



## eXpress Dialog Form Designer



## Table of Contents

Dialog Forms .....	1
Other Data Types .....	1
Dialog Form Designer .....	1
General Features .....	1
The Dialog Form .....	1
The Dialog Form Designer Tool Bar .....	1
Adding Controls and Fields .....	1
Moving and Sizing Controls .....	2
Aligning and Sizing Controls .....	2
Automatic Properties Assignment .....	2
Automatic Alignment to Client Area .....	2
File menu .....	2
Edit menu .....	3
View menu .....	3
Help menu .....	3
The Designer buttons .....	3
Property Editor .....	6
Property and Methods Used with Dialog Forms .....	7
DialogResult Property .....	7
FreeDialogForm Subroutine .....	7
GetChecked Function .....	7
GetCtlEnabled Function .....	7
GetDialogDoubleVar Function .....	7
GetDialogIntVar Function .....	7
GetDialogLongVar Function .....	7
GetDialogStringVar Function .....	7
GetString Function .....	8
GetVisible Function .....	8
ListAddItem Subroutine .....	8
ListClear Subroutine .....	8
ListGetColText Function .....	8
ListGetCount Function .....	8
ListGetIndex Function .....	9
ListGetItem Function .....	9
ListSetColHeader Subroutine .....	9
ListSetColText Function .....	9
ListSetIndex Subroutine .....	9
ListSetItem Subroutine .....	9
LoadDialogForm Function .....	10
LoadImage Subroutine .....	10
SetChecked Subroutine .....	10
SetCtlEnabled Subroutine .....	10
SetDialogDoubleVar Subroutine .....	10
SetDialogIntVar Subroutine .....	10
SetDialogLongVar Subroutine .....	10

- SetDialogStringVar Subroutine ..... 10
- SetString Subroutine ..... 11
- SetVisible Subroutine ..... 11
- ShowDialogForm Subroutine ..... 11
- Examples Using Dialog Forms ..... 13
  - Passing and Retrieving Variables ..... 13
  - Doing a File Transfer ..... 14
  - Transaction Processing with Dialogs ..... 16
- Controls on Dialog Forms ..... 21
  - Form Properties ..... 21
    - Form Initial Action ..... 21
    - Form Activate Action ..... 21
    - Form Properties ..... 21
  - Text Labels ..... 21
  - Edit Boxes ..... 22
  - Command Buttons ..... 23
  - Speed Buttons ..... 23
  - Check Boxes ..... 24
  - Option Buttons ..... 25
  - List Boxes ..... 25
  - Drop-down List Boxes ..... 26
  - Multi-column List Boxes ..... 27
  - Memo ..... 28
  - Bevels ..... 29
  - Button Groups ..... 29
  - Group Boxes ..... 30
  - Panels ..... 31
  - Splitters ..... 32
  - Images ..... 32
  - Media Players ..... 33
  - Date/Time Labels ..... 33
  - Browser ..... 34
  - URL Link ..... 34
- Control Properties ..... 37
  - Align Property ..... 37
  - Alignment Property ..... 37
  - AllowAllUp Property ..... 37
  - AutoCenter Property ..... 37
  - AutoSize Property ..... 37
  - AutoSnap Property ..... 37
  - BackColor Property ..... 37
  - BevelInner Property ..... 37
  - BevelOuter Property ..... 37
  - BevelWidth Property ..... 37
  - Bitmap Property ..... 37
  - BitmapPosition Property ..... 37
  - BlinkColor Property ..... 38

Blinking Property .....	38
BlinkIntervalOff Property .....	38
BlinkIntervalOn Property .....	38
Border Property .....	38
BorderStyle Property .....	38
BorderWidth Property .....	38
ButtonStyle Property .....	38
Cancel Property .....	38
Caption Property .....	38
Center Property .....	38
CharCase Property .....	39
Checked Property .....	39
ColumnHeaders Property .....	39
Columns Property .....	39
Ctl3d Property .....	39
DataSource Property .....	39
Default Property .....	39
Down Property .....	39
EditMask Property .....	39
Enabled Property .....	39
Flat Property .....	39
FlatColor Property .....	39
Font Property .....	39
ForeColor Property .....	39
Format Property .....	39
FrameStyle Property .....	40
GroupIndex Property .....	40
Height Property .....	40
HelpContext Property .....	40
Hint Property .....	40
ItemIndex Property .....	40
Items Property .....	40
Left Property .....	40
Lines Property .....	40
MaximizedButton Property .....	40
MaxLength Property .....	40
MinimizedButton Property .....	40
MinSize Property .....	40
MonoChromeButtons Property .....	40
Name Property .....	41
NumBitMaps Property .....	41
NumericSort Property .....	41
ParentColor Property .....	41
ParentCtl3d Property .....	41
ParentFont Property .....	41
PassWordChar Property .....	41
Picture Property .....	41

- ReadOnly Property ..... 41
- ScrollBars Property ..... 41
- Shape Property ..... 41
- ShowAccelChar Property..... 41
- ShowHint Property..... 42
- SortColumn Property ..... 42
- SortDescending Property ..... 42
- Sorted Property..... 42
- Strech Property..... 42
- Style Property..... 42
- ParentShowHint Property..... 42
- TabOrder Property..... 42
- TabStop Property ..... 42
- Text Property..... 42
- Top Property..... 42
- Transparent Property ..... 42
- URL Property ..... 42
- Visible Property..... 42
- WantsReturns Property ..... 42
- Width Property..... 43
- WindowState Property..... 43
- WinHelpFile Property..... 43
- WordWrap Property ..... 43
- Property Dialogs ..... 45
  - Parent Controls ..... 45
    - Color Selection..... 45
      - Standard Color ..... 45
      - Custom..... 45
    - Edit Mask ..... 45
      - Input Edit Mask ..... 45
      - Pre-defined Standard Edit Mask..... 46
      - Character for Blanks ..... 46
      - Save Literal Characters ..... 46
      - Test Input..... 46
    - Edit Mask ..... 46
      - Input Edit Mask ..... 46
      - Pre-defined Standard Edit Mask..... 47
      - Character for Blanks ..... 47
      - Save Literal Characters ..... 48
      - Test Input..... 48
    - Date Time Format Editor ..... 48
      - Date Time Format..... 48
      - Predefined Formats..... 48
- Global Controls ..... 49
  - Dialog Form Menu Designer ..... 49
    - Menu Item Caption ..... 49
    - Name..... 49

Short Cut.....	49
Insert Item .....	49
Indent Level.....	49
Move.....	49
Checked .....	49
Visible .....	49
Enabled .....	49
Preview .....	49
Menu Items .....	49
Action Editor Used with Dialog Form Designer.....	51
Dialog Form Action Editor .....	51
File menu .....	51
Edit menu.....	51
Search menu .....	52
Bookmarks .....	52
Options menu .....	52
Tools.....	53
Help.....	53
Editor Properties .....	53
Edit Window Font.....	53
Tab Size .....	53
Highlight Colors.....	53
OK .....	53
Cancel.....	53
Help.....	53
Language Elements .....	55
Comments.....	55
Statements.....	55
Line Continuation Character.....	55
Numbers .....	55
Variable and Constant Names .....	55
Variables.....	57
Variable Types .....	57
Variant.....	57
Variants and Concatenation .....	57
Other Data Types .....	57
Other Data Types .....	58
Other Data Types .....	58
Flow of Control .....	61
Control Structures .....	61
The GoTo .....	61
The Do Loops.....	61
The While Loop .....	61
The For ... Next Loop .....	61
The If and Select Statements.....	62
Subroutine and Functions.....	63
Subroutines and Function Naming Conventions .....	63

ByRef and ByVal.....	63
Calling Procedures in DLLs.....	64
Files .....	65
File Input/Output.....	65
Arrays .....	67
Arrays.....	67
User Defined Types .....	69
User Defined Types.....	69
Dialogs and Dialog Controls.....	71
Dialog Support .....	71
OK and Cancel .....	71
List Boxes and Drop-down List Boxes.....	72
Check Boxes in Dialog.....	73
Text Boxes and Text .....	74
Option Buttons and Group Boxes.....	75
The Dialog Function .....	76
The Dialog Box Controls .....	76
The Dialog Function Syntax .....	76
OLE Automation.....	79
What is OLE Automation? .....	79
Accessing an Object.....	79
What is an OLE Object?.....	80
OLE Fundamentals.....	80
OLE Object .....	81
OLE Automation .....	81
Class.....	81
OLE Automation and Word example .....	81
Data Types, Operators and Precedence.....	83
Data Types, Operators and Precedences.....	83
Data Types .....	83
Arithmetic Operators.....	83
Operator Precedence.....	83
Relational Operators .....	83
Logical Operators.....	83
Functions, Statements, Subroutines and Events .....	85
Abs Function.....	85
AppActivate Statement.....	85
Asc Function .....	85
Atn Function .....	85
Beep Statement .....	86
Begin Dialog Statement.....	86
CalendarDialog Function (eXpress Plus).....	87
Call Statement .....	87
CancelButton Statement.....	87
CBool Function .....	88
CDate Function .....	88
CDBl Function .....	88



ChangeCursorStyle Subroutine.....	89
ChDir Statement .....	89
ChDrive Statement .....	90
CheckBox Statement .....	90
Choose Function.....	90
Chr Function .....	91
CInt Function .....	91
CLng Function .....	91
Close Statement.....	92
CloseApp Subroutine (eXpress Plus) .....	92
ComboBox Statement .....	92
Const Statement .....	92
Cos Function.....	93
CreateObject Function.....	93
CSng Function.....	94
CStr Function.....	95
CurDir Function .....	95
CVar Function .....	95
Date Function .....	95
DateSerial Function .....	96
DateValue Function.....	96
Day Function .....	96
Declare Statement.....	97
Dialog Function .....	97
Dim Statement.....	99
Dir Function.....	99
DlgEnable Statement .....	99
DlgText Statement .....	100
DlgVisible Statement.....	101
Do...Loop Statement.....	101
DropListBox Statement .....	101
End Statement .....	102
EOF Function .....	102
Erase Statement .....	103
Exit Statement.....	103
Exp Function.....	103
FileCopy Function .....	104
FileLen Function .....	104
FileOpenDialog Function (eXpress Plus) .....	104
FileSaveDialog Function (eXpress Plus) .....	104
Fix Function .....	105
For Each...Next Statement.....	105
For...Next Statement .....	105
Format Function .....	105
FreeFile Function .....	110
Function Statement .....	110
Get Statement .....	111

GetColor Function (eXpress Plus) .....	111
GetNumericProp Function (eXpress Plus) .....	111
GetObject Function .....	112
GetState Function (eXpress Plus) .....	112
GetString Function (eXpress Plus) .....	113
GetStringProp Function (eXpress Plus) .....	113
Global Statement .....	113
GroupBox Statement .....	114
GoTo Statement .....	114
Hex Function.....	114
Hour Function .....	115
If...Then...Else Statement .....	115
ImageOpenDialog Function (eXpress Plus).....	116
ImageSaveDialog Function (eXpress Plus) .....	116
Input Function .....	116
InputBox Function .....	117
InputBox Function .....	117
Int Function.....	118
IsArray Function.....	118
IsDate Function.....	118
IsEmpty Function .....	118
IsNull Function .....	118
IsNumeric Function.....	119
IsObject Function .....	119
Kill Statement .....	119
LBound Function.....	120
LCase Function.....	120
Left Function.....	121
Len Function.....	121
Let Statement .....	121
Line Input # Statement.....	122
ListBox Statement .....	122
ListClear Subroutine (eXpress Plus) .....	122
ListColHeader Subroutine (eXpress Plus) .....	123
ListCount Function (eXpress Plus).....	123
ListGetColText Function (eXpress Plus).....	123
ListGetIndex Function (eXpress Plus) .....	123
ListGetItem Function (eXpress Plus) .....	123
ListItemAdd Subroutine (eXpress Plus).....	124
ListItemRemove Subroutine (eXpress Plus).....	124
ListSetColText Subroutine (eXpress Plus) .....	124
ListSetIndex Subroutine (eXpress Plus) .....	124
ListSetItem Subroutine (eXpress Plus) .....	124
LoadImage Function (eXpress Plus) .....	124
LoadMMFile Function (eXpress Plus).....	125
LOF Function.....	125
Log Function.....	125

Mid Function .....	126
Minute Function.....	126
MkDir Statement .....	126
Month Function .....	127
MsgBox Function, MsgBox Statement .....	127
Name Statement .....	128
Now Function.....	129
Oct Function .....	129
OKButton Statement.....	129
On Error Statement .....	129
Open Statement.....	131
Option Base Statement .....	132
Option Explicit Statement .....	132
OptionButton Statement.....	133
OptionGroup Statement .....	133
Print # Statement .....	133
Print Statement.....	135
PrintBeginDoc Subroutine (eXpress Plus) .....	136
PrintDlg Function (eXpress Plus).....	136
PrintDraw Subroutine (eXpress Plus).....	136
PrintEndDoc Subroutine (eXpress Plus) .....	136
PrintMoveTo Subroutine (eXpress Plus) .....	137
PrintNewPage Subroutine (eXpress Plus) .....	137
PrintPageHeight Function (eXpress Plus) .....	137
PrintPageWidth Function (eXpress Plus) .....	137
PrintRect Subroutine (eXpress Plus).....	137
PrintSetFont Subroutine (eXpress Plus) .....	137
PrintSetFontSize Subroutine (eXpress Plus) .....	138
PrintSetFontStyleSubroutine (eXpress Plus) .....	138
PrintSetOrientation Subroutine (eXpress Plus).....	138
PrintTextHeight Function (eXpress Plus) .....	138
PrintTextWidth Function (eXpress Plus) .....	139
PushButton Statement .....	139
Put Statement.....	140
Randomize Statement.....	140
ReDim Statement .....	140
Rem Statement .....	141
Right Function.....	141
Rmdir Statement .....	142
Rnd Function .....	142
ScriptDir Function (eXpress Plus).....	142
Second Function.....	143
Seek Function .....	143
Seek Statement .....	143
Select Case Statement .....	144
SendKeys Statement .....	145
SendMail Subroutine (exPress Plus) .....	146

Set Statement .....	146
SetColor Subroutine (eXpress Plus) .....	147
SetFocus Subroutine (eXpress Plus) .....	148
SetNumericProp Subroutine (eXpress Plus) .....	148
SetState Subroutine (eXpress Plus) .....	149
SetString Subroutine (eXpress Plus) .....	149
SetStringProp Subroutine (eXpress Plus) .....	149
Sgn Function.....	149
Shell Function .....	150
Sin Function .....	150
Space Function.....	150
Sqr Function .....	150
Static Statement .....	151
Stop Statement.....	151
Str Function.....	152
StrComp Function.....	152
String Function .....	152
Sub Statement.....	153
Tan Function.....	153
Text Statement .....	153
TextBox Statement .....	154
Time Function, Time Statement.....	154
Timer Event.....	154
TimeSerial Function .....	155
TimeValue Function .....	155
Trim, LTrim, RTrim Functions.....	155
Type Statement .....	156
UBound Function .....	157
UCase Function .....	157
Val Function.....	158
VarType Function .....	158
Wait Subroutine (eXpress Plus) .....	158
Weekday Function .....	158
While...Wend Statement.....	159
With Statement.....	159
Write # - Statement .....	160
Year Function.....	160
Predefined Constants.....	161
Predefined Constants .....	161
Color Types (also see GetColor and SetColor): .....	161
Defined Colors (also see SetColor): .....	161
Message Box Constants (also see MsgBox): .....	162
Print Font Styles (also see PrintSetFont): .....	162
State Types (also see GetState and): .....	162

## Dialog Forms

### Other Data Types

The six data types available are shown below with their declaration character suffixes:

<u>Data type</u>	<u>Suffix</u>	<u>Type Declaration</u>	<u>Size</u>	<u>Range</u>
String	\$	Dim StrVar As String	String of characters	0 to 65,500 characters
Integer	%	Dim IntVar As Integer	2-byte integer	-32,768 to 32,767
Long	&	Dim LongVar As Long	4-byte integer	-2,147,483,648 to 2,147,483,647
Single	!	Dim SingVar As Single	4-byte floating-point number	-3.402823E38 to -1.401298E-45 (negative values) 1.401298E-45 to 3.402823E38 (positive values)
Double	#	Dim DbIVar As Double	8-byte floating-point number	-1.79769313486232D308 to -4.94065645841247D-324 (negative values) 4.94065645841247D-324 to 1.79769313486232D308 (positive values)
Variant		Dim X As Variant	Date/time, floating-point number or string	Date values: January 1, 0000 through December 31, 9999; numeric values: same range as Double; string values: same range as String
Currency				(Currency datatype is not supported)

### Dialog Form Designer

Use the Dialog Form Designer toolbar and the companion dialogs (Dialog Form, Property Editor and Dialog Form Action Script Editor) to alter the appearance of the basic terminal screen (as initially captured or converted), add new controls and automate end-user interface functions.

### General Features

The Dialog Form Designer is very similar in appearance to the development environment of several object-oriented visual languages. The Dialog Form is the canvas upon which you place Windows controls (buttons, list boxes, etc.). The Dialog Form Designer toolbar is the pallet from which you select the controls that you want to place on the form. Each control has its own unique set of Properties, dependent upon the type of control selected, that allow you to specify how the control will appear and behave on the form.

### The Dialog Form

The initial design form contains a menu bar at the top that has three disabled (grayed out) menu names. These menus are enabled at runtime and used to access standard options (e.g., Print Setup) at runtime. You may add other menus specific to the form and they will be displayed here, but they will not cause any actions to be invoked during the design phase.

The dialog form is a Windows control, and like any Windows control, has a set of properties associated with that control. For properties and actions, see Form Properties.

### The Dialog Form Designer Tool Bar

The tool bar (initially at the top of the monitor viewing area) can be moved anywhere on the window that is convenient. To move, first select Float Tool Bar from the tool bar's View menu. Next, place the mouse cursor over the caption bar (over the words, "Dialog Form Designer"). While holding down the left mouse button, drag the toolbar to the desired location.

### Adding Controls and Fields

To add controls to the Dialog Form, click the appropriate control button on the tool bar. Note: Adding menu controls are done by selecting Menu Designer from the Tools menu or clicking the corresponding button on the tool bar.

With a control button selected on the toolbar, move the mouse cursor over an empty area of the form window and click the left mouse button. The new control will be placed on the form where you clicked. Alternately, you may hold down the left mouse button and drag the mouse to create the initial size and location of the control. A box will grow and shrink as you drag in any direction. When the left mouse button is released, the control will appear, already selected and ready to move or edit.

To edit, click the left mouse button over the control and use the Property Editor window to change the desired property.

### **Moving and Sizing Controls**

Dialog controls may be moved and resized with the mouse. Select the desired control by moving the mouse cursor over the control and clicking the left mouse button. When selected, the control will have sizing handles around it.

To size, move the mouse cursor over one of the sizing handles (the mouse cursor will change to sizing arrows), then press and hold the left mouse button and drag the sizing handle in the desired direction. When the mouse button is released, the control will snap to the new size.

To change the location on the form, move the mouse cursor into the center area of the control and press and hold the left mouse button. With the mouse, drag and drop the control to the desired location.

### **Aligning and Sizing Controls**

The alignment and sizing feature allows any group of controls to be automatically aligned with or sized to a "base" control (any single Dialog control). The process is made simple by first selecting the base control with the left mouse button. Next, while holding down the Shift key, other controls to be aligned or sized are selected with the left mouse button (notice that the sizing handles become gray). Finally, right click on any of the selected controls and choose an alignment or sizing option from the popup menu.

Alternately, you may select a group of controls by moving the mouse cursor to a point on the dialog form away from any control (the gray part). While holding the left mouse button, drag the mouse to encompass the controls you want to align or size. Note: The last control selected becomes the base control.

Alignment and sizing are based on the original alignment and size of the base control. Alignment allows vertical alignment to be left justified, centered or right justified to the base control; horizontal alignment to be aligned across the top, middle or bottom of the base control. Sizing changes the width or height of the selected controls to match that of the base control.

### **Automatic Properties Assignment**

Much like the ability to automatically size and align groups of controls (see above), controls may have their properties (font, color, etc.) set as a group. The procedure is practically the same. While holding down the left mouse button, drag the mouse over the controls whose properties are to change. Next, change the property or properties desired on the Property Editor window.

Most controls have "Parent" properties (ParentCtl3D, ParentFont and/or ParentShowHint). If a parent property is set to True, then the control takes on the property of the parent control. Currently, there are only two types of parent controls: the form itself and the panel control. For example, if a Button on the form has the ParentFont property set to True, the Button caption will use the same font assigned to the form. Note: Changing the Font property on the Button will automatically set ParentFont property to False.

A Panel can have controls placed on it so that whenever you move the panel during the design phase, all the controls on the panel move with it. Likewise, if an action disables the Panel at runtime, all the controls on the Panel are also disabled since the Panel is Parent to all the controls placed on the panel.

### **Automatic Alignment to Client Area**

Some controls (panels, list boxes, text labels, etc.) can be aligned to all or part of the client area of a parent control. The Align property allows the automatic alignment of the control to the top ("alTop"), bottom (alBottom"), left (alLeft") or right ("alRight") edge of the parent control. If the Align property is set to "alNone" (the default), the control remains wherever it is placed on the parent control. A sixth setting, "alClient", can be used to align the control within the client area of the control. If multiple controls are aligned on the parent control, they are aligned in the precedence they were placed on the control.

An example of using aligned controls would be when you wanted to place a splitter between two list boxes thus giving the user the ability to determine how much of the list boxes would be visible at a given time.

### **File menu**

#### **New Form**

Use this selection to close the current dialog form and start a new form.

#### **Open**

Use this selection to open an existing form.

#### **Save**

Use this selection to save alterations to the current form.

#### **Save As**

Use this selection to save the current form to a different binary form file.

**Edit menu****Cut (Ctrl+X)**

Use this selection to cut the selected control(s) to the Windows clipboard.

**Copy (Ctrl+C)**

Use this selection to copy the selected control(s) to the Windows clipboard.

**Paste (Ctrl+V)**

Use this selection to paste the contents of the Windows clipboard to the form.

**Delete (Del)**

Use this selection to delete the selected control(s).

**Delete All Controls**

Use this selection to clear the form.

**Cut to File**

Use this selection to cut the selected control(s) to a file (.clp).

**Copy to File**

Use this selection to copy the selected control(s) to a file (.clp).

**Paste from File**

Use this selection to paste the contents a file to the form.

**Select All**

Use this command to select all controls on the form.

**Form Edits...**

Use this selection to display the alignment, sizing, tab order and grid options popup menu. This selection is the same as doing a right mouse click over a selected control.

**View menu****Tool Bar (F6)**

Use this selection to return emphasis to the Dialog Form Designer toolbar.

**Properties Window (F7)**

Use this selection to open the Property Editor window.

**Design Window (F8)**

Use this selection to return emphasis to the Dialog Form window from the Dialog Form Designer toolbar.

**Action Edit Window (F9)**

Use this selection to open the Dialog Form Action Script Editor.

**Float Tool Bar**

Use this selection to unlock the toolbar. This selection allows you to drag the tool bar by holding down the left mouse button over the toolbar caption and dragging it to another location.

**Help menu****Contents**

This selection starts the Windows Help program and displays all help topics for the Dialog Form Designer.

**About**

Use this selection to display version and copyright notification.

**The Designer buttons**

The buttons along the bottom of the Dialog Form Designer toolbar are used to select the type of control you want to place on the form.

**Pointer button**

Use this selection to cancel adding a control, thus returning the mouse to a pointer tool. Use the pointer tool to select control(s) on the form.

**Text Label**

Use this selection to add a text label to a parent control (form or panel). This control is useful to label controls that have no caption property (e.g., Edit Boxes).

For properties, see Text Labels.

**Edit Box**

Use this selection to add an edit box to a parent control.

For properties, see Edit Boxes.

**Command Button**

Use this selection to add a command button to a parent control (e.g., OK button, Cancel button, Query button, etc.).

For properties, see Command Buttons.

**Speed Button**

Use this selection to add a speed button to a parent control. A speed button is a non-windowed control meaning it takes less system resources. Since it is a non-windowed control, you may not tab to it.

You can make a group of speed buttons act like radio buttons by setting the group index to something other than 0.

Speed buttons can also have that flat property that makes the outline show on mouse fly-over.

For properties, see Speed Buttons.

**Check Box**

Use this selection to add a check box to a parent control. Use check boxes when multiple options may be selected by the user.

For properties, see Check Boxes.

**Option Button**

Use this selection to add option or radio buttons to a parent control. Use option buttons when the options are mutually exclusive for selection by the user.

Option buttons placed on a form are mutually exclusive for the entire form; i.e., only one may be selected by the user on a given form. Use the Button Group control when option buttons need to be mutually exclusive within a group (see below).

For properties, see Option Buttons.

**List Box**

Use this selection to add a single column list box to a parent control that can be used for display or data value selection by the user. If more items are displayed than can be contained by the size of the list box on the, scroll bars will appear. If a list is to contain many values, it might be desirable to use a drop-down list box (see below) to conserve space on the form.

For properties, see List Boxes.

**Drop-Down List Box**

Use this selection to add a drop-down list box to a parent control that can be used for display or data value selection by the user. The drop-down list box occupies only the amount of space on the form that it takes to display a single item in the list. Other items are made visible by clicking the drop-down arrow on the box.

For properties, see Drop-down List Boxes.

**Multi-Column List Box**

Use this selection to add a multi-column list box to a parent control. Unlike the single column list box, the multi-column list box control supports header options that are displayed as part of the form.

For properties, see Multi-column List Boxes.



**Bevel**

Use this selection to add a bevel to a parent control. The bevel has very few properties. It is primarily used to for appearance.

You can place controls within a bevel but those controls, unlike those on a panel or button group, may not be moved as a group.

For properties, see Bevels.

**Button Group**

Use this selection to add an option button group to a parent control. When the button group is first placed on the form, no option buttons appear in the group. Use the Items property to add the option buttons. Next, use the button group sizing handles to size the group to contain the option items entered. The Item Index property can be used to specify which option in the group is initially set when the form is loaded (e.g., -1 means that no option is set; 0 means the first option is set; 1, the second, etc.).

For properties, see Button Groups.

**Group Box**

Use this selection to add a group box to a parent control. Group boxes are generally used to logically group controls on the form (e.g., a group of controls that presents the user the Account Number, Balance, Amount Due, etc. might be placed in a group box captioned, Account Information).

You can place controls within a group box but those controls, unlike those on a panel or button group, may not be moved as a group.

For properties, see Group Boxes.

**Panel**

Use this control to add a panel to a parent control. Controls can be placed on a panel and moved as a group when you select the panel and drag it to another location on the form. To place a control on a panel, select the panel on the form with the mouse. Next, click the Panel button on the DialogForm Designer toolbar and then click the mouse again over the panel.

To cut a control from another location (on the form or other panel) and paste it on a panel, select the control and enter Ctrl+X on the keyboard. Next, with the mouse, select the panel that is to receive the control and press Ctrl+V. Note: The control will be placed on the panel in the same relative location on which it was originally located; therefore, it may be necessary to increase the size of the panel in order to see the control.

For properties, see Panels.

**Splitter**

Use this control to add a splitter to a parent control. A splitter, like those seen in popular Web browsers, allows the user to adjust the panels on a form to their own liking.

Splitters must be aligned and placed between other aligned controls. For example, if a splitter is to be placed vertically between two panels the left panel could be aligned left, followed by aligning the splitter left also. The right-hand panel then could be aligned to fill the remaining portion of the client area.

For properties, see Splitters.

**Image**

Use this control to add an image to a parent control. Only pictures with .bmp, .wmf, .emf or .ico extensions are supported.

For properties, see Images.

**Media Player**

Use this selection to add a media player control to a parent control. To use the Media Player control, use the LoadMMFile function in an action.

For properties, see Media Players.

**Date/Time Label**

Use this selection to add a date/time stamp to a parent control. It is useful in a status bar (a bottom aligned panel).

For properties, see Date/Time Labels.

**Menu Designer**

Use this selection to open the Dialog Form Menu Designer window to add or edit menus on the form.

**Property Editor**

The Property Editor window is where the properties and actions for individual controls placed on the dialog form are set. Correspondingly, the window contains two tabs: Properties and Actions. The Properties tab is where you name the control, as it will be referenced in dialog form actions, initially set how the control will appear to the user at runtime, attach runtime help for the control, etc. The Action tab is used to assign an action to the control.

The dialog form can also be considered a control and, as such, may have properties assigned or changed by first selecting the form (left click on any empty space on the form).

Note: The dialog form is a "parent" control. Any form property (Font, Ctl3d, etc.) will be assigned to a control placed on the form if that control has its corresponding "parent" control set to True (see Parent Controls).

To change a property, first select the control with the mouse on the dialog form. With a control selected, the properties for the control will appear on the Properties tab. The left-hand column contains the property name; the right-hand column, the property value. Next, select a property with the mouse. A control (button, text box, etc.) will appear in the cell to the right of the property name that will allow you to change the property value.

## Property and Methods Used with Dialog Forms

### DialogResult Property

This is an integer value set by the Dialog Form's user-written Action Script. The Dialog Form's Action Script must contain the following declaration:

```
Global DialogFormResult as Integer
```

See Examples: Passing and Retrieving Variables, Doing a File Transfer and Transaction Processing with Dialogs.

### FreeDialogForm Subroutine

Free the current dialog form object. This procedure is automatically called when a dialog form is destroyed. It must be used when a new dialog form is created using the same object variable.

Format:

```
FreeDialogForm
```

### GetChecked Function

Get the Checked state of a Checkbox or Radio Button on the Dialog Form.

Format:

```
GetChecked (CtlName)
```

The *CtlName* parameter is any string expression containing the name of a checkbox or option button.

This function returns an Integer; returns True (1) if checked, else returns False (0).

See Doing a File Transfer.

### GetCtlEnabled Function

Get the Enabled state of a control on the Dialog Form.

Format:

```
GetCtlEnabled (CtlName)
```

The *CtlName* parameter is any string expression containing the name of a control.

This function returns an Integer; returns True (1) if enabled, else returns False (0).

### GetDialogDoubleVar Function

Get the value of a Global Dialog Form Double variable.

Format:

```
GetDialogDoubleVar(Name)
```

The *Name* parameter is any string expression containing the name of the variable.

See Passing and Retrieving Variables.

### GetDialogIntVar Function

Get the value of a Global Dialog Form Integer variable.

Format:

```
GetDialogIntVar (Name)
```

The *Name* parameter is any string expression containing the name of the variable.

Integers are the same as Longs in the 32-bit environment. The only difference is that the type definition in the form's action script must match.

See Passing and Retrieving Variables.

### GetDialogLongVar Function

Get the value of a Global Dialog Form Long variable.

Format:

```
GetDialogLongVar (Name)
```

The *Name* parameter is any string expression containing the name of the variable.

See Passing and Retrieving Variables.

### GetDialogStringVar Function

Get the value of a Global Dialog Form String variable.

Format:

```
GetDialogStringVar (Name)
```

See Passing and Retrieving Variables.

### GetString Function

Get string value of controls on the Dialog Form as follows:

<u>Control Type</u>	<u>Value Returned</u>
Menu Item	Caption
Text Label	Caption
Edit Box	Text
Command Button	Caption
Speed Button	Caption
Checkbox	Caption
Option Button	Caption
Button Group	Caption
Group Box	Caption
Panel	Caption

Format:

GetString (*CtlName*)

This function returns a String.

The *CtlName* parameter is any string expression containing the name of a valid control.

See Doing a File Transfer.

### GetVisible Function

Get the visible state of a control on the Dialog Form.

Format:

GetVisible (*CtlName*)

The *CtlName* parameter is any string expression containing the name of a control.

This function returns an Integer; returns True (1) if visible, else returns False (0).

### ListAddItem Subroutine

Add an item to a standard, drop-down or multi-column list box.

Format:

ListAddItem *CtlName*, *Value* [, *Value2* ... , *Valuen*]

The *CtlName* parameter is any string expression containing the name of a list box. A *Value* parameter is any string expression.

To add a line to a multi-column list, use multiple *Value* parameters separated by commas. Each item is put into the corresponding column from left to right.

See Transaction Processing with Dialogs.

### ListClear Subroutine

Clear or removes all items from any type of list box.

Format:

ListClear *CtlName*

The *CtlName* parameter is any string expression containing the name of a list box.

### ListGetColText Function

Retrieve the text in a specified column from the currently selected item of a multi-column list box. Columns are numbered from left to right starting with zero (0).

Format:

ListGetColText (*CtlName*, *Column*)

This function returns a String.

The *CtlName* parameter is any string expression containing the name of a multi-column list box. The *Column* parameter is any integer expression representing a column relative to zero (0).

### ListGetCount Function

Return the number of items currently in the list.

Format:

`ListGetCount (CtlName)`

The *CtlName* parameter is any string expression containing the name of a list box.

This function returns an Integer.

### ListGetIndex Function

Retrieve the index of the currently selected item of a standard or drop-down list box.

Format:

`ListGetIndex (CtlName)`

This function returns an Integer.

The *CtlName* parameter is any string expression containing the name of a list box.

The index can be in the range -1 to the value of ListGetCount - 1. A returned value of -1 indicates the list is either empty or nothing is currently selected.

See Examples: Passing and Retrieving Variables, Doing a File Transfer and Transaction Processing with Dialogs.

### ListGetItem Function

Get a row of data from the specified list box. The currently selected item (row) is retrieved.

Format:

`ListGetItem (CtlName)`

This function returns a String.

The *CtlName* parameter is any string expression containing the name of a list box or drop-down list box.

See Passing and Retrieving Variables and Transaction Processing with Dialogs.

### ListSetColHeader Subroutine

Set the column header value for a specified column in a multi-column list box. Columns are numbered from left to right starting with zero (0).

Format:

`ListSetColHeader CtlName, Column, Value`

The *CtlName* parameter is any string expression containing the name of a multi-column list box. The *Column* parameter is any integer expression representing a column relative to 0. The *Value* parameter is any string expression.

### ListSetColText Function

Set the text of a specified column in the currently selected item of a multi-column list box. Columns are numbered from left to right starting with zero (0).

Format:

`ListSetColText (CtlName, Column)`

The *CtlName* parameter is any string expression containing the name of a multi-column list box. The *Column* parameter is any integer expression representing the column relative to zero (0). The *Value* parameter is any string expression.

### ListSetIndex Subroutine

Set the selected list item index of a standard or drop-down list box.

Format:

`ListSetIndex CtlName, Index`

The *CtlName* parameter is any string expression containing the name of a list box. The *Index* parameter is any integer expression representing the row relative to zero (0).

See Passing and Retrieving Variables and Transaction Processing with Dialogs.

### ListSetItem Subroutine

Set a row of data into the specified list box. Note: This function may not be used with multi-column list boxes (see ListSetColText Subroutine, above).

Format:

`ListSetItem CtlName, Value`

The *CtlName* parameter is any string expression containing the name of a list box or drop-down list box. The *Index* parameter is any integer expression representing the row relative to zero (0). The *Value* parameter is any string expression containing the text of the row.

**LoadDialogForm Function**

Load a Dialog Form from a binary form file (BFM).

Format:

LoadDialogForm (*FormFile*)

The *FormFile* parameter is any string expression containing the complete drive, path and file specification of the binary form file (.bfm).

**LoadImage Subroutine**

Load an image file into the specified image control. The image file may be a Windows or OS/2 bitmap (.bmp), icon (.ico), Windows metafile (.wmf) Windows enhanced metafile (.emf) or JPEG compliant files (.jpg and .jpeg).

Format:

LoadImage CtlName, ImgFileName

The *CtlName* parameter is any string expression containing the name of an image control. The *ImageFileName* parameter is any string expression containing the complete drive, path and file specification of the image file.

**SetChecked Subroutine**

Set the Checked state of a Checkbox or Radio Button on the Dialog Form.

Format:

SetChecked CtlName, Value

The *CtlName* parameter is any string expression containing the name of a Check Box or Radio Button. The *Value* parameter can be any integer expression containing True (1) to check/set, False (0), to uncheck/unset.

**SetCtlEnabled Subroutine**

Set the Enabled state of a control on the Dialog Form.

Format:

SetCtlEnabled CtlName, Value

The *CtlName* parameter is any string expression containing the name of a control. The *Value* parameter can be any integer expression containing True (1) to enable, False (0), to disable.

**SetDialogDoubleVar Subroutine**

Set the value of a Global Dialog Form Double variable.

Format:

SetDialogDoubleVar Name, Value

The *Name* parameter is any string expression containing the name of a control. The *Value* parameter is any expression containing a global dialog double variable.

See Passing and Retrieving Variables.

**SetDialogIntVar Subroutine**

Set the value of a Global Dialog Form Integer variable.

Format:

SetDialogIntVar Name, Value

The *Name* parameter is any string expression containing the name of a control. The *Value* parameter is any expression containing a global dialog integer variable.

Integers are the same as Longs in the 32-bit environment. The only difference is that the type definition in the form's action script must match.

See Passing and Retrieving Variables.

**SetDialogLongVar Subroutine**

Set the value of a Global Dialog Form Long variable.

Format:

SetDialogLongVar Name, Value

The *Name* parameter is any string expression containing the name of a control. The *Value* parameter is any expression containing a global dialog long variable.

See Passing and Retrieving Variables.

**SetDialogStringVar Subroutine**

Get the value of a Global Dialog Form String variable.

Format:

`SetDialogStringVar Name, Value`

The *Name* parameter is any string expression containing the name of a control. The *Value* parameter is any expression containing a global dialog string variable.

See Passing and Retrieving Variables.

#### **SetString Subroutine**

Get string value of controls on the Dialog Form as follows:

Format:

`GetString CtlName, Value`

#### **SetVisible Subroutine**

Set the visible state of a control on the Dialog Form.

Format:

`SetVisible CtlName, Value`

The *CtlName* parameter is any string expression containing the name of a control. The *Value* parameter can be any integer expression containing True (1) to make visible, False (0), to hide.

#### **ShowDialogForm Subroutine**

Show the current Dialog Form in the Modal state. All values required in the dialog form must be set before this method is called. Execution of the calling program or script is suspended until the Dialog Form is closed.

Once the Dialog Form is closed, its values may be retrieved.

Format:

`ShowDialogForm`

See Examples: Passing and Retrieving Variables, Doing a File Transfer and Transaction Processing with Dialogs.





## Examples Using Dialog Forms

### Passing and Retrieving Variables

This example passes variable values to the dialog form's action script and then retrieves them when the dialog form closes. The variables must be defined in the action script of the dialog form.

Below, the global variables UserId, Password, etc. are defined in the dialog's action script. They are preset with values before the form is shown. The new values of the variables are retrieved when the form is closed.

```
Sub Main()
    set df = CreateObject("TEQDlgFormx.TEQDlgForm")

    df.LoadDialogForm("C:\AA_DialogForm\Demo\GeneralSignOn.bfm")
    df.SetDialogStringVar "UserId", "J Doe"
    df.SetDialogStringVar "Password", "Green Stuff"
    df.SetDialogIntVar "Appx", 1
    df.SetDialogLongVar "LongVal", 9999
    df.SetDialogDoubleVar "Dbl", 1.99
    df.ShowDialogForm

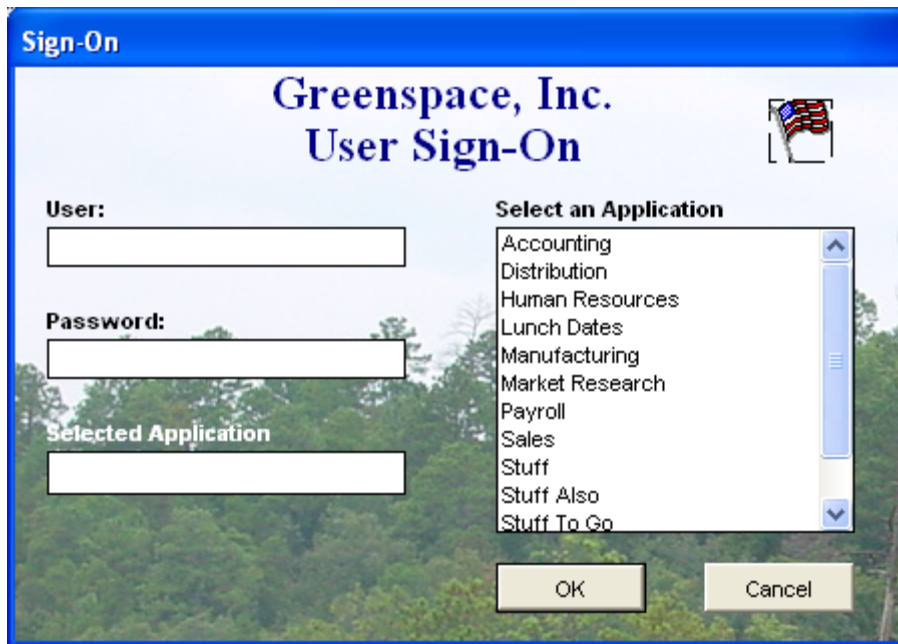
    If df.DialogResult = 1 Then
        MsgBox "Dialog OK'ed" & chr$(13) & "UserId=" &
df.GetDialogStringVar("UserId") & chr$(13) & "Password = " &
df.GetDialogStringVar("Password") & chr$(13) & "Application = " &
df.GetDialogStringVar("Application")
        MsgBox "Appx = " & df.GetDialogIntVar("Appx")
        MsgBox "LongVal = " & Df.GetDialogLongVar("LongVal") & " Dbl = " &
df.GetDialogDoubleVar("Dbl")
    Else
        MsgBox "Dialog Cancelled"
    End If
    df = nothing
End Sub
```

Note: The above script was written using the eXpress Plus Script Editor and provides input to the dialog action script shown below. The dialog action script is developed using the Dialog Form Designer that stores the script as a binary form file (.bfm).

The following is the action script for the **GeneralSignOn** dialog form:

```
Global DialogFormResult as Integer
Global UserId as String
Global Password as String
Global Application as String
Global Appx as Integer
Global LongVal as long
Global Dbl as double
Sub Btn_Cancel()
    ' Action for Btn_Cancel
    DialogFormResult = 0
    CloseApp
End Sub
Sub Btn_OK()
    ' Action for Btn_OK
    Application = GetString("Lbl_App")
    DialogFormResult = 1
    CloseApp
End Sub
Sub Lst_Apps()
    ' Action for Lst_Apps
    SetString "Lbl_App", ListGetItem("Lst_Apps", ListGetIndex("Lst_Apps"))
End Sub
Sub FormInitial()
    ' Action for FormInitial
    SetString "Ed_User", UserId
    SetString "Ed_Password", Password
    ListSetIndex "Lst_Apps", Appx
```

```
SetString "Lbl_App", ListGetItem("Lst_Apps", ListGetIndex("Lst_Apps"))
End Sub
```



Other examples are shown at the end of this topic.

### Doing a File Transfer

The following script and dialog form interact with the user to provide necessary information to do an FTP file transfer to/from the host mainframe. Please note that this example is for a Unisys 2200 mainframe. Also, this example is actually installed with eXpress and may be modified to suit your needs (see ftpxfer.bas in your scripts directory, and [ftp.act](#) and [ftp.bfm](#) in the installation directory).

```
Sub Main()
  Dim X, Msg
  Set df = CreateObject("TEQDlgFormx.TEQDlgForm")
  df.LoadDialogForm("C:\Program Files\KMSystems\UTSPlus32\4.0\ftp.bfm")
  df.SetString "TB_HostIP", "system.domain.com"
  df.SetString "TB_HostFile", "qual*file.elt/vers"
  df.SetString "TB_PCFile", "FTPFile.txt"
  df.ShowDialogForm

  If df.DialogResult = 1 Then
    ChDrive "c:"
    ChDir "\"
    X = Shell("cmd.exe", 1) ' Shell DOS.
    Wait 1000
    Open "FTPCmds.txt" For Output as #1
    Print #1, df.GetString("TB_UserID")
    Print #1, df.GetString("TB_Passwd")
    Print #1, df.GetString("TB_Account")
    Y = df.GetChecked("CB_Binary")
    If df.GetChecked("CB_Binary") = 1 Then
      Print #1, "BINARY"
    End If
    If df.ListGetIndex("BG_Opts") = 0 Then
      Print #1, "GET " & df.GetString("TB_HostFile") _
        & " " & df.GetString("TB_PCFile")
    Else
      Print #1, "PUT " & df.GetString("TB_PCFile") _
        & " " & df.GetString("TB_HostFile")
    End If
    Print #1, "Bye"
    Close #1
  End Sub
```

```

SendKeys "ftp -s:FTPCmds.txt " & df.GetString ("TB_HostIP") & "~"
Msg = "Click OK after FTP session complete and TELNET is closed."
MsgBox Msg, 0, "Please Wait for BYE"      ' Display OK prompt.
AppActivate "c:\windows\system32\cmd.exe"  ' Return focus to DOS.
SendKeys "exit~"                          ' Exit DOS.
End If

```

```
End Sub
```

The dialog contains all the fields necessary to connect and sign on to the host, name the host and PC files, and specify the type of transfer to be performed:

The dialog's actions, FormClose and FormInitial, are used to retain parameter settings from session (FTP transfer) to session:

```

Global DialogFormResult as Integer
Sub Btn_OK()
' Action for Btn_OK
DialogFormResult = 1
CloseApp
End Sub
Sub Btn_Cancel()
' Action for Btn_Cancel
DialogFormResult = 0
CloseApp
End Sub
Sub FormClose()
' Action for FormClose
Open "FTPParams.txt" For output as #1
Print #1, GetString ("TB_UserID")
Print #1, GetString ("TB_Account")
Print #1, GetString ("TB_HostIP")
Print #1, GetString ("TB_HostFile")

```

```

        Print #1, GetString ("TB_PCFile")
        Close #1
    End Sub
    Sub FormInitial()
    ' Action for FormInitial
        Dim UserID as String
        Dim Account as String
        Dim HostIP as String
        Dim HostFile as String
        Dim PCFile as String
        ChDrive "c:"
        ChDir "\Program Files\KMSystems\UTSPlus32\4.0\"
        If Dir("FTPparams.txt", 0) <> "" Then
            Open "FTPparams.txt" For input as #1
            Line Input #1, UserID
            Line Input #1, Account
            Line Input #1, HostIP
            Line Input #1, HostFile
            Line Input #1, PCFile
            SetString "TB_UserID", UserID
            SetString "TB_Account", Account
            SetString "TB_HostIP", HostIP
            SetString "TB_HostFile", HostFile
            SetString "TB_PCFile", PCFile
            Close #1
        End If
    End Sub

```

The script starts a DOS shell that executes the transfer as follows:

```

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\>ftp -s:FTPCmds.txt system.domain.com
Connected to system.domain.com.
220 1100JD1100 Service ready for new user.
User (system.domain.com:(none)):
331 User name okay, need password.
332 Need account for login.
230 User logged in, proceed.
ftp> GET qualifier*file.element/version c:\FTPFile.txt
200 Command okay.
150 File status okay; about to open data connection.
226 Closing data connection; requested file action successful.
ftp: 1018 bytes received in 0.19Seconds 5.44Kbytes/sec.
ftp> Bye
221 Service closing TELNET connection.
C:\>

```

### Transaction Processing with Dialogs

The following eXpress Script Editor script was developed to execute a series of transactions (programs) on the host. The script employs dialogs to allow user selection of account numbers and to present totals.

**Note:** The initial development took place using the Script Recorder feature of eXpress Plus. Subsequently, code was added to provide the dialogs.

```

' *** Script recorded by eXpress Plus Script Recorder
Option Explicit
Sub Main()
    Dim tmp as String
    Dim x as Integer
    Dim Acc as String
    Dim tot as double
    Dim cnt as Integer

    UTSKey UK_CURSOR_TO_HOME
    UTSKey UK_Erase_DISPLAY
    EnterText "cust3"

```

```

    UTSKey UK_TRANSMIT_KEY
    If not WaitForSpecificString(3, 22, 33, "Customer Address File Maintenance") Then
    Exit Sub

    Set df = CreateObject("TEQDlgFormx.TEQDlgForm")
    df.LoadDialogForm(ScriptDir + "\ACCOUNTSELECTOR.BFM")
    df.ShowDialogForm
    If Df.DialogResult = 0 Then
        df = nothing
        MsgBox "Cancelled"
        Exit Sub
    End If

    Acc = df.ListGetItem("Lst_Accounts")
    df = nothing

    SetScreenText 2, 23, 1, "X"

    EnterText "Q"
    EnterText Acc$
    UTSKey UK_TRANSMIT_KEY

    Cnt = 0
    For x = 1 to 20
        If GetScreenText(2, 23, 1) <> "X" Then
            Exit For
        End If
        Cnt = Cnt + 1
        If Cnt > 20 Then
            MsgBox "Timed out waiting for a Query response"
            Exit Sub
        End If
        Wait 500
    Next x

    If GetScreenText(2, 23, 6) = "UNABLE" Then                                ' Did not find
    account
        Exit Sub
    End If
    If x > 20 Then
        Exit Sub
    End If
    UTSKey UK_TAB_FORWARD
    UTSKey UK_TAB_FORWARD
    UTSKey UK_TAB_FORWARD
    UTSKey UK_TAB_FORWARD
    UTSKey UK_TAB_FORWARD
    UTSKey UK_TAB_FORWARD
    UTSKey UK_TAB_FORWARD
    UTSKey UK_TAB_FORWARD
    UTSKey UK_TAB_FORWARD
    UTSKey UK_TAB_FORWARD

    SetScreenText 2, 23, 1, "X"
    UTSKey UK_TRANSMIT_KEY

    Cnt = 0
    For x = 1 to 20
        If GetScreenText(2, 23, 1) = "X" or GetScreenText(2, 23, 1) = " " Then
            Cnt = Cnt + 1
            If Cnt > 20 Then
                MsgBox "Timed out waiting for a Details response"
                Exit Sub
            End If

```

```

        Wait 500
    Else
        Exit For
    End If
Next x
Wait 500
If GetScreenText(2, 23, 6) = "UNABLE" Then
    Exit Sub
End If

    If not WaitForSpecificString(2, 25, 20, "Customer Detail List") Then Exit Sub' If
expected screen not displayed stop here
Set df = CreateObject("TEQDlgFormx.TEQDlgForm")
df.LoadDialogForm(ScriptDir + "\ORDERSUMMARY.BFM")
Tot = 0
For x = 1 to 13
    Tmp = GetScreenText(47, 7 + x, 10)
    If Trim(Tmp) = "" Then
        Exit For
    End If
    Tot = Tot + Val(Trim(Tmp))
    df.ListAddItem "Btn_OrderDetails", Trim(GetScreenText(19, x + 7, 11))
    cnt = cnt + 1
Next x

df.SetString "Lbl_Total", "Total of all orders: " + Format$(Tot, "#####.00")
df.SetString "Lbl_Average", "Average amount: " & Format$(tot / cnt, "#####.00")
df.ShowDialogForm
x = df.DialogResult
df = Nothing
If x = 0 Then
    Exit Sub
End If
SetCursor 63, x + 7
UTSKey UK_TRANSMIT_KEY

' *** End of recorded script
End Sub

```

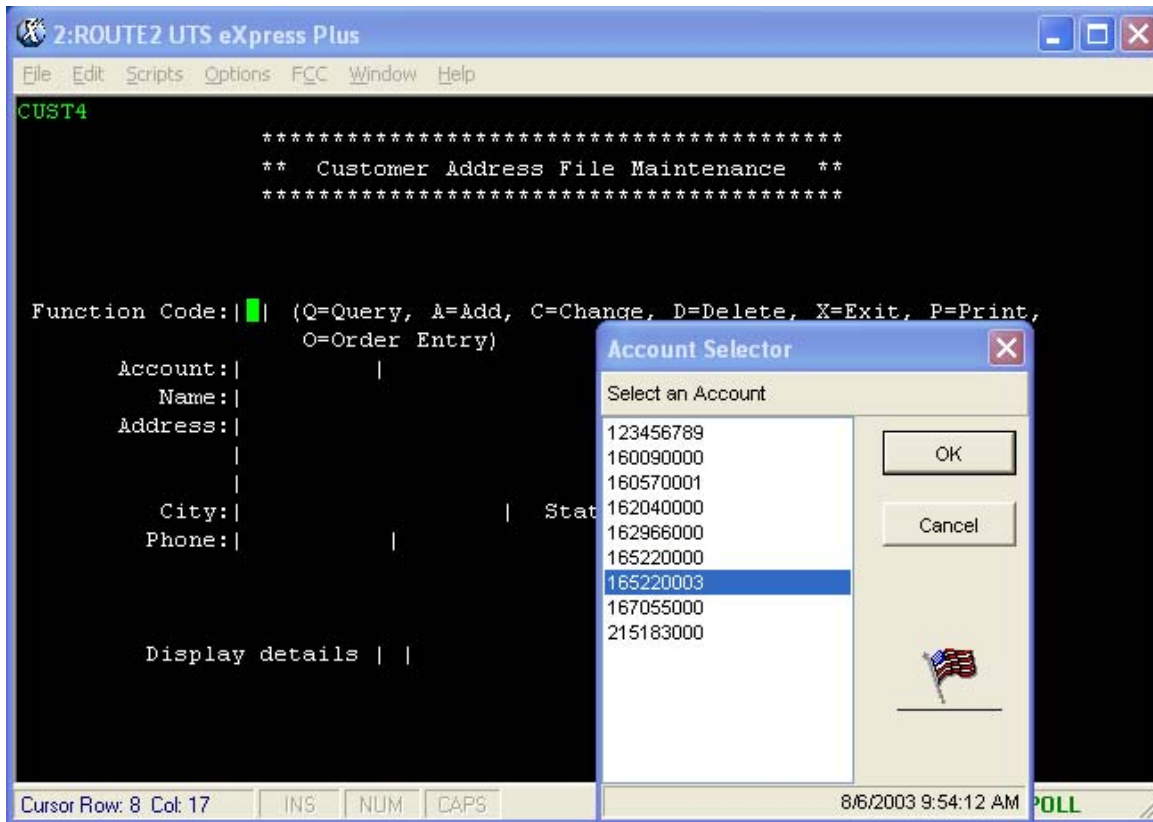
The following are the dialog action scripts that are in effect when their respective dialogs are displayed.

The **ACCOUNTSELECTOR.BFM** dialog presents a list of account numbers from which the user may choose.

```

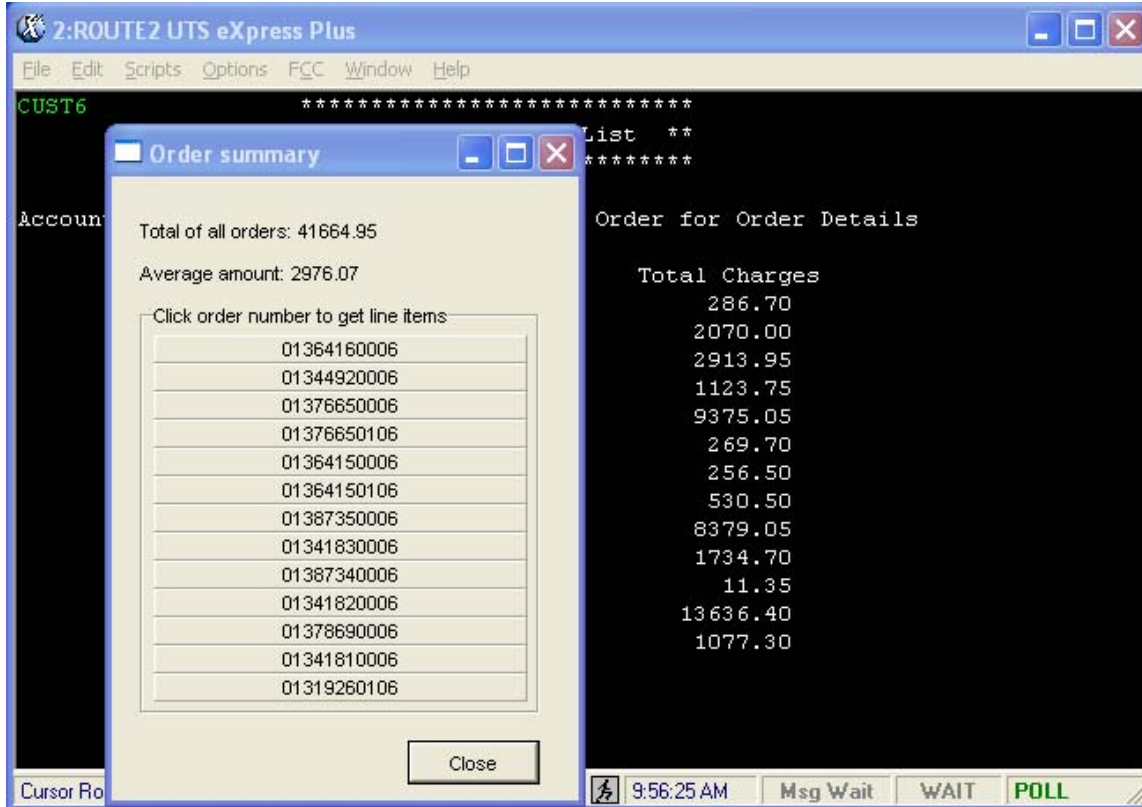
Global DialogFormResult as Integer
Sub Btn_OK()
' Action for Btn_OK
    DialogFormResult = 1
    CloseApp
End Sub
Sub Lst_AccountsDoubleClick()
' Action for Lst_AccountsDoubleClick
    Call Btn_OK
End Sub
Sub Btn_Cancel()
' Action for Btn_Cancel
    DialogFormResult = 0
    CloseApp
End Sub
Sub FormInitial()
' Action for FormInitial
    ListSetIndex "Lst_Accounts", 0
End Sub
Sub SpeedButton_1()
' Action for SpeedButton_1
    MsgBox "GO USA", mb_Iconexclamation, "USA, USA, USA"
End Sub

```



The **ORDERSUMMARY.BFM** dialog is user to display the total of all items ordered for an account.

```
Global DialogFormResult as Integer
Sub Btn_OrderDetails()
' Action for Btn_OrderDetails
DialogFormResult = ListGetIndex("Btn_OrderDetails") + 1
CloseApp
End Sub
Sub Btn_Close()
' Action for Btn_Close
DialogFormResult = 0
CloseApp
End Sub
```





## Controls on Dialog Forms

### Form Properties

Form properties are set in the same manner as any control, from the Property Editor. Select form properties in one of two ways: 1) click the mouse on a blank portion of the form or 2) select the form from the drop-down list box at the top of the Property Editor dialog.

There are two separate actions that can be associated with a form: Form Initial Action and Form Activate Action.

### Form Initial Action

The Form Initial Action is executed each time the form is loaded.

### Form Activate Action

The Form Activate Action takes place when the form regains focus.

### Form Properties

Forms have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
AutoCenter	Integer	Y	Y
BackColor	Integer	Y	Y
BorderStyle	Integer	Y	Y
Caption	String	Y	Y
Ctl3d	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Height	Integer	Y	Y
HelpContext	N/A	Y	
Left	Integer	Y	Y
MaximizedButton	Integer	Y	Y
MinimizedButton	Integer	Y	Y
ShowHint	Integer	Y	Y
Top	Integer	Y	Y
Width	Integer	Y	Y
WindowState	Integer	Y	Y
WinHelpFile	N/A	Y	

### Text Labels

Text labels can be used to create form titles, captions for edit boxes, captions for groups of controls, areas to receive messages from actions, and in general, a more Windows-like appearance to the form.

Actions associated with a text label are executed when the text label is clicked.

Related Action Statements:

```

GetNumericProp
GetState
GetString
GetStringProp
SetNumericProp
SetState
SetString
SetStringProp

```

Text labels have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
Align	Integer	Y	Y
Alignment	Integer	Y	Y
AutoSize	Integer	Y	Y
BackColor	Integer	Y	Y
Border	Integer	Y	Y
Caption	String	Y	Y

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Ctl3d	Integer	Y	Y
DataSource	N/A	Y	
Enabled	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Height	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	
ParentColor	Integer	Y	Y
ParentCtl3d	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowAccelChar	Integer	Y	Y
ShowHint	Integer	Y	Y
Top	Integer	Y	Y
Transparent	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

**Edit Boxes**

Actions associated with an edit box are executed when the user clicks in the edit box.

Related Action Statements:

GetNumericProp  
 GetState  
 GetString  
 GetStringProp  
 SetNumericProp  
 SetState  
 SetString  
 SetStringProp

Edit Boxes have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
Alignment	Integer	Y	Y
AutoSize	Integer	Y	Y
BackColor	Integer	Y	Y
CharCase	Integer	Y	
<a href="#">Ctl3d</a>	Integer	Y	Y
EditMask	String	Y	Y
<a href="#">Font</a>	N/A	Y	
<a href="#">ForeColor</a>	Integer	Y	Y
<a href="#">Height</a>	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
<a href="#">Left</a>	Integer	Y	Y
MaxLength	String	Y	Y
Name	String	Y	
ParentColor	Integer	Y	Y
ParentCtl3d	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
PassWordChar	String	Y	Y
ShowHint	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Integer	Y	Y
Text	String	Y	Y

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Top	Integer	Y	Y
Visible	Integer	Y	Y
<u>Width</u>	Integer	Y	Y

### Command Buttons

A command button is used to carry out a command or action when a user clicks the button. Any number of command buttons may be placed on a form or panel (see also, Panels).

Actions associated with a command button are executed when the button is clicked.

Related Action Statements:

```
GetNumericProp
GetState
GetString
GetStringProp
SetNumericProp
SetState
SetString
SetStringProp
```

Command buttons have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
Bitmap	N/A	Y	
BitmapPosition	Integer	Y	Y
Cancel	Integer	Y	Y
Caption	String	Y	Y
Default	Integer	Y	Y
Enabled	Integer	Y	Y
Font	N/A	Y	
Height	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	
NumBitMaps	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

### Speed Buttons

A speed button is a non-windowed control taking fewer system resources.

You can make a group of speed buttons act like radio buttons by setting the GroupIndex property to something other than 0.

The Down property specifies whether the button is down or up. The Down property is only in effect when the GroupIndex is not zero; otherwise, it will remain False. The AllowAllUp property affects this property also.

Speed buttons can also have a flat property that makes the outline show on mouse fly-over.

Actions associated with a speed button are executed when the button is clicked.

Related Action Statements:

```
GetNumericProp
GetState
GetString
GetStringProp
SetNumericProp
SetState
```

SetString  
SetStringProp

Speed buttons have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
AllowAllUp	Integer	Y	Y
Bitmap	N/A	Y	
BitmapPosition	Integer	Y	Y
Caption	String	Y	Y
Down	Boolean	Y	Y
Enabled	Integer	Y	Y
Flat	Integer	Y	Y
Font	N/A	Y	
GroupIndex	Integer	Y	Y
Height	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	
NumBitMaps	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

### Check Boxes

A check box is used to display a true/false, on/off or yes/no option that the user can set or clear by clicking. A "3" in a check box indicates that it is selected, set to on/true/yes. Any number of check boxes on a form can be checked at one time.

Actions associated with a check box are executed after the check box's state has changed (i.e., from unchecked to checked, or checked to unchecked). If an action changes the state of a check box, the check box's action will be triggered.

Related Action Statements:

GetNumericProp  
GetState  
GetString  
GetStringProp  
SetNumericProp  
SetState  
SetString  
SetStringProp

Check boxes have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
Alignment	Integer	Y	Y
<a href="#">BackColor</a>	Integer	Y	Y
Caption	String	Y	Y
Checked	Integer	Y	Y
Ctl3d	Integer	Y	Y
Enabled	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Height	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
Left	Integer	Y	Y

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Name	String	Y	
ParentColor	Integer	Y	Y
ParentCtl3d	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

See also, Edit Mask.

### Option Buttons

An option button (frequently referred to as a Radio button) is used in conjunction with other option buttons to offer multiple choices, from which the user can select only one. Option buttons may be placed directly on the form, on a panel or in a button group. The buttons will be mutually exclusive upon the control on which they are placed.

Actions associated with an option button are executed when the option button's state has changed. If an action changes the state of an option button, the option button's action will be triggered.

Related Action Statements:

```
GetNumericProp
GetState
GetString
GetStringProp
SetNumericProp
SetState
SetString
SetStringProp
```

Option buttons have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
Alignment	Integer	Y	Y
<a href="#">BackColor</a>	Integer	Y	Y
Caption	String	Y	Y
Checked	Integer	Y	Y
Ctl3d	Integer	Y	Y
Enabled	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Height	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	
ParentColor	Integer	Y	Y
ParentCtl3d	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

### List Boxes

A list box is used to display a list of items from which a user may view or select (to select from the list requires that an action be associated with the list box). The list box size on the form is not dependent upon the number of

values to be placed in the box. If a list box contains more values than can be accommodated by the size of the list box, a scroll bar will appear making all items in the list accessible. If a list is to contain many values, it might be desirable to use a drop-down list box to conserve space on the form.

Actions associated with a list box are executed when an item is clicked or double-clicked (two separate actions) from the list.

Related Action Statements:

```
GetNumericProp
GetState
GetStringProp
ListClear
ListCount
ListGetIndex
ListGetItem
ListSetIndex
ListSetItem
ListItemAdd
ListItemRemove
SetNumericProp
SetState
SetStringProp
```

List boxes have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
Align	Integer	Y	Y
<a href="#">BackColor</a>	Integer	Y	Y
Ctl3d	Integer	Y	Y
Enabled	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Height	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
Items	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentColor	Integer	Y	Y
ParentCtl3d	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
Sorted	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

### Drop-down List Boxes

Like the list box, the drop-down list box is used to display a list of items from which a user can select; however, the drop-down list box takes up less room on the form, and as such, is useful when the selection list contains many values. The drop-down list box only occupies one line on the form. In addition, the drop-down list box is limited to a single column.

Actions associated with a drop-down list box are executed when an item is selected (clicked) from the list.

Related Action Statements:

```
GetNumericProp
GetState
GetStringProp
ListClear
ListCount
ListGetIndex
```

ListGetItem  
 ListSetIndex  
 ListSetItem  
 ListItemAdd  
 ListItemRemove  
 SetNumericProp  
 SetState  
 SetStringProp

Drop-down list boxes have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
<a href="#">BackColor</a>	Integer	Y	Y
Ctl3d	Integer	Y	Y
Enabled	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Height	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
Items	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentColor	Integer	Y	Y
ParentCtl3d	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
Sorted	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

### Multi-column List Boxes

A multi-column list box is similar to the standard list box in usage and behavior; however, in addition to supporting multiple columns, it allows optional column headings.

Actions associated with a multi-column list box are executed when an item is clicked or double-clicked (two separate actions) from the list.

Related Action Statements:

GetNumericProp  
 GetState  
 GetStringProp  
 ListClear  
 ListColHeader  
 ListGetColText  
 ListCount  
 ListGetIndex  
 ListSetColText  
 ListSetIndex  
 ListItemAdd  
 ListItemRemove  
 SetNumericProp  
 SetState  
 SetStringProp

Multi-column list boxes have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
Align	Integer	Y	Y

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
<a href="#">BackColor</a>	Integer	Y	Y
ColumnHeaders	Integer	Y	Y
Columns	N/A	Y	
Ctl3d	Integer	Y	Y
Enabled	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Height	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
NumericSort	Boolean	Y	Y
ParentColor	Integer	Y	Y
ParentCtl3d	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
SortColumn	Integer	Y	Y
SortDescending	Boolean	Y	Y
Sorted	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

See also, Multi-Column List Setup.

### **Memo**

A memo control is a multi-line edit box. The user can type and edit large amounts of text in a Memo control. Access to a Memo from an Action Script is similar to handling items in a List Box. Use the following list-oriented Action statements to manipulate lines in a memo control: ListClear, ListCount, ListItemAdd and ListItem.

If GetString and SetString are used with Memo control, the following should be considered:

- With SetString, text is moved into the memo and split into multiple lines depending upon the setting of the WordWrap property. For line breaks, carriage returns (Chr\$(13)) may be inserted into the text before using SetString. If WordWrap is false and no carriage returns are present, the memo will contain a single line of text.
- With GetString, all lines of the memo are concatenated into a single string with spaces added between lines replacing carriage return characters. Only lines ending with a carriage return are modified. The carriage return is replaced with a single space. Lines automatically wrapped due to the control's width do not contain carriage returns.

Related Action Statements:

```
GetString
ListClear
ListCount
ListItem
ListItemAdd
ListItemRemove
SetString
```



Memo controls have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Run-time</u>
Action	N/A	Y	
Align	Integer	Y	Y
BackColor	Integer	Y	Y
BorderStyle	Integer	Y	Y
Ctl3d	Integer	Y	Y
Enabled	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Lines	String	Y	Y
MaxLength	Integer	Y	Y
Name	String	Y	Y
ParentCtl3d	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ReadOnly	Integer	Y	Y
ScrollBars	Integer	Y	Y
ShowHint	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
WantsReturns	Integer	Y	Y
Width	Integer	Y	Y
WordWrap	Integer	Y	Y

### Bevels

A bevel may be used to enhance the appearance of the form.

Actions may NOT be associated with a bevel.

Related Action Statements:

```
GetNumericProp
GetStringProp
SetNumericProp
SetStringProp
```

Bevels have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Align	Integer	Y	Y
Name	String	Y	Y
Shape	Integer	Y	Y
Style	Integer	Y	Y
Visible	Integer	Y	Y

### Button Groups

Button groups are used to group option buttons logically onto a single control. In other words, when the options in a group need to be mutually exclusive (only one option may be selected), the button group should be used. Note: Option buttons may be placed on a panel when tab control is required between buttons.

When the button group is first placed on the form, no option buttons appear in the group. Use the Items property to add the option buttons. Next, use the button group sizing handles to size the group to contain the option items entered. The Item Index property can be used to specify which option in the group is initially set when the form is loaded (e.g., -1 means that no option is set; 0 means the first option is set; 1, the second; etc.).

Actions associated with a button group are executed when the state of any option button in the group has changed. If an action changes the state of an option button, the button group's action will be triggered.

Button Groups may be populated at run-time just like list boxes.

Related Action Statements:

GetNumericProp  
 GetState  
 GetStringProp  
 ListClear  
 ListCount  
 ListGetIndex  
 ListGetItem  
 ListSetIndex  
 ListSetItem  
 ListItemAdd  
 ListItemRemove  
 SetNumericProp  
 SetState  
 SetStringProp

Button groups have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
Align	Integer	Y	Y
<a href="#">BackColor</a>	Integer	Y	Y
ButtonStyle	Integer	Y	Y
Caption	String	Y	Y
Columns	Integer	Y	Y
Ctl3d	Integer	Y	Y
Enabled	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Height	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
ItemIndex	Integer	Y	Y
Items	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentCtl3d	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

### Group Boxes

A group box may be used to logically group or frame a set of controls. The group box may not be used to establish option groups, and as such, has no physical functionality.

Actions may NOT be associated with group boxes.

Related Action Statements:

GetNumericProp  
 GetState  
 GetString  
 GetStringProp  
 SetNumericProp  
 SetState  
 SetString  
 SetStringProp

Group boxes have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
<a href="#">BackColor</a>	Integer	Y	Y

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Caption	String	Y	Y
Ctl3d	Integer	Y	Y
Enabled	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Height	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentColor	Integer	Y	Y
ParentCtl3d	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

### Panels

A panel may be used to house one or more controls in a logical group. The controls behave on the panel as if they were placed on a form. In other words, if you move the panel the controls on the panel will move with the panel. In addition, if a "parent" property of a control is set to True, the control will take the property of the panel, the parent.

Actions associated with a panel are executed when the panel is clicked. The panel must be enabled and visible to click it. A panel can be used as a big button since its caption text can wrap from line to line. Panel text can also be aligned left, right or centered. The caption of regular buttons can contain only a single line of text centered in the client area.

Related Action Statements:

```
GetNumericProp
GetState
GetString
GetStringProp
SetNumericProp
SetState
SetString
SetStringProp
```

Panels have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
Align	Integer	Y	Y
Alignment	Integer	Y	Y
<a href="#">BackColor</a>	Integer	Y	Y
BevelInner	Integer	Y	Y
BevelOuter	Integer	Y	Y
BevelWidth	Integer	Y	Y
BorderStyle	Integer	Y	Y
BorderWidth	Integer	Y	Y
Caption	String	Y	Y
Ctl3d	Integer	Y	Y
Enabled	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Height	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
ParentColor	Integer	Y	Y
ParentCtl3d	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

### Splitters

A splitter is used to allow the end user to adjust the viewing area of controls like list boxes and panels at run-time. Actions may NOT be associated with splitters.

Related Action Statements:

```
GetNumericProp
GetStringProp
SetNumericProp
SetStringProp
```

Splitters have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Align	Integer	Y	Y
AutoSnap	Integer	Y	Y
<a href="#">BackColor</a>	Integer	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
MinSize	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

### Images

Images not only can be used to enhance the appearance of a form but also can be functional Windows controls that when selected perform a prescribed action.

Actions associated with an image are executed when the image is clicked.

Related Action Statements:

```
GetNumericProp
GetStringProp
LoadImage
SetNumericProp
SetStringProp
```

Images have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Action	N/A	Y	
Align	Integer	Y	Y
AutoSize	Integer	Y	Y
Border	Integer	Y	Y
Center	Integer	Y	Y
Enabled	Integer	Y	Y
Height	Integer	Y	Y
HelpContext	N/A	Y	
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	
ParentShowHint	Integer	Y	Y
Picture	N/A	Y	
ShowHint	Integer	Y	Y

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Strech	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

### Media Players

The media player control bar can be placed on a form to activate and play a movie. Normally, the form on which the control is placed loads the multi-media file when the form is first displayed. The control itself has no action associated with it; therefore, some other event, like the initial form action or command button, must be used to load the media file. For Example:

```
Sub FormInitial()  
  ' Action for FormInitial  
  Rslt = LoadMMFile("Player", "C:\Data\TESTx\SPEEDIS.AVI")  
End Sub
```

"Player" is the name of the media player control as set in the Name property.

Actions may NOT be associated with the media player control.

Related Action Statements:

```
GetNumericProp  
GetStringProp  
LoadMMFile  
SetNumericProp  
SetStringProp
```

The media player control has the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Enabled	Integer	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
MonoChromeButtons	Integer	Y	Y
Name	String	Y	
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

### Date/Time Labels

The data/time label is used to supply the current date and time (machine time) to the user on the form or control. For example, this control could be placed on a panel as a part if a status bar.

Actions may NOT be associated with a date/time label.

Related Action Statements:

```
GetNumericProp  
GetState  
GetString  
GetStringProp  
SetNumericProp  
SetState  
SetString  
SetStringProp
```

Date/Time labels have the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Align	Integer	Y	Y
Alignment	Integer	Y	Y
<a href="#">BackColor</a>	Integer	Y	Y
BlinkColor	Integer	Y	Y

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Runtime</u>
Blinking	Integer	Y	Y
BlinkIntervalOff	Integer	Y	Y
BlinkIntervalOn	Integer	Y	Y
Enabled	Integer	Y	Y
FlatColor	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Format	String	Y	Y
FrameStyle	Integer	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
<u>Top</u>	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

See also, Date Time Format Editor.

### Browser

The browser control can be placed on a form to embed the browser on a form. The URL is set at run-time by an action using the "SetString" statement.

```
Sub FormInitial()
  ' Action for FormInitial
  SetString "Browser_1", "http://www.kmsys.com"
End Sub
```

"Browser\_1" is the name of the media player control as set in the Name property.

eQuate actions may NOT be associated with the browser control.

Related eQuate Action Statements:

```
GetNumericProp
GetStringProp
SetNumericProp
SetString
SetStringProp
```

The browser control has the following defined properties, some of which can be manipulated under the control of your application:

<u>Property</u>	<u>Type</u>	<u>Design-time</u>	<u>Run-time</u>
Enabled	Integer	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	
ParentShowHint	Integer	Y	Y
ShowHint	Integer	Y	Y
TabOrder	String	Y	Y
TabStop	Integer	Y	Y
Top	Integer	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y

### URL Link

URL links can be used to place a URL link on a form. A URL link is similar to a Text Label in that its caption appears on the form, but it differs in that when clicked, it opens a separate browser window to the URL specified in the URL property. Furthermore, a URL link differs from a Browser control in that no browser is embedded on the form.

Actions associated with a URL link are executed when the URL link is clicked.

Related Action Statements:

GetNumericProp  
 GetState  
 GetString  
 GetStringProp  
 SetNumericProp  
 SetState  
 SetString  
 SetStringProp

Text labels have the following defined properties, some of which can be manipulated under the control of your application:

Property	Type	Design-time	Runtime
Action	N/A	Y	
Align	Integer	Y	Y
Alignment	Integer	Y	Y
AutoSize	Integer	Y	Y
BackColor	Integer	Y	Y
Caption	String	Y	Y
Enabled	Integer	Y	Y
Font	N/A	Y	
ForeColor	Integer	Y	Y
Height	Integer	Y	Y
Hint	String	Y	Y
Left	Integer	Y	Y
Name	String	Y	
ParentColor	Integer	Y	Y
ParentFont	Integer	Y	Y
ParentShowHint	Integer	Y	Y
ShowAccelChar	Integer	Y	Y
ShowHint	Integer	Y	Y
Top	Integer	Y	Y
Transparent	Integer	Y	Y
URL	String	Y	Y
Visible	Integer	Y	Y
Width	Integer	Y	Y





## Control Properties

### Align Property

Use this property to align and automatically size the control on the parent control. Predefined constants are `alNone` (default), `alTop`, `alBottom`, `alLeft`, `alRight` and `alClient`.

`alTop` and `alBottom` will align the control at the top or bottom edge of the parent control, respectively. The control will take the height of the parent control.

`alLeft` and `alRight` will align the control at the left or right edge of the parent control, respectively. The control will take the width of the parent control.

`alClient` will align at the width and height of the parent control.

### Alignment Property

This property is used to align the text in a control. Predefined constants are `alLeftJustify`, `alRightJustify` and `alCenter` (not applicable for edit boxes).

### AllowAllUp Property

The `AllowAllUp` property is used to force all speed buttons in a group to be up by default.

`AllowAllUp` is used in conjunction with `GroupIndex`. If you want a group of speed buttons to toggle like car radio buttons, you can set the `GroupIndex` of all of all buttons in the group to the same number. By default, one in the group will always be down. `AllowAllUp` changes the default behavior so that all buttons in such a group can be up.

### AutoCenter Property

Use this control to cause the form to be centered on the desktop at run-time.

### AutoSize Property

This property is used to size the control according to the size of the text or image placed in the control. The default is `False`.

### AutoSnap Property

The `AutoSnap` property is used to determine if the splitter will automatically snap to the edge of a parent control. The default is `True`.

### BackColor Property

Use this control to set the background (non-text) color of a control. The default for most controls is `CP-BtnFace`. The default for edit boxes is `CP-Window`.

For forms, there is a rule regarding the form's `Ctl3d` and `BackColor` properties. If the `BackColor` is `CP-BtnFace`, when you toggle `Ctl3d` from `True` to `False`, the `BackColor` will toggle from `CP-BtnFace` to `CP-Window`. If the `BackColor` is something else, the current color is not change when `Ctl3d` is toggled.

See also Predefined Constants.

### BevelInner Property

Use this property along with the `BorderWidth` property to add an inner bevel to a Panel control. The `BorderWidth` is the number of pixels from the outer edge of the panel that the inner bevel will begin. Predefined constants are `bvNone` (default), `bvLowered` and `bvRaised`. See also, `BevelOuter`, `BevelWidth`, `BorderStyle` and `BorderWidth`.

### BevelOuter Property

Use this property to apply a lowered or raised bevel on the outer edge on a Panel control. Predefined constants are `bvNone`, `bvLowered` and `bvRaised` (default). See also, `BevelInner`, `BevelWidth`, `BorderStyle` and `BorderWidth`.

### BevelWidth Property

The `BevelWidth` property is used to specify the width of any bevel on a Panel control. The default for this property is 1 pixel. See also, `BevelInner`, `BevelOuter`, `BorderStyle` and `BorderWidth`.

### Bitmap Property

Use this property to select an optional bitmap to place on a button face.

### BitmapPosition Property

Use the `BitmapPosition` property to specify the position of the bitmap on the button face. Predefined constants are `blLeft` (default), `blRight`, `blTop` and `blBottom`.

**BlinkColor Property**

Use this property to set the color for blinking text on the DateTimeLabel control when the Blinking property is set to True. The default color is CP-Highlight. This color is only visible on the control if the Blinking property is set to True. See also, Blinking, BlinkIntervalOff and BlinkIntervalOn.

**Blinking Property**

Use the Blinking property to make the text on the DateTimeLabel control blink. The default is False. See also, BlinkColor, BlinkIntervalOff and BlinkIntervalOn.

**BlinkIntervalOff Property**

This property is used to set the interval (in milliseconds) that the text in the DateTimeLabel control will NOT be displayed in BlinkColor property color. The default is 500 milliseconds. This value is only used if the Blinking property is set to True. See also, BlinkColor, Blinking and BlinkIntervalOn.

**BlinkIntervalOn Property**

Use this property to set the interval (in milliseconds) that the text in the DateTimeLabel control will be displayed in the BlinkColor property color. The default is 500 milliseconds. This value is only used if the Blinking property is set to True. See also, BlinkColor, Blinking and BlinkIntervalOff.

**Border Property**

This property may be used to place a border around a Text Label or Image control. The default is False. For Text Labels, this control is impacted by the setting of the Ctl3d property.

**BorderStyle Property**

Use this property to set the border style for Forms, Memo and Panel controls. Predefined constants are bsNone, bsSingle, bsSizeable (default) and bsDialog. The bsSizeable and bsDialog settings only apply to forms.

The bsSizeable value displays the form as a standard resizable dialog.

The bsDialog value displays the form as a non-resizable dialog with a standard border. In addition, with the bsDialog value, the system menu (upper-left) will not be shown, and the Minimize, Maximize and Close buttons (upper-right) will not be shown.

The bsSingle value causes a non-resizable dialog with a single-line border to be displayed.

The bsNone value displays a non-resizable dialog with no border.

For Panels, also see, BevelInner, BevelOuter, BevelWidth and BorderWidth.

**BorderWidth Property**

Use the BorderWidth property in conjunction with the BevelInner property to place a border on a Panel control. The BorderWidth is the number of pixels from the outer edge of the panel that the inner bevel will begin. The default is 1 pixel. See also, BevelInner, BevelOuter, BevelWidth and BorderStyle.

**ButtonStyle Property**

This property may be used to select the button style for the buttons in a Button Group control. Predefined constants are bsRadio (default) and bsPush.

**Cancel Property**

Use the Cancel property to specify whether a button's Action executes when the Escape key is pressed. If Cancel is True, the button's Action executes when the user presses Esc. Although a Form can have more than one Cancel button, the form calls the Action only for the first visible button in the tab order. The default value is False. To set the initial tab order, select Form Edits | Tab Order from the Edit menu on the Dialog Form Designer Toolbar or right-click anywhere on the form and select Tab Order. You may also order the tab order at runtime by using the TabOrder property.

**Caption Property**

This property is used to set text that will appear as a caption on a control; e.g., the caption in blue at the top of a dialog, the text in a text label, etc.

To add an accelerator key, add an ampersand (&) in front of any character in the caption. That character will appear underlined and becomes the accelerator key for that control. Accelerator key values should be unique amongst all accelerator key values on the form. When an accelerator key is pressed at runtime, the control's action is executed. See also, ShowAccelChar.

Note: On Text Labels, the accelerator character has no effect, other than how it appears in the text.

Note: Most control captions are a single line of characters; however, Text Labels allow for multiple lines in the caption. In a Text Label, use the Carriage Return key (chr\$(13)) to begin the next line.

**Center Property**

Use this property to center the picture in an Image control. The default is False.

**CharCase Property**

This property may be used to specify the case of the characters to be typed into an Edit Box. Predefined constants are ecNormal (default, allowing both uppercase and lowercase), ecUpperCase and ecLowerCase.

**Checked Property**

Use this property to initially or programmatically check a checkbox or set an option button. The default is False.

**ColumnHeaders Property**

Use this property to establish column headers on a Multi-column List control. The default is False. When set to True, use the Columns property to create header text.

**Columns Property**

This property may be used to alter the number of columns (default is 5), change the row height (default is 17 pixels), specify headers and dividers. See also, the ColumnHeaders.

**Ctl3d Property**

Use the Ctl3d property to set disable on enable a three-dimensional look on a control. The default is True. See also, ParentCtl3d and the notes for the BackColor.

**DataSource Property**

This property may be used to alter the data source for an Edit Box or Text Label control. The DataSource property must be a valid field name in the screen associated with the current form.

**Default Property**

Use the Default property to give focus to a Command Button control when the form is first displayed. The default is False. If True and the user hits the Enter key, the button will be clicked.

**Down Property**

This property is used to initially at design time or programmatically at runtime press down a Speed Button control. The default is False. When used in conjunction with the GroupIndex property, and when multiple Speed Buttons have the same group index value, setting this property to True will raise another Speed Button in the same group, like car radio buttons.

See also Predefined Constants.

**EditMask Property**

Use this property to change the input edit mask for the Edit Box control. See Edit Mask dialog.

**Enabled Property**

This property is used to enable or disable a control. The default is True (enabled).

**Flat Property**

Use this property to give a speed button a flat instead of raised appearance. The default is False. If this property is set to True, the outline of the button will appear upon mouse flyover.

**FlatColor Property**

This property may be used to change the color of the frame on a Date/Time Label control when the FrameStyle is set to fsFlat. The default is Black.

**Font Property**

Use the Font property to change the font on a control's caption or the font of a parent control (see Parent Controls). See also, ParentFont.

**ForeColor Property**

Use this property to set the foreground (text) color of a control. The default for most controls is CP-BtnFace. The default for edit boxes is CP-Window.

For forms, there is a rule regarding the form's Ctl3d and BackColor properties. If the BackColor is CP-BtnFace, when you toggle Ctl3d from True to False, the BackColor will toggle from CP-BtnFace to CP-Window. If the BackColor is something else, the current color is not change when Ctl3d is toggled.

See also Predefined Constants.

**Format Property**

This property is used to change to format of the date and time on the Date/Time Label control. See the Date Time Format Editor.

**FrameStyle Property**

Use this property to change the frame style on a Date/Time Label control. The available styles are fsNone, fsFlat, fsGrove, fsBump, fsLowered, fsButtonDown, fsRaised, fsButtonUp, fsStatus (default) and fsPopup. If the fsFlat value is selected, the color of the frame may be changed with the FlatColor property.

**GroupIndex Property**

This property is used to group Speed Buttons on a parent control. When used in conjunction with the Down property, and when multiple Speed Buttons have the same group index value, setting this property to True will raise another Speed Button in the same group, car radio buttons.

**Height Property**

Use this property to change the height of the control. It is used in conjunction with the Top property. The value is specified in pixels. For design purposes and ease, all controls have sizing handles and may be sized/aligned with other controls by right clicking on a group of selected controls (see "Aligning and Sizing Controls" on the [Dialog Form Designer Toolbar](#) help page).

**HelpContext Property**

This property may be used to assign help for a control from a standard windows help file (.hlp).

**Hint Property**

Use this property to add a hint to a control. At runtime, when the user holds the mouse cursor over the control, the hint will appear briefly. See ShowHint and ParentShowHint.

**ItemIndex Property**

The ItemIndex property is used to specify which button in a Button Group is to be on initially at runtime. The default is -1 (no button initially set). 0 is the first button, 1 the second, etc. The Items property must be set prior to setting this property.

**Items Property**

Use this property to name the buttons in a Button Group. Each line entered is a separate button name. Use the Return key between lines.

**Left Property**

This property may be used to change the horizontal starting position of the control. It is used in conjunction with the Width property. The value is specified in pixels. For design purposes and ease, all controls have sizing handles and may be sized/aligned with other controls by right clicking on a group of selected controls (see "Aligning and Sizing Controls" on the [Dialog Form Designer Toolbar](#) help page).

**Lines Property**

Use this property to enter or edit in a Memo control at design time.

**MaximizedButton Property**

This property is used to add or remove the standard Windows maximize button to a form. The default is True.

**MaxLength Property**

This property is used to change the maximum length (in characters) of an Edit Box or Memo control. The default is 0 (no maximum). For a Memo control, this represents the total number of characters in the control including carriage returns (chr\$(13)). See also, Memo control.

**MinimizedButton Property**

Use this property to add or remove the standard Windows minimize button to a form. The default is True.

**MinSize Property**

The MisSize property is used to specify the minimum size of the panes (in pixels) on either side of the Splitter control. The default is 30.

Set MinSize to provide a minimum size the splitter must leave when resizing its neighboring control. For example, if the Align property is alLeft or alRight, the splitter cannot resize the regions to its left or right any smaller than MinSize pixels. If the Align property is alTop or alBottom, the splitter cannot resize the regions above or below it any smaller than MinSize pixels.

Note: Always set MinSize to a value less than half the client width of its parent. When MinSize is half the client width of the splitter's parent, the splitter cannot move because to do so would be to resize one of the panes less than MinSize pixels.

**MonoChromeButtons Property**

Use this property to change the buttons on the Media Player control to appear in monochrome as opposed to color. The default is False.

**Name Property**

This property is used to assign a label to a control that can be referenced in an action script.

**NumBitMaps Property**

Use this property to specify the number of bitmaps that are to be used on a Command Button or Speed Button when the button is enabled or disabled. The default is 1. The maximum is 4. This property is used in conjunction with the Bitmap property.

Buttons can show different images depending on the state of the button: Up, Disabled, Clicked and Down. The Bitmap property can reference a bitmap that is divided into four images of equal size, side-by-side in a row. The first image will be shown when the button is up or has focus; the second if the button is disabled, the third when the button is pushed and the fourth if the button remains down.

If there is only one image in the bitmap, this image is used for all four states.

Note: The lower left pixel of the bitmap is reserved for the "transparent" color. Any pixel in the bitmap that matches that lower left pixel will be transparent.

**NumericSort Property**

Use this property in conjunction with the SetNumericProp subroutine when sorting multi-column list boxes.

The property is Boolean, specifying that the data in the SortColumn property is to be compared numerically. The default is False (alpha compare).

This property can be changed in the Form Designer or, dynamically, at runtime.

See also, SortColumn and SortDescending properties. For an example, see the SetNumericProp subroutine.

**ParentColor Property**

If this property is set to True, the control will use the same color property of the parent control. The default is False. Also see, Parent Controls.

**ParentCtl3d Property**

This property is used to specify if a control is to take on the three-dimensional look (Ctl3d property) of the parent control. The default is True. Also see, Parent Controls.

**ParentFont Property**

If this property is set to True, the control will use the same Font property of the parent control. The default is True. Also see, Parent Controls.

**PassWordChar Property**

This property is used to specify a password character for an Edit Box control. If a password character is specified, the user will only see password characters as they type into the edit box. The default is no password character in which case all characters typed are visible.

**Picture Property**

This property is used to place a bitmap into an Image control. The type of files that may be loaded are Windows or OS/2 bitmap (.bmp), icon (.ico), Windows metafile (.wmf) Windows enhanced metafile (.emf) or JPEG compliant files (.jpg and .jpeg).

**ReadOnly Property**

The ReadOnly property is used to determine whether the Memo control may have data entered at runtime or will be "read only." The default is False (read/write).

**ScrollBars Property**

Use this property to add or remove scrollbars to a Memo control. Predefined constants are ssNone (default), ssHorizontal, ssVertical and ssBoth.

**Shape Property**

This property determines the shape of a Bevel control. Predefined constants are bsBox (default), bsFrame, bsTopLine, bsBottomLine, bsLeftLine, bsRightLine and bsSpacer.

**ShowAccelChar Property**

Use this property to display or not display accelerator characters in a Text Label control. The default is True.

The accelerator character is an ampersand (&). If the ShowAccelChar property is set to true and an ampersand is entered to the left of a character, the character will appear underlined in the text label. For example, "S&ample" would appear as "Sample".

Note: On Text Labels, the accelerator character has no effect, other than how it appears in the text.

**ShowHint Property**

The ShowHint property along with the Hint property determines if a brief pop-up is to be displayed when the user moves the mouse cursor over the control. The default is True. See also, ParentShowHint Property.

**SortColumn Property**

Use this property in conjunction with the SetNumericProp subroutine when sorting multi-column list boxes.

The property is an Integer in the range of 0 to columns -1. The default is 0 (the first column).

This property can be changed in the Form Designer or, dynamically, at runtime.

See also, NumericSort and SortDescending properties. For an example, see the SetNumericProp subroutine.

**SortDescending Property**

Use this property in conjunction with the SetNumericProp subroutine when sorting multi-column list boxes.

The property is Boolean, specifying that the sort order is descending. The default is False (ascending sort).

This property can be changed in the Form Designer or, dynamically, at runtime.

See also, NumericSort and SortColumn properties. For an example, see the SetNumericProp subroutine.

**Sorted Property**

Use this property to determine if the items in a List Box, Drop-down List Box or Multi-column List Box are to be sorted. The default is False.

**Strech Property**

This property may be used to force a picture to fit the size if the Image control. The default is False.

**Style Property**

Use this property to specify the style of a Bevel control. Predefined constants are bsLowered (default) and bsRaised.

**ParentShowHint Property**

Use this property to specify if the control is to use the ShowHint property of the parent control. The default is True. Also see, Parent Controls.

**TabOrder Property**

This property may be used to set the tab order of the controls when the user clicks the tab key. The tab order is like an index beginning with 1 and incremented by 1. It is used in conjunction with the TabStop property. An alternate and quick way to set tab order on multiple controls is to select Form Edits | Tab Order from the Edit menu on the Dialog Form Designer Toolbar or right-click anywhere on the form and select Tab Order.

**TabStop Property**

Use this property to specify whether a control is to be a tab stop when the user presses the tab key. The default is True.

**Text Property**

The Text property may be used to place text into an Edit Box control. The text is for documentation purposes only and will not be displayed at runtime.

**Top Property**

Use the Top property to change the vertical starting position of the control. It is used in conjunction with the Height property. The value is specified in pixels. For design purposes and ease, all controls have sizing handles and may be sized/aligned with other controls by right clicking on a group of selected controls (see "Aligning and Sizing Controls" on the [Dialog Form Designer Toolbar](#) help page).

**Transparent Property**

This property is used to determine if the background of a Text Label control is see-through. The default is False.

**URL Property**

This property is used to set the URL link to be accessed by the browser. When this control is clicked at runtime, it automatically opens the browser to the URL specified. No user code is needed.

**Visible Property**

Use this property to make a control visible or hidden. The default is True (visible).

**WantsReturns Property**

This property determines how the return character is to be handled in a Memo control. If True, the application accepts return characters into the text and does not pass the return character to any other control. If false, return

characters are not accepted by the Memo control and may be passed to another control such as a default button. The default is True.

**Width Property**

Use this property to change the width of the control. It is used in conjunction with the Left property. The value is specified in pixels. For design purposes and ease, all controls have sizing handles and may be sized/aligned with other controls by right clicking on a group of selected controls (see "Aligning and Sizing Controls" on the [Dialog Form Designer Toolbar](#) help page).

**WindowState Property**

This property is used to set the window state of the form. Predefined constants are wsNormal (default), wsMinimized and wsMaximized.

**WinHelpFile Property**

This property may be used to enter the name of an externally developed help file (.hlp). Help files are normally placed in the installation directory. Note: Do not enter the .hlp extension.

**WordWrap Property**

Use this property to specify if words will wrap to the next line when a line is full on a Memo control. The default is True. If True, lines automatically wrap based on the width of the control.





## Property Dialogs

### Parent Controls

Most controls have "Parent" properties (ParentCtl3D, ParentFont and/or ParentShowHint). If a parent property is set to True, then the control takes on the property of the parent control. Currently, there are only two types of parent controls other than the form itself: the group box and the panel control. For example, if a button on the form has the ParentFont property set to True, the button caption will use the same font assigned to the form. Note: Changing the Font property on the button will automatically set ParentFont property to False.

A panel or a group box can have controls placed on it so that whenever you move the panel during the design phase, all the controls on the panel move with it. Likewise, if an action disables the panel at runtime, all the controls on the panel are also disabled since the panel is parent to all the controls placed on the panel.

### Color Selection

This dialog is used to assign a color property of a control.

### Standard Color

From the list box on the left, select the desired Windows color.

### Custom...

To select a non-standard color, click this button to initiate the Color dialog palette.

### Edit Mask

This dialog is used to apply the EditMask property to an edit box.

### Input Edit Mask

The Input Edit Mask is the mask that is used to limit the data that can be put into a masked edit box. A mask restricts the characters the user can enter to valid characters and formats. If the user attempts to enter a character that is not valid, the edit box does not accept the character. Validation is performed on a character-by-character basis.

If no edit mask is specified, the end-user is not restricted, except by maximum length if specified.

If a Custom Edit Mask is selected, the Input Edit Mask text box may be used to specify a mask other than the standard field edit masks supplied with the Dialog Form Designer. The Input Edit Mask is a case-sensitive text box used to specify the type of input that will be allowed, and the position in the field where each character will appear.

A mask consists of three fields with semicolons (;) separating the fields. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of a mask are saved as part of the data. The third part of the mask is the character used to represent a blank in the mask.

#### Part 1:

The first part of the Edit Mask can contain any of the following characters:

<u>Character</u>	<u>Meaning in Mask</u>
!	If an exclamation (!) character appears in the mask, leading blanks do not appear in the data. If an exclamation character is not present, trailing blanks do not appear in the data.
>	If a greater than (>) character appears in the mask, all characters that follow are in uppercase until the end of the mask or until a greater than character is encountered.
<	If a less than (<) character appears in the mask, all characters that follow are in lowercase until the end of the mask or until a less than character is encountered.
<>	If these two characters appear together in a mask, no case checking is done and the data is formatted with the case the user uses to enter the data.
\	The character that follows a back slash (\) character is a literal character. Use this character when you want to allow any of the mask special characters as a literal in the data.
L	The "L" character requires an alphabetic character only in this position. For the US, this is A-Z, a-z.
l	The "l" character permits only an alphabetic character in this position, but does not require it.
A	The "A" character requires an alphanumeric character only in this position. For the US, this is A-Z, a-z, and 0-9.
a	The "a" character permits an alphanumeric character in this position, but does not require it.
C	The "C" character requires a character in this position.

<u>Character</u>	<u>Meaning in Mask</u>
c	The "c" character permits a character in this position, but does not require it.
0	The zero (0) character requires a numeric character only in this position.
9	The nine (9) character permits a numeric character in this position, but does not require it.
#	The pound (#) character permits a numeric character or a plus or minus sign in this position, but does not require it.
:	The colon (:) character is used to separate hours, minutes, and seconds in times. If the character that separates hours, minutes, and seconds is different in the International settings of the Control Panel utility on your computer system, that character is used instead of the colon.
/	The slash (/) character is used to separate months, days, and years in dates. If the character that separates months, days, and years is different in the International settings of the Windows Control Panel utility on your computer system, that character is used instead of the slash.

**Part 2:**

In the second part of the edit mask, the "0" character means that the mask is not saved as part of the data. The "1" character means that the mask is saved as part of the data. For example, a telephone number could have parentheses around the area code as part of the mask. If the second part of the edit mask is "0", the parentheses do not become part of the data, making the size of the field slightly smaller.

**Part 3:**

In the third part of the Edit Mask, the underscore (\_) character may be used to automatically insert underscores in the edit box for positions that are not yet filled. You may change this character to any desired fill character or a space.

Examples:

<u>Edit Mask</u>	<u>Display</u>	<u>Internal</u>
\(999\)999\ -9999;0;	(770)635-6363	7706356363
\(999\)999\ -9999;1;	(770)635-6363	(770)635-6363
999-999;0;_	123-4__	1234

**Pre-defined Standard Edit Mask**

This list box contains pre-defined edit masks from which you may select one. If you wish a customized edit mask, type directly into the Input Edit mask text box.

**Character for Blanks**

In this text box, enter the character that will appear in the edit box in place of a blank value.

**Save Literal Characters**

Check this box to save mask characters as part of the data. See, "Part 2," above.

**Test Input**

Use this text box to test type input.

**Edit Mask**

This dialog is used to apply the EditMask property to an edit box.

**Input Edit Mask**

The Input Edit Mask is the mask that is used to limit the data that can be put into a masked edit box. A mask restricts the characters the user can enter to valid characters and formats. If the user attempts to enter a character that is not valid, the edit box does not accept the character. Validation is performed on a character-by-character basis.

If no edit mask is specified, the end-user is not restricted, except by maximum length if specified.

If a Custom Edit Mask is selected, the Input Edit Mask text box may be used to specify a mask other than the standard field edit masks supplied with the Dialog Form Designer. The Input Edit Mask is a case-sensitive text box used to specify the type of input that will be allowed, and the position in the field where each character will appear.

A mask consists of three fields with semicolons (;) separating the fields. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of a mask are saved as part of the data. The third part of the mask is the character used to represent a blank in the mask.

**Part 1:**

The first part of the Edit Mask can contain any of the following characters:

<u>Character</u>	<u>Meaning in Mask</u>
!	If an exclamation (!) character appears in the mask, leading blanks do not appear in the data. If an exclamation character is not present, trailing blanks do not appear in the data.
>	If a greater than (>) character appears in the mask, all characters that follow are in uppercase until the end of the mask or until a greater than character is encountered.
<	If a less than (<) character appears in the mask, all characters that follow are in lowercase until the end of the mask or until a less than character is encountered.
<>	If these two characters appear together in a mask, no case checking is done and the data is formatted with the case the user uses to enter the data.
\	The character that follows a back slash (\) character is a literal character. Use this character when you want to allow any of the mask special characters as a literal in the data.
L	The "L" character requires an alphabetic character only in this position. For the US, this is A-Z, a-z.
l	The "l" character permits only an alphabetic character in this position, but does not require it.
A	The "A" character requires an alphanumeric character only in this position. For the US, this is A-Z, a-z, and 0-9.
a	The "a" character permits an alphanumeric character in this position, but does not require it.
C	The "C" character requires a character in this position.
c	The "c" character permits a character in this position, but does not require it.
0	The zero (0) character requires a numeric character only in this position.
9	The nine (9) character permits a numeric character in this position, but does not require it.
#	The pound (#) character permits a numeric character or a plus or minus sign in this position, but does not require it.
:	The colon (:) character is used to separate hours, minutes, and seconds in times. If the character that separates hours, minutes, and seconds is different in the International settings of the Control Panel utility on your computer system, that character is used instead of the colon.
/	The slash (/) character is used to separate months, days, and years in dates. If the character that separates months, days, and years is different in the International settings of the Windows Control Panel utility on your computer system, that character is used instead of the slash.

**Part 2:**

In the second part of the edit mask, the "0" character means that the mask is not saved as part of the data. The "1" character means that the mask is saved as part of the data. For example, a telephone number could have parentheses around the area code as part of the mask. If the second part of the edit mask is "0", the parentheses do not become part of the data, making the size of the field slightly smaller.

**Part 3:**

In the third part of the Edit Mask, the underscore (\_) character may be used to automatically insert underscores in the edit box for positions that are not yet filled. You may change this character to any desired fill character or a space.

Examples:

<u>Edit Mask</u>	<u>Display</u>	<u>Internal</u>
\(999\)999\.-9999;0;	(770)635-6363	7706356363
\(999\)999\.-9999;1;	(770)635-6363	(770)635-6363
999-999;0;_	123-4__	1234

**Pre-defined Standard Edit Mask**

This list box contains pre-defined edit masks from which you may select one. If you wish a customized edit mask, type directly into the Input Edit mask text box.

**Character for Blanks**

In this text box, enter the character that will appear in the edit box in place of a blank value.

**Save Literal Characters**

Check this box to save mask characters as part of the data. See, "Part 2," above.

**Test Input**

Use this text box to test type input.

**Date Time Format Editor**

This dialog is used to establish the format of the date/time stamp shown in the DateTimeLabel control.

**Date Time Format**

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

<u>Character</u>	<u>Meaning</u>
c	Display the date as dddd and display the time as tt, in that 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 the day as a full name (Sunday-Saturday).
dddd	Display the date as m/d/yy.
dddddd	Display the date as dddd, mmmm d, yyyy (e.g., Friday, October 27, 2000).
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).
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 (0000-9999).
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).
t	Display the time as h:n AM/PM.
tt	Display the time as h:n:s AM/PM.
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.
A/P	Use the 12-hour clock display an uppercase A/P.
a/p	Use the 12-hour clock display a lowercase a/p

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

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

**Predefined Formats**

Select from this list if you want to use a predefined date/time format. See Date Time Format, above, for a description of the various predefined formats.

## Global Controls

### Dialog Form Menu Designer

This dialog is used to add user menu items to the form.

Once a menu has been updated with the Dialog Form Menu Designer, the action for the menu item may be updated by selecting the menu item from the dialog form displayed in the Dialog Form Designer. See Dialog Form Designer.

### Menu Item Caption

This entry specifies the menu item's caption. Captions may include accelerator keys by typing an ampersand character (&) in front of the desired letter within the caption text.

Menu separators may be indicated by entering a hyphen (-) in the caption.

Blank menu items are not permitted.

### Name

In this text box, enter the name of the menu or menu selection. This name can be referenced in an action.

### Short Cut

From the drop-down list box, select a short cut key for the menu selection. The shot cut key is optional.

### Insert Item

Insert a blank menu item before the currently selected menu item. All menu items below and including the current menu item will be pushed down one position.

### Indent Level

Use the arrow controls in this group to raise or lower the level of the selected menu or menu item in the Menu Items list. A menu item without a preceding asterisk represents a menu on the form's menu bar. Clicking the right arrow lowers the item level as indicated by an added asterisk (\*) preceding the item. Clicking the left arrow raises the item level. Using the left and right arrow keys on the keyboard will also raise or lower the item level.

An item of a higher level with items below it at the next lower level becomes a cascading menu. For example, clicking an item called "Updates" (denoted by a single asterisk preceding it) might reveal a cascading menu containing "Add", "Replace" and "Delete" (each preceded by double asterisks).

### Move

Use the arrow controls in this group to position the menu selection up or down in the list. You may also move an item with the keyboard by holding down the shift key and pressing the up and down arrow keys.

### Checked

Check this control to preset the menu item to its selected state. Note: Only menu items may be checked – menus may not.

### Visible

Check this box to make the menu or menu selection initially visible.

### Enabled

Check this box to enable the menu or menu item initially.

### Preview

Click this button to open a small dialog that reveals the designed menu bar. Test the menu and menu selection by clicking the menu.

### Menu Items

This list box contains the menu and menu items defined. To work with an existing item, select it with the mouse or up and down arrows. Use the Indent and Move controls to rearrange the menus and menu items.



## Action Editor Used with Dialog Form Designer

### Dialog Form Action Editor

This window provides the means to develop and edit dialog form actions. The editor contains multiple tabs that allow more than one action file to be edited at a time.

Following is a description of each Dialog Form Action Editor command. All the commands may be performed by making a menu selection. Toolbar buttons, shortcut keys and right mouse click actions are available for frequently used commands. A right mouse click anywhere in the test area will produce a pop-up menu of commands.

The menu items below show an image of the toolbar button and the shortcut key combination (in parentheses) where applicable.

#### File menu

The File menu contains commands to maintain action files and setup printing.

**New (Ctrl+N)**

Use this command to create a new action file (.ACT).

**Open (Ctrl+O)**

Use this command to open an existing action file.

**Save (Ctrl+S)**

Use this command to save the current action file.

**Save As...**

Use this command to save the current action file to another file name.

**Close**

Close the currently selected action file.

**Close All**

Close all open action files.

**Print (Ctrl+P)**

Use this command to print the entire action file.

**Printer Setup...**

Allow margins to be set and allow printers and printer fonts to be selected for printing.

**Clear Previous File List**

Remove all file names from the list of previously accessed files.

**Editor Properties**

Use this command to edit the properties of the action editor: window font, highlight colors and tab stops.

**Exit**

Exit the Action Editor.

**Previous File List**

Select (open) from the list of previous accessed action files.

#### Edit menu

The Edit menu contains commands to manage selected text between the editor and the Windows clipboard.

**Undo (Ctrl+Z)**

Use this command to reverse the effects of the most recent change.

**Redo (Ctrl+Shift+Z)**

Use this command to reverse the effects of the most recent **Undo** command.

**Cut (Ctrl+X)**

Use this command to place the selected text on the clipboard and delete.

**Copy (Ctrl+C)**

Use this command to copy the selected text to the clipboard.

**Paste (Ctrl+V)**

Use this command to paste the contents of the clipboard to the current cursor position.

**Delete (Ctrl+D)**

Use this command to delete the selected text without copying to the clipboard.

**Word Wrap (Ctrl+W)**

Use this command as a toggle. By default, long lines may only be viewed/edited by first bringing the excess text into view with the horizontal scroll bar or by using the cursor keys (arrows). When **Word Wrap** is set, long lines wrap to the next line and are viewable within the confines (width) of the window.

**Search menu**

The Search menu contains commands to locate and change text within the action file.

**Find... (Ctrl+F)**

Use this command to enable the **Find** dialog used to locate text strings.

**Find Again (F3)**

Use this command to find the next occurrence of the same string used on the previous find.

**Replace... (Ctrl+R)**

Use this command to enable the **Replace** dialog used to locate and replace text strings.

**Go to Line (Ctrl+G)**

Use this command to go to a specific line.

**Bookmarks**

The Bookmarks menu contains commands to mark lines and navigate within the action file.

**Set Bookmark 1 through 5 (Shift+F1 through Shift+F5)**

Use one of these commands to mark a line at the current text cursor position. A book marked line will appear with a gray background.

**Go to Bookmark 1 through 5 (Ctrl+F1 through Ctrl+F5)**

Use one of these commands to go to a line previously book marked by one of the five corresponding **Set Bookmark** commands.

**Options menu**

The Option menu contains commands to specify color, font and tab stop preferences.

**Show Tool Bar**

Use this command to toggle the display of the toolbar.

**Syntax Highlight**

Use this command to toggle the display of the action syntax. This command is affected by the settings of the **Editor Properties** command, above.

**View Permanent Declarations**

Use this command to show the permanent declarations that are automatically included with all actions. You browse permanent declarations in read-only mode; however, you may copy code from the browse window to the Windows clipboard using the Ctrl+C key.



## Tools

The **Tools** menu contains commands to check action syntax.



### **Check Script (F4)**

Use this command to check the syntax of the entire action file.



### **Compile Script**

Use this command to compile the action and save it in encrypted form (.BAX). This option is useful for sites that wish to secure the contents of script files from general viewing (e.g., user-id/password). Either the text form (.BAS) or the compiled form of the script may be made available to the user for execution.

## Help

The **Help** menu contains commands to display on-line help and information about the product.

### **Contents**

Use this command to display the contents of the on-line help.

### **This Window**

Use this command to receive on-line help for this window.

### **About...**

Use this command to display copyright and product version information.

## Editor Properties

This dialog is used to change the properties or appearance of a script window.

### **Edit Window Font**

The controls in this group affect the font typeface, size and intensity used to display the script.

#### **Font Name**

From this drop-down list box, choose from the list of non-proportional, fixed fonts installed on your PC.

#### **Size**

With this spin wheel, increase or decrease the font size.

#### **Bold**

Check this box to increase the font intensity.

### **Tab Size**

With this spin wheel, increase or decrease the number characters between tab characters.

### **Highlight Colors**

Use this group to assign colors to different parts of the script text.

#### **Set Text Color**

To change color, select the type of text (Normal text, Strings, etc.) and select from the Set Text Color drop-down list box to change the foreground.

#### **Set Background Color**

To change the background color, select from the Set Background Color drop-down list box.

### **OK**

Click this button to accept the changes made and exit the dialog.

### **Cancel**

Click this button to discard the changes made and exit the dialog.

### **Help**

Click this button to receive on-line help for this dialog.



## Language Elements

### 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. Rem, all other keywords, and most names in an action 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

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)
```

This is equivalent to:

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

### Line Continuation Character

The underscore (`_`) is the line continuation character. There must be a space before and after the line continuation character.

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

### Numbers

Three representations of numbers are supported: 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 may be comprised of uppercase letters (A through Z), lowercase letters (a through z), underscore (`_`) characters, and numeric digits (0 through 9). Variable and constant names can be no longer than 40 characters and cannot be reserved words. For a table of reserved words, see Language Reference. One exception to this rule is that object member names and property names may be reserved words.



## Variables

### Variable Types

As is the case with Visual Basic, when a variable is introduced, it is not necessary to declare it first (see Option Explicit for an exception to this rule).

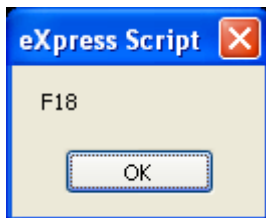
Note: Use the browse buttons (>>) (<<) at the top of this window to view the variable types.

### Variant

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 convert a variable explicitly from one data type to another. This data type conversion is handled automatically.

Example:

```
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 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 ampersand (&) operator instead of the plus (+) operator. The ampersand will not perform arithmetic on your numeric values; it will simply concatenate them as if they were strings.

Example:

```
X = String_1 & Integer_2 ' Concatenate a string and a number - OK
```

Instead of:

```
X = String_1 + Integer_2 ' Error
```

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

### Other Data Types

The six data types available are shown below with their declaration character suffixes:

<u>Data type</u>	<u>Suffix</u>	<u>Type Declaration</u>	<u>Size</u>	<u>Range</u>
String	\$	Dim StrVar As String	String of characters	0 to 65,500 characters
Integer	%	Dim IntVar As Integer	2-byte integer	-32,768 to 32,767
Long	&	Dim LongVar As Long	4-byte integer	-2,147,483,648 to 2,147,483,647

<u>Data type</u>	<u>Suffix</u>	<u>Type Declaration</u>	<u>Size</u>	<u>Range</u>
Single	!	Dim SingVar As Single	4-byte floating-point number	-3.402823E38 to -1.401298E-45 (negative values) 1.401298E-45 to 3.402823E38 (positive values)
Double	#	Dim DbIVar As Double	8-byte floating-point number	-1.79769313486232D308 to -4.94065645841247D-324 (negative values) 4.94065645841247D-324 to 1.79769313486232D308 (positive values)
Variant		Dim X As Variant	Date/time, floating-point number or string	Date values: January 1, 0000 through December 31, 9999; numeric values: same range as Double; string values: same range as String
Currency		(Currency datatype is not supported)		

**Other Data Types**

The six data types available are shown below with their declaration character suffixes:

<u>Data type</u>	<u>Suffix</u>	<u>Type Declaration</u>	<u>Size</u>	<u>Range</u>
String	\$	Dim StrVar As String	String of characters	0 to 65,500 characters
Integer	%	Dim IntVar As Integer	2-byte integer	-32,768 to 32,767
Long	&	Dim LongVar As Long	4-byte integer	-2,147,483,648 to 2,147,483,647
Single	!	Dim SingVar As Single	4-byte floating-point number	-3.402823E38 to -1.401298E-45 (negative values) 1.401298E-45 to 3.402823E38 (positive values)
Double	#	Dim DbIVar As Double	8-byte floating-point number	-1.79769313486232D308 to -4.94065645841247D-324 (negative values) 4.94065645841247D-324 to 1.79769313486232D308 (positive values)
Variant		Dim X As Variant	Date/time, floating-point number or string	Date values: January 1, 0000 through December 31, 9999; numeric values: same range as Double; string values: same range as String
Currency		(Currency datatype is not supported)		

**Other Data Types**

The six data types available are shown below with their declaration character suffixes:

<u>Data type</u>	<u>Suffix</u>	<u>Type Declaration</u>	<u>Size</u>	<u>Range</u>
String	\$	Dim StrVar As String	String of characters	0 to 65,500 characters
Integer	%	Dim IntVar As Integer	2-byte integer	-32,768 to 32,767
Long	&	Dim LongVar As Long	4-byte integer	-2,147,483,648 to 2,147,483,647
Single	!	Dim SingVar As Single	4-byte floating-point number	-3.402823E38 to -1.401298E-45 (negative values) 1.401298E-45 to 3.402823E38 (positive values)
Double	#	Dim DbIVar As Double	8-byte floating-point number	-1.79769313486232D308 to -4.94065645841247D-324 (negative values) 4.94065645841247D-324 to 1.79769313486232D308 (positive values)

<u>Data type</u>	<u>Suffix</u>	<u>Type Declaration</u>	<u>Size</u>	<u>Range</u>
Variant		Dim X As Variant	Date/time, floating- point number or string	Date values: January 1, 0000 through December 31, 9999; numeric values: same range as Double; string values: same range as String
Currency		(Currency datatype is not supported)		





## Flow of Control

### Control Structures

The scripting language 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, one branching statement is available: GoTo.

Note: Use the browse buttons (>>) (<<) to view each control structure.

#### The GoTo

The GoTo statement branches to the label specified on the Goto statement.

```
Goto label1
:
:
label1:
```

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

#### The 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 condition
    statement(s)...
Loop
Do Until condition
    statement(s)...
Loop
Do
    statement(s)...
Loop While condition
Do
    statement(s)...
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.

#### The 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
    statement(s)...
Wend
```

#### The 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 = beginvalue To endvalue [Step increment]
    statement(s)...
Next
```

### The 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 statement(s) ' single line syntax

If condition Then           ' multiple line syntax
    statement(s)...
End If
```

The other variation of the If statement is the If...Then...Else statement. This statement should be used when there are different statement blocks to be executed depending on the condition. A variation of the If...Then...Else is the If...Then...ElseIf.

```
If condition Then
    statement(s)...
ElseIf condition Then
    statement(s)...
Else
    statement(s)...
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
        statement(s)...
    Case 2
        statement(s)...
    Case 3
        statement(s)...
    Case Else
        statement(s)...
End Select
```

## Subroutine and Functions

### Subroutines and Function Naming Conventions

Subroutine and function names may be comprised of uppercase letters (A through Z), lowercase letters (a through z), underscore (\_) and numeric digits (0 through 9). The only limitation is that subroutine and function names must begin with a letter, be no longer than 40 characters and not be a reserved word. For a list of reserved words, see the list of reserved words in Language Reference.

Script developers can create their own functions or subroutines or make DLL calls. Subroutines are created with the syntax:

```
Sub subname
.
.
End Sub
```

Functions have a similar syntax:

```
Function funcname As type
.
.
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 as parameters. The keyword ByVal denies this permission and the parameters cannot be reassigned outside their local procedure. ByRef is the 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 is when using the ByVal keyword to protect the variable or using parentheses, which indicate the variable is ByVal.

If the arguments or parameters are passed with parentheses around them, you are passing them ByVal.

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

The var3 parameter 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 "R" function is receiving two parameters, "X" and "n". The second parameter, "n", is passed by value and the contents cannot be changed from within the "R" function.

In the following code samples, scalar variable types 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 ' 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 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 execute functions and subroutines located within the libraries. There are two ways scripts can be extended: 1) calling functions and subroutines in DLLs and 2) calling 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 the global declaration section of the script. All declarations 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 is 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 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.

## Files

### 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:

```

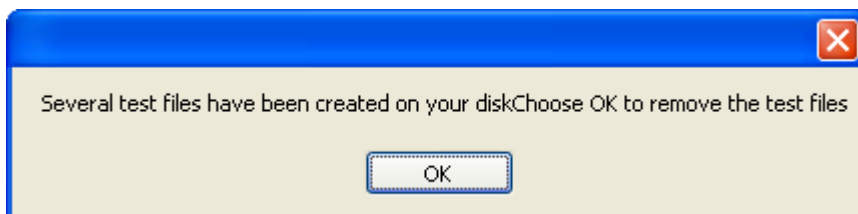
Sub Main
  Open "TESTFILE" For Input As #1      ' Open file.
  Do While Not EOF(1)                  ' Loop until end of file.
    Line Input #1, TextLine            ' Read line into variable.
    Print TextLine                       ' Print to Debug window.
  Loop
  Close #1                         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

### Arrays

Enable supports single-dimensioned and multidimensional arrays. By 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 than necessary.

All the elements in an array have the same data type. Enable supports arrays of integers, singles, double and strings.

Ways to declare a fixed size array:

- Global array – use the Global or Dim statement outside the general declarations section of a module to declare the array;
- Local array – use the Dim statement inside a procedure or function.

Enable does not support dynamic arrays.

### Single-dimensioned Arrays:

When declaring an array, the array name must be followed by the upper bound (boundary) 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 Global in place of Dim:

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

The same declarations within a procedure use Static or Dim:

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

The Counters declaration creates an array with 13 elements, with index numbers running from 0 to 12. The Sums declaration creates an array with 27 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 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. With Enable Basic, 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 a value of five (5):

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

```

### Multidimensional Arrays:

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) = L * 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

### User Defined Types

Users can define their own types that are composites of other built-in or user-defined types. Variables of these new user types can be declared. Member variables of the new type can be accessed using dot notation. Variables of user-defined types cannot be passed to DLL functions expecting "C" structures.

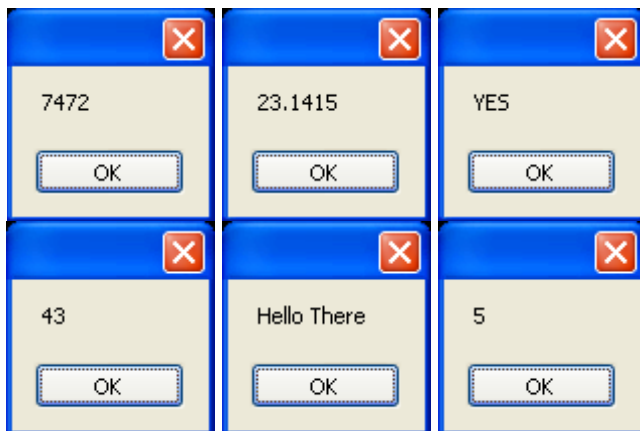
User-defined types are created using the type statement, which must be placed outside the procedure in your Enable code. User-defined types are global. The variables that are declared as user-defined types can be either global or local. User-defined types in Enable cannot contain arrays at this time.

```
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 typela As type1

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





## Dialogs and Dialog Controls

### Dialog Support

This topic covers the code that is used to establish and maintain dialogs in a script. Written manually, this code can be very complicated; therefore, it is recommended that you use the Enable Dialog Designer, from within the Action Script Editor whenever possible.

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 types of dialogs supported are outlined below.

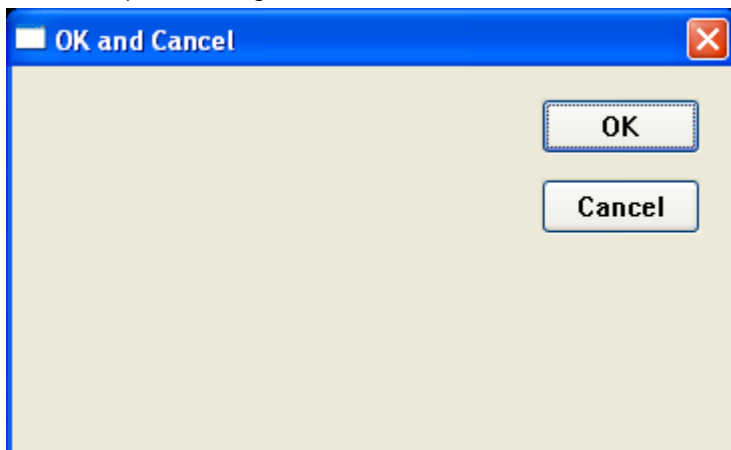
Most of the standard Windows dialog box controls are supported.

Use the browse buttons (>>) (<<) to review the controls available for custom dialog boxes and the guidelines for using them.

### OK and Cancel

```
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 - an OK button or a Cancel button. Enable includes separate dialog box definition statements for each of these two types of buttons.



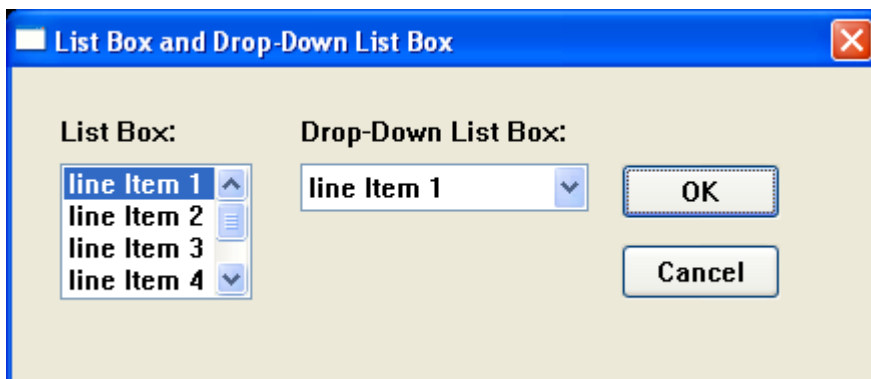
### List Boxes and Drop-down List Boxes

You can use a list box or drop-down list 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). The items displayed in a list box or drop-down list box are stored in an array that is defined before the instructions that define the dialog box.

```
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 159,175, 216, 78, "List Box and Drop-Down List Box"
    OKButton 152,24,40,14
    CancelButton 152,44,40,14
    ListBox 12,24,48,40, MyList$ (), .Lstbox
    DropListBox 72,24,72,40, MyList$ (), .DrpList
    Text 12,12,32,8, "List Box:"
    Text 72,12,68,8, "Drop-Down List Box:"
End Dialog

    Dim Dlg1 As BoxSample
    Button = Dialog ( Dlg1 )

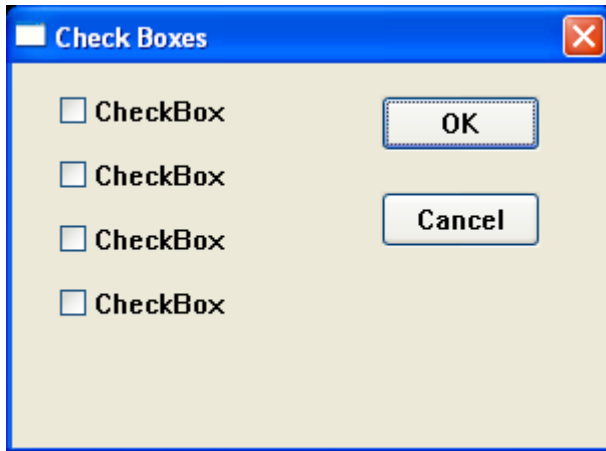
End Sub
```



### Check Boxes in Dialog

```
Sub Main
  Begin Dialog CheckSample 15,32,149,96,"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.

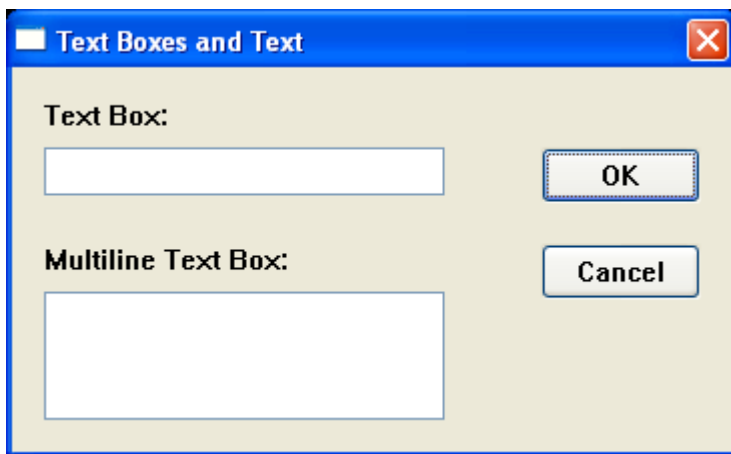


### Text Boxes and Text

```
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
```

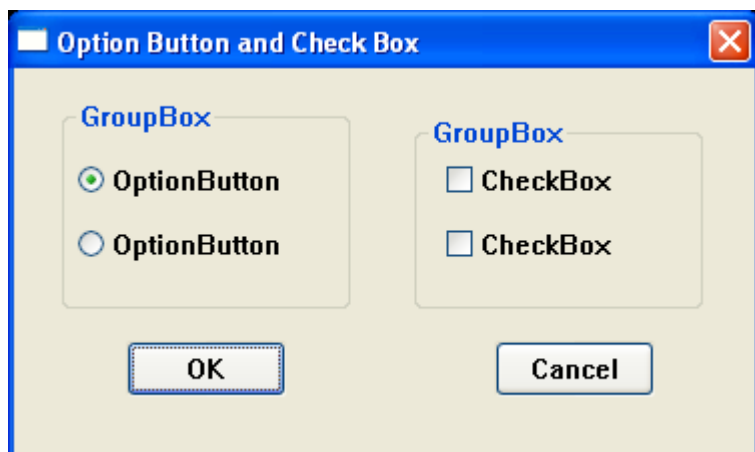
A text box control is a box in which the user can enter text while the dialog box is displayed. By default, a text box holds a single line of text. Enable does not support multiline text boxes in this version - this feature will be included in later versions.



### 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
```



## The Dialog Function

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 be taken.

The dialog box and its function are connected in the dialog definition. A function name argument (e.g., UserDialog1, below) 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 upon which some action will take place. 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 zero (0) 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.

ControlID\$ receives the identifier of the dialog box control.

Action identifies the action that calls the dialog function. Enable supports two actions:

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.

SuppValue receives supplemental information about a change in a dialog box control. The information SuppValue receives depends upon which control calls the dialog function:

Checkbox	0 if cleared, 1 if selected.
Option Button	Number of option buttons selected, where zero is the first option button within a group.
Command Button	A value identifying the button chosen. SuppValues for push buttons are internal only.
OK Button	-1
Cancel Button	0

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

```



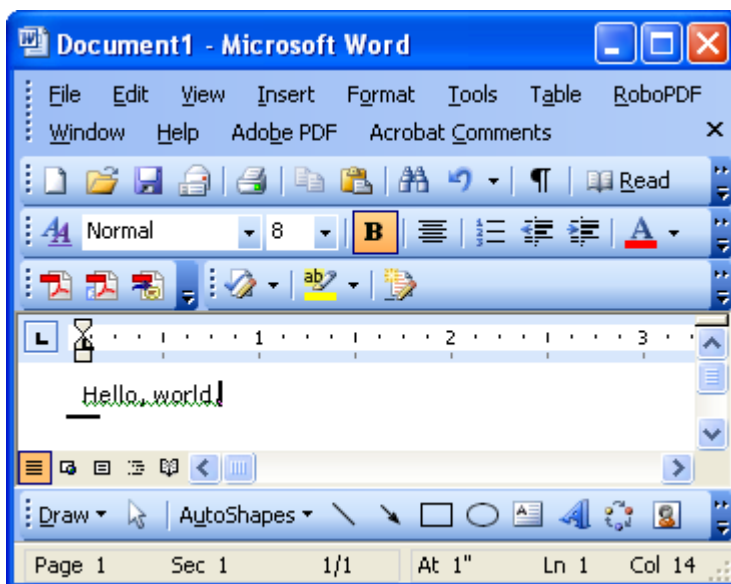
## OLE Automation

### What is OLE Automation?

OLE Automation is an emerging standard, promoted by Microsoft, which 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 (e.g., Microsoft Excel 5.0). A word processor might expose objects such as an application, paragraph, sentence, bookmark or selection (e.g., Microsoft Word 6.0).

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 objects 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 follows to manipulate the object:

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



The following syntax is supported for the GetObject function:

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

"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:

<u>Name</u>	<u>Description</u>
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. These instances 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.

Microsoft suggests the following objects could be exposed by implementing the listed methods and properties for each object.

#### Applications:

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

#### Documents:

OLE Automation Object	Methods	Properties
Document	Activate Close NewWindow Print PrintPreview RevertToSaved Save SaveAs	Application Author Comments FullName Keywords Name Parent 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 Close Item Open	Application Count Parent

### OLE Fundamentals

Object Linking and Embedding (OLE) is a technology that allows programmers of Windows-based applications to create applications that can display data from many different applications. OLE 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 manipulate the data programmatically 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

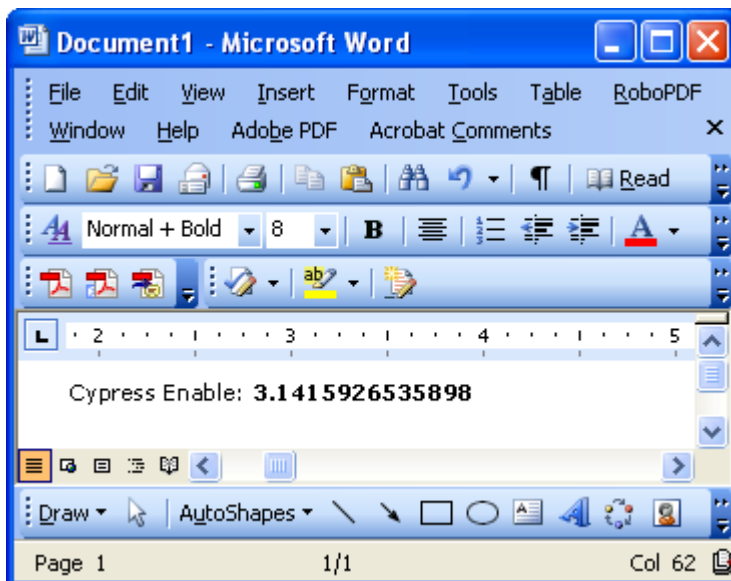
The class of an object determines the application that provides the object's data and the type of data the object contains. The class names of some commonly used Microsoft applications include MSGraph, MSDraw, WordDocument and ExcelWorksheet.

## OLE Automation and Word example

```
Sub OLEexample()
    Dim word6 As Object
    Dim myData As String

    myData = 4 * Atn(1)
        ' Demonstrates Automatic type conversion
    Set word6 = CreateObject("Word.Basic")
    word6.AppShow
    word6.FileNewDefault
    word6.Insert "The following was computed in Cypress Enable: "
    word6.Bold 1          ' Show value in boldface
    word6.Insert myData
    word6.Bold 0

    MsgBox "Done"
End Sub
```





## Data Types, Operators and Precedence

### Data Types, Operators and Precedences

This topic is a quick reference of the data types, operators and precedence used in scripting.

#### Data Types

<u>Variable</u>		<u>Type Declaration</u>	<u>Size</u>
String	\$	Dim Str_Var As String	0 to 65500 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 Variant	
Currency		(Currency datatype is not supported)	

#### Arithmetic Operators

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

\*Arithmetic operators follow mathematical rules of precedence

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

#### Operator Precedence

<u>Operator</u>	<u>Description</u>	<u>Order</u>
()	Parenthesis	Highest
^	Exponentiation	
-	Unary Minus	
/, *	Division/Multiplication	
mod	Modulo	
=, >, <, <=, >=	Relational	
not	Negation	
and	And	
or	Or	Lowest

#### Relational Operators

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

#### Logical Operators

<u>Operator</u>	<u>Function</u>	<u>Usage</u>
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, Subroutines and Events

### Abs Function

Return the absolute value of a *number*.

Format:

*Abs(number)*

The data type of the return value is the same as that of the *number* argument. 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

Activate an application.

Format:

*AppActivate "application"*

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

Related Topics: Shell, SendKeys

Example:

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

### Asc Function

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

Format:

*Asc(string)*

Example:

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

### Atn Function

Return the arctangent of a number.

Format:

*Atn(string)*

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

Related Topics: Cos, Tan, Sin

Example:

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

### Beep Statement

Sound a tone through the computer's speaker.

Format:

**Beep**

The frequency and duration of the beep depends on hardware, which may vary among computers.

Example:

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

### Begin Dialog Statement

Mark the beginning of a dialog and the overall dimensions of the dialog box.

Note: Dialogs when written manually can be very difficult; therefore, it is recommended that you use the Dialog Designer, from within the Action Script Editor whenever possible.

Format:

**Begin Dialog** *name starting-x-pos, starting-y-pos, width, height, "caption"* [, . *functionname*]

Related Topics: Dialog Support, CancelButton, CheckBox, Dialog, DropListBox, GroupBox, ListBox, OKButton, OptionButton, OptionGroup, PushButton, Text, TextBox

Example:

```
Sub LST()
  Begin Dialog DIALOG_1 0,0, 128, 60, "Order Amount Update"
    OKButton 80,12,36,12
    CancelButton 80,28,36,12
    Text 4,8,36,8, "Amount"
    TextBox 4,16,64,12, .TXT_AMT
  End Dialog

  Dim Dlg As Dialog_1
  Dim amt As Single
  Dim x As integer

  if ListCount("LST") > 0 then
    x = Val(ListGetIndex("LST")) + 1
    Amt = Val(ListGetColText("LST", 1))
    Dlg.Txt_Amt = Format$(Amt, "#0.00")
    if Dialog(Dlg) then
      Amt = Val(Dlg.Txt_Amt)
      ListSetColText "LST", 1, Format$(Amt, "#0.00")
      SetString "TOTAL_CHARGES:" + Str$(x), Format$(Amt, "#####.00")
    end if
  else
    beep
  end if
End SUB
```

**CalendarDialog Function (eXpress Plus)**

Display a calendar from which a user can select a date.

Format:

`CalendarDialog (InitialDate, ControlName)`

The *InitialDate* parameter is any string variable containing a date on the calendar to select initially. The date is entered as a string in the YYYYMMDD format. It must be exactly eight characters in length. If an empty string is used, the current date is selected.

The *ControlName* parameter is any string variable containing the name of the control in the current form over which the calendar dialog will be positioned. If an empty string is used, the dialog will be positioned in the upper left corner of the form.

The dialog simply displays a calendar with which the user can select a date. Initially, the calendar displays a single month, but the dialog may be expanded to show up to an entire year. The date is returned as a string in the format "YYYYMMDD". Canceling returns what ever was supplied as an initial date.

**Call Statement**

Activate an Enable subroutine or DLL function.

Formats:

`Call name [(parameters(s))]`

or

`name [parameter(s)]`

Activate an Enable or DLL subroutine called *name*. The first parameter is the *name* of the function or subroutine to call, and the second is the list of *parameter(s)* to pass to the called function or subroutine.

Note: Script Functions and Subroutines may be placed in a user library (USER.LIB) and become global to any script that has need to call them. The user library must reside in the SCRIPTS directory under the installation directory.

You are never required to use the Call statement when calling an Enable or DLL subroutine. 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
```

**CancelButton Statement**

Use to close a dialog when omitting changes.

Format:

`CancelButton starting-x-pos, starting-y-pos, width, height`

Related Topics: Begin Dialog, CheckBox, Dialog, DropListBox, GroupBox, ListBox, OKButton, OptionButton, OptionGroup, PushButton, Text, TextBox

Example:

```
Sub BTN_1()
Begin Dialog Dialog_1 0,0, 252, 136, "Dialog Title"
  OptionGroup .GRP_1
    OptionButton 32,48,80,12, "1st Radio - Group 1"
    OptionButton 32,64,80,12, "2nd Radio - Group 1"
  OptionGroup .GRP_2
    OptionButton 144,48,84,12, "1st Radio - Group 2"
    OptionButton 144,64,84,12, "2nd Radio - Group 2"
  OKButton 24,96,68,20
  CancelButton 156,96,52,20
  GroupBox 24,36,92,52, "Option Group 1"
  GroupBox 136,36,100,52, "Option Group 2"
  GroupBox 24,4,212,28, "Check Group"
  CheckBox 36,16,76,12, "Check_Box_1", .CHECKBOX_1
  CheckBox 124,16,76,12, "Check_Box_1", .CHECKBOX_2
End Dialog
Dim Dlg1 As Dialog_1
Dlg1.Grp_1 = 0          ' Set 1st button - group 1
Dlg1.Grp_2 = 1          ' Set 2nd button - group 2
button = Dialog ( Dlg1 )
```

```

If button = 0 Then Return
MsgBox "Grp1: " + Dlg1.Grp_1 + ", Grp2: " + Dlg1.Grp_2
Dialog Dlg1
End Sub

```

**CBool Function**

Convert expression from one data type to a Boolean expression.

Format:

**CBool** (*expression*)

The *expression* parameter 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**

Converts any valid expression to a Date variable with a vartype of 7.

Format:

**CDate** (*expression*)

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**

Convert expressions from one data type to a double.

Format:

**CDbl** (*expression*)

The *expression* parameter must be a valid string or numeric expression.

Example:

```
Sub Main ()
  Dim y As Integer
  y = 25
  If VarType(y) = 2 Then
    Print y
    x = CDbl(y)
    'Converts integer value of y to a double value in x
    Print x
  End If
End Sub
```

### ChangeCursorStyle Subroutine

Control the cursor style from within an eQuate Action script.

Format:

ChangeCursorStyle (*CursorStyle*)

The following are available cursor styles:

23) <u>Name</u>	24) <u>Value</u>	25) <u>Comment</u>
26) crDefault	27) 0	28) Usually crArrow
29) crNone	30) -1	31)
32) crArrow	33) -2	34)
35) crCross	36) -3	37)
38) crIBeam	39) -4	40)
41) crSizeNESW	42) -6	43)
44) crSizeNS	45) -7	46)
47) crSizeNWSE	48) -8	49)
50) crSizeWE	51) -9	52)
53) crUpArrow	54) -10	55)
56) crHourGlass	57) -11	58)
59) crDrag	60) -12	61)
62) crNoDrop	63) -13	64)
65) crHSplit	66) -14	67)
68) crVSplit	69) -15	70)
71) crMultiDrag	72) -16	73)
74) crSQLWait	75) -17	76)
77) crNo	78) -18	79)
80) crAppStart	81) -19	82)
83) crHelp	84) -20	85)
86) crHandPoint	87) -21	88)
89) crSizeAll	90) -22	91)

### ChDir Statement

Change the default directory.

Format:

ChDir *pathname*

*Pathname* syntax:

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

The *pathname* parameter 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: ChDrive, CurDir, Dir, Mkdir, Rmdir

Example:

```
Sub Main ()
  Dim Answer, Msg, NL      ' Declare variables.
  NL = Chr(10)             ' Define newline.
  CurPath = CurDir()       ' Get current path.
  ChDir "\"                ' Change to the root directory.
  Msg = "The current directory has been changed to "
```

```

Msg = Msg & CurDir() & NL & NL & "Press OK to change "
Msg = Msg & "back 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**

Change the current drive.

Format:

`ChDrive drivename`

The *drivename* parameter 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 "\"                 ' Change to the root directory.
  ChDrive "G:"              ' Change to G drive.
  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, Dir, Mkdir, Rmdir

**CheckBox Statement**

Use a checkbox in a dialog for selecting one or more in a series of choices.

Format:

`CheckBox starting-x-pos, starting-y-pos, width, height, "caption", .name`

Related Topics: Begin Dialog, CancelButton, Dialog, DropDownList, GroupBox, ListBox, OKButton, OptionButton, OptionGroup, PushButton, Text, TextBox

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

```

**Choose Function**

Return a value from a list of arguments.

Format:

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

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

Return a one-character string whose ASCII number is the argument.

Format:

*Chr(integer)*

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

Convert any valid expression to an integer.

Format:

*CInt(expression)*

Example:

```
Sub Main ()
    Dim y As Long

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

End Sub
```

### CLng Function

Convert any valid expression into a Long.

Format:

*CLng(expression)*

Example:

```
Sub Main ()
    Dim y As Integer

    y = 25
    If VarType(y) = 2 Then
        Print y
        x = CLng(y)
        'Converts integer value of y to a long value in x
        Print x
    End If

End Sub
```

**Close Statement**

Close active file.

Format:

Close [*filename*]

If the optional *filename* parameter is omitted, all open files will be closed.

Related Topics: EOF, Input, Line Input, Open, Print #, Write #

Example:

```
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
```

**CloseApp Subroutine (eXpress Plus)**

Close the application immediately upon exiting the current action.

Format:

CloseApp

**ComboBox Statement**

Use a combo box in a dialog to allow the user to make a selection either by typing text into the combo box or by selecting an item from its list. If there are more items in the list than will fit in the combo box, a scroll bar will appear allowing access to all items in the list.

Format:

ComboBox *starting-x-pos, starting-y-pos, width, height, listsource, .name*

Related Topics: Begin Dialog, CancelButton, CheckBox, Dialog, DropListBox, GroupBox, ListBox, OKButton, OptionButton, OptionGroup, PushButton, Text, TextBox

Example:

```
Sub BTN_3()
    Dim LISTSRC$(2)
    LISTSRC(0) = "Item 1"
    LISTSRC(1) = "Item 2"
    LISTSRC(2) = "Item 3"
    Begin Dialog DIALOG_2 16,16, 204, 96, "Test ComboBox"
        ComboBox 36,4,128,32, LISTSRC(), .COMBOBOX_1
        TextBox 36,40,128,20, .TEXTBOX_2
        OKButton 8,64,76,20
        CancelButton 108,64,72,20
    End Dialog
    Dim Dlg2 As DIALOG_2
    Dlg2.COMBOBOX_1 = "Combo List"
    button = Dialog(Dlg2)
    If button = 0 Then Return
    x = Dlg2.COMBOBOX_1
    Dlg2.TEXTBOX_2 = LISTSRC(x)
    MsgBox "Text box is set to: " + Dlg2.TEXTBOX_2
    Dialog Dlg2
End SUB
```

**Const Statement**

Assign a symbolic name to a constant value.

Format:

[Global] Const *name* = *expression*

A constant must be defined before it is used.



The Global statement must be outside the procedure section (i.e., not in a Sub or Function) of Enable. Global variables are available to all functions and subroutines.

The definition of a Const in Enable outside the procedure is global. The syntax, Global Const and Const, used below, outside the procedure, are identical.

A type declaration character may be used (see Other Data Types). If no type declaration character is used, Enable will automatically assign one of the following data types to the constant by evaluating *expression*:

Long (if *expression* evaluates to a long or integer),  
 Double (if a decimal place is present) or  
 String (if *expression* evaluates to a string).

Example:

```
Global Const GloConst = 142
Const MyConst = 122

Sub Main ()
    Dim Answer, Msg, NL
    Const PI = 3.14159
    NL = Chr(10)
    CurPath = CurDir()
    ChDir "\"
    Msg = "The current directory has been changed to "
    Msg = Msg & CurDir() & NL & NL & "Press OK to change "
    Msg = Msg & "back to your previous default directory."
    Answer = MsgBox(Msg)
    ChDir CurPath
    Msg = "Directory changed back to " & CurPath & "."
    MsgBox Msg
    Myvar = myConst + PI + GloConst
    Print MyVar
End Sub
```

### Cos Function

Return the cosine of an angle.

Format:

Cos (*expression*)

The *radian* argument must be expressed in radians and must be a valid numeric expression. Cos will, by default, return a Double unless a Single or Integer is specified as the return value.

Example:

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

### CreateObject Function

Create an OLE automation object.

Format:

CreateObject (*class*)

The *class* parameter has the following Format:

"*appname.objecttype*"

The *class* parameter has the following parts:

<u>Part</u>	<u>Description</u>
<i>appName</i>	Name of the application providing the object.
<i>Objecttype</i>	Type or class of object to create.

Example:

```

Sub BUTTON_BAR_02()
  Dim word6 as object
  Set word6 = CreateObject("Word.Basic")
  word6.AppShow
  word6.FileNewDefault
  word6.ViewPage
  word6.InsertPara
  word6.Insert DF_CUST_NAME + Chr$(13) + DF_ADDRESS1 + Chr$(13)
  if Left$(DF_ADDRESS2,1) <> " " then
    word6.Insert DF_ADDRESS2 + Chr$(13)
  end if
  if Left$(DF_ADDRESS3,1) <> " " Then
    word6.Insert DF_ADDRESS3 + Chr$(13)
  end if
  word6.insert DF_CITY + ", " + DF_STATE + " " + DF_ZIP
  word6.InsertPara
  word6.InsertPara
  word6.Insert "Dear Valued Customer:"
  word6.InsertPara
  word6.InsertPara
  word6.Insert "The following is your account number and "
  word6.Insert "telephone number as stored in our database:"
  word6.InsertPara
  word6.Bold 1
  word6.Insert "Account:" + Chr$(9) + DF_ACCOUNT + Chr$(13)
  word6.insert "Phone number:" + Chr$(9) + DF_PHONE
  word6.bold 0
  word6.InsertPara
  word6.Insert "If either of the above is incorrect, please "
  word6.Insert "contact us immediately."
  word6.InsertPara
  word6.InsertPara
  word6.Insert "Sincerely,"
  word6.InsertPara
  word6.Insert " John J. Doe"
  word6.Insert " Customer Services Rep."
  word6.InsertPara
  MsgBox "Your letter is ready to print. Make adjustments " _
    & "in Word and print before pressing the OK button below."
End Sub

```

### CSng Function

Convert any valid expression to a Single.

Format:

CSng (*expression*)

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

```

End Sub

### CStr Function

Convert any valid expression to a String.

Format:

CStr (*expression*)

Use CStr to force the result to be expressed as a String. Use the CStr function instead of Str to provide internationally aware conversions from any other data type to a String; e.g., different decimal separators are properly recognized depending on the local setting of your system.

The data in *expression* determines what is returned according to the following:

<u>If <i>expression</i> is</u>	<u>CStr returns</u>
Boolean	A string containing true (-1) or false (0).
Other numeric	A string containing the number.

Related Topics: Str

### CurDir Function

Return the current path for the specified drive.

Format:

CurDir (*drive*)

CurDir returns a Variant.

Related Topics: ChDir, ChDrive, Dir, Mkdir, Rmdir

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

Convert any valid expression to a Variant.

Format:

CVar (*expression*)

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

Return the current system date.

Format:

Date  
or  
Date()

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

Related Topics: Day, Format, Hour, Minute, Month, Now, Second, Weekday, Year

Examples:

```
x = Date()
```

```

Print Date
Print x
Print VarType(Date)

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

' 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"

```

**DateSerial Function**

Return a variant (Date) corresponding to the year, month and day specified.

Format:

```
DateSerial (year, month, day)
```

All three parameters for the DateSerial Function are required and must be valid.

Related Topics: DateValue, Day, Month, TimeSerial, TimeValue, Year

Example:

```

Sub Main

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

End Sub

```

**DateValue Function**

Return a variant (Date) corresponding to the year, month and day specified.

Format:

```
DateValue (dateexpression)
```

The *dateexpression* parameter can be a string or any expression that can represent a date, time or both a date and a time.

Related Topics: DateSerial, Day, Month, TimeSerial, TimeValue, Year

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**

Return an integer between 1 and 31 that is the portion of the *date* parameter representing the day of the month.

Format:

```
Day (date)
```

The *date* parameter is any string expression.

The returned integer represents the day of the *date* parameter.

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

Related Topics: Date, Format, Hour, Minute, Month, Now, Second, Weekday, Year

Example:

```

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

```

```
End Sub
```

### Declare Statement

Refer to an external procedure in a Dynamic Linking Library (DLL).

Formats:

```
Declare Sub procedure Lib "library" [Alias "aliasname"] [(argumentlist)]
```

or

```
Declare Function procedure Lib "library" [Alias "aliasname"] [(argumentlist)] [As type]
```

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

Supply the name of the DLL that contains the *procedure* on the *library* parameter.

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 *argumentlist* 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).

The optional *As type* parameter is used to specify the data type. Valid types are String, Integer, Single, Double, Long and Variant (see Other Data Types).

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

```
Declare Sub OneTime Lib "Check" ()
```

The following syntax is an Enable extension to the Declare statement but is not supported by Microsoft Visual Basic.

```
Declare Function procedure App [Alias "aliasname"] [(argumentlist)] [As type]
```

This form of the Declare statement refers 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 Function

Return a value corresponding to the button the user chooses.

Format:

```
Dialog [(dialogrecord)]
```

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

The return value or button:

<u>Value</u>	<u>Description</u>
-1	OK button.

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

Related Topics: Begin Dialog, CancelButton, CheckBox, DropDownListBox, GroupBox, ListBox, OKButton, OptionButton, OptionGroup, PushButton, Text, TextBox

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$(2)
  MyList(0) = "Banana"
  MyList(1) = "Orange"
  MyList(2) = "Apple"
Begin Dialog DIALOGNAME1 59,111, 240, 147, "Test Dialog"
  OptionGroup .GRP1
    OptionButton 42,56,48,12, "Option&1"
    OptionButton 42,66,48,12, "Option&2"
  OptionGroup .GRP2
    OptionButton 42,92,48,12, "Option&3"
    OptionButton 42,102,48,12, "Option&4"
  GroupBox 132,81,70,36, "Group"
  Text 10,10,28,12, "Name:"
  TextBox 40,10,50,12, .joe
  ListBox 102,10,108,16, MyList$, .MyList1
  DropDownListBox 42,32,108,36, MyList$, .DropList1$
  CheckBox 142,56,48,12, "Check&A", .Check1
  CheckBox 142,66,48,12, "Check&B", .Check2
  CheckBox 142,92,48,12, "Check&C", .Check3
  CheckBox 142,102,48,12, "Check&D", .Check4
  CancelButton 42,124,40,12
  OKButton 90,124,40,12
  PushButton 140,124,40,12, "&Push Me 1"
  PushButton 190,124,40,12, "Push &Me 2"
End Dialog
Dim Dlg1 As DialogName1
Dlg1.joe = "Def String"
Dlg1.MyList1 = 1
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 return val
If button = 0 Then Return
MsgBox "TextBox: " & Dlg1.joe
MsgBox "ListBox: " & Dlg1.MyList1
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.MyList1
```

```
End If
End Sub
```

### Dim Statement

Allocate storage for, and declare the data type of, variables and arrays in a module.

Format:

```
Dim variablename[(subscripts)] [As type][,name] [As type]
```

The types currently supported are Integer, Long, Single, Double and String.

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
```

### Dir Function

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

Format:

```
Dir [(path, attributes)]
```

The *path* parameter is a string expression that contains a file name. The file name may include drive and directory specifications. In addition, standard wildcard characters, asterisk (\*) and question mark (?), may be included.

The Dir function returns the first file name that matches the path specified on a previous Dir function. A Dir function with no parameters specified will return the next file name that matches the path specification. When no more files meet the specification, an empty string is returned.

The *attributes* parameter is a numeric expression containing a number equal to the sum of all required attributes. The attributes are:

<u>Value</u>	<u>Attribute</u>
0	Normal
2	Hidden
4	System file
8	Volume label
16	Directory

Related Topics: ChDir, ChDrive, CurDir, MkDir, RmDir

### DlgEnable Statement

Enable or disable a particular control on a dialog box.

Format:

```
DlgEnable "controlname", value
```

The *controlname* parameter is the name of the control on the dialog box. The *value* parameter is the value to which it is set: 1 = Enable, 0 = Disable. "On" is equal to 1 in the example below. If the *value* parameter is omitted, the status of the control toggles.

Related Topics: DlgText, DlgVisible

Example:

```

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",. chl
    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

```

### DlgText Statement

Set or change the text of a dialog control.



Format:

`DlgText "controlname", string`

The *controlname* parameter is the name of the control on the dialog box. The *string* parameter is the value to which it is set.

Related Topics: DlgEnable, DlgVisible

Example:

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

### DlgVisible Statement

Hide or make visible a particular control on a dialog box.

Format:

`DlgVisible "controlname", value`

The *controlname* parameter is the name of the control on the dialog box. The *value* parameter is the value to which it is set: 1 = Visible, 0 = Hidden. "On" is equal to 1. If the *value* parameter is omitted, the status of the control toggles.

Related Topics: DlgEnable, DlgText

Example:

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

See the DlgEnable Statement for a complete example.

### Do...Loop Statement

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

Formats:

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

The *condition* parameter may be a numeric or string expression.

Related Topics: While..Wend, With

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
```

### DropListBox Statement

Use a drop-down list box in a dialog to allow the user to make a selection from a drop-down list. The drop-down list is useful when you wish to conserve space on the dialog.

Format:

`DropListBox starting-x-pos, starting-y-pos, width, height, listsource, .name`

Related Topics: Begin Dialog, CancelButton, CheckBox, Dialog, GroupBox, ListBox, OKButton, OptionButton, OptionGroup, PushButton, Text, TextBox

Example:

```

Sub BTN_3()
Dim LISTSRC$(2)
LISTSRC(0) = "Item 1"
LISTSRC(1) = "Item 2"
LISTSRC(2) = "Item 3"
Begin Dialog DIALOG_2 0,0, 204, 96, "Test DropListBox"
  DropListBox 36,12,128,44, LISTSRC$, .DROPLISTBOX_1
  TextBox 36,36,128,20, .TEXTBOX_2
  OKButton 8,64,76,20
  CancelButton 108,64,72,20
End Dialog
Dim Dlg2 As DIALOG_2
Dlg2.DROPLISTBOX_1 = 0
button = Dialog(Dlg2)
If button = 0 Then Return
x = Dlg2.DROPLISTBOX_1
Dlg2.TEXTBOX_2 = LISTSRC(x)
MsgBox "Text box is set to: " + Dlg2.TEXTBOX_2
Dialog Dlg2
End SUB

```

### End Statement

End a program or a block of statements such as a Sub procedure or a Function.

Format:

End [Dialog | Function | If | Sub | Select | Type | With]

The End statement, without any parameters, may be used anywhere within a procedure to close files opened with the Open statement and to clear variables. To suspend execution without clearing variables or closing files, use the Stop statement.

Related Topics: Dialog, Do...Loop, Exit, Function, If...Then...Else, Select Case, Stop, Sub, Type, With

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

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

Format:

EOF (*filenumber*)

Related Topics: Close, Input, Line Input, Open, Print #, Write #

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**

Reinitialize the elements of a fixed array.

Format:

Erase *arrayname*[,*arrayname*] ...

Related Topics: Dim

Example:

```

' This example demonstrates some of the features of
' arrays. The lower bound for an array is 0 unless
' option base has been set as 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
  Erase( a ) ' If this line is uncommented,
             ' then the values will all be 0
  Print a(0),a(1),a(2), a(3)
End Sub

```

**Exit Statement**

Exit a loop or procedure.

Format:

Exit {Do | For | Function | Sub}

Related Topics: End, Stop

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 Function**

Return *e* to a power (*e* ^ *number*).

Format:

Exp (*number*)

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**

Copy a file from source to destination.

Format:

`FileCopy (sourcefile, destinationfile)`

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

**FileLen Function**

Return a Long integer that is the length of the file in bytes.

Format:

`FileLen(filename)`

Example:

```

Sub Main

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

End Sub

```

**FileOpenDialog Function (eXpress Plus)**

Open the File Open dialog.

Format:

`FileOpenDialog (initialdirectory, filename, defaultextension, filter, title)`

The *initialdirectory* is any string expression containing the initial directory to use when the dialog is opened.

The *filename* parameter is any string expression containing the file name to be displayed initially when the dialog is opened (usually left blank).

The *defaultextension* parameter is any string expression containing the default file extension to use if one is not supplied by the user.

The *filter* parameter is any string expression containing the filter or filters used to restrict file selection. A *filter* is setup as follows:

*file-type-description-1|file-filter-1|file-type-description-2|filter-2|...file-type-description-n|filter-n*

For example, to show all text files or all files use the following filter string:

"Text files(\*.txt)|\*.txt|All files(\*)|\*.\*"

"Text files(\*.txt)" is the file-type-description and the second "\*.\*" is the *file-filter*.

The *title* parameter is any string expression containing the title to display in the file dialog.

This function returns the full file name, including path, of the selected file. If a file is not selected, the dialog is cancelled, and an empty string is returned.

Related topics: FileSaveDialog Function, ImageOpenDialog Function, ImageSaveDialog Function

**FileSaveDialog Function (eXpress Plus)**

Open the File Save dialog.

Format:

`FileSaveDialog (initialdirectory, filename, defaultextension, filter, title)`

The *initialdirectory* is any string expression containing the initial directory to use when the dialog is opened.

The *filename* parameter is any string expression containing the file name to be displayed initially when the dialog is opened (usually left blank).

The *defaultextension* parameter is any string expression containing the default file extension to use if one is not supplied by the user.

The *filter* parameter is any string expression containing the filter or filters used to restrict file selection. A *filter* is setup as follows:

*file-type-description-1|file-filter-1|file-type-description-2|filter-2|...file-type-description-n|filter-n*

For example, to show all text files or all files use the following filter string:

"Text files(\*.txt)\*.txt|All files(\*.\*)\*.\*"

"Text files(\*.txt)" is the file-type-description and the second "\*.txt" is the *file-filter*.

The *title* parameter is any string expression containing the title to display in the file dialog.

This function returns the full file name, including path, of the selected file. If a file is not selected, the dialog is cancelled, and an empty string is returned.

Related Topics: FileOpenDialog Function, ImageOpenDialog Function, ImageSaveDialog Function

### Fix Function

Return the integer portion of a number.

Format:

Fix (*number*)

Related Topics: Int

### For Each...Next Statement

Repeat the group of statements for each element in an array or collection.

For Each *element* in *group*

[ *statement(s)* ]

[Exit For]

[ *statement(s)* ]

Next [*element*]

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

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

Format:

For *counter* = *expression1* To *expression2* [Step *increment*]

[*statement(s)*]

Next [*counter*]

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 a string, number or variant (date/time) data type to a format expression.

Format:

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

Format returns a string.

The Format has two parts:

<u>Part</u>	<u>Description</u>
<i>expression</i>	Expression to be formatted.
<i>fmt</i>	A string of characters that specify how the expression is to be 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>fmt</i> parameter.

If the *fmt* parameter is omitted or is zero-length and the *expression* parameter is a numeric, Format provides the same functionality as the Str function by converting the numeric value to the appropriate return data type. Positive numbers converted 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:

<u>Format Name</u>	<u>Description</u>
General Number	Display the number as is – without thousand separators.
Fixed	Display at least one digit to the left and two digits to the right of the decimal separator.
Standard	Display the number with thousand separators, if appropriate; display two digits to the right of the decimal separator.
Percent	Display the number multiplied by 100 with a percent sign (%) appended to the right; display two digits to the right of the decimal separator.
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:

<u>Character</u>	<u>Meaning</u>
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 format expression's decimal separator, 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.
%	Percentage placeholder. The percent character (%) is inserted in the position where it appears in the format string. The expression is multiplied by 100.
,	Thousand separators. The thousands 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 format and E or e is inserted between the number and its

<u>Character</u>	<u>Meaning</u>
	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 output depends on Date Format specified in the International section of the Control Panel.
- \$ ( ) space	Display a literal character. To display a character other than one of those listed, precede it with a backslash (\).
\	Display the next character in the format string. The backslash itself is not displayed. To display a backslash, use two backslashes (\\). Examples of characters that can't be displayed as literal characters are the date- and time-formatting characters (a, c, d, h, m, n, p, q, s, t, w, y and /:), the numeric-formatting characters (#, 0, %, E, e, comma, and period), and the string-formatting characters (@, &, <, >, and !).
"String"	Display the string inside the double quotation marks. To include a string in <i>fmt</i> from within 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. Each section must be separated by semicolons.

<u>If you use</u>	<u>The result is</u>
One section only	The format expression applies to all values.
Two	The first section applies to positive values, the second to negative values.
Three	The first section applies to positive values, the second to negative values, and the third to zeros.
Four	The first section applies to positive values, the second to negative 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\ev\r\o"

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

<u>Format (fmt)</u>	<u>Positive 3</u>	<u>Negative 3</u>	<u>Decimal 3</u>	<u>Null</u>
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 formats that have been predefined in Enable, or create user-defined date and time formats using standard characters that have special meaning when used in a format expression.

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

<u>Format Name</u>	<u>Description</u>
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 Date	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 minutes and the 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.

<u>Character</u>	<u>Meaning</u>
c	Display the date as dddd and display the time as tttt, in that 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 the day as a full name (Sunday-Saturday).
w	Display the day of the week as a number (1 for Sunday through 7 for Saturday).
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 (0000-9999).
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.
A/P	Use the 12-hour clock display a uppercase A/P.



<u>Character</u>	<u>Meaning</u>
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 23:59 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:

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

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

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

<u>Character</u>	<u>Meaning</u>
@	Character placeholder. Displays a character or a space. Placeholders are filled from right to left unless there is an exclamation 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

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 = "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 AMPM") ' Returns "05:04:23 PM".
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**

Return the next valid unused file number.

Format:

```
FreeFile [()]
```

Example:

```

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

```

**Function Statement**

Declare and define a procedure that can receive arguments and return a value of a specified data type.

Format:

```

Function functionname [(argumentlist)] [As type]
    [statement(s)]
        functionname = expression
[Exit Function]
    [statement(s)]
        functionname = expression

```

End Function

When the optional *argumentlist* 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).

The optional As *type* parameter is used to specify the data type. Valid types are String, Integer, Single, Double, Long and Variant (see Other Data Types).

Related Topics: Dim, End, Exit, Sub

Example:

```
Sub Main
  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

Read from a disk file into a variable.

Format:

Get [#] *filename*, [*recordnumber*,] *variablename*

The Get statement has three parts:

Parameter	Description
<i>filename</i>	The number used to open the file.
<i>recordnumber</i>	For files opened in Binary mode, <i>recordnumber</i> is the byte position where reading starts.
<i>variablename</i>	The name of the variable used to receive the data from the file.

Related Topics: Open, Put

### GetColor Function (eXpress Plus)

Retrieve either the foreground or the background color of a control.

Format:

GetColor (*name*, *colortype*)

This function returns a Long integer.

The *name* parameter is any string expression containing the name of a valid control.

The *colortype* parameter specifies either foreground or background color. The *colortype* parameter may be specified as an Integer or Constant:

Color	Integer	Constant
Foreground color	0	tpForeColor
Background color	1	tpBackColor

Note: Since variables do not have to be declared prior to first reference, use an **Option Explicit** statement (external to the procedure) when using constants to assure that they are spelled correctly.

Related Topics: SetColor

### GetNumericProp Function (eXpress Plus)

Retrieve the current value for the specified numeric property. Numeric properties include Height, Width, Left, Top, TabOrder, etc.

Format:

`GetNumericProp(name, property)`

This function returns an Integer.

The *name* parameter is any string expression containing the name of the control. The *property* parameter is any string expression containing the name of the property.

Related Topics: GetStringProp, SetNumericProp, SetStringProp

**GetObject Function**

Retrieve an OLE Automation object from a file.

Format:

`GetObject ("filename" [, "class"])`

The GetObject function has these parts:

<u>Part</u>	<u>Description</u>
<i>filename</i>	Full path and name of file containing the object. If filename is an empty string (""), class is required, and the function returns the currently active object of the specified type.
<i>class</i>	String representing the class of the object.

The optional *class* parameter has the following Format:

`"appname.objecttype"`

The *class* parameter has the following parts:

<u>Part</u>	<u>Description</u>
<i>appname</i>	Name of the application providing the object.
<i>objecttype</i>	Type or class of object to get.

If the optional *class* is not supplied, the OLE2 DLLs determine the application based on the filename provided. Use the optional *class* parameter when the file supports more than one class.

Example:

```
Dim WordObject as Object
Set WordObject = GetObject ("C:\WINWORD\LETTERS\OLETST.DOC")
```

**GetState Function (eXpress Plus)**

Retrieve the specified state of controls. The value returned will be either True (1) or False (0).

Format:

`GetState (name, statetype)`

Returns an Integer.

The *name* parameter is any string expression containing the name of a valid control.

The *statetype* parameter is the property type for which the state is to be retrieved.

The *statetype* parameter may be expressed as an Integer or a Constant:

<u>Effect</u>	<u>Integer</u>	<u>Constant</u>
Enabled	0	tpEnabled
Visible*	1	tpVisible
Checked**	2	tpChecked

\* Visible does not apply to user menu items.

\*\* Checked only applies to Option Button and Check Box controls.

Note: Since variables do not have to be declared prior to first reference, use an **Option Explicit** statement (external to the procedure) when using constants to assure that they are spelled correctly.

Related Topics: SetState

Example:

```
' If ACCOUNT is enabled, disabled it.
If GetState("ACCOUNT", tpEnabled) Then
    SetState "ACCOUNT", tpEnabled, False
End If
```

### GetString Function (eXpress Plus)

Retrieve a string value from a control. For controls like buttons and text labels, the string returned is the caption. For edit boxes, the string returned is the actual data value.

Format:

GetString (*name*)

This function returns a String.

The *name* parameter is any string expression containing the name of a valid control or global variable.

### GetStringProp Function (eXpress Plus)

Retrieve the current value for the specified string property. String properties include Hints, Captions, Pictures, etc.

Format:

GetStringProp (*name, property*)

This function returns a String.

The *name* parameter is any string expression containing the name of the control. The *property* parameter is any string expression containing the name of the property.

Related Topics: GetNumericProp, SetNumericProp, SetStringProp

### Global Statement

Declare a global variable and allocate storage space.

Format:

[Global] Const *constant*

A constant must be defined before it is used.

The Global statement must be outside the procedure section (i.e., not in a Sub or Function) of Enable. Global variables are available to all functions and subroutines in your program.

The definition of a Const in Enable, outside the procedure, is global. The syntax, Global Const and Const (used below, outside the Sub) are identical.

A type declaration character may be used (see Other Data Types). If no type declaration character is used, Enable will automatically assign one of the following data types to the constant by evaluating *expression*:

Long (if *expression* evaluates to a long or integer),  
 Double (if a decimal place is present) or  
 String (if *expression* evaluates to a string).

Related Topics: Const, Dim, Type

Example:

```
Global Const GloConst = 142
Const MyConst = 122      ' Global to all procedures in a module
Sub Main ()
    Dim Answer, Msg, N    ' Declare variables
    Const PI = 3.14159
    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 "
Msg = Msg & "back 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
myvar =myConst + PI + GloConst
Print MyVar
End Sub

```

### GroupBox Statement

Use a group box in a dialog to logically group controls.

Format:

*GroupBox starting-x-pos, starting-y-pos, width, height, "caption"*

Related Topics: Begin Dialog, CancelButton, CheckBox, Dialog, DropDownList, ListBox, OKButton, OptionButton, OptionGroup, PushButton, Text, TextBox

Example:

```

GroupBox 24,4,212,28, "Check Group"
CheckBox 36,16,76,12, "Check_Box_1", .CHECKBOX_1
CheckBox 124,16,76,12, "Check_Box_1", .CHECKBOX_2

```

### GoTo Statement

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

Format:

*GoTo linelabel*

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 Labell
                End If
            Next z
        Next y
    Next x
    Labell:

End Sub

```

### Hex Function

Return the hexadecimal value of a decimal parameter.

Format:

*Hex(number)*

Hex returns a string.

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

Related Topics: 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

Return an integer between 0 and 23 that is the portion of the *time* parameter representing the hour of the day.

Format:

Hour(*time*)

The *time* parameter is any string expression that can represent a time.

Related Topics: Date, Day, Format, Minute, Month, Now, Second, Weekday, Year

Exxample:

```
MyTime = "08:04:23 PM"
MsgBox MyTime

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

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

Allow conditional statements to be executed in the code.

Formats:

```
If condition Then
    [statement(s)]
[Elseif condition Then
    [statement(s)]]
[Else
    [statement(s)]]
End If

or

If condition Then statement [Else statement]
```

Related Topics: Select Case

Example:

```
Sub IfTest
    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
```

### ImageOpenDialog Function (eXpress Plus)

Open the Open Picture Image dialog. ImageOpenDialog differs from the File Open dialog in that it includes an image preview window. In addition, the filter is only applied in the Image Open dialog when it contains something, because default filters (\*.jpg, \*.bmp, etc.) are already present (not in file dialogs).

Format:

ImageOpenDialog (*initialdirectory*, *filename*, *defaultextension*, *filter*, *title*)

The *initialdirectory* is any string expression containing the initial directory to use when the dialog is opened.

The *filename* parameter is any string expression containing the file name to be displayed initially when the dialog is opened (usually left blank).

The *defaultextension* parameter is any string expression containing the default file extension to use if one is not supplied by the user.

The *filter* parameter is any string expression containing the filter or filters used to restrict file selection. A *filter* is setup as follows:

*file-type-description-1|file-filter-1|file-type-description-2|filter-2|...file-type-description-n|filter-n*

For example, to show all text files or all files use the following filter string:

"Text files(\*.txt)|\*.txt|All files(\*.\*)\*.\*"

"Text files(\*.txt)" is the file-type-description and the second "\*.txt" is the *file-filter*.

The *title* parameter is any string expression containing the title to display in the file dialog.

This function returns the full file name, including path, of the selected file. If a file is not selected, the dialog is cancelled, and an empty string is returned.

Related topics: FileOpenDialog Function, FileSaveDialog Function, ImageSaveDialog Function

### ImageSaveDialog Function (eXpress Plus)

Open the File Save dialog. ImageSaveDialog differs from the File Save dialog in that it includes an image preview window. In addition, the filter is only applied in the Image Save dialog when it contains something, because default filters (\*.jpg, \*.bmp, etc.) are already present (not in file dialogs).

FileSaveDialog (*initialdirectory*, *filename*, *defaultextension*, *filter*, *title*)

The *initialdirectory* is any string expression containing the initial directory to use when the dialog is opened.

The *filename* parameter is any string expression containing the file name to be displayed initially when the dialog is opened (usually left blank).

The *defaultextension* parameter is any string expression containing the default file extension to use if one is not supplied by the user.

The *filter* parameter is any string expression containing the filter or filters used to restrict file selection. A *filter* is setup as follows:

*file-type-description-1|file-filter-1|file-type-description-2|filter-2|...file-type-description-n|filter-n*

For example, to show all text files or all files use the following filter string:

"Text files(\*.txt)|\*.txt|All files(\*.\*)\*.\*"

"Text files(\*.txt)" is the file-type-description and the second "\*.txt" is the *file-filter*.

The *title* parameter is any string expression containing the title to display in the file dialog.

This function returns the full file name, including path, of the selected file. If a file is not selected, the dialog is cancelled, and an empty string is returned.

Related topics: FileOpenDialog Function, FileSaveDialog Function, ImageOpenDialog Function

### Input Function

Return characters from a sequential file.

Format:

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

The Input function has two parameters: *n* and *filenumber*. The *n* parameter 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.

Related Topics: Close, EOF, Line Input, Open, Print #, Write #



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
```

### InputDialog Function

Display a prompt in a dialog box.

Format:

```
InputDialog (prompt[[title][,default][,x-pos,y-pos]])
```

InputDialog returns a String.

The Input function has these parts:

<u>Part</u>	<u>Description</u>
<i>prompt</i>	String expression displayed as the message in the dialog box.
<i>title</i>	String expression displayed in the title bar of the dialog.
<i>default</i>	String expression displayed in the textbox as the default response if no other input is provided.
<i>x-pos</i>	Numeric expression that specifies, in twips, the horizontal distance of the left edge of the dialog from the left edge of the screen. If omitted, <i>y-pos</i> must also be omitted. If <i>x-pos</i> and <i>y-pos</i> are omitted, the dialog box is horizontally centered and vertically positioned approximately one-third the way down the screen.
<i>y-pos</i>	Numeric expression that specifies, in twips, the vertical distance of the upper edge of the dialog from the top edge of the screen.

Note: A twip is 1/20 of a printer's point (1,440 twips equal an inch and 567 twips equal a centimeter).

Example:

```
MyTime = InputBox$ ("Enter Time <hh:mm:ss>." + Chr(13) + _
  Chr(10) + "(Not military time).", "Time Entry")
MyDate = InputBox$ ("Enter Date.", "Date Entry")
MsgBox Format(MyDate, "Long Date") + " at " + _
  Format(MyTime, "Long Time")
```

### InputDialog Function

Display a prompt in a dialog box.

Format:

```
InputDialog (prompt[[title][,default][,x-pos,y-pos]])
```

InputDialog returns a String.

The Input function has these parts:

<u>Part</u>	<u>Description</u>
<i>prompt</i>	String expression displayed as the message in the dialog box.
<i>title</i>	String expression displayed in the title bar of the dialog.
<i>default</i>	String expression displayed in the textbox as the default response if no other input is provided.
<i>x-pos</i>	Numeric expression that specifies, in twips, the horizontal distance of the left edge of the dialog from the left edge of the screen. If omitted, <i>y-pos</i> must also be omitted. If <i>x-pos</i> and <i>y-pos</i> are omitted, the dialog box is horizontally centered and vertically positioned approximately one-third the way down the screen.
<i>y-pos</i>	Numeric expression that specifies, in twips, the vertical distance of the upper edge of the dialog from the top edge of the screen.

Note: A twip is 1/20 of a printer's point (1,440 twips equal an inch and 567 twips equal a centimeter).

Example:

```
MyTime = InputBox$ ("Enter Time <hh:mm:ss>." + Chr(13) + _
  Chr(10) + "(Not military time).", "Time Entry")
MyDate = InputBox$ ("Enter Date.", "Date Entry")
```

```
MsgBox Format(MyDate, "Long Date") + " at " + _
    Format(MyTime, "Long Time")
```

**Int Function**

Return the integer portion of a number.

Format:

```
Int (number)
```

Related Topics: Fix

**IsArray Function**

Return a Boolean value True or False indicating whether the *variablename* parameter is an array.

Format:

```
IsArray (variablename)
```

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 Function**

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

Format:

```
IsDate (variant)
```

The IsDate function returns True if the variant can be converted to a date; otherwise, it returns False.

Related Topics: IsEmpty, IsNull, IsNumeric, VarType

Example:

```
If IsDate(inputvar)
    MsgBox Format(inputvar, "Long Date")
Else
    MsgBox "Input not a valid date."
EndIf
```

**IsEmpty Function**

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

Format:

```
IsEmpty (variant)
```

The IsEmpty function returns True if the variant is empty; otherwise, it returns False.

Related Topics: IsDate, IsNull, IsNumeric, VarType

Example:

```
Sub BTN_5()
    Dim new, answer
    If IsEmpty(new) Then
        answer = MsgBox ("Start at the beginning?", 36)
        If answer = 6 Then
            new = 1
            MsgBox "Starting at the beginning now."
        End If
    End If
End Sub
```

**IsNull Function**

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

Format:

```
IsNull (variant)
```

IsNull returns a True if *variant* contains NULL; otherwise, it returns False.

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

Related Topics: IsDate, IsEmpty, IsNumeric, VarType

### IsNumeric Function

Return a value that indicates if a variant contains a numeric value.

Format:

IsNumeric (*variant*)

IsNumeric returns a True if *variant* is recognized as a number; otherwise, it returns False.

The *variant* parameter 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 or a letter:")
    If IsNumeric(TestVar) Then ' Evaluate variable
        MsgBox "Entered data is numeric." ' Message if number
    Else
        MsgBox "Entered data is not numeric." ' Message if not
    End If
End Sub
```

### IsObject Function

Return a Boolean value True or False indicating whether the *objectname* parameter is an object.

Format:

IsObject (*objectname*)

Related Topics: IsEmpty, IsNumeric, VarType, IsArray

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

Delete files from a disk. Kill will only delete files. To remove a directory, use the Rmdir Statement.

Format:

Kill *filename*

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 "
    Msg = Msg & "disk. You may see them by switching tasks."
    Msg = Msg & " Choose OK to remove the test files."
```

```

MsgBox Msg
For I = 1 To NumberOfFiles
    Kill "TEST" & I      ' Remove data files from disk.
Next I
End Sub

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

```

**LBound Function**

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

Format:

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

Related Topics: Dim, Global, Option Base, Static, UBound

Example:

```

' This example demonstrates some of the features of
' arrays. The lower bound for an array is 0 unless it is
' specified or an Option Base is set as 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**

Return a string in which all *string* parameter letters have been converted to lower case.

Format:

**LCase** (*string*)

Related Topics: UCase

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 Function

Return the left most *number* of characters of a string parameter.

Format:

**Left** (*string*, *number*)

The *string* parameter is the string expression from which the leftmost characters are returned.

The *number* parameter is the numeric expression indicating the number of characters that will be returned.

Related Topics: Len, Mid, Right

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.
        RWord = Right(UsrInp, Len(UsrInp) - SpcPos)
                                           ' Get right word.
        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 Function

Return the number of characters in a string.

Format:

**Len** (*string*)

Related Topics: InStr

Example:

```

Sub Main ()
    A$ = "Enable"
    StrLen% = Len(A$)    ' the value of StrLen is 6
    MsgBox StrLen%
End Sub

```

### Let Statement

Assign a value to a variable.

Format:

**[Let]** *variablename* = *expression*

The Let keyword is optional and only rarely used. The Let keyword 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**

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

Format:

Line Input # *filename*, *name*

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

Related Topics: Close, Input, EOF, Open, Print #, Write #

Example:

```
Sub FORM_INITIAL_ACTION()
    ' Load current part descriptions into list box from a file.
    dim fl as integer
    dim dta as string

    fl = FreeFile
    Open "H:\MSQDTA\PARTS.TXT" for Input as fl
    ListClear "LST_PARTDESC"
    While not Eof(fl)
        Line Input #fl, dta
        ListItemAdd "LST_PARTDESC", dta
    WEnd
    Close fl
End SUB
```

**ListBox Statement**

Use a list box in a dialog to allow the user to make a selection from a list box. If there are more items in the list than will fit in the list box, a scroll bar will appear allowing access to all items in the list.

Format:

ListBox *starting-x-pos*, *starting-y-pos*, *width*, *height*, *listsource*, *.name*

Related Topics: Begin Dialog, CancelButton, CheckBox, Dialog, DropListBox, GroupBox, OKButton, OptionButton, OptionGroup, PushButton, Text, TextBox

Example:

```
Sub BTN_3()
    Dim LISTSRC$(2)
    LISTSRC(0) = "Item 1"
    LISTSRC(1) = "Item 2"
    LISTSRC(2) = "Item 3"
    Begin Dialog DIALOG_2 0,0, 204, 96, "Test ListBox"
        ListBox 36,8,128,16, LISTSRC$, .LISTBOX_1
        TextBox 36,36,128,20, .TEXTBOX_2
        OKButton 8,64,76,20
        CancelButton 108,64,72,20
    End Dialog
    Dim Dlg2 As DIALOG_2
    Dlg2.LISTBOX_1 = 1
    button = Dialog(Dlg2)
    If button = 0 Then Return
    x = Dlg2.LISTBOX_1
    Dlg2.TEXTBOX_2 = LISTSRC(x)
    MsgBox "Text box is set to: " + Dlg2.TEXTBOX_2
    Dialog Dlg2
End SUB
```

**ListClear Subroutine (eXpress Plus)**

Clear or removes all items from any type of list box.

Format:

ListClear *name*

The *name* parameter is any string expression containing the name of a list box.

Example:

(See ListItemAdd)

**ListColHeader Subroutine (eXpress Plus)**

Set the column header value for a specified column in a multi-column list box. Columns are numbered from left to right starting with zero (0).

Format:

`ListColHeader name, column, headertext`

The *name* parameter is any string expression containing the name of a multi-column list box. The *column* parameter is any integer expression representing a column relative to 0. The *headertext* parameter is any string expression.

Related Topics: ListCount, ListGetColText, ListGetIndex, ListItemAdd, ListItemRemove, ListSetColText, ListSetIndex

Example:

(see ListItemAdd)

**ListCount Function (eXpress Plus)**

Retrieve the item count for a standard or Drop-down List Box.

Format:

`ListCount (name)`

The *name* parameter is any string expression containing the name of a list box.

This function returns an Integer.

Related Topics: ListColHeader, ListGetColText, ListGetIndex, ListItemAdd, ListItemRemove, ListSetColText, ListSetIndex

Example:

(see ListGetColText)

**ListGetColText Function (eXpress Plus)**

Retrieve the text in a specified column from the currently selected item of a multi-column list box. Columns are numbered from left to right starting with zero (0).

Format:

`ListGetColText (name, column)`

This function returns a String.

The *name* parameter is any string expression containing the name of a multi-column list box. The *column* parameter is any integer expression representing a column relative to zero (0).

Related Topics: ListColHeader, ListCount, ListGetIndex, ListItemAdd, ListItemRemove, ListSetColText, ListSetIndex

Example:

```
' Get account from the second column of the selected row.
If ListCount("CUST_LIST") > 0 then
    ' Make sure list contains something.
    MsgBox "Selected account:" + ListGetColText("CUST_LIST", 1)
End If
```

**ListGetIndex Function (eXpress Plus)**

Retrieve the index of the currently selected item of a standard or drop-down list box.

Format:

`ListGetIndex (name)`

This function returns an Integer.

The *name* parameter is any string expression containing the name of a list box.

The index can be in the range -1 to the value of ListCount - 1. A returned value of -1 indicates the list is either empty or nothing is currently selected.

Related Topics: ListColHeader, ListCount, ListGetColText, ListItemAdd, ListItemRemove, ListSetColText, ListSetIndex

**ListGetItem Function (eXpress Plus)**

Get a row of data from the specified list box. Note: This function may not be used with multi-column list boxes (see ListGetColText).

Format:

`ListGetItem (name, index)`

This function returns a String.

The *name* parameter is any string expression containing the name of a list box or drop-down list box. The *index* parameter is any integer expression representing the row relative to zero (0).

Related Topics: ListColHeader, ListCount, ListGetColText, ListGetIndex, ListItemAdd, ListItemRemove, ListSetColText, ListSetIndex, ListSetItem

#### ListItemAdd Subroutine (eXpress Plus)

Add an item to a standard, drop-down or multi-column list box.

Format:

```
ListItemAdd name, item1 [, item2 ... , itemn]
```

The *name* parameter is any string expression containing the name of a list box. An *item* parameter is any string expression.

To add a line to a multi-column list, use multiple *item* parameters separated by commas. Each item is put into the corresponding column from left to right.

Related Topics: ListColHeader, ListCount, ListGetColText, ListGetIndex, ListItemRemove, ListSetColText, ListSetIndex

#### ListItemRemove Subroutine (eXpress Plus)

Remove the currently selected item from a standard or drop-down list box.

Format:

```
ListItemRemove name
```

The *name* parameter is any string expression containing the name of a list box.

If no item is selected or the list is empty, this function has no effect.

Related Topics: ListColHeader, ListCount, ListGetColText, ListGetIndex, ListItemAdd, ListSetColText, ListSetIndex

#### ListSetColText Subroutine (eXpress Plus)

Set the text of a specified column in the currently selected item of a multi-column list box. Columns are numbered from left to right starting with zero (0).

Format:

```
ListSetColText name, column, value
```

The *name* parameter is any string expression containing the name of a multi-column list box. The *column* parameter is any integer expression representing the column relative to zero (0). The *value* parameter is any string expression.

Related Topics: ListColHeader, ListCount, ListGetColText, ListGetIndex, ListItemAdd, ListItemRemove, ListSetIndex

#### ListSetIndex Subroutine (eXpress Plus)

Set the selected list item index of a standard or drop-down list box.

Format:

```
ListSetIndex name, index
```

The *name* parameter is any string expression containing the name of a list box. The *index* parameter is any integer expression representing the row relative to zero (0).

Related Topics: ListColHeader, ListCount, ListGetColText, ListGetIndex, ListItemAdd, ListItemRemove, ListSetColText

#### ListSetItem Subroutine (eXpress Plus)

Set a row of data into the specified list box. Note: This function may not be used with multi-column list boxes (see ListSetColText Subroutine).

Format:

```
ListSetItem (name, index, value)
```

This function returns a String.

The *name* parameter is any string expression containing the name of a list box or drop-down list box. The *index* parameter is any integer expression representing the row relative to zero (0). The *value* parameter is any string expression containing the text of the row.

Related Topics: ListColHeader, ListCount, ListGetColText, ListGetIndex, ListGetItem, ListItemAdd, ListItemRemove, ListSetColText, ListSetIndex

#### LoadImage Function (eXpress Plus)

Load an image file into the specified image control. The image file may be a Windows or OS/2 bitmap (.bmp), icon (.ico), Windows metafile (.wmf) Windows enhanced metafile (.emf) or JPEG compliant files (.jpg and .jpeg).



Format:

`LoadImage (name, imagefile)`

The *name* parameter is any string expression containing the name of an image control. The *imagefile* parameter is any string expression containing the complete drive, path and file specification of the image file. To clear the image and load a blank image, specify a null string for the *imagefile* property.

Examples:

```
Rslt = LoadImage(ImControl, "c:\Image.bmp")
```

or:

```
Rslt = LoadImage(ImControl, "")
```

### LoadMMFile Function (eXpress Plus)

Load a multimedia file into the specified multimedia player control. The file type may be any supported by the Windows Media Player.

Format:

`LoadMMFile (name, mmfile)`

The *name* parameter is any string expression containing the name of a media player control. The *mmfile* parameter is any string expression containing the complete drive, path and file specification of the multimedia file.

Example:

In this example, a button on one form creates an object with the TEQDlgFormX.ocx (supplied with this product), loads MOVIE.BFM into the dialog form and shows the form:

```
Sub Btn_Movie()
' Action for Btn_Movie
    set df = CreateObject("TEQDlgFormx.TEQDlgForm")
    df.LoadDialogForm("C:\Data\TESTx\MOVIE.BFM")
    df.ShowDialogForm
End Sub
```

A second form, MOVIE (hence, MOVIE.BFM) contains the control that actually activated the movie when the "Play" button on the control is clicked:

```
Sub FormInitial()
' Action for FormInitial
    Rslt = LoadMMFile("Player", "C:\Data\TESTx\SPEEDIS.AVI")
End Sub
```

Also see, Media Players.

### LOF Function

Return a long number for the number of bytes in the open file.

Format:

`LOF (filenumber)`

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 Function

Return the natural log of a number.

Format:

`Log (number)`

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

Related Topics: Cos, Exp, Sin, Tan

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**

Return a substring within a string.

Format:

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

Mid returns a String.

The Mid function has these parts:

<u>Part</u>	<u>Description</u>
<i>string</i>	String expression from which another string is created.
<i>begin</i>	Long expression that indicates the character position in <i>stringexpression</i> at which the part to be taken begins.
<i>length</i>	Long expression that indicates the number of characters to return.

Related Topics: Left, Len, Right

Example:

```
Sub BTN_6()
  Dim MidWord, Msg, TstStr, SpcPos1, SpcPos2, WordLen
  TstStr = "Mid Function Demo"
  SpcPos1 = InStr(1, TstStr, " ") ' Find 1st space
  SpcPos2 = InStr(SpcPos1 + 1, TstStr, " ") ' Find 2nd space
  WordLen = (SpcPos2 - SpcPos1) - 1 ' Get 2nd word length
  MidWord = Mid(TstStr, SpcPos1 + 1, WordLen) ' Get 2nd word
  Msg = "the word in the middle of Title is '" & MidWord & "'."
  MsgBox Msg, 0, TstStr
End SUB
```

**Minute Function**

Return an integer between 0 and 59 that is the portion of the *time* parameter representing the minute of the hour.

Format:

```
Minute (time)
```

The *time* parameter is any string expression that can represent a time.

Related Topics: Date, Day, Format, Hour, Month, Now, Second, Weekday, Year

Example:

```
MyTime = "08:04:23 PM"
MsgBox MyTime

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

**MkDir Statement**

Create a new directory.

Format:

```
Mkdir path
```

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

Related Topics: ChDir, ChDrive, CurDir, Dir, Rmdir

Example:

```
Sub Main
  Dim DST As String
```

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

**Month Function**

Return an integer between 1 and 12 that is the portion of the *date* parameter representing the month of the year.

Format:

```
Month (date)
```

The *date* parameter is any string expression that can represent a date.

The returned integer represents the month of the *date* parameter.

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

Related Topics: Date, Day, Format, Hour, Minute, Now, Second, Weekday, Year

Example:

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

**MsgBox Function, MsgBox Statement**

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

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

Function Format:





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

Statement Format:

```
MsgBox msg [, type] [, title]
```

The *msg* parameter 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.

The *type* is the sum of the values specifying the type of buttons to display, the icon style to use, the identity of the default button and the modality. The following illustrates the values and meaning of each group:

<u>Constant</u>	<u>Value</u>	<u>Meaning</u>
MB_OK	0	Display OK button only.
MB_OKCANCEL	1	Display OK and Cancel buttons.
MB_ABORTRETRYIGNORE	2	Display Abort, Retry and Ignore buttons.
MB_YESNOCANCEL	3	Display Yes, No and Cancel buttons.
MB_YESNO	4	Display Yes and No buttons.
MB_RETRYCANCEL	5	Display Retry and Cancel buttons.
<hr/>		
MB_ICONSTOP	16	 Display:
MB_ICONQUESTION	32	 Display:
MB_ICONEXCLAMATION	48	 Display:
MB_ICONINFORMATION	64	 Display:
<hr/>		
MB_DEFBUTTON1	0	First button is default.
MB_DEFBUTTON2	256	Second button is default.
MB_DEFBUTTON3	512	Third button is default.

MB_APPLMODAL	0	Application modal. The user must respond to the message box before continuing work in the current application.
MB_SYSTEMMODAL	4096	System modal. All applications are suspended until the user responds to the message box.

The first group of values (0-5) 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. 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 zero (0).

The *title* parameter is a 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:

Constant	Value	Meaning
IDOK	1	OK button selected.
IDCANCEL	2	Cancel button selected.
IDABORT	3	Abort button selected.
IDRETRY	4	Retry button selected.
IDIGNORE	5	Ignore button selected.
IDYES	6	Yes button selected.
IDNO	7	No button selected.

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

Note: Since variables do not have to be declared prior to first reference, use an **Option Explicit** statement (external to the procedure) when using constants to assure that they are spelled correctly.

Example:

This 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 Yes 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.

```

Sub Main
  Dim DgDef, Msg, Response, Title
  Title = "MsgBox Sample Question"
  Msg = "This is a sample of Close Without Saving?."
  Msg = Msg & " Do you want to save changes?"
  DgDef = MB_YESNOCANCEL + MB_ICONQUESTION + MB_DEFBUTTON1
  Response = MsgBox(Msg, DgDef, Title)
  If Response = IDYES Then
    Msg = "You chose Yes or pressed Enter."
  ElseIf Response = IDCANCEL
    Msg = "You chose Cancel or pressed Esc."
  Else
    Msg = "You chose No."
  End If
  MsgBox Msg
End Sub
    
```

**Name Statement**

Change the name of a directory or a file.

Format:

```
Name oldname As newname
```

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

Related Topics: ChDir, Kill

Example:

```

Sub Main
  Name "testfile" As "newtest"
End Sub
    
```

```
End Sub
```

### Now Function

Return a date that represents the current date and time according to the settings in the computer's system date and time.

Format:

**Now**

Related Topics: Date, Day, Format, Hour, Minute, Month, Second, Weekday, Year

Example:

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

### Oct Function

Return the octal value of the decimal parameter.

Format:

**Oct** (*number*)

Oct returns a string.

Related Topics: Hex, 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 Statement

Use to close a dialog when accepting changes.

Format:

**OKBUTTON** *starting-x-pos, starting-y-pos, width, height*

Related Topics: Begin Dialog, CancelButton, CheckBox, Dialog, DropDownList, GroupBox, ListBox, OptionButton, OptionGroup, PushButton, Text, TextBox

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
```

### On Error Statement

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

Format:

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

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

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	Cannot 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	Cannot 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 does not 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 does not match format of data
480	Cannot create AutoRedraw image
481	Invalid picture
482	Printer error
483	Printer driver does not support specified property
484	Problem getting printer information from from the system – make sure the printer is setup correctly
485	invalid picture type
520	Cannot empty Clipboard
521	Cannot open Clipboard

Example:

```

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

```

### Open Statement

Open a file for input and output operations.

Format:

Open *file* [For *mode*] [Access *access*] As [#]*filenumber*

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

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

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

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

<u>Mode</u>	<u>Description</u>
Input	Sequential input mode.
Output	Sequential output mode.
Append	Sequential output mode. Append sets the file pointer to the end of the file. A Print # or Write # statement then extends (appends to) the file.

The *access* argument 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 does not support networking, a "feature unavailable" error occurs. The *access* argument can be one of the following reserved words:

<u>Access</u>	<u>Description</u>
Read	Opens the file for reading only.

Write            Opens the file for writing only.  
 Read Write      Opens the file for both reading and writing. This access is valid only for files opened for Append mode.

Related Topics: Close, EOF, Input, Line Input, Open, Print #, Write

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
```

### Option Base Statement

Declare the default lower bound for array subscripts.

Format:

Option Base *number*

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

The value of *number* must be either zero (0) or one (1). The default base is zero.

The **To** clause in the Dim, Global, and Static statements provides a more flexible way to control the range of an array's subscripts. If you do not explicitly set the lower bound with a **To** clause, you can use Option Base to change the default lower bound to one (1).

Related Topics: Dim, Global, Lbound, Static, UBound

The following example uses the Option Base statement to override the default base array subscript value of zero (0):

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

### Option Explicit Statement

Require all variables referenced to be explicitly declared.

Format:

Option Explicit

The Option Explicit statement is used outside of the script in the Declarations section.

*It is highly recommended that an Option Explicit statement be used with all actions.*

Related Topics: Const, Global

Example :



```

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

End Sub

```

### OptionButton Statement

Use an option button (radio button) in a dialog for selecting one, and only one, option from a group of options.

Format:

`OptionButton starting-x-pos, starting-y-pos, width, height, "caption"`

Related Topics: Begin Dialog, CancelButton, CheckBox, Dialog, DropListBox, GroupBox, ListBox, OKButton, OptionGroup, PushButton, Text, TextBox

Example:

```

Sub BTN_1()
Begin Dialog Dialog_1 0,0, 252, 136, "Dialog Title"
    OptionGroup .GRP_1
        OptionButton 32,48,80,12, "1st Radio - Group 1"
        OptionButton 32,64,80,12, "2nd Radio - Group 1"
    OptionGroup .GRP_2
        OptionButton 144,48,84,12, "1st Radio - Group 2"
        OptionButton 144,64,84,12, "2nd Radio - Group 2"
    OKButton 24,96,68,20
    CancelButton 156,96,52,20
    GroupBox 24,36,92,52, "Option Group 1"
    GroupBox 136,36,100,52, "Option Group 2"
    GroupBox 24,4,212,28, "Check Group"
    CheckBox 36,16,76,12, "Check_Box_1", .CHECKBOX_1
    CheckBox 124,16,76,12, "Check_Box_1", .CHECKBOX_2
End Dialog
Dim Dlg1 As Dialog_1
Dlg1.Grp_1 = 0          ' Set 1st button - group 1
Dlg1.Grp_2 = 1          ' Set 2nd button - group 2
button = Dialog ( Dlg1 )
If button = 0 Then Return
MsgBox "Grp1: " + Dlg1.Grp_1 + ", Grp2: " + Dlg1.Grp_2
Dialog Dlg1
End Sub

```

### OptionGroup Statement

Use an option group in a dialog for grouping mutually exclusive option buttons.

Format:

`OptionGroup .name`

Related Topics: Begin Dialog, CancelButton, CheckBox, Dialog, DropListBox, GroupBox, ListBox, OKButton, OptionButton, PushButton, Text, TextBox

Example:

(See OptionButton.)

### Print # Statement

Write data to a sequential file.

Format:

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

The Print statement consists of the following parts:

<u>Part</u>	<u>Description</u>
<i>filename</i>	Number used in an Open statement to open a sequential file. This parameter can be any numeric expression that evaluates to the number of an open file. Note that the number sign (#) preceding filename is not optional.
<i>Spc(n)</i>	Name of the Basic function optionally used to insert <i>n</i> spaces into the printed output. Multiple use is permitted.
<i>Tab(n)</i>	Name of the Basic function optionally used to tab to the <i>n</i> 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.

Related Topics: Close, EOF, Input, Line Input, Open, Write #

The following example writes data to a test file:

```
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
```

### Print Statement

Print a string to the default printer (specified by the user in the Windows Control Panel).

Format:

*Print expression*

Print text on the current printer at the current x, y coordinates. The x and y coordinates are set using the PrintMoveTo subroutine.

Use the following technique for printing text and graphics to the default printer:

Send text and graphics to the Printer Object and print them using the PrintNewPage and PrintEndDoc subroutines.

The Printer Object is a device-independent drawing space that supports Print, PrintBeginDoc, PrintEndDoc, PrintMoveTo, PrintNewPage, PrintPageHeight, PrintPageWidth, PrintRect, PrintSetFont, PrintSetFontStyle, PrintTextHeight and PrintTextWidth subroutines. When you finish placing the information on the Printer Object, you use the PrintEndDoc or PrintNewPage subroutines to send the output to the printer.

Related Topics: PrintBeginDoc, PrintEndDoc, PrintMoveTo, PrintNewPage, PrintPageHeight, PrintPageWidth, PrintRect, PrintSetFont, PrintSetFontStyle, PrintTextHeight, PrintTextWidth

Example:

```
Option Explicit
Sub Main()
  dim x as integer
  dim LineHeight as integer
  dim Margin as integer

  PrintBeginDoc
  ' Set printer font
  PrintSetFont "Courier New"
  PrintSetFontStyle fsFontBold
```

```

' Calculate margin as one inch
Margin = PrintPageHeight / 11
' Calculate line height based on current font
LineHeight = PrintTextHeight("X")

'Print a title
PrintMoveTo Margin, Margin - LineHeight
Print "List of Orders for Account " + GetScreenText(10, 5, 9)

'Print order ids and amounts
PrintSetFontStyle fsNormal
PrintMoveTo Margin, Margin
For x = 1 to 13
    PrintMoveTo Margin, Margin + x * LineHeight
    Print GetScreenText(18, 7 + Str(x), 11), _
        GetScreenText(44, 7 + Str(x), 12)
Next x

' Print a box around orders ids and amounts
PrintRect Margin, Margin, Margin + (Margin * 6), _
    Margin + (LineHeight * (x + 1)), 5
PrintEndDoc
MsgBox "Printing Complete", MB_ICONINFORMATION
End Sub

```

**PrintBeginDoc Subroutine (eXpress Plus)**

Initialize the Printer Object (page) context.

Format:

```
PrintBeginDoc
```

Related Topics: Print, PrintEndDoc, PrintMoveTo, PrintNewPage, PrintPageHeight, PrintPageWidth, PrintRect, PrintSetFont, PrintSetFontSize, PrintSetFontStyle, PrintTextHeight, PrintTextWidth

Example:

(See Print Statement).

**PrintDlg Function (eXpress Plus)**

Display a standard print dialog.

Format:

```
PrintDlg()
```

This function returns an integer. Returns True (1) if OK, else returns False (0).

Related topics: PrintDraw, PrintSetOrientation

**PrintDraw Subroutine (eXpress Plus)**

Draw an image from the named image control on the print page within the specified boundary. The image is stretched to fit the boundary.

Format:

```
PrintDraw ImageCtlName, left, top, right, bottom
```

The *ImageCtlName* parameter is any string expression containing a valid image control name. The *left*, *top*, *right*, *bottom* and *width* parameters are any integer expression and are specified in pixels.

To maintain the aspect ratio of image, *bottom* or *right* may be set to zero (0) indicating the following. If *right* is set to zero, the image will be scaled for height only and the width will be scaled proportionally. If *bottom* is set to zero, the image will be scaled for width only and the height will be scaled proportionally.

Related topics: PrintDlg Function, PrintSetOrientation Subroutine

**PrintEndDoc Subroutine (eXpress Plus)**

Terminate printing. If print is in the current printer context, it is printed.

Format:

```
PrintEndDoc
```

Related Topics: Print, PrintBeginDoc, PrintMoveTo, PrintNewPage, PrintPageHeight, PrintPageWidth, PrintRect, PrintSetFont, PrintSetFontSize, PrintSetFontStyle, PrintTextHeight, PrintTextWidth

Example:

(See Print Statement).

### **PrintMoveTo Subroutine (eXpress Plus)**

Move the current x- and y-coordinates of the print object. The new x- and y-coordinates will be used as the upper left position of text printed using the Print statement.

Format:

*PrintMoveTo x-coordinate, y-coordinate*

The *x-coordinate* and *y-coordinate* parameters are any integer expression.

Related Topics: Print, PrintBeginDoc, PrintEndDoc, PrintNewPage, PrintPageHeight, PrintPageWidth, PrintRect, PrintSetFont, PrintSetFontSize, PrintSetFontStyle, PrintTextHeight, PrintTextWidth

Example:

(See Print Statement).

### **PrintNewPage Subroutine (eXpress Plus)**

Cause the current content of the Printer Object to be printed immediately.

Format:

*PrintNewPage*

Related Topics: Print, PrintBeginDoc, PrintEndDoc, PrintMoveTo, PrintPageHeight, PrintPageWidth, PrintRect, PrintSetFont, PrintSetFontSize, PrintSetFontStyle, PrintTextHeight, PrintTextWidth

Example:

(See Print Statement).

### **PrintPageHeight Function (eXpress Plus)**

Retrieve the current page height in pixels. The pixel height and width of the page will vary depending on the currently selected printer.

Format:

*PrintPageHeight()*

Related Topics: Print, PrintBeginDoc, PrintEndDoc, PrintMoveTo, PrintNewPage, PrintPageWidth, PrintRect, PrintSetFont, PrintSetFontSize, PrintSetFontStyle, PrintTextHeight, PrintTextWidth

Example:

(See Print Statement).

### **PrintPageWidth Function (eXpress Plus)**

Retrieve the current page width in pixels. The pixel height and width of the page will vary depending on the currently selected printer.

Format:

*PrintPageWidth()*

Related Topics: Print, PrintBeginDoc, PrintEndDoc, PrintMoveTo, PrintNewPage, PrintPageHeight, PrintRect, PrintSetFont, PrintSetFontSize, PrintSetFontStyle, PrintTextHeight, PrintTextWidth

Example:

(See Print Statement).

### **PrintRect Subroutine (eXpress Plus)**

Print a rectangle using the specified left, top, right and bottom coordinates, and line width.

Format:

*PrintRect left, top, right, bottom, width*

The *left*, *top*, *right*, *bottom* and *width* parameters are any integer expression and are specified in pixels.

Related Topics: Print, PrintBeginDoc, PrintEndDoc, PrintMoveTo, PrintNewPage, PrintPageHeight, PrintPageWidth, PrintSetFont, PrintSetFontSize, PrintSetFontStyle, PrintTextHeight, PrintTextWidth

Example:

(See Print Statement).

### **PrintSetFont Subroutine (eXpress Plus)**

Set the current print font.

Format:

*PrintSetFont name*

The *name* parameter is any string expression containing a valid font name. If the font does not exist on the user's system, Windows will substitute a default font of the selected printer.

Related Topics: Print, PrintBeginDoc, PrintEndDoc, PrintMoveTo, PrintNewPage, PrintPageHeight, PrintPageWidth, PrintRect, PrintSetFont, PrintSetFontStyle, PrintTextHeight, PrintTextWidth

Example:

(See Print Statement).

### **PrintSetFontSize Subroutine (eXpress Plus)**

Set the size of the current print font in points.

Format:

PrintSetFontSize *size*

The *size* parameter is any integer expression.

Related Topics: Print, PrintBeginDoc, PrintEndDoc, PrintMoveTo, PrintNewPage, PrintPageHeight, PrintPageWidth, PrintRect, PrintSetFont, PrintSetFontStyle, PrintTextHeight, PrintTextWidth

Example:

(See Print Statement).

### **PrintSetFontStyleSubroutine (eXpress Plus)**

Set the style of the current print font.

Format:

PrintSetFontStyle *style*

The *style* parameter is a numeric expression containing a number equal to the sum of all required attributes.

The *style* parameter may be stated as an Integer or a Constant. Available styles are:

<u>Effect</u>	<u>Integer</u>	<u>Constant</u>
Normal	0	fsNormal
Bold	1	fsFontBold
Italic	2	fsFontItalic
Underline	4	fsFontUnderline
Strikethrough	8	fsFontStrikeThru

Styles may be ORed together to create combined effects. For example, to set bold and italic, use "fsFontBold or fsFontItalic".

Note: Since variables do not have to be declared prior to first reference, use an **Option Explicit** statement (external to the procedure) when using constants to assure that they are spelled correctly.

Related Topics: Print, PrintBeginDoc, PrintEndDoc, PrintMoveTo, PrintNewPage, PrintPageHeight, PrintPageWidth, PrintRect, PrintSetFont, PrintSetFontStyle, PrintTextHeight, PrintTextWidth

Example:

(See Print Statement).

### **PrintSetOrientation Subroutine (eXpress Plus)**

Set print orientation to portrait or landscape.

Format:

PrintSetOrientation *orientation*

The *orientation* parameter is any integer expression, where Portrait = zero (0) and Landscape = one (1).

Related topics: PrintDlg Function, PrintDraw Subroutine

### **PrintTextHeight Function (eXpress Plus)**

Retrieve the height in pixels of a specified text string. The text height can be used to determine vertical spacing between lines.

Format:

PrintTextHeight (*textstring*)

The *textstring* parameter is any string expression.

Related Topics: Print, PrintBeginDoc, PrintEndDoc, PrintMoveTo, PrintNewPage, PrintPageHeight, PrintPageWidth, PrintRect, PrintSetFont, PrintSetFontStyle, PrintTextHeight, PrintTextWidth

Example:

(See Print Statement).

**PrintTextWidth Function (eXpress Plus)**

Retrieve the width in pixels of a specified text string. The text width can be used to determine horizontal spacing.

Format:

`PrintTextWidth (textstring)`

The *textstring* parameter is any string expression.

Related Topics: Print, PrintBeginDoc, PrintEndDoc, PrintMoveTo, PrintNewPage, PrintPageHeight, PrintPageWidth, PrintRect, PrintSetFont, PrintSetFontSize, PrintSetFontStyle, PrintTextHeight

Example:

(See Print Statement).

**PushButton Statement**

Use a push button in a dialog for assigning a button to a command.

Format:

`PushButton starting-x-pos, starting-y-pos, width, height, "caption" .name`

Related Topics: Begin Dialog, CancelButton, CheckBox, Dialog, DropListBox, GroupBox, ListBox, OKButton, OptionButton, OptionGroup, Text, TextBox

Example:

```

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", .chk1
  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

```

**Put Statement**

Write to a disk file from a variable.

Format:

```
Put [#] filename, [recordnumber,] variablename
```

The Put statement has three parts:

<u>Parameter</u>	<u>Description</u>
<i>filename</i>	The number used to open the file.
<i>recordnumber</i>	For files opened in Binary mode, <i>recordnumber</i> is the byte position where writing starts.
<i>variablename</i>	The name of the variable containing the data to be written to the file.

Related Topics: Open, Get

**Randomize Statement**

Initialize the random number generator.

Format:

```
Randomize [number]
```

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**

Declare dynamic arrays and reallocate storage space.

Format:

```
ReDim varname (subscripts) [As type][,varname(subscripts) [As type]] ...
```

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 change the number of elements repeatedly in an array, but not to change the number of dimensions in an array or the type of the elements in the array.

Related Topics: Dim, Option Base, Set, Static

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

Include explanatory remarks in a program.

Format:

**Rem** *remark*

Or anywhere after a statement on a line:

*' remark*

The *remark* parameter 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

Return the right most *number* of characters of the string parameter.

Format:

**Right** (*string, number*)

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

The *number* parameter is the numeric expression indicating the number of characters that will be returned.

Related Topics: Left, Len, Mid

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
    SpcPos = InStr(1, UsrInp, " ")        ' Find space
    If SpcPos Then
        LWord = Left(UsrInp, SpcPos - 1) ' Get left word
        RWord = Right(UsrInp, Len(UsrInp) - SpcPos)
                                           ' Get right word
        Msg = "The first word you entered is <" & LWord & ">"
        Msg = Msg & RWord & ". "
    Else
        Msg = "You didn't enter two words."
    End If
End Sub

```

```

        MsgBox Msg                ' Display message
    End Sub

```

**Rmdir Statement**

Remove an existing directory.

Format:

```
Rmdir path
```

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

Related Topics: ChDir, ChDrive, CurDir, Dir, Mkdir

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**

Return a random number.

Format:

```
Rnd [(number)]
```

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

The Rnd function returns a Single value less than one (1) but greater than or equal to zero (0).

The value of *number* determines how Rnd generates a random number:

<u>Value of number</u>	<u>Number returned</u>
< 0	The same number every time as determined by <i>number</i> .
> 0	The next random number in the sequence.
= 0	The number most recently generated.
<i>number</i> omitted	The next random number in the sequence.

Example:

```

' The example uses the Rnd function to simulate rolling a
' pair of dice by generating random values from 1 to 6.

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

```

**ScriptDir Function (eXpress Plus)**

Return the full directory path of the current script file. Note: There is no trailing backslash (\).

Format:

```
ScriptDir()
```

This function returns a String.

Example:

```
) Set df = CreateObject("TEQDlgFormx.TEQDlgForm")
```

```

) df.LoadDialogForm(ScriptDir + "\ACCOUNTSELECTOR.BFM")
) df.ShowDialogForm
) If Df.DialogResult = 0 Then
)     df = nothing
)     MsgBox "Cancelled"
0)     Exit Sub
1) End If

```

### Second Function

Return an integer between 0 and 59 that is the portion of the *time* parameter representing the second of the minute.

Format:

**Second** (*time*)

The *time* parameter is any string expression that can represent a time.

Related Topics: Date, Day, Format, Hour, Minute, Month, Now, Weekday, Year

Example:

```

MyTime = "08:04:23 PM"
MsgBox MyTime

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

```

### Seek Function

Return 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.

Format:

**Seek** (*filenumber*)

The *filenumber* parameter is used on an Open statement and must be a valid numeric expression.

Related Topics: Open

Example:

```

Sub Main
    Open "TESTFILE" For Input As # ' 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

Set the position in a file for the next read or write.

Format:

**Seek** *filenumber*, *position*

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

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**

Execute one of the sets of statement(s) in the case, based on the test variable.

Format:

```
Select Case testexpression
    [Case expressionlist1
        statement(s)]
    [Case expressionlist2
        statement(s)]
    [Case Else
        statement(s)]
End Select
```

The Select Case statement has these parts:

<u>Part</u>	<u>Description</u>
Select Case	Begins the Select Case decision control sequence.
<i>testexpression</i>	Any numeric or string expression. If <i>testexpression</i> matches the <i>expressionlist</i> associated with the Case clause, the <i>statement(s)</i> following the Case clause are executed.
Case	Sets apart a group of statements to be executed if an expression in <i>expressionlist</i> matches the <i>testexpression</i> .
<i>expressionlist</i>	The <i>expressionlist</i> consists of a comma-delimited list of one or more of the following forms: <i>expression</i> <i>expression</i> To <i>expression</i> Is <i>compare-operator</i> <i>expression</i>
<i>statement(s)</i>	Any number of statements on one or more lines.
Case Else	Begins the statement(s) to be executed if no match is found between the <i>testexpression</i> and an <i>expressionlist</i> in any of the other Case selections.
End Case	Ends the Select Case.

The *expression* parameter may be any numeric or string expression; however, it must be compatible with the type of *testexpression*.

The *compare-operator* may be any valid comparison operator, except Is and Like.

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

Example:

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

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

End Sub

### SendKeys Statement

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

Format:

SendKeys *keys*

The *keys* parameter is a string and is sent to the active window.

To send a single keyboard character, use the character itself. To send the letter A, use "A". To send multiple keyboard characters, one behind the other, include them in the string in the order you want them sent. To send a D followed by an E and then followed by an F, use "DEF".

Ten keyboard characters have special significance when used with the SendKeys statement:

<u>Character(s)</u>	<u>Usage</u>
Braces { }	Used to enclose a special character or key name being sent. For example, {F4} sends function key 4.
Plus sign +	The SHIFT key.
Caret ^	The CTRL key.
Percent sign %	The ALT key.
Tilde ~	The ENTER key.
Parentheses ( )	Used to enclose multiple keystrokes in combination with the SHIFT, CTRL and ALT keys. For example, "%(EF)" would be the same as holding down the ALT key while pressing E followed by F.
Brackets [ ]	No special significance but must be enclosed in braces when sent; e.g., "{[]}" and "{[]}".

To send any special character, enclose it in braces. For example, "{{}" sends an open brace and "{+}" sends a plus sign.

To send keys that do not display when you press them, use the following substitution codes:

<u>Key</u>	<u>Substitution Code</u>
BACKSPACE	{BACKSPACE}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER} or ~
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS	{INSERT}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}
:	:
F16	{F16}

To repeat a key, follow the key by the number of times to repeat the keystroke. For example, "{UP 10}" is the same as pressing the UP ARROW 10 times. Note: A space is required between the key and the number.

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 & "{+}" ' Send keystrokes to Calculator.
    Next I
End Sub
```

```

Next I                                ' to add each value of I.

Msg = "Choose OK to close the Calculator."
MsgBox Msg                             ' Display OK prompt.
AppActivate "Calculator"                ' Return focus to Calculator.
SendKeys "%{F4}"                        ' Alt+F4 to close Calculator.
End Sub

```

### SendMail Subroutine (exPress Plus)

Create an e-mail message in a dialog form's Action.

Format:

*SendMail Recipients, Subject, CcRecipients, BccRecipients, MessageText, Attachments, NoPrompt*

The *Recipients, Subject, CcRecipients, BccRecipients, MessageText* and *Attachments* parameters are any string expressions.

The *Recipients, CcRecipients, BccRecipients, MessageText* and *Attachments* are actually lists of strings allowing multiple items. Items must be separated by semicolons. For example, to send to two recipients, you use the following: "Alice@kmsys.com;Ralph@kmsys.com".

Empty parameters must be entered as "" (zero length strings).

When the Mail is sent, the user's e-mail handler will allow the message to be reviewed before sending. If the user does not send the message or an error occurs, a message box containing the message "Message Not Sent" will pop up.

*NoPrompt* is an integer value (which is also a Boolean in VB). If *NoPrompt* is non-zero (True) the mail is sent without the review prompt.

Example:

```

Sub BtnSendMail()
    SendMail
    "Ralph@KMSys.com", "TextSubject", "Alice@KMSys.Com;Steve@xys.com", "", "This
    is the message text line 1;line two", "C:\Docs\MyFile.txt", True
End Sub

```

### Set Statement

Assign an object to an object variable.

Format:

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

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
```

### SetColor Subroutine (eXpress Plus)

Set either the foreground or the background color of a control.

Format:

```
SetColor–name, colortype, color
```

The *name* parameter is a string expression containing the name of a valid control.

The *colortype* is an integer expression specifying the property type for which the color is to be set. The *color* is a long integer expression designating the desired color. Both the *colortype* and *color* are expressed as Constants.

Valid *colortype* entries are:

<u>Constant</u>	<u>Description</u>
tpForeColor	The color of text or box border.
tpBackColor	The color of control's background area.

Valid *color* entries are:

<u>Constant</u>	<u>Description</u>
clBlack	Black
clMaroon	Maroon
clGreen	Green
clOlive	Olive green
clNavy	Navy blue
clPurple	Purple
clTeal	Teal
clGray	Gray
clSilver	Silver
clYellow	Yellow
clRed	Red
clLime	Lime green
clBlue	Blue
clFuchsia	Fuchsia
clAqua	Aqua
clWhite	White

*The following colors refer to the default colors set in the current Windows environment:*

clScrollBar	Current color of Windows scrollbar.
clBackground	Current color of Windows background.
clActiveCaption	Current color of the title bar of the active window.
clInactiveCaption	Current color of the title bar of inactive windows.
clMenu	Current background color of menus.
clWindow	Current background color of windows.
clWindowFrame	Current color of window frames.
clMenuText	Current color of text on menus.
clWindowText	Current color of text in windows.
clCaptionText	Current color of the text on the title bar of the active window.
clActiveBorder	Current border color of the active window.
clInactiveBorder	Current border color of inactive windows.
clAppWorkSpace	Current color of the application workspace.
clHighlight	Current background color of selected text.
clHightlightText	Current color of selected text.
clBtnFace	Current color of a button face.
clBtnShadow	Current color of a shadow cast by a button.
clGrayText	Current color of text that is dimmed.
clBtnText	Current color of text on a button.
clInactiveCaptionText	Current color of the text on the title bar of an inactive window.
clBtnHighlight	Current color of the highlighting on a button.

Note: Since variables do not have to be declared prior to first reference, use an **Option Explicit** statement (external to the procedure) when using constants to assure that they are spelled correctly.

Related Topics: GetColor

Example:

```
Dim clr as long
Clr = GetColor("ACCOUNT", tpForeColor)
If Clr = clWindowText Then
    SetColor "ACCOUNT", tpForeColor, clBlue
End If
```

### SetFocus Subroutine (eXpress Plus)

Set the windows focus to the specified control. The only control types that may receive focus are: edit box, command button, standard list box, drop-down list box, multi-column list box, check box or option button.

Format:

**SetFocus** *name*

The *name* parameter is any string expression containing a valid control name.

Example:

This example sets the focus to either a edit box or a command button depending on whether an option button is currently checked.

```
If GetState(OPT_1, tpChecked) then
    SetFocus "ACCOUNT"
Else
    SetFocus "BTN_FIND_ACCOUNTS"
End If
```

### SetNumericProp Subroutine (eXpress Plus)

Set the integer property to the value specified. Numeric properties include Height, Width, Left, Top, TabOrder, etc.

Format:

**SetNumericProp** *name, property, value*

The *name* parameter is any string expression containing the name of the control. The *property* parameter is any string expression containing the name of the property. The *value* parameter is any integer expression.

Related Topics: GetNumericProp, GetStringProp, SetStringProp

Example:

```
Option Explicit

Sub Btn_Load()
' Load some items into the list

    ListItemAdd "MCList", "1234, Spring, 5.12"
    ListItemAdd "MCList", "2312, Nut, .02"
    ListItemAdd "MCList", "3334, Bolt, .12"
    ListItemAdd "MCList", "2230, TBolt, .35"
    ListItemAdd "MCList", "9324, Flange, 1.23"
    ListItemAdd "MCList", "0534, Washer, 1.10"
    ListItemAdd "MCList", "7801, Lock Washer, 1.77"
    ListItemAdd "MCList", "0725, Lock Washer 2, -1.05"
    ListItemAdd "MCList", "3012, Lock Washer 3, -1.77"
    ListItemAdd "MCList", "3012, Trailing Sign Invalid, 1.77-"

End Sub

Sub Sb_SortPartNo()
' Change sort the first column and an alpha sort
    SetNumericProp "MCList", "SortColumn", 0
    SetNumericProp "MCList", "NumericSort", 0 End Sub

Sub Sb_SortDesc()
' Change sort the second column and an alpha sort
    SetNumericProp "MCList", "SortColumn", 1
    SetNumericProp "MCList", "NumericSort", 0 End Sub

Sub Sb_SortPrice()
```



```

' Change sort the third column and a numeric sort
  SetNumericProp "MCList", "SortColumn", 2
  SetNumericProp "MCList", "NumericSort", 1 End Sub

Sub Chk_Desc()
' Switch between ascending and descending sort.
  SetNumericProp "MCList", "SortDescending", GetState("Chk_Desc", tpChecked) End
Sub

```

### SetState Subroutine (eXpress Plus)

Set the specified state of controls.

Format:

**SetState** *name, statetype, state*

The *name* parameter is a string expression containing a control name.

The *statetype* parameter is an integer expression specifying the property type for which the state is to be retrieved.

The *statetype* parameter may be expressed as an Integer or a Constant. Valid *statetype* entries are:

<u>Effect</u>	<u>Integer</u>	<u>Constant</u>
Enabled	0	tpEnabled
Visible*	1	tpVisible
Checked**	2	tpChecked

\* Visible does not apply to user menu items.

\*\* Checked only applies to Option Button and Check Box controls.

Note: Since variables do not have to be declared prior to first reference, use an **Option Explicit** statement (external to the procedure) when using constants to assure that they are spelled correctly.

The *state* parameter is an integer expression containing either True (non-zero) or False (zero).

Related Topics: GetState

Example:

(see GetState).

### SetString Subroutine (eXpress Plus)

Set a control or global session variable string value. For edit boxes, the data text is set. For controls, like command buttons and labels, the caption text is set. For list boxes, the currently selected line is set.

Format:

**SetString** *name, value*

The *name* parameter is any string expression containing the name of a valid control name or global session variable. The *value* parameter is the string expression to be assigned to the named control or session variable.

Related Topics: GetString

Example:

(see GetString).

### SetStringProp Subroutine (eXpress Plus)

Set the string property to the value specified. String properties include Hints, Captions, Pictures, etc.

Format:

**SetStringProp** *name, property, value*

The *name* parameter is any string expression containing the name of the control. The *property* parameter is any string expression containing the name of the property. The *value* parameter is any string expression containing the value to be set in the property.

Note: The Caption property of Text Label controls cannot be set using SetStringProp — use the SetString subroutine, instead.

Related Topics: GetNumericProp, GetStringProp, SetNumericProp, SetString

### Sgn Function

Return an integer indicating the sign (+, -, 0) of a number.

Format:

**Sgn**(*number*)

The number parameter can be any valid numeric expression. The Sgn function returns the following values:

<u>Value</u>	<u>Condition</u>
--------------	------------------

```

1      number > 0
0      number = 0
-1     number < 0
    
```

**Shell Function**

Run an executable program.

Format:

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

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. The second argument is also optional, and if omitted, the program is opened minimized with focus.

<u>Value</u>	<u>Window Style</u>
1, 5, 9	Normal with focus.
2	Minimized with focus (default).
3	Maximized with focus.
4, 8	Normal without focus.
6, 7	Minimized without focus.

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 & "{+}"   ' Send keystrokes to Calculator
  Next I                 ' to add each value of I
  Msg = "Choose OK to close the Calculator."
  MsgBox Msg             ' Display OK prompt
  AppActivate "Calculator" ' Return focus to Calculator
  SendKeys "%{F4}"      ' Alt+F4 to close Calculator
End Sub
    
```

**Sin Function**

Return the sine of an angle that is expressed in radians.

Format:

Sin (*radian*)

Example:

```

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

**Space Function**

Skip a specified number of spaces in a Print # statement.

Format:

Space (*number*)

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

Example:

```

Sub Main
  MsgBox "Hello" & Space(20) & "There"
End Sub
    
```

**Sqr Function**

Return the square root of a number.

Format:

*Sqr (number)*

The *number* parameter 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

Declare variables and allocate storage space. These variables will retain their value through the program run.

Format:

*Static variable*

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

End the execution of the program.

Format:

**Stop**

The **Stop** statement can be placed anywhere in your code.

Related Topics: End, Exit

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

Return the value of a numeric expression.

Format:

```
Str (numericexpression)
```

Str returns a String.

Use the Format function to convert numeric values you want formatted as dates, times or in other user-defined formats.

The Str function recognizes only the period (.) as a valid decimal separator. When a possibility exists that different decimal separators may be used (e.g., in international applications), you should use CStr to convert a number to a string.

Related topics: CStr, Format, Val

Example:

```
Sub main ()
  Dim msg
  a = -1
  MsgBox "Num = " & Str(a)
  MsgBox "_Abs(Num) =" & Str(_Abs(a))
End Sub
```

### StrComp Function

Return a variant that is the result of the comparison of two strings.

Format:

```
StrComp(string1,string2,[compare])
```

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 is used to create a string that consists of one character repeated repeatedly.

Formats:

```
String (numeric, charcode)
```

or

```
String (numeric, string)
```

String returns a string.

Related topics: Space

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**

Declare and define a Sub procedure name, parameters and code.

Format:

```
Sub subname [(argumentlist)]
    [statement(s)]
        subname = expression
[Exit Sub]
[statement(s)]
    subname = expression
End Sub
```

When the optional *argumentlist* 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).

The optional As *type* parameter is used to specify the data type. Valid types are String, Integer, Single, Double, Long and Variant (see Other Data Types).

Related Topics: Call, Dim, Function

Example:

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

**Tan Function**

Return the tangent of an angle as a double.

Format:

```
Tan(angle)
```

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

Related Topic: Atn, Cos, Sin

Example:

```
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**

Create a text field for titles and labels.

Format:

```
Text starting-x-pos, starting-y-pos, width, height, label
```

Related Topics: Begin Dialog, CancelButton, CheckBox, Dialog, DropDownList, GroupBox, ListBox, OKButton, OptionButton, OptionGroup, PushButton, TextBox

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

Create a Text Box for typing in numbers and text.

Format:

`TextBox starting-x-pos, starting-y-pos, width, height, .default_string, [32]`

The optional string, "32", instructs enable to provide password protection characters as the user types into the text box. The password protection character is the asterisk (\*).

Related Topics: Begin Dialog, CancelButton, CheckBox, Dialog, DropListBox, GroupBox, ListBox, OKButton, OptionButton, OptionGroup, PushButton, 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 Statement

Returns the current system time or sets the system time.

Time function returns a value; the Time statement does not.

Function Format:

`Time{()}`

Statement Format:

`Time = time`

The time parameter is any numeric or string expression that represents a time.

```

x = Time$(Now)
Print x

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

```

To set the system time use the TIME statement:

```

SysTime = "8:00:00 AM"
Time = SysTime

```

### Timer Event

Timer Event is used to track elapsed time. It can also be displayed as a stopwatch in a dialog. The timer's value is the number of seconds from midnight.

Format:

`Timer`

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

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

Format:

`TimeSerial (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

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

Format:

`TimeValue (timestring)`

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

Return a copy of a string with leading, trailing or both leading and training spaces removed.

Format:

`[L | R]Trim (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**

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

Format:

```

Type usertype elementname [(subscripts)] As typename
    [ elementname [(subscripts)] As typename]
    ...

```

End Type

The Type statement has these parts:

<u>Part</u>	<u>Description</u>
Type	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 (not currently implemented).
<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> cannot 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 are not 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.

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 typela.a
MsgBox typela.d
MsgBox typela.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**

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

Format:

`Ubound (arrayname[, dimension])`

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

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
    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**

Return a copy of a string in which all lowercase characters have been converted to uppercase.

Format:

`UCase (string)`

Related Topics: LCase

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 = "<-Trim->"
    MsgBox "|" & TrimString & "|"      ' Using the Trim function
                                        ' alone achieves the same
                                        ' result.
    TrimString = UCase(Trim(MyString)) ' TrimString = "<-TRIM->"
    MsgBox "|" & TrimString & "|"
End Sub

```

**Val Function**

Return the numeric value of a string of characters.

Format:

Val (*string*)

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 Function**

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

Format:

VarType (*varname*)

The *varname* parameter is a variant data type.

<u>VarType</u>	<u>Return Values</u>
Empty	0
Null	1
Integer	2
Long	3
Single	4
Double	5
Currency	6 (not available at this time)
Date/Time	7 (mapped to a string)
String	8

Related Topics: IsNull, IsNumeric

Example:

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

**Wait Subroutine (eXpress Plus)**

Wait for a fixed amount of time. A call to this subroutine causes the script to wait for a period before continuing on to the next script statement, subroutine or function.

Format:

Wait *milliseconds*

The *milliseconds* parameter is an integer expression containing the amount of time to wait expressed in milliseconds.

**Weekday Function**

Return an integer between 1 (Sunday) and 7 (Saturday) that represents the day of the week for a date argument.

Format:

Weekday (*date*)

The *date* parameter is any string expression that can represent a date.

The returned integer represents the day of the *date* parameter.

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

Related Topics: Date, Day, Format, Hour, Minute, Month, Now, Second, Year

Example:

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

**While...Wend Statement**

Execute a series of statements as long as a condition is true.

Format:

```
While condition
    [statement(s)]
Wend
```

The While...Wend statement has these parts:

<u>Part</u>	<u>Description</u>
While <i>condition</i>	Begins the While...Wend flow of control structure. Any numeric or expression that evaluates to true or false. If the condition is true, the statements are executed.
<i>statement(s)</i>	Any number of valid statements.
Wend	Ends the While...Wend flow of control structure.

Related Topics: Do...Loop Statement, With

Example:

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

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

**With Statement**

Execute a series of statements on a single object or user-defined type.

Format:

```
With object
    [statement(s)]
End With
```

The With statement allows you to perform a series of commands or statements on a particular object without referring to the name of that object again. 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: Do...Loop, While...Wend

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
typela.s = "YES"
MsgBox typela.a
MsgBox typela.d
MsgBox typela.s
MsgBox type2a.a
MsgBox type2a.o.s
End Sub

```

### Write # - Statement

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

Format:

```
Write #filename [,parameterlist]
```

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: Close, EOF, Input, Line Input, Open, Print #

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

Return an integer between 100 and 9999 that is the portion of the *date* parameter representing a year.

Format:

```
Year (date)
```

The *date* parameter is any string expression that can represent a date.

The returned integer represents the year of the *date* parameter.

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

Related Topics: Date, Day, Format, Hour, Minute, Month, Now, Second, Weekday

Example:

```
ThisYear = Year(Now)
```

## Predefined Constants

### Predefined Constants

This topic contains all interface constants that are predefined when an action script is invoked.

Note: Since variables do not have to be declared prior to first reference, use an **Option Explicit** statement (external to the procedure) when using constants to assure that they are spelled correctly.

### Color Types (also see **GetColor** and **SetColor**):

<u>Constant</u>	<u>Value</u>	<u>Description</u>
tpForeColor	0	The color of text or box border.
tpBackColor	1	The color of the control's background area.





### Defined Colors (also see **SetColor**):

<u>Constant</u>	<u>Description</u>
clBlack	Black
clMaroon	Maroon
clGreen	Green
clOlive	Olive green
clNavy	Navy blue
clPurple	Purple
clTeal	Teal
clGray	Gray
clSilver	Silver
clYellow	Yellow
clRed	Red
clLime	Lime green
clBlue	Blue
clFuchsia	Fuchsia
clAqua	Aqua
clWhite	White

*The following colors refer to the default colors set in the current Windows environment:*

clScrollBar	Current color of Windows scrollbar.
clBackground	Current color of Windows background.
clActiveCaption	Current color of the title bar of the active window.
clInactiveCaption	Current color of the title bar of inactive windows.
clMenu	Current background color of menus.
clWindow	Current background color of windows.
clWindowFrame	Current color of window frames.
clMenuText	Current color of text on menus.
clWindowText	Current color of text in windows.
clCaptionText	Current color of the text on the title bar of the active window.
clActiveBorder	Current border color of the active window.
clInactiveBorder	Current border color of inactive windows.
clAppWorkSpace	Current color of the application workspace.
clHighlight	Current background color of selected text.
clHighlightText	Current color of selected text.
clBtnFace	Current color of a button face.
clBtnShadow	Current color of a shadow cast by a button.
clGrayText	Current color of text that is dimmed.
clBtnText	Current color of text on a button.
clInactiveCaptionText	Current color of the text on the title bar of an inactive window.
clBtnHighlight	Current color of the highlighting on a button.

**Message Box Constants (also see MsgBox):**

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<i>MsgBox Buttons:</i>		
MB_OK	0	OK button only.
MB_OKCANCEL	1	OK and Cancel buttons.
MB_ABORTRETRYIGNORE	2	Abort, Retry and Ignore buttons.
MB_YESNOCANCEL	3	Yes, No and Cancel buttons.
MB_YESNO	4	Yes and No buttons.
MB_RETRYCANCEL	5	Retry and Cancel buttons
<i>MsgBox Icons:</i>		
MB_ICONSTOP	16	 Critical message.
MB_ICONQUESTION	32	 Warning query.
MB_ICONEXCLAMATION	48	 Warning message.
MB_ICONINFORMATION	64	 Information message.
<i>MsgBox Defaults:</i>		
MB_APPLMODAL	0	Application Modal Message Box. The user must respond to the message before continuing work in the current application (Default).
MB_DEFBUTTON1	0	First button is default.
MB_DEFBUTTON2	256	Second button is default.
MB_DEFBUTTON3	512	Third button is default.
MB_SYSTEMMODAL	4096	System Modal. All applications are suspended until the user responds to the message box.
<i>MsgBox return values:</i>		
IDOK	1	OK button pressed.
IDCANCEL	2	Cancel button pressed.
IDABORT	3	Abort button pressed.
IDRETRY	4	Retry button pressed.
IDIGNORE	5	Ignore button pressed.
IDYES	6	Yes button pressed.
IDNO	7	No button pressed.

**Print Font Styles (also see PrintSetFont):**

<u>Constant</u>	<u>Value</u>	<u>Description</u>
fsNormal	0	Normal font.
fsFontBold	1	Bold font.
fsFontItalic	2	Italic font.
fsFontUnderline	4	Underlined font.
fsFontStrikeThru	8	Strikethrough font.

**State Types (also see GetState and):**

<u>Constant</u>	<u>Value</u>	<u>Description</u>
tpEnabled	0	Whether the control is enabled or disabled (grayed). Applies to all controls.
tpVisible	1	Whether the control is visible to the user or not. Applies to all controls.
tpChecked	2	Whether the control is checked or unchecked. Applies only to option buttons and check boxes.

## Index

<b>A</b>	
Abs Function.....	85
Accessing an Object .....	79
Action Key Assignment .....	1
Adding Controls and Fields .....	1
Align Property.....	37
Aligning and Sizing Controls .....	1
Alignment .....	1
Alignment Property.....	37
AllowAllUp Property .....	37
AppActivate Statement .....	85
Arithmetic Operators .....	83
Arrays .....	67
Asc Function.....	85
Atn Function.....	85
AutoCenter Property.....	37
Automatic Alignment .....	1
Automatic Properties Assignment .....	1
Automation .....	79
AutoSize Property .....	37
AutoSnap Property .....	37
<b>B</b>	
BackColor Property.....	37
Beep Statement .....	86
Begin Dialog Statement .....	86
Bevel .....	1
BevelInner Property.....	37
BevelOuter Property .....	37
Bevels .....	29
BevelWidth Property.....	37
Bitmap Property.....	37
BitmapPosition Property.....	37
BlinkColor Property.....	38
Blinking Property.....	38
BlinkIntervalOff Property.....	38
BlinkIntervalOn Property.....	38
Border Property .....	38
BorderStyle Property .....	38
BorderWidth Property .....	38
Browser .....	34
Button Group.....	1
Button Groups .....	29
ButtonStyle Property .....	38
ByRef and ByVal .....	63
<b>C</b>	
Calendar Function .....	87
CalendarDialog Function.....	87
Call Statement.....	87
Calling Procedures in DLLs .....	64
Cancel button.....	71
Cancel Property .....	38
CancelButton Statement.....	87
Caption Property.....	38
CBool Function .....	88
CDate Function.....	88
CDBl Function.....	88
Center Property .....	38
ChangeCursorStyle Subroutine .....	89
CharCase Property.....	39
ChDir Statement.....	89
ChDrive Statement .....	90
Check Box .....	1
Check Boxes .....	24
Check Boxes in Dialog.....	73
Check Script .....	51
CheckBox Statement.....	90
Checked Property .....	39
Choose Function .....	90
Chr Function .....	91
CInt Function .....	91
Class.....	80
CLng Function .....	91
Close Statement .....	92
CloseApp Subroutine.....	92
Color Selection .....	45
ColumnHeaders Property .....	39
Columns Property .....	39
ComboBox Statement .....	92
Command Button.....	1
Command Buttons .....	23
Comments .....	55
Compile Script.....	51
Concatenation .....	57
Const Statement .....	92
Constant Names .....	55
Continuation .....	55
Control Structures .....	61
Controls .....	1
Cos Function .....	93
CreateObject Function.....	93
CSng Function .....	94
CStr Function .....	95
Ctl3d Property.....	39
CurDir Function .....	95
CVar Function .....	95

**D**

Data Types.....	57, 83
DataSource Property .....	39
Date Function .....	95
Date Time Format Editor .....	48
Date/Time Label.....	1
Date/Time Labels .....	33
DateSerial Function .....	96
DateValue Function .....	96
Day Function .....	96
Declaration of Variables .....	58
Declare Statement .....	97
Default Property.....	39
Dialog Box Controls .....	76
Dialog Designer .....	51
Dialog Form Action Editor .....	51
Dialog Form Designer .....	1
Dialog Form Designer Toolbar.....	1
Dialog Form Example 1 .....	13
Dialog Form Example 2 .....	14
Dialog Form Example 3 .....	16
Dialog Form Menu Designer.....	49
Dialog Function.....	76, 97
Dialog Support.....	71
DialogResult .....	1
DialogResult Property .....	7
Dim .....	67
Dim Statement .....	99
Dir Function.....	99
DlgEnable Statement .....	99
DlgText Statement .....	100
DlgVisible Statement .....	101
Do Loops.....	61
Do...placeLoop Statement .....	101
Doing a File Transfer .....	14
Double .....	57, 83
Down Property.....	39
Drop-Down List Box.....	1
Drop-down List Boxes .....	26, 72
DropListBox Statement .....	101

**E**

Edit Box .....	1
Edit Boxes .....	22
Edit Mask .....	45
EditMask Property .....	39
Editor Properties .....	53
Enabled Property.....	39
End Statement.....	102
EOF Function .....	102
Erase Statement .....	103
Exit Statement.....	103

Exp Function .....	103
--------------------	-----

**F**

Fields .....	1
File Input/Output .....	65
File Transfer .....	14
FileCopy Function .....	104
FileLen Function .....	104
FileOpenDialog Function .....	104
FileSaveDialog Function.....	104
Fix Function .....	105
Flat Property .....	39
FlatColor Property.....	39
Font Property .....	39
For ... Next placeLoop .....	61
For Each...Next Statement.....	105
For...Next Statement .....	105
ForeColor Property.....	39
Form Activate Action.....	21
Form Designer.....	1
Form Designer Toolbar .....	1
Form Initial Action .....	21
Form Properties.....	21
Format Function .....	105
Format Property .....	39
FrameStyle Property .....	40
FreeDialogForm .....	1
FreeDialogForm Subroutine.....	7
FreeFile Function .....	110
Function Naming Conventions .....	63
Function Statement.....	110

**G**

Get Statement.....	111
GetChecked .....	1
GetChecked Function .....	7
GetColor Function .....	111
GetCtlEnabled .....	1
GetCtlEnabled Function .....	7
GetDialogDoubleVar .....	1
GetDialogDoubleVar Function .....	7
GetDialogIntVar.....	1
GetDialogIntVar Function.....	7
GetDialogLongVar .....	1
GetDialogLongVar Function .....	7
GetDialogStringVar .....	1
GetDialogStringVar Function .....	7
GetNumericProp Function .....	111
GetObject Function .....	112
GetState Function .....	112
GetString.....	1
GetString Function.....	8, 113
GetStringProp Function.....	113



GetVisible.....	1	List Box .....	1
GetVisible Function.....	8	List Boxes.....	25, 72
Global array .....	67	ListAddItem .....	1
Global Statement .....	113	ListAddItem Subroutine.....	8
GoTo .....	61	ListBox Statement .....	122
GoTo Statement.....	114	ListClear .....	1
Group Box.....	1	ListClear Subroutine.....	8, 122
Group Boxes.....	30, 75	ListColHeader Subroutine.....	123
GroupBox Statement .....	114	ListCount Function .....	123
GroupIndex Property .....	40	ListGetColText.....	1
<b>H</b>		ListGetColText Function.....	8, 123
Height Property.....	40	ListGetCount .....	1
HelpContext Property.....	40	ListGetCount Function .....	8
Hex Function .....	114	ListGetIndex .....	1
Hint Property.....	40	ListGetIndex Function .....	9, 123
Host Error Action.....	21	ListGetItem.....	1
Host Message Action.....	21	ListGetItem Function.....	9, 123
Hour Function .....	115	ListItemAdd Subroutine.....	124
<b>I</b>		ListItemRemove Subroutine .....	124
If and Select Statements.....	62	ListSetColHeader .....	1
If...Then...Else Statement .....	115	ListSetColHeader Subroutine .....	9
Image.....	1	ListSetColText .....	1
ImageOpenDialog Function.....	116	ListSetColText Function .....	9
Images .....	32	ListSetColText Subroutine.....	124
ImageSaveDialog Function .....	116	ListSetIndex.....	1
Input Edit Mask.....	45	ListSetIndex Subroutine .....	9, 124
Input Function .....	116	ListSetItem .....	1
InputBox Function .....	117	ListSetItem Subroutine.....	9, 124
InStr Function.....	117	LoadDialogForm .....	1
Int Function.....	118	LoadDialogForm Function.....	10
Integer .....	57, 83	LoadImage .....	1
IsArray Function .....	118	LoadImage Function.....	124
IsDate Function.....	118	LoadImage Subroutine .....	10
IsEmpty Function .....	118	LoadMMFile Function .....	125
IsNull Function.....	118	Local array.....	67
IsNumeric Function .....	119	LOF Function .....	125
IsObject Function .....	119	Log Function .....	125
ItemIndex Property .....	40	Logical Operators.....	83
Items Property.....	40	Long .....	57, 83
<b>K</b>		LTrim Function .....	155
Kill Statement.....	119	<b>M</b>	
<b>L</b>		MaximizedButton Property .....	40
LBound Function .....	120	MaxLength Property .....	40
LCase Function .....	120	Media Player .....	1
Left Function .....	121	Media Players.....	33
Left Property .....	40	Memo.....	28
Len Function.....	121	Menu Designer .....	1, 49
Let Statement.....	121	Methods .....	1, 80
Line Continuation Character.....	55	Mid Function .....	126
Line Input # Statement .....	122	MinimizedButton Property .....	40
Lines Property.....	40	MinSize Property.....	40

Minute Function .....	126	Precedences .....	83
MkDir Statement .....	126	Predefined Constants .....	161
MonoChromeButtons Property .....	40	Print # Statement .....	133
Month Function .....	127	Print Statement .....	135
Moving and Sizing Controls .....	1	PrintBeginDoc Subroutine .....	136
MsgBox Function .....	127	PrintDlg Function .....	136
MsgBox Statement .....	127	PrintDraw Subroutine .....	136
Multi-Column List Box .....	1	PrintEndDoc Subroutine .....	136
Multi-column List Boxes .....	27	PrintMoveTo Subroutine .....	137
Multi-Column List Setup .....	46	PrintNewPage Subroutine .....	137
<b>N</b>		PrintPageHeight Function .....	137
Name Property .....	41	PrintPageWidth Function .....	137
Name Statement .....	128	PrintRect Subroutine .....	137
Naming Conventions .....	63	PrintSetFont Subroutine .....	137
Now Function .....	129	PrintSetFontSize Subroutine .....	138
Numbers .....	55	PrintSetFontStyleSubroutine .....	138
NumBitMaps Property .....	41	PrintSetOrientation Subroutine .....	138
NumericSort Property .....	41	PrintTextHeight Function .....	138
<b>O</b>		PrintTextWidth Function .....	139
Object .....	79	Properties .....	80
Oct Function .....	129	Properties Assignment .....	1
OK button .....	71	Property .....	1
OKButton Statement .....	129	PushButton Statement .....	139
OLE Automation .....	79, 80	Put Statement .....	140
OLE Automation and Word 6.0 example .....	81	<b>Q</b>	
OLE Fundamentals .....	80	Operator Precedence .....	83
OLE Object .....	80	<b>R</b>	
On Error Statement .....	129	Randomize Statement .....	140
Open Statement .....	131	ReadOnly Property .....	41
Operator Precedence .....	83	ReDim Statement .....	140
Option Base .....	67	Relational Operators .....	83
Option Base Statement .....	132	Rem .....	55
Option Button .....	1	Rem Statement .....	141
Option Buttons .....	25, 75	Right Function .....	141
Option Explicit Statement .....	132	Rmdir Statement .....	142
OptionButton Statement .....	133	Rnd Function .....	142
OptionGroup Statement .....	133	RTrim Function .....	155
Other Data Types .....	57	<b>S</b>	
<b>P</b>		Scope of Variables .....	58
Panel .....	1	Screen Complete Check Action .....	21
Panels .....	31	ScriptDir Function eXpress Plus .....	142
Parent Controls .....	45	ScrollBars Property .....	41
ParentColor Property .....	41	Second Function .....	143
ParentCtl3d Property .....	41	Seek Function .....	143
ParentFont Property .....	41	Seek Statement .....	143
ParentShowHint Property .....	42	Select Case Statement .....	144
Passing and Retrieving Variables .....	13	Select Statements .....	62
PasswordChar Property .....	41	SendKeys Statement .....	145
Picture Property .....	41	SendMail Subroutine .....	146
Player .....	1	Set Statement .....	146
Pointer button .....	1	SetChecked .....	1

SetChecked Subroutine ..... 10  
 SetColor Subroutine ..... 147  
 SetCtlEnabled ..... 1  
 SetCtlEnabled Subroutine ..... 10  
 SetDialogDoubleVar ..... 1  
 SetDialogDoubleVar Subroutine ..... 10  
 SetDialogIntVar ..... 1  
 SetDialogIntVar Subroutine ..... 10  
 SetDialogLongVar ..... 1  
 SetDialogLongVar Subroutine ..... 10  
 SetDialogStringVar ..... 1  
 SetDialogStringVar Subroutine ..... 10  
 SetFocus Subroutine ..... 148  
 SetNumericProp Subroutine ..... 148  
 SetState Subroutine ..... 149  
 SetString ..... 1  
 SetString Subroutine ..... 11, 149  
 SetStringProp Subroutine ..... 149  
 SetVisible ..... 1  
 SetVisible Subroutine ..... 11  
 Sgn Function ..... 149  
 Shape Property ..... 41  
 Shell Function ..... 150  
 ShowAccelChar Property ..... 41  
 ShowDialogForm ..... 1  
 ShowDialogForm Subroutine ..... 11  
 ShowHint Property ..... 42  
 Sin Function ..... 150  
 Single ..... 57, 83  
 Sizing Controls ..... 1  
 SortColumn Property ..... 42  
 SortDescending Property ..... 42  
 Sorted Property ..... 42  
 Space Function ..... 150  
 Speed Button ..... 1  
 Speed Buttons ..... 23  
 Splitter ..... 1  
 Splitters ..... 32  
 Sqr Function ..... 150  
 Statements ..... 55  
 Static array ..... 67  
 Static Statement ..... 151  
 Stop Statement ..... 151  
 Str Function ..... 152  
 StrComp Function ..... 152  
 Stretch Property ..... 42  
 String ..... 57, 83  
 String Function ..... 152  
 Style Property ..... 42  
 Sub Statement ..... 153  
 Subroutine and Function Naming Conventions ..... 63

**T**  
 TabOrder Property ..... 42  
 TabStop Property ..... 42  
 Tan Function ..... 153  
 Text ..... 74  
 Text Boxes ..... 74  
 Text Label ..... 1  
 Text Labels ..... 21  
 Text Property ..... 42  
 Text Statement ..... 153  
 TextBox Statement ..... 154  
 Time Function ..... 154  
 Time Statement ..... 154  
 Timer Event ..... 154  
 TimeSerial Function ..... 155  
 TimeValue Function ..... 155  
 Top Property ..... 42  
 Transaction Processing with Dialogs ..... 16  
 Transparent Property ..... 42  
 Trim Function ..... 155  
 Type Statement ..... 156

**U**  
 UBound Function ..... 157  
 UCase Function ..... 157  
 upload ..... 122  
 URL Link ..... 34  
 URL Property ..... 42  
 User Defined Types ..... 69  
 Using Dialog Forms ..... 1

**V**  
 Val Function ..... 158  
 Values ..... 63  
 Variable and Constant Names ..... 55  
 Variable Types ..... 57  
 Variables ..... 58, 69  
 Variant ..... 57, 83  
 Variants and Concatenation ..... 57  
 VarType Function ..... 158  
 Visible Property ..... 42

**W**  
 Wait Subroutine ..... 158  
 WantsReturns Property ..... 42  
 Weekday Function ..... 158  
 What is an OLE Object? ..... 80  
 What is OLE Automation? ..... 79  
 While placeLoop ..... 61  
 While...Wend Statement ..... 159  
 Width Property ..... 43  
 WindowState Property ..... 43  
 WinHelpFile Property ..... 43  
 With Statement ..... 159

WordWrap Property ..... 43  
Write # - Statement ..... 160

**Y**

Year Function ..... 160