Microsoft® Excel for Windows®

Microsoft®

Product Support Services
Application Note

Versions: 5.0, 5.0c

Subject: **Sample Visual Basic® Code for Controlling Dialog Boxes**

Contents: 18 Pages, 1 Disk

5/95 - WE1162

Table Of Contents

Overview

This Application Note will help you learn some techniques for writing Visual Basic, Applications Edition, code for use with custom dialog boxes. The DLOGSMPL.XLS file included with this Application Note contains code examples you can run. These code examples can be used with the following elements: labels, edit boxes, group boxes, buttons, check boxes, option buttons, list boxes, drop-down boxes, combination list-edit boxes, combination drop-down edit boxes, scroll bars, and spinners. The DLOGSMPL.XLS file also contains code that demonstrates how to: set the focus in a dialog box, create a "masked" password dialog box, adjust the dialog box height, and use the **.Caller** property.

The text of this Application Note includes additional explanations for some of these elements, and it also points you to other sources of information. Each section of the text contains an introduction for a particular control, followed by some sample Visual Basic code. Some sections also contain a brief description of the commonly used properties and methods for that control.

Note that although the examples in this Application Note are created to illustrate the use of controls on a dialog sheet, many of these controls can be used on a worksheet or a chart sheet if you make minor changes to the code. For more information about using controls in other types of sheets, see Chapter 13 of the *User's Guide* or Chapter 11 of the *Visual Basic User's Guide*.

Microsoft provides examples of Visual Basic procedures for illustration only, without warranty either expressed or implied, including but not limited to the implied warranties of merchantability and/or fitness for a particular purpose. These Visual Basic procedures are provided 'as is' and Microsoft does not guarantee that they can be used in all situations. Microsoft does not support modifications of these procedures to suit customer requirements for a particular purpose.

How to Use the Examples in This Application Note

Using the DLOGSMPL.XLS File

To use the DLOGSMPL.XLS file, copy it to a directory on your hard disk drive, and open it in Microsoft Excel.

To install DLOGSMPL.XLS on your hard disk

1. If you received a disk with this Application Note, insert the disk in the appropriate floppy disk drive. If you downloaded this Application Note from an online service, skip to step 2. This procedure assumes that you have already downloaded and extracted the compressed file.

2. Copy the DLOGSMPL.XLS file from the WE1162 disk (or from the directory to which you downloaded and extracted WE1162.EXE) to the appropriate directory on your hard disk drive.

For more information about copying files, see your Windows *User's Guide* or Windows online Help.

To use DLOGSMPL.XLS

1. Start Microsoft Excel, and choose Open from the File menu.

2. Select the directory to which you installed the DLOGSMPL.XLS file, select DLOGSMPL.XLS, and choose the Open button.

Using the Examples in the Text of This Application Note

The examples in the text portion of this Application Note demonstrate ways you can use Visual Basic, Applications Edition, code to accomplish basic tasks with dialog box controls.

To use this code, set up your workbook as follows:

- The module containing the code must be located in the same workbook as the dialog sheet that contains the controls.

- The dialog sheet that contains the controls must be named "Dialog1" and must be the first dialog sheet in the workbook.

In some examples, to run the sample code, you can choose Macro from the Tools menu, select the name of the macro, and choose the Run button. In other examples, you will run the code by assigning it to a control (usually a button) in the dialog box, running the dialog box, and then choosing the control. To run a dialog box, select the dialog sheet that contains the dialog box and choose the Run Dialog button on the Forms toolbar. (Examples that require you to run the code this way contain instructions about which control to assign the code to.)

For more information about associating a macro with a button, see the "Assigning Code to Controls and Dialog Boxes" section of the *Visual Basic User's Guide*.

For more information about running a macro, see pages 14–15 of the *Visual Basic User's Guide,* or choose the Search button in Help and type:

```
macros, running
```

Also, note that in the sample code, objects are referred to by their ordinal numbers rather than by name. For example, in code referring to the first button in a dialog box, the button is referred to as Buttons(1) rather than as Buttons("Button 12").

Examples of Visual Basic Code to Use with Dialog Box Controls

Labels

Labels are mainly used to add descriptive text to other controls on a dialog box. The most commonly used properties of this control are the **.Text** and the **.Caption** properties, which are interchangeable. Labels cannot be edited by the user while the dialog box is running, although a macro can make changes to the text at any time.

> ***Note:*** *The font size of the text in a label is fixed. If you want to have a control that resembles a label but has a font size and background color that can be changed, use a text box. The Text Box button is located on the Drawing toolbar.*

To use a With statement to change the text property of a label

1. Create a dialog box with at least three label controls.

2. Enter the following code in a Visual Basic module:

```
Sub LabelExample1()
   With ThisWorkbook.DialogSheets("Dialog1")
      labels(1).Text = "Labels can be an empty string"
      labels(2).Text = ""
      labels(3).Text = "...such as Label 2 above."
   End With
End Sub
```

To change the .Text property of a label using a For Each...Next statement

1. Create a dialog box with at least one label control.

2. Enter the following code in a Visual Basic module:

```
Sub LabelExample2()
   Dim Label As Variant
   For Each Label In ThisWorkbook.DialogSheets(1).labels
      Label.Caption = "Example text for" & Label.Name
   Next Label
```
End Sub

To cycle through (index) a collection of dialog box labels

1. Create a dialog box with at least three label controls.

2. Enter the following code in a Visual Basic module:

```
Sub LabelExample3()
   Dim MyDlgLbls As Object, x as Integer
   Set MyDlgLbls = DialogSheets(1).labels
   For x = 1 To 3
      MyDlgLbls(x).Text = Format(Now + x, "dddd - mmmm,yyyy hh:mm AM/PM")
   Next
End Sub
```

Edit Boxes

Edit boxes are framed boxes that accept user input when the dialog box is running. The font size for text in an edit box is fixed (that is, it cannot be customized).

If you want text to wrap in an edit box, set the **.MultiLine** property to **true**. To restrict the type of information the edit box will accept (its input type), use the **.InputType** property. To set these properties, use macro code, or select the edit box and choose Object from the Format menu. If you set the **.InputType** property with code, you can use the constants **xlFormula, xlInteger, xlNumber, xlReference,** and **xlText.** The default input type is text (or **xlText**).

To retrieve the text in an edit box that has restricted input type

1. Create a dialog box with at least one edit box control.

2. Enter the following code in a Visual Basic module:

```
Sub EditExample1()
   Dim theText As String
   With ThisWorkbook.DialogSheets(1).EditBoxes(1)
      .InputType = xlInteger
      theText = .Text
   End With
   MsgBox theText,,"Edit Box Value"
End Sub
```

To display information in an edit box

1. Create a dialog box with at least one edit box.

2. Enter the following code in a Visual Basic module

```
Sub EditExample2()
  ThisWorkbook.DialogSheets(1).EditBoxes(1).Text = "123"
End Sub
```

Group Boxes

Group boxes allow you to group controls on a dialog sheet, worksheet, or chart sheet. Group boxes are especially useful for grouping option buttons—when option buttons are in a group box, only one of the option buttons can be selected at a time. The font size and background color of a group box cannot be changed.

Although group boxes display a group of objects, changes to the group box do not affect the objects within that group box (that is, the objects within a group box do not become a collection).

To change the text that is displayed along the top edge of a group box control, use the **.Caption** property.

To change the caption text of a group box

1. Create a dialog box with at least one group box control.

2. Enter the following code in a Visual Basic module:

```
Sub GroupBoxExample1()
   With DialogSheets(1).GroupBoxes(1)
      If .Caption = "Example Caption Text" Then
         .Caption = "Another Example Caption Text"
      Else
         .Caption = "Example Caption Text"
      End If
   End With
End Sub
```

To hide a group box or make a group box visible

1. Create a dialog box with a group box control.

2. Enter the following code in a Visual Basic module:

```
Sub GroupBoxExample2()
   With DialogSheets(1).GroupBoxes(1)
      If .Visible = True Then .Visible = False Else .Visible = True
   End With
End Sub
```

3. Add a button control and assign the GroupBoxExample2 macro to the button.

Buttons

Buttons are primarily used for triggering an event. Some commonly used button properties include: **.DefaultButton, .CancelButton, .DismissButton,** and **.HelpButton.** You can set these properties using code, or you can set them by selecting the button, choosing Object from the Format menu, and selecting the Default, Cancel, Dismiss, and Help check boxes. These properties control what happens to the dialog box when the user chooses a button in that dialog box. For example, if you set a **.DefaultButton** property to **true,** that button will be activated when the user presses the ENTER key (if no other button has the focus on that dialog box). Only one button in a dialog box can have the **.DefaultButton** property. For additional information, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID:	Q105552
TITLE :	XL5: Static and Dynamic Default Buttons

You cannot change the font or color of a button. However, once you place the control on a dialog sheet, you *can* paste a picture over the button to accomplish a similar effect. For additional information, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID:	Q115046
TITLE :	XL5: Customizing Button Fonts on Dialog Sheets

To use the .OnAction property to assign a macro to a button

1. Create a dialog box with at three button controls.

2. Enter the following code in a Visual Basic module:

```
Sub ButtonExample1()
   DialogSheets(1).Buttons(3).OnAction = "ChangeText"
End Sub
```

3. To run the ButtonExample1 macro, you must also create the following function macro (this macro is assigned to the third button on the dialog box by the ButtonExample1 macro with the .OnAction property):

```
Function ChangeText()
   With DialogSheets(1).Buttons(3)
      If .Text = "Run" Then
         .Text = "Stop"
      Else
         .Text = "Run"
      End If
   End With
End Function
```

To associate a Help file with a dialog box button

1. Create a dialog box with at least three button controls.

2. Enter the following code in a Visual Basic module:

```
Sub ButtonExample2()
   With DialogSheets(1).Buttons(3)
      .Text = "Help"
      .HelpButton = True
      .OnAction = "GetHelp"
   End With
End Sub

Sub GetHelp()
   Application.Help "C:\WINDOWS\CALC.HLP"
End Sub
```

Note: *In the code example above, because the Help property for button 3 is set to* ***True****, if you run the dialog box and press F1 (or choose Help), Windows Calculator Help is displayed.*

For additional information, please see the following articles in the Microsoft Knowledge Base:

ARTICLE-ID:	Q109424
TITLE :	XL5: How to Use the Help Button in Custom Dialog Boxes
ARTICLE-ID:	Q116059
TITLE :	XL5: Pressing F1 on Custom Dialog May Not Activate Help Button

To assign an accelerator key to a button

If the letter to which you assign the accelerator key exists in the text of the button, that letter will be underlined only when the dialog box is run. Accelerator keys are activated by pressing the letter on the keyboard or by pressing the letter in combination with the ALT key. An accelerator key is not case sensitive.

1. Create a dialog box with at least one button control.

2. Enter the following code in a Visual Basic module:

```
Sub ButtonExample3()
   DialogSheets(1).Buttons(1).Text = "Test"
   With DialogSheets(1).Buttons(1)
      If .Accelerator = "T" Then
         .Accelerator = "e"
      Else
         .Accelerator = "T"
      End If
   End With
End Sub
```

When you run this code, the accelerator key for Button 1, which has the **.Text** property set to "Text," is toggled between "T" and "e."

> ***Note:*** *To set the accelerator key manually, choose Object from the Format menu, select the Control tab, and type the letter you want to assign to the accelerator key in the Accelerator Key box.*

Check Boxes

Check boxes enable the user to select an item. When you use check boxes in a group box, the user can select one or more items in the group. The font and color of a check box are fixed (that is, they cannot be customized). A check box can be in one of three states: on, off, or mixed. To indicate the state of a check box, set the **.Value**. property with the **xlOn, xlOff,** or **xlMixed** constant.

To obtain the value of the first check box in the dialog box

1. Create a dialog box with at least one check box control.

2. Enter the following code in a Visual Basic module:

```
Sub CheckBoxExample1()
    If DialogSheets(1).CheckBoxes(1).Value = xlOn Then
       MsgBox "Is checked."
    Else
       MsgBox "Is not checked."
    End If
End Sub
```

3. Assign the CheckBoxExample1 macro to the check box you created in step 1.

To discern which check boxes are selected

1. Create a dialog box with at least two check box controls.

2. Enter the following code in a Visual Basic module:

```
Sub CheckBoxExample2()
   Dim myCheckBoxes As Object
   Dim chk As Variant
   Set myCheckBoxes = DialogSheets(1).CheckBoxes
   For Each chk in myCheckBoxes
      If chk = xlOn Then MsgBox chk.Name & " is selected."
   Next
End Sub
```

3. Assign the CheckBoxExample2 macro to the check boxes you created in step 1.

To discern the state of check boxes on a dialog box

1. Create a dialog box with at least three check box controls. Do the following to assign a different state to each check box:

   a. Select a check box.

   b. From the Format menu, choose Object, select the Control tab, and select the Unchecked, Checked, or Mixed option.

   c. Repeat steps a and b for each check box so that each box is assigned a different state.

2. Enter the following code in a Visual Basic module:

```
Sub CheckBoxExample3()
   Dim myCheckBoxes As Object, chk As Variant
   Set myCheckBoxes = DialogSheets(1).CheckBoxes
   For Each chk in myCheckBoxes
      Select Case chk
      Case xlOn
         MsgBox chk.Name & " is Checked."
      Case xlOff
         MsgBox chk.Name & " is Unchecked."
      Case xlMixed
         MsgBox chk.Name & " is Mixed."
      End Select
   Next
End Sub
```

3. Add a button control and assign the CheckBoxExample3 macro to the button.

Option Buttons

Unlike check boxes, only one option button in a group can be selected at a time. To separate option buttons into groups, create a group box, and then draw the option buttons inside the box. The font and background color of an option button are fixed (that is, they cannot be customized).

To discern which option button is on

1. Create a dialog box with at least two option buttons.

2. Enter the following code in a Visual Basic module:

```
Sub OptionExample1()
   Dim myButtons As Object, btn As Variant
   Set mybuttons = DialogSheets(1).OptionButtons
   For Each btn In mybuttons
      If btn = xlOn Then MsgBox btn.Name & " is selected."
   Next
End Sub
```

3. Add a button control and assign the OptionExample1 macro to the button.

To select or clear the first option button

1. Create a dialog box with at least one option button control.

2. Enter the following code in a Visual Basic module:

```
Sub OptionExample2()
   With DialogSheets(1).OptionButtons(1)
      If .Value = xlOn Then: .Value = xlOff: Else .Value = xlOn
   End With
End Sub
```

3. Add a button control and assign the OptionExample2 macro to the button.

List Boxes

List boxes present the user with a list of scrollable items that can be selected. Commonly used list box methods are **.RemoveItem** and **.RemoveAllItems.** Note that these methods do not work if the list box is linked to a worksheet. If you use a macro to set the selected property in a list box item to **false**, it will not be reflected in the list box while the dialog box is running. The font in a list box is fixed (that is, it cannot be customized).

To populate a list box with cells on a worksheet using the .ListFillRange method

1. Create a dialog box with at least one list box control.

2. In cells A1:A10 on sheet1, type the values that you want to use to populate the list box.

3. Enter the following code in a Visual Basic module:

```
Sub ListBoxExample1()
   DialogSheets(1).ListBoxes(1).ListFillRange = "Sheet1!A1:A10"
End Sub
```

To populate a list box using an array of data

1. Create a dialog box with at least one list box control.

2. Enter the following code in a Visual Basic module:

```
Sub ListBoxExample2()
   DialogSheets(1).ListBoxes(1).List = Array("Mon", "Tue", "Wed", "Thu",
"Fri")
End Sub
```

To return all items in a list box using a For Each...Next statement

1. Create a dialog box with one list box control, and populate the list box using either ListBoxExample1 or ListBoxExample2.

2. Enter the following code in a Visual Basic module:

```
Sub ListBoxExample3()
   Dim mTemp As Object, myList As Variant, LItem As Variant
   Set mTemp = DialogSheets(1).ListBoxes(1)
      myList = mTemp.List
   For Each LItem In myList
      MsgBox LItem
   Next
End Sub
```

3. Add a button control to the dialog box and assign the ListBoxExample3 macro to the button.

   ***Note**: Using a For Each...Next statement with a list box may cause an error if you don't use an object for the list box. For additional information, please see the following article in the Microsoft Knowledge Base:*

   ARTICLE-ID: Q112330
   TITLE : XL5: 'For Each Item in List' Doesn't Work

To obtain the selected item in a single-select list box

1. Create a dialog box with at least one list box control, and populate the list box using either ListBoxExample1 or ListBoxExample2.

2. Enter the following code in a Visual Basic module:

```
Sub ListBoxExample4()
   Dim theContents As String
   With DialogSheets(1).ListBoxes(1)
      theContents = .List(.ListIndex)
   End With
   MsgBox theContents
End Sub
```

3. Add a button control to the dialog box and assign the ListBoxExample4 macro to the button you created in step 1.

To obtain the selected items of a multi-select list box

1. Create a dialog box with one list box control, and populate the list box using either ListBoxExample1 or ListBoxExample2.

2. Enter the following code in a Visual Basic module:

```
Sub ListBoxExample5()
   Dim CurList As Object, ListTemp As Variant, ListItem As Variant
   Dim MultiList As ListBox, counter As Integer
   'Set an object name for easy referencing of the list box.
   Set CurList = DialogSheets(1).ListBoxes(1)
   'Put the selected array into the variable ListTemp
   ListTemp = CurList.Selected
   'Initialize a counter variable.
```

```
        counter = 1
        'Iterate through the loop once for each item in the array.
        For Each ListItem In ListTemp
           'If the value of the current item is True . . .
           If ListItem = True Then
              'Show a message box indicating the item is selected.
               MsgBox CurList.List(counter)
           End If
           'Increment the counter to get the next selected item.
           counter = counter + 1
        Next
     End Sub
```

3. Add a button control to the dialog box and assign the ListBoxExample5 macro to the button.

For additional information, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID:	Q111564
TITLE :	XL5: Determining Which Items Are Selected in a List Box

To use a horizontal array of cells on a worksheet to populate a list box

1. Create a dialog box with one list box control.

2. In the range A1:F1 of sheet1 of your workbook, enter the values that you want to appear in the list box.

3. Enter the following code in a Visual Basic module:

```
     Sub ListBoxExample6()
        DialogSheets(1).ListBoxes(1).List = Worksheets("Sheet1").Range("A1:F1")
     End Sub
```

Note: *Ordinarily, list boxes are populated with a column of data. The above example makes it possible to populate a list box with a row of data.*

To clear all items in a list box using the .RemoveAllItems method

1. Create a dialog box with one list box control, and populate the list box using either ListBoxExample1 or ListBoxExample2 above.

2. Enter the following code in a Visual Basic module:

```
     Sub ListBoxExample7()
       DialogSheets(1).ListBoxes(1).RemoveAllItems
     End Sub
```

3. Place a button control on the dialog box and assign the ListBoxExample7 macro to the button.

Drop-Down List Boxes

A drop-down list box allows the user to select a single item from a list. The main difference between a drop-down list box and a regular list box is the amount of space the control takes up in the dialog box.

To add items to a drop-down list box using values on a worksheet with .ListFillRange

1. Create a dialog box with at least one drop-down list box control.

2. Type the values that will appear in the drop-down list box into cells A2:A10 on sheet1 of your workbook.

3. Enter the following code in a Visual Basic module:

```
Sub DropDownExample1()
   DialogSheets(1).DropDowns(1).ListFillRange = "Sheet1!A2:A10"
End Sub
```

To return the selected item of a drop-down list box

1. Create a dialog box with one drop-down list box control, and populate the drop-down list box using the DropDownExample1 code.

2. Enter the following code in a module:

```
Sub DropDownExample2()
   Dim theContents As String
   With DialogSheets(1).DropDowns(1)
      theContents = .List(.ListIndex)
   End With
   MsgBox theContents
End Sub
```

3. Add a button control to the control box and assign the DropDownExample2 macro to the button.

To clear all items from a drop-down list box

1. Create a dialog box with one drop-down list box control, and populate the drop-down list box using DropDownExample1.

2. Enter the following code in a module:

```
Sub DropDownExample3()
   DialogSheets(1).DropDowns(1).RemoveAllItems
End Sub
```

3. Add a button control to the dialog box and assign the DropDownExample3 macro to the button.

Combination List-Edit Boxes

A combination list-edit box is similar to a standard list box, except that there is an edit box associated with the list box. The edit box portion of the combination list-edit box contains the value selected in the list portion. This value can be edited and subsequently added to the list box. Note that combination list-edit boxes cannot be used on a worksheet. For additional information, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID:	Q104303
TITLE :	XL5: Some Limitations for Controls on Sheets and Dialogs

Although a combination list-edit box is a built-in dialog box element, it can also be created by placing an edit box and a list box on a dialog sheet and then using the following code to link the two objects:

```
ActiveDialog.DrawingObjects(Array("List Box 1", "Edit Box 1")).LinkCombo
```

To obtain the selected value in the list box portion of a combination list-edit box

1. Create a dialog box with one combination list-edit box control.

2. To populate the combination list-edit box, choose Object from the Format menu, select the Control tab, and then enter a cell reference in the Input Range box.

3. Enter the following code in a Visual Basic module:

```
Sub ListEditExample1()
   Dim myAnswer As String
   With DialogSheets(1).ListBoxes(1)
      myAnswer = .List(.ListIndex)
      MsgBox myAnswer,,"Selected List Item"
   End With
End Sub
```

4. Place a button control on the dialog box and assign the ListEditExample1 macro to the button.

   Note: *The ListEditExample1 macro is exactly the same method used to obtain the selected value of a list box. The edit box portion of the combination list-edit box is the same as an edit box control.*

To obtain the value in the edit box portion of a combination list-edit box

1. Create a dialog box with one combination list-edit box control.

2. To populate the combination list-edit box, choose Object from the Format menu, select the Control tab, and then enter a cell reference in the Input Range box.

3. Enter the following code in a Visual Basic module:

```
Sub ListEditExample2()
   Dim myText As String
   myText = DialogSheets(1).EditBoxes(1).Text
   MsgBox myText,,"Edit Box Value"
End Sub
```

4. Add a Button control to the dialog box and assign the ListEditExample2 macro to the button.

To add the edit box value to the list box portion of a combination list-edit box

1. Create a dialog box with one combination list-edit box control.

2. To populate the combination list-edit box, choose Object from the Format menu, select the Control tab, and then enter a cell reference in the Input Range box.

3. Enter the following code in a Visual Basic module:

```
Sub ListEditExample3()
   Dim theText As String
   theText = DialogSheets(1).EditBoxes(1).Text
   DialogSheets(1).ListBoxes(1).AddItem Text:=theText
End Sub
```

4. Add a button control to the dialog box and assign the ListEditExample3 macro to the button.

   Note: *The* ***.AddItem*** *method will clear any* ***.ListFillRange*** *that is being used. If you want to add an item to an existing list that comes from a range of cells on a worksheet, you need to place the edit box value into the appropriate cell and then redefine the* ***.ListFillRange*** *of the combination list-edit box.*

Combination Drop-Down Edit Boxes

A combination drop-down edit box is similar to a standard drop-down box except that the text in the caption portion of the drop-down box can be edited. A combination drop-down edit box cannot be used on a worksheet.

To add items to a combination drop-down edit box using .ListFillRange

1. Create a dialog box with one combination drop-down edit box control.

2. Enter the following code in a Visual Basic module:

```
Sub ComboDropDownExample1()
  DialogSheets(1).DropDowns(1).ListFillRange = "MyWorksheet!A2:A10"
End Sub
```

To return the selected item from a combination drop-down edit box

1. Create a dialog box with one combination drop-down edit box control.

2. To populate the combination drop-down edit box, choose Object from the Format menu, select the Control tab, and then enter a cell reference in the Input Range box.

3. Enter the following code in a module:

```
Sub ComboDropDownExample2()
   Dim textAnswer As String
   With DialogSheets(1).DropDowns(1)
      textAnswer = .List(.ListIndex)
      MsgBox textAnswer
   End With
End Sub
```

4. Add a button control to the dialog box and assign the ComboDropDownExample2 macro to the button.

   Note: *When an item in a combination drop-down edit box has been edited, don't try to get the value of .List(.ListIndex). The* ***.ListIndex*** *property has a value of 0 in this case, and List(0) results in an error because there is no element 0.*

To add the edited text value to the drop-down list

1. Create a dialog box with one combination drop-down edit box control.

2. To populate the drop-down list, choose Object from the Format menu, select the Control tab, and then enter a cell reference in the Input Range box.

3. Enter the following code in a module:

```
Sub ComboDropDownExample3()
   Dim captionText As String
   captionText = DialogSheets(1).DropDowns(1).Caption
   DialogSheets(1).DropDowns(1).AddItem Text:=captionText
   MsgBox captionText & " has been added to the list.",,"Add Item"
End Sub
```

4. Add a button control to the dialog box and assign the ComboDropDownExample3 macro to the button.

> ***Note:*** *The* ***.AddItem*** *method will clear any ListFillRange that is being used. If you want to add an item to an existing list, and if the list comes from a range of cells on a worksheet, you need to place the edit box value into the appropriate cell and then redefine the* ***.ListFillRange*** *of the combination drop-down edit box.*

To clear all items from a combination drop-down edit box

1. Create a dialog box with one combination drop-down edit box control.

2. To populate the combination drop-down edit box, choose Object from the Format menu, select the Control tab, and then enter a cell reference in the Input Range box.

3. Enter the following code in a Visual Basic module:

```
Sub ComboDropDownExample4()
   DialogSheets(1).DropDowns(1).RemoveAllItems
End Sub
```

4. Add a button control to the dialog box and assign the ComboDropDownExample4 macro to the button.

Scroll Bars

You can create a vertical or horizontal scroll bar. To create a horizontal scroll bar, press and hold the CTRL key when you choose the Scroll Bar tool. In general, a scroll bar is used to increment or decrement the value of a cell on a worksheet, which in turn changes all the cells linked to that cell in a "what-if" scenario.

To obtain the value of the scroll bar

1. Create a dialog box with one edit box.

2. Enter the following code in a module:

```
Sub ScrollBarExample1()
  DialogSheets(1).EditBoxes(1).Text = DialogSheets(1).ScrollBars(1).Value
End Sub
```

3. Add a scroll bar control to the dialog box and assign ScrollBarExample1 macro to the scroll bar.

Spinners

A spinner is similar to a scroll bar, except that a spinner does not have the LargeChange property. Spinners are often placed next to edit boxes so that the user can increment or decrement a value without having to type in a number. For an edit box to have an associated spinner control, create a separate spinner object and add the code to link the spinner value to the edit box.

To associate a spinner with an edit box

1. Create a dialog box with one edit box and one spinner control.

2. Enter the following code in a Visual Basic module:

```
Sub SpinnerExample()
   DialogSheets(1).EditBoxes(1).Text = DialogSheets(1).Spinners(1).Value
End Sub
```

3. Add a spinner control to the dialog box and assign the SpinnerExample macro to the spinner.

Other Examples and Tips

Avoiding the "Out of Stack Space" Error Message

In Microsoft Excel, when you choose a control in a dialog box that is assigned to an event macro when there are a total of three dialog boxes on the screen that have not been dismissed, you may receive the following error message(s):

Not Enough Stack Space to Run Macro

-or-

Error 28: Out of Stack Space

For information about how to avoid these error messages, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID:	Q111867
TITLE :	XL5 Err Msg: "Not Enough Stack Space to Run Macro"

Using the .Focus Property

Use the **.Focus** dialog box property when you want a specific control to be active when the dialog box is run. A dialog box needs to be running before the **.Focus** property can be set. In other words, you cannot set the **.Focus** property and then run the dialog box. The **.Focus** property must be set as the dialog box is opening or while it is running.

> ***Note:*** *The control that has the initial focus when a dialog box is run can also be set by moving the control name to the top of the tab order. To set the tab order, activate the dialog sheet, and choose Tab Order from the Tools menu.*

To set the focus of an edit box

1. Create a dialog box with four edit boxes.

2. In a Visual Basic module, enter the following code:

```
Sub FocusExample1()
   DialogSheets(1).Focus = "Edit Box 4"
End Sub

Sub FocusExample2()
   DialogSheets(1).Focus = ActiveDialog.EditBoxes(1).Name
End Sub
```

3. To use one of the code examples, assign either macro to the dialog frame.

Using an Edit Box As a Password Entry Control

Although edit boxes do not contain the "show password as asterisks" feature that is available in Microsoft Visual Basic 3.0, this capability can be emulated by placing an edit box on the dialog sheet, outside of the dialog frame. By placing the edit box in this manner, you can create code that will show asterisks in an edit box placed within the dialog frame when the user types in the password. In the following code, Edit Box 4 is the edit box that has been placed outside of the dialog frame.

Assign this macro to the hidden edit box that is outside of the dialog frame:

```
Sub DisplayAsterisks()
  Var1 = DialogSheets(1).EditBoxes("Edit Box 5").Text
  DialogSheets(1).EditBoxes("Edit Box 5").Text = Var1 & "*"
End Sub
```

Assign this subroutine to the dialog frame of the dialog sheet:

```
Sub SetFocus()
  DialogSheets(1).Focus = "Edit Box 4"
End Sub
```

For additional information, please see the following article in the Microsoft Knowledge Base:

ARTICLE-ID:	Q125422
TITLE :	XL5: Creating a Masked Password Dialog Box in Visual Basic

Changing the Height of the Dialog Box Frame

You may want a dialog box to change size while the dialog box is running. This can be accomplished by changing the **.Height** property of the dialog frame. For example, in the following code, choosing Button 1 increases and shrinks the size of the dialog frame:

1. Create a dialog box with a button control.

2. Enter the following code in a module:

```
Sub Button_Click()
   If DialogSheets(1).DialogFrame.Height = 180 Then
      DialogSheets(1).DialogFrame.Height = 110
   Else
      DialogSheets(1).DialogFrame.Height = 180
   End If
End Sub
```

3. Assign the Button_Click macro to the button in the dialog box.

Using the .Caller Property

The **.Caller** property can return the name of the control that called a subroutine. This property is useful when a subroutine is designed to perform a specific action based on the dialog box control that called it. For instance, a dialog box may have four different spinners that change the contents of four associated Labels. Instead of having four separate subroutines, the spinners can all be assigned to the same subroutine. The following code assumes this type of situation.

```
Sub CallerExample1()
   Dim ControlName As String
   Dim ControlNum As Integer
   ControlName = Application.Caller
   'Note: The right function obtains the Spinner number
   ControlNum = Right(ControlName, 1)
   DialogSheets(1).Labels(ControlNum).Text = _
   DialogSheets(1).Spinners(ControlName).Value
End Sub
```

Where to Find More Information

The Object Browser

A complete list of all of the properties and methods for a specific dialog box control is available in the Object Browser. To find this information, switch to a Visual Basic module, choose Object Browser from the View menu, and then select the name of the desired control from the list of Excel Libraries/Workbooks.

For more information about using the Object Browser, see pages 77–79 of the *Visual Basic User's Guide.*

Microsoft Knowledge Base

The Microsoft Knowledge Base is a primary Microsoft product information source for Microsoft support engineers and is also available to Microsoft customers. This comprehensive database contains more than 40,000 detailed articles with technical information about Microsoft products, fix lists, documentation errors, and answers to commonly asked technical support questions. These articles are also available through CompuServe®, GEnie®, the Microsoft TechNet CD-ROM, and the Microsoft Developer Network CD-ROM.

FastTips Technical Library Catalog

Microsoft FastTips is an automated, toll-free service that provides technical information about key Microsoft products and is available 24 hours a day, 7 days a week in the United States and Canada. Through the FastTips system, you can receive automated answers to common technical problems and access popular articles from the Microsoft Knowledge Base. This information is delivered over the phone through recorded voice scripts, by fax, or through the U.S. mail.

Home Products FastTips	(800) 936-4100
Desktop Applications FastTips	(800) 936-4100
Personal Operating Systems FastTips	(800) 936-4200
Development Tools FastTips	(800) 936-4300
Business Systems FastTips	(800) 936-4400