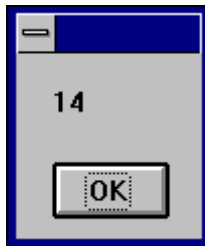
Related Topics: InStr

### Example:

```

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

```




---

## Let Statement

### [Let] *variablename* = *expression*

Let assigns a value to a variable.

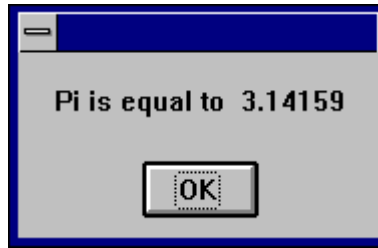
Let is an optional keyword that is rarely used. The Let statement is required in older versions of BASIC.

### Example:

```

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

```



---

## Line Input # Statement

Line Input # *filenumber* and *name*

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

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

Related Topics: Open

### Example:

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

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

```
End Sub
```

---

## LOF

LOF(*filenumber*)

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

Related Topics: FileLen

### Example:

```
Sub Main

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

End Sub

---

## Log

**Log(*num*)**

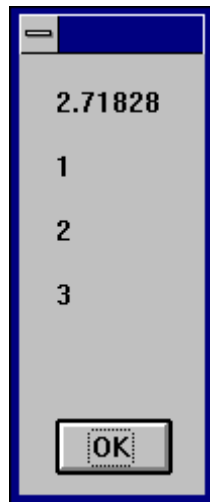
Returns the natural log of a number

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

Related Topics: Exp, Sin, Cos

### Example:

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



---

## Mid Function

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

Returns a substring within a string.

### Example:

```
Sub Main ( )
    Dim LWord, Msg, RWord, SpcPos, UsrInp ' Declare variables.
    Msg = "Enter two words separated by a space."
```

```

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

```

---

## Minute Function

**Minute**(*string*)

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

' Format Function Example

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

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

Sub Main

```

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

```

MsgBox Now

MsgBox MyTime

MsgBox Second( MyTime ) & " Seconds"

MsgBox Minute( MyTime ) & " Minutes"

```

MsgBox Hour( MyTime ) & " Hours"

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

End Sub

```

---

## MkDir

**MkDir** *path*

Creates a new directory.

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

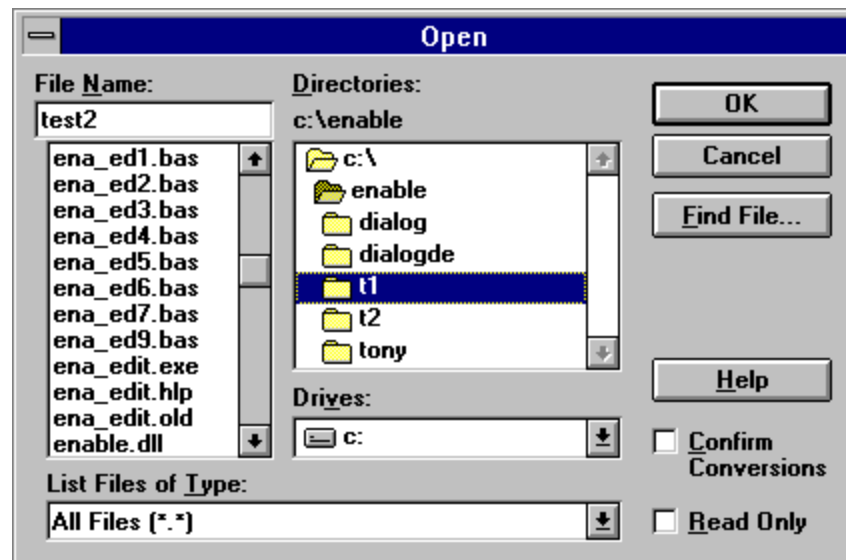
### Example:

```

Sub Main
    Dim DST As String

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

```




---

## Month Function

Month(*number*)

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

Related Topics: Day, Hour, Weekday, Year

### Example:

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

---

## MsgBox Function MsgBox Statement

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

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

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

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

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

768	Fourth button is default
0	Application modal.
4096	System modal

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

**title:**

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

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

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

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

### MsgBox Function, MsgBox Statement Example

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

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

### **Example:**

```

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

```

---

## Name Statement

Name *oldname* As *newname*

Changes the name of a directory or a file.

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

Related Topics: Kill, ChDir

---

## Now Function

Now

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

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

Related Topics:

### Example:

```

Sub Main ()
    Dim Today
    Today = Now
End Sub

```



---

## Oct Function

### Oct (*num*)

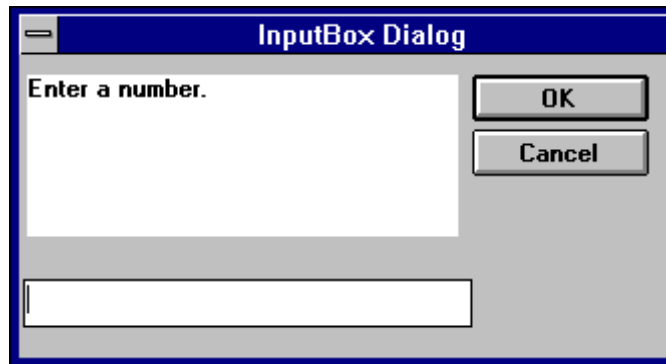
Returns the octal value of the decimal parameter

Oct returns a string

Related Topics: [Hex](#)

### Example:

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



---

## OKButton

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

For selecting options and closing dialog boxes

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

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




---

## On Error

`On Error { GoTo line / Resume Next / GoTo 0 }`

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

Related Topics: Resume

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

### Example:

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

Errors can be raised with the syntax:

`Err.Raise x`

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

- 5: "Invalid procedure call";
- 6: "Overflow";

7: "Out of memory";  
 9: "Subscript out of range";  
 10: "Array is fixed or temporarily locked";  
 11: "Division by zero";  
 13: "Type mismatch";  
 14: "Out of string space";  
 16: "Expression too complex";  
 17: "Can't perform requested operation";  
 18: "User interrupt occurred";  
 20: "Resume without error";  
 28: "Out of stack space";  
 35: "Sub, Function, or Property not defined";  
 47: "Too many DLL application clients";  
 48: "Error in loading DLL";  
 49: "Bad DLL calling convention";  
 51: "Internal error";  
 52: "Bad file name or number";  
 53: "File not found";  
 54: "Bad file mode";  
 55: "File already open";  
 57: "Device I/O error";  
 58: "File already exists";  
 59: "Bad record length";  
 60: "Disk full";  
 62: "Input past end of file";  
 63: "Bad record number";  
 67: "Too many files";  
 68: "Device unavailable";  
 70: "Permission denied";  
 71: "Disk not ready";  
 74: "Can't rename with different drive";  
 75: "Path/File access error";  
 76: "Path not found";  
 91: "Object variable or With block variable not set";  
 92: "For loop not initialized";  
 93: "Invalid pattern string";  
 94: "Invalid use of Null";  
 // OLE Automation Messages  
 429: "OLE Automation server cannot create object";  
 430: "Class doesn't support OLE Automation";  
 432: "File name or class name not found during OLE Automation operation";  
 438: "Object doesn't support this property or method";  
 440: "OLE Automation error";  
 443: "OLE Automation object does not have a default value";  
 445: "Object doesn't support this action";

```

446: "Object doesn't support named arguments";
447: "Object doesn't support current local setting";
448: "Named argument not found";
449: "Argument not optional";
450: "Wrong number of arguments";
451: "Object not a collection";
// Miscellaneous Messages
444: "Method not applicable in this context";
452: "Invalid ordinal";
453: "Specified DLL function not found";
457: "Duplicate Key";
460: "Invalid Clipboard format";
461: "Specified format doesn't match format of data";
480: "Can't create AutoRedraw image";
481: "Invalid picture";
482: "Printer error";
483: "Printer driver does not supported specified property";
484: "Problem getting printer information from from the system.";
    // Make sure the printer is setp up correctly.
485: "invalid picture type";
520: "Can't empty Clipboard";
521: "Can't open Clipboard";

```

---

## Open Statement

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

Opens a file for input and output operations.

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

Part	Description
<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, <i>filenumber</i> is associated with the file

as long as it is open. Other I/O statements can use the                      number to refer to the file.
-----------------------------------------------------------------------------------------------------------

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

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

Mode	Description
<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.

Access type	Description
<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               'Pints 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
```

---

## Randomize Statement

Randomize[*number*]

Used to Initialize the random number generator.

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

### Example:

```
Sub Main

    Dim MValue

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

End Sub
```

---

## ReDim Statement

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

Used to declare dynamic arrays and reallocate storage space.

The ReDim statement is used to size or resize a dynamic array that has already been declared using the Dim statement with empty parentheses. You can use the ReDim statement to repeatedly change the number of elements in 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 (*filenumber*)

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

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

Related Topics: Open

### Example:

```

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

```

---

## Seek Statement

Seek *filenumber*, *position*

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

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

Related Topics: Open

### Example:

```

Sub Main

```

```

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

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

```

---

## Select Case Statement

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

```

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

```

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

```

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

```

Related Topics: [If...Then...Else](#)

### Example:

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

```

Sub Test ()
  For x = 1 to 5
    print x
    Select Case x
      Case 2
        Print "Outer Case Two"
      Case 3
        Print "Outer Case Three"
    '
    Exit For
  
```

```

        Select Case x
        Case 2
            Print "Inner Case Two"
        Case 3
            Print "Inner Case Three"
        Case Else
            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"

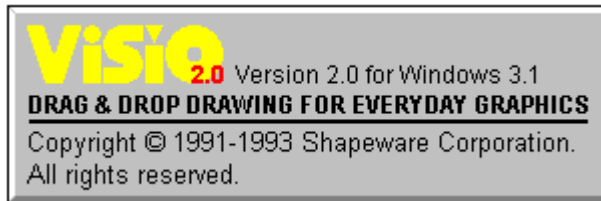
Dim cyan As Object
Set cyan = page.DrawOval (2.5, 8.5, 5.75, 5.25)
cyan.FillStyle = "Cyan fill"

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

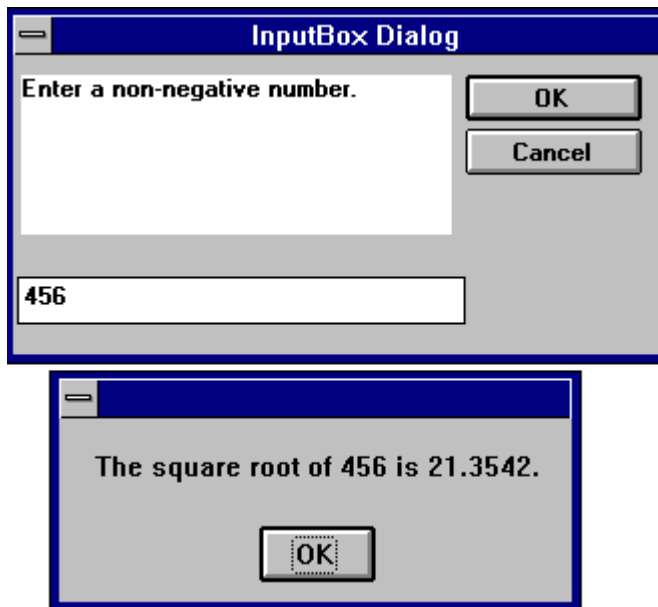
**Sqr**(*num*)

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:

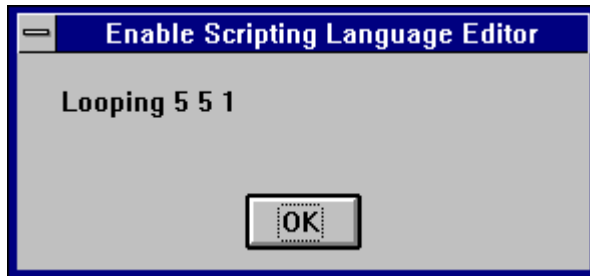
```

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 ()

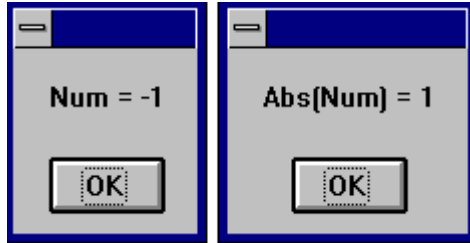
```

```

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.

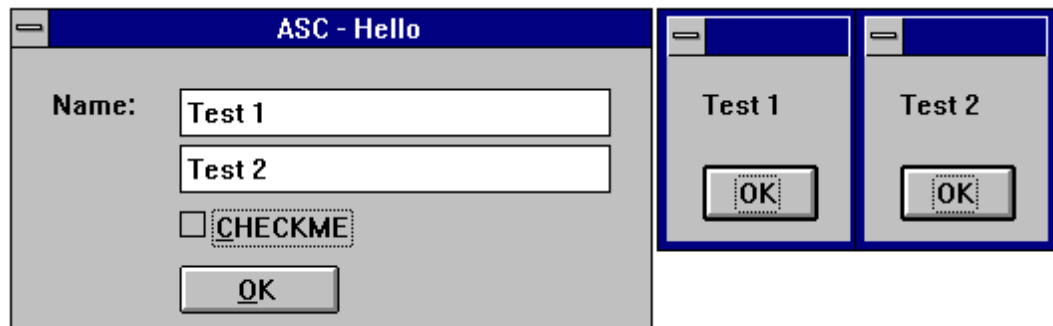
### Example:

```

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 TIME\$ statement.

### Example:

```

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

```

---

## Timer Event

### Timer

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

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

### Example:

```

Sub Main

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

    TS = Timer

    MsgBox "Starting Timer"

    TE = Timer

    TT = TE - TS
    Print TT

```



```
End Sub
```

---

## TimeSerial - Function

TimeSerial ( *hour, minute, second* )

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

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

### Example:

```
Sub Main

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

End Sub
```

---

## TimeValue - Function

TimeValue ( *TimeString* )

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

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

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

### Example:

```
Sub Main

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

End Sub
```

---

## Trim, LTrim, RTrim Functions

[L| R] Trim (*String* )

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

Ltrim, Rtrim and Trim all return a string

Ltrim removes leading spaces.

Rtrim removes trailing spaces.

Trim removes leading and trailing spaces.

### Example:

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

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

---

## Type Statement

Type usertype elementname As typename  
[ elementname As typename]

...

End Type

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

The **Type** statement has these parts:

Part	Description
<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.
<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 <i>typename</i> can't be an object type. End Type Marks the end of a user-defined type.

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

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

### Example:

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

```

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

Type type2
  a As String
  o As type1
End Type

Type type3
  b As Integer
  c As type2
End Type

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

Sub Form_Click ()
  a = 5
  type1a.a = 7472
  type1a.d = 23.1415
  type1a.s = "YES"
  type2a.a = "43 - forty three"
  type2a.o.s = "Yaba Daba Doo"

```

```

type3a.c.o.s = "COS"
type2b.a = "943 - nine hundred and forty three"
type2b.o.s = "Yogi"
MsgBox type1a.a
MsgBox type1a.d
MsgBox type1a.s
MsgBox type2a.a
MsgBox type2a.o.s
MsgBox type2b.a
MsgBox type2b.o.s
MsgBox type3a.c.o.s
MsgBox a
End Sub

```

---

## UBound Function

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

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

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

### Example:

```

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

Option Base 1

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

```

---

## UCase Function

Ucase (*String* )

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

Related Topics: Lcase, Lcase\$ Function

### Example:

```

' This example uses the LTrim and RTrim functions to strip leading and
' trailing spaces, respectively, from a string variable. It
' uses the Trim function alone to strip both types of spaces.

```

```
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

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

---

## Val

Val(*string*)

Returns the numeric value of a string of characters.

### Example:

```
Sub main
    Dim Msg
    Dim YourVal As Double
    YourVal = Val(InputBox$("Enter a number"))
    Msg = "The number you 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 Enable Basic statements. Wend ends the While...Wend flow of control structure.

Related Topics:    Do...Loop Statement

### Example:

```

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

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

```

---

## With Statement

```

With object
    [STATEMENTS]
End With

```

The With statement allows you to 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 typela
        .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.
        Print FileData ' Construct message.

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

```

---

## Year Function

**Year**(*serial#* )



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

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

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

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

### Example:

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



# Index

## A

Abs Function 153  
Accessing an object 143  
    CreateObject Function 143  
    GetObject Function 143  
Activate 143  
AppActivate Statement 154  
Application 143  
Arrays 130  
Asc Function 154  
Atn Function 155

## B

BASIC Language Reference 10  
Beep Statement 155  
BoundaryEquivalent function 39  
Break Point 14  
BusPicker 38

## C

Call Statement 156  
Calling Procedures in DLLs 128  
CBool Function 156  
CDate Function 157  
CDBl Function 157  
Change file *See* ReadChangeFile, *See*  
    ReadChangeFile, *See* ReadChangeFile, *See*  
    ReadChangeFile, *See* ReadChangeFile  
ChDir 149, 152, 158  
ChDrive 149  
ChDrive Statement 158  
**Check Boxes** 134  
CheckBox 159  
Choose Function 160  
Chr, Function 160  
Cint Function 161  
Class 145  
CLng Function 161  
Close Statement 161  
Comments 121

Const Statement 162  
Control Structures 121, 124  
Cos 163  
CreateObject 164  
CSng Function 165  
CStr Function 165  
CurDir Function 166  
CVar Function 166  
Cypress Enable Scripting Language Elements 121

## D

Data  
    access 23  
    modification 23  
**Data Types** 150  
Date Function 167  
DateSerial 168  
DateValue 168  
Day Function 169  
Debugging 13  
Declare Statement 169  
Dialog Box Editor 15  
Dialog Dialog Function 171  
Dialog Support 133  
Dim Statement 172  
Dir\$ Function 172  
DlgEnable Statement 173  
**DlgFocus Statement, DlgFocus() Function** 140  
**DlgListBoxArray, DlgListBoxArray()** 140  
**DlgSetPicture** 141  
DlgText Statement 174  
**DlgValue, DlgValue()** 141  
DlgVisible Statement 175  
Do...Loop Statement 175  
DoArchFlash function 44  
DoFault 33, 39, 42  
DoPF 33, 50  
DoVS 48  
DoVSEx 49

## E

End Statement 176  
Eof 177  
Equipment Type Code 22  
EquipmentType 55  
Erase 177  
**ErrorString** 85  
Example  
    Fault location 9  
    Per-Unit Calculator 11  
    Power flow outage studies 9  
    Power transfer studies 9  
Exit Statement 178  
Exp 149, 178

## F

FaultSelector 57  
FaultDescription 56  
File Input/Output 129  
FileCopy 149, 179  
FileLen Function 179  
FindBusByName 59  
Fix Function 179  
For...Next Statement 180  
Format Statement 181  
FreeFile Function 191  
FullBusName 64  
FullRelayName function 64  
Function Statement 191  
Functions list 35

## G

Get Object Function 192, 193  
GetBusEquipment 22, 66  
GetCurrent 33  
GetData 22, 67  
GetEquipment 22, 68  
GetFlow 69  
GetPFCCurrent 73  
GetPFVVoltage 74  
GetPSCVoltage 76  
GetRelay 22, 77  
GetRelayTime 33, 78  
GetSCCurrent 80  
GetSCVoltage 82  
GetVoltage 33  
GetVSVoltage 84  
Global Statement 193  
GoTo Statement 194

## H

Handles 21  
    Pre-defined 21  
Hex, 194, 196  
Hour Function 195  
HTMLDialog 196

## I

If...Then...Else Statement 125, 196  
Input # Statement 197  
Input, Function 198  
InputBox Function 198  
Installation 241  
InStr 199  
Int Function 200  
IsArray Function 200  
IsDate 200

IsEmpty 201  
IsNull 201  
IsNumeric 201  
IsObject Function 202

## K

Kill Statement 203

## L

Language Reference 10  
LBound Function 203  
LCase, Function 204  
Left 204  
Len 205  
Let Statement 205  
Line Input # Statement 206  
**List Boxes, Combo Boxes and Drop-down List Boxes** 133  
LoadDataFile function 87  
LOF 206  
Log 207

## M

Macros 9  
Making Applications Work Together 146  
Methods 143  
Mid Function 207  
Minute Function 208  
MkDir 209  
Month Function 209  
MsgBox 210

## N

Name Statement 212  
**NextBusByName** 89  
NextBusByNumber 91  
Now Function 212  
Numbers 122

## O

Object Handles *See* Handles  
Oct Function 213  
**OK and Cancel Buttons** 133  
OKButton 213  
OLE Automation 142, 145, 146, 147  
    What is OLE Automation? 142, 146, 147  
OLE Fundamentals 145  
OLE Object 145  
On Error 214  
Open Statement 216  
**Operators** 150

Option Base Statement 218  
**Option Buttons and Group Boxes** 136  
Option Explicit 219  
Other Data Types 123  
    Declaration of Variables 123  
    Scope of Variables 123

## P

Parameter Code 23  
PickFault 33, 92  
**PostData** 23, 93  
Power Flow Solution 33  
Print # Statement 220  
Print Method 219  
PrintTTY 95  
ProgressDialog function 94  
Properties 143  
**PutData** 23

## R

Randomize Statement 221  
ReadChangeFile 87, 88, 89, 97, 115  
ReDim Statement 222  
Rem Statement 222  
Reserved keywords 34  
Right, Function 223  
Rmdir Statement 224  
Rnd 224  
Running Script as Program Command 19

## S

Script File  
    debugging 13  
    editing 11  
    insert break point 14  
Second Function 224  
Seek Function 226  
SendKeys 228  
Set Statement 228  
SetData 116  
**Shell** 147, 229  
Short Circuit Solution 33  
ShowFault 33, 119  
Sin 230  
Space 230  
Sqr 230  
Statements and Functions Used in Dialog Functions  
    139  
Static 231  
Stop 232  
Str Function 232  
StrComp Function 233  
String, Function 233

Sub Statement 234  
Subroutines and Functions 126  
    Naming conventions 126  
Subroutines list 35

## T

Text 235  
**Text Boxes and Text** 135  
TextBox 235  
**The Dialog Function** 137  
**The Dialog Function Syntax** 138  
Time, Function 236  
Timer Event 236  
TimeSerial - Function 237  
TimeValue - Function 237  
Trim, LTrim Rtrim Functions 237  
Tutorial 11  
Type Code *See* Equipment Type Code  
Type Statement 238  
**Type/Functions/Statements** 149

## U

UBound Function 240  
UCase, Function 240  
User Defined Types 132, 239

## V

Val 241  
Variable and Constant Names 122  
Variable Types 122  
    Variants and Concatenation 122  
Variable Types  
    Variant 122  
VarType 241

## W

Weekday Function 242  
What is an OLE Object? 143  
While...Wend Statement 242  
With Statement 243  
Write # - Statement 244

## Y

Year 244