WinFBE – Visual Designer

Display a popup modal form

Use the *ShowDialog* method. Pass the form that is the parent of the popup (in the example below that would be frmMain).

```
frmPopup.ShowDialog( frmMain )
```

Allow the Enter key to simulate pressing the Tab key

```
''
''
Function frmMain_txtAddress_KeyUp( ByRef sender As wfxTextBox, ByRef e As EventArgs) As LRESULT

   ' Catch the RETURN key to simulate TAB and Shift-TAB
   ' If you were wanting to test multiple textboxes controls (eg. a series of textboxes) then
   ' I would the following code in a dedicated function rather than duplicating it each time.
   if e.KeyCode = VK_RETURN then
      if e.Shift then
         sender.SelectNextControl(false)
      else
         sender.SelectNextControl(true)
      end if
      e.Handled = true
   END IF

   Function = 0
End Function
```

Filter non-numeric characters from a TextBox

```
''
''  Key events are always processed in the following order:
''  (1) KeyDown
''  (2) KeyPress
''  (3) KeyUp
''
''  Based on example from: https://msdn.microsoft.com/en-
us/library/system.windows.forms.control.keypress(v=vs.110).aspx
''  Listing of virtual keycodes can be found at: https://docs.microsoft.com/en-
us/windows/desktop/inputdev/virtual-key-codes
''

'' Boolean flag used to determine when a character other than a number is entered.
dim shared nonNumberEntered as boolean = false

'' Handle KeyDown event to determine the type of character entered into the control.
Function Form1_TextBox1_KeyDown( ByRef sender As wfxTextBox, ByRef e As EventArgs) As
LRESULT

   '' Initialize the flag to false.
   nonNumberEntered = false

   '' Determine whether the keystroke is a number from the top of the keyboard.
   if (e.KeyCode < VK_0 or e.KeyCode > VK_9) then
      '' Determine whether the keystroke is a number from the keypad.
      if (e.KeyCode < VK_NumPad0 or e.KeyCode > VK_NumPad9) then
         '' Determine whether the keystroke is a backspace.
         if (e.KeyCode <> VK_BACK) then
```

```
            '' A non-numerical keystroke was pressed.
            '' Set the flag to true and evaluate in KeyPress event.
            nonNumberEntered = true
         end if
      end if
   end if

   '' If shift key was pressed, it's not a number.
   if e.Shift = true then
      nonNumberEntered = true
   end if

   function = 0
end function

'' This event occurs after the KeyDown event and can be used to prevent
'' characters from entering the control.
Function Form1_TextBox1_KeyPress( ByRef sender As wfxTextBox, ByRef e As EventArgs) As
LRESULT
   '' Check for the flag being set in the KeyDown event.
   if nonNumberEntered = true then
      '' Stop the character from being entered into the control since it is non-
numerical.
      e.Handled = true
   end if
   function = 0
end function
```

Form KeyPreview (filter keyboard characters at the form level)

```
''
''  Key events are always processed in the following order:
''  (1) KeyDown
''  (2) KeyPress
''  (3) KeyUp
''
''  *** Ensure that the form KeyPreview property is set to True ***
''

'' Boolean flag used to determine if numeric character is handled.
dim shared bNumberEntered as boolean = false

''
''
Function Form1_KeyDown( ByRef sender As wfxForm, ByRef e As EventArgs) As LRESULT
   bNumberEntered = false
   select case e.KeyCode
      ' Determine whether the keystroke is a number.
      CASE VK_0 to VK_9, VK_NumPad0 to VK_NumPad9
         ? chr(e.KeyCode)
         bNumberEntered = true
      case VK_RETURN
         ? "ENTER"
         bNumberEntered = true
   END SELECT
   Function = 0
End Function

''
''
Function Form1_KeyPress( ByRef sender As wfxForm, ByRef e As EventArgs) As LRESULT
   if bNumberEntered = true then
      e.Handled = true
```

```
   END IF
   Function = 0
End Function
```

Working with a ListBox

Adding items (Text and 32-bit user defined value)

```
nIndex = Form1.ListBox1.Items.Add("First item in ListBox”, 12345)
```

Adding items (Text only)

```
nIndex = Form1.ListBox1.Items.Add("Second item in ListBox”)
```

Deleting an Item (the selected item)

```
Form1.ListBox1.Items.Remove(Form1.ListBox1.SelectedIndex)
```

Deleting an Item (based on index)

```
Form1.ListBox1.Items.Remove(0)  ' removes the first listbox item
```

Remove all Items from the ListBox

```
Form1.ListBox1.Items.Clear
```

Iterate all items in ListBox

```
for i as long = 0 to Form1.ListBox1.Items.Count - 1
   ? Form1.ListBox1.Item(i).Text, Form1.ListBox1.Item(i).Data32
next
```

Iterate all selected items in the ListBox

```
' Determine the type of listbox selection method and then display items
select case Form1.ListBox1.SelectionMode
   case ListSelectionMode.None
      ' Obviously nothing in the list can be selected

   case ListSelectionMode.One
      ' Show the selected item (single select listbox)
      ? "Method #1: "; Form1.ListBox1.Item(Form1.ListBox1.SelectedIndex).Text, _
                       Form1.ListBox1.Item(Form1.ListBox1.SelectedIndex).Data32
      ' Alternate approach, show the selected item (single select listbox)
      ? "Method #2: "; Form1.ListBox1.SelectedItem.Text, _
                       Form1.ListBox1.SelectedItem.Data32

   case ListSelectionMode.MultiSimple, ListSelectionMode.MultiExtended
      ' Show the selected items (multiselect listbox). Must iterate
      ' through the list and test if entry is selected.
      ? "Count: "; Form1.ListBox1.Items.Count
      ? "SelectedCount: "; Form1.ListBox1.Items.SelectedCount
      for i as long = 0 to Form1.ListBox1.Items.Count - 1  '<- Count not SelectedCount
         if Form1.ListBox1.Item(i).Selected then
            ? Form1.ListBox1.Item(i).Text, Form1.ListBox1.Item(i).Data32
         end if
      next
end select
```