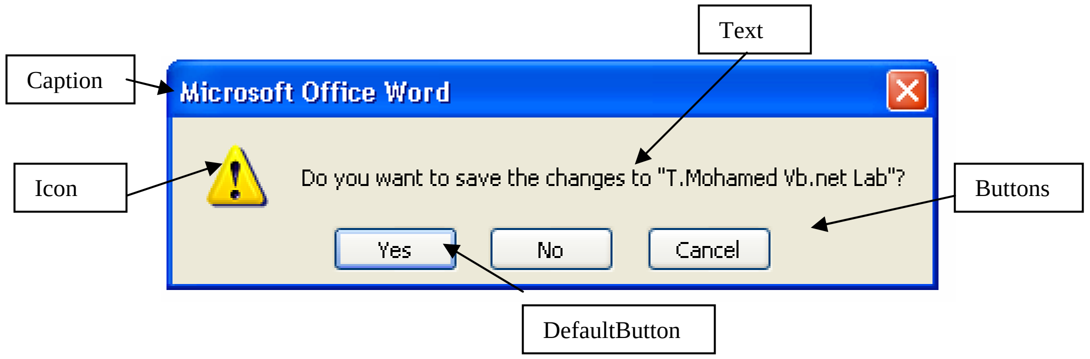
Labs MessageBox Dialog & Controls

MessageBox Dialog



In this message box, the different parts that you control have been labeled. You will see how you can format a message box any way you desire.

- To use the **MessageBox** ,you decide what the **Text** of the message should be, what **Caption** you desire, what **Icon** and **Buttons** are appropriate, and which **DefaultButton** you want. To display the message box in code, you use the MessageBox **Show** method.

- The MessageBox.Show function is **overloaded** with several ways to implement the **Show** method. Some of the more common ways are:

```
MessageBox.Show(Text)
MessageBox.Show(Text, Caption)
MessageBox.Show(Text, Caption, Buttons)
MessageBox.Show(Text, Caption, Buttons, Icon)
MessageBox.Show(Text, Caption, Buttons, Icon,
DefaultButton)
```

In these implementations, if **DefaultButton** is omitted, the first button is default. If **Icon** is omitted, no icon is displayed. If **Buttons** is omitted, an 'OK' button is displayed. And, if **Caption** is omitted, no caption is displayed.

- You decide what you want for the message box **Text** and **Caption** information (string data types). The other arguments are defined by Visual Basic .NET predefined constants. The **Buttons** constants are defined by the **MessageBoxButtons** constants:

Member	**Description**
AbortRetryIgnore	Displays Abort, Retry and Ignore buttons
OK	Displays an OK button
OKCancel	Displays OK and Cancel buttons
RetryCancel	Displays Retry and Cancel buttons

YesNo	Displays Yes and No buttons
YesNoCancel	Displays Yes, No and Cancel buttons

The syntax for specify a choice of buttons is the usual dot-notation:

MessageBoxButtons.Member

So, to display an OK and Cancel button, the constant is:

`MessageBoxButtons.OKCancel`

You don't have to remember this, however. When typing the code, the Intellisense feature will provide a drop-down list of button choices when you reach that argument! Again, like magic! This will happen for all the arguments in the MessageBox function.

- The displayed Icon is established by the **MessageBoxIcon** constants:

Member	**Description**
IconAsterisk	Displays an information icon
IconInformation	Displays an information icon
IconError	Displays an error icon (white X in red circle)
IconHand	Displays an error icon
IconStop	Displays an error icon
IconExclamation	Displays an exclamation point icon
IconWarning	Displays an exclamation point icon
IconQuestion	Displays a question mark icon

To specify an icon, the syntax is:

MessageBoxIcon.Member

Note there are eight different members of the **MessageBoxIcon** constants, but only four icons (information, error, exclamation, question) available. This is because the current Windows operating system only offers four icons. Future implementations may offer more.

- When a message box is displayed, one of the displayed buttons will have focus or be the default button. If the user presses <Enter>, this button is selected. You specify which button is default using the **MessageBoxDefaultButton** constants:

Member	**Description**
DefaultButton1	First button in message box is default
DefaultButton2	Second button in message box is default
DefaultButton3	Third button in message box is default

To specify a default button, the syntax is:

MessageBoxDefaultButton.Member

The specified default button is relative to the displayed buttons, left to right. So, if you have Yes, No and Cancel buttons displayed and the second button is selected as default, the No button will have focus (be default).

- When you invoke the Show method of the MessageBox function, the function returns a value from the **DialogResult** constants. The available members are:

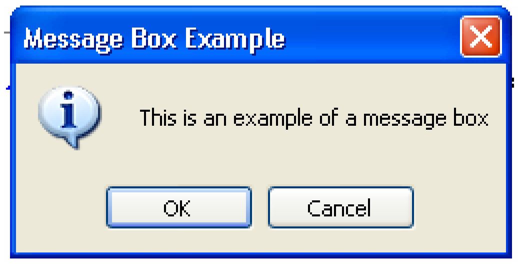
Member	**Description**
Abort	The Abort button was selected
Cancel	The Cancel button was selected
Ignore	The Ignore button was selected
No	The No button was selected
OK	The OK button was selected
Retry	The Retry button was selected
Yes	The Yes button was selected

- MessageBox **Example**:

This little code snippet (the first line is very long):

```
If MessageBox.Show("This is an example of a message box",
"Message Box Example", MessageBoxButtons.OKCancel,
MessageBoxIcon.Information,
MessageBoxDefaultButton.Button1) = DialogResult.OK Then
  'everything is OK
Else
  'cancel was pressed
End If
```

displays this message box:



This is an example of a message box

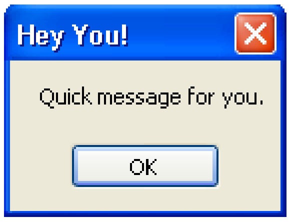
Of course, you would need to add code for the different tasks depending on whether OK or Cancel is clicked by the user.

- Another MessageBox **Example**:

  Many times, you just want to display a quick message to the user with no need for feedback (just an OK button). This code does the job:

  ```
  MessageBox.Show("Quick message for you.", "Hey You!")
  ```

  The resulting message box:



NumericUpDown Control

- The **NumericUpDown** Control is used to obtain a numeric input. It looks like a text box control with two small arrows. Clicking the arrows changes the displayed value, which ranges from a specified minimum to a specified maximum. The user can even type in a value, if desired. Such controls are useful for supplying a date in a month or are used as volume controls in some Windows multimedia applications.

- NumericUpDown **Properties**:

Name	Gets or sets the name of the numeric updown (three letter prefix for numeric updown name is **nud**).
BorderStyle	Gets or sets the border style for the updown control.
Increment	Gets or sets the value to increment or decrement the updown control when the up or down buttons are clicked.
Maximum	Gets or sets the maximum value for the updown control.
Minimum	Gets or sets the minimum value for the updown control.
ReadOnly	Gets or sets a value indicating whether the text may be changed by the use of the up or down buttons only.
Value	Gets or sets the value assigned to the updown control.

- NumericUpDown **Methods**:

DownButton	Decrements the value of the updown control.
UpButton	Increments the value of the updown control.

- NumericUpDown **Events**:

LostFocus	Occurs when the updown control loses focus.
ValueChanged	Occurs when the Value property has been changed in some way.

RadioButton Control

RadioButton controls provide the capability to make a "mutually exclusive" choice among a group of potential candidate choices. This simply means, radio buttons work as a group, only one of which can be selected.

- RadioButton **Properties**:

Name	Gets or sets the name of the radio button (three letter prefix for radio button name is **rdo**).
Checked	Gets or sets a value indicating whether the radio button is checked.
TextAlign	Gets or sets the alignment of text of the radio button.

- RadioButton **Methods**:

Focus	Moves focus to this radio button.
PerformClick	Generates a Click event for the button, simulating a click by a user.

- RadioButton **Events**:

CheckedChange	Occurs when the value of the Checked property changes.
Click	Triggered when a button is clicked. **Checked** property is automatically changed by Visual Basic .NET.

CheckBox Control

The **CheckBox** control provides a way to make choices from a list of potential candidates

- CheckBox **Properties**:

Name	Gets or sets the name of the check box (three letter prefix for check box name is **chk**).
Checked	Gets or sets a value indicating whether the check box is in the checked state.
Text	Gets or sets string displayed next to check box.
TextAlign	Gets or sets the alignment of text of the check box.

- CheckBox **Methods**:

Focus	Moves focus to this check box.

- CheckBox **Events**:

CheckedChange	Occurs when the value of the Checked property changes.
Click	Triggered when a check box is clicked. **Checked** property is automatically changed by Visual Basic .NET.

GroupBox Control

- The **GroupBox** control provides a convenient way of grouping related controls in a Visual Basic .NET application. And, in the case of radio buttons, group boxes affect how such buttons operate.
- GroupBox **Properties**:

Name	Gets or sets the name of the group box (three letter prefix for group box name is **grp**).
Enabled	Gets or sets a value indicating whether the panel is enabled. If False, all controls in the group box are disabled.
Text	Gets or sets string displayed in title region of group box.
Visible	If False, hides the group box (and all its controls).

Panel Control

The **Panel** control is another Visual Basic .NET grouping control. It is nearly identical to the **GroupBox** control in behavior. The Panel control lacks a Text property (titling information), but has optional scrolling capabilities.

- Panel **Properties**:

Name	Gets or sets the name of the panel (three letter prefix for panel name is **pnl**).

AutoScroll	Gets or sets a value indicating whether the panel will allow the user to scroll to any controls placed outside of its visible boundaries.
BorderStyle	Get or set the panel border style.
Enabled	Gets or sets a value indicating whether the panel is enabled. If False, all controls in the panel are disabled.
Visible	If False, hides the panel (and all its controls).

ComboBox Control

The ComboBox allows the selection of a single item from a list. And, in some cases, the user can type in an alternate response.

- **ComboBox Properties**:

Name	Gets or sets the name of the combo box (three letter prefix for combo box name is **cbo**).
DropDownStyle	Specifies one of three combo box styles.
Items	Gets the Items object of the combo box.
MaxDropDownItems	Maximum number of items to show in dropdown portion.
SelectedIndex	Gets or sets the zero-based index of the currently selected item in list box portion.
SelectedItem	Gets or sets the currently selected item in the list box portion.
SelectedText	Gets or sets the text that is selected in the editable portion of combo box.
Sorted	Gets or sets a value indicating whether the items in list box portion are sorted alphabetically.

- ComboBox **Events**:

KeyPress	Occurs when a key is pressed while the combo box has focus.
SelectedIndexChanged	Occurs when the SelectedIndex property has changed.

- The **Items** object for the ComboBox control is identical to that of the ListBox control. You add and remove items in the same manner and values are read with the same properties.

- The **DropDownStyle** property has three different values. The values and their description are:

Value	**Description**
DropDown	Text portion is editable; drop-down list portion.
DropDownList	Text portion is not editable; drop-down list portion.
Simple	The text portion is editable. The list portion is always visible. With this value, you'll want to resize the control to set the list box portion height.

PictureBox Control

- Visual Basic .NET has powerful features for graphics. The **PictureBox** control is the primary tool for exploiting these features. The picture box control can display graphics files (in a variety of formats), Here, we concentrate on using the control to display a graphics file.

- PictureBox **Properties**:

Name	Gets or sets the name of the picture box (three letter prefix for picture box name is **pic**).
BackColor	Get or sets the picture box background color.
BorderStyle	Indicates the border style for the picture box.
Image	Establishes the graphics file to display in the picture box.
SizeMode	Indicates how the image is displayed.

- PictureBox **Events**:

Click	Triggered when a picture box is clicked.

- The **Image** property specifies the graphics file to display. It can be established in design mode or at run-time. To set the Image property at design time, simply display the **Properties** window for the picture box control and select the Image property. An ellipsis (…) will appear. Click the ellipsis and an **Open File** dialog box will appear. Use that box to locate the graphics file to display.

- Five types of graphics files can be viewed in a picture box:

File Type	**Description**
Bitmap	An image represented by pixels and stored as a collection of bits in which each bit corresponds to one pixel. This is the format commonly used by scanners and paintbrush programs. Bitmap filenames have a **.bmp** extension.
Icon	A special type of bitmap file of maximum 32 x 32 size. Icon filenames have an **.ico** extension. We'll create icon files in Class 5
Metafile	A file that stores an image as a collection of graphical objects (lines, circles, polygons) rather than pixels. Metafiles preserve an image more accurately than bitmaps when resized. Many graphics files available for download from the internet are metafiles. Metafile filenames have a **.wmf** extension.
JPEG	JPEG (Joint Photographic Experts Group) is a compressed bitmap format which supports 8 and 24 bit color. It is popular on the Internet and is a common format for digital cameras. JPEG filenames have a .**jpg** extension.
GIF	GIF (Graphic Interchange Format) is a compressed bitmap format originally developed by CompuServe. It supports up to 256 colors and is also popular on the Internet. GIF filenames have a .**gif** extension.

- To set the **Image** property at run-time, you use the **FromFile** method associated with the **Image** object. As an example, to load the file **c:\\sample\myfile.bmp** into a picture box name **picExample**, the proper code is:

```
picExample.Image =
Image.FromFile("c:\\sample\myfile.bmp")
```

The argument in the **Image.From** method must be a legal, complete path and file name, or your program will stop with an error message.

- To clear an image from a picture box control at run-time, simply set the corresponding Image property to **Nothing** (a Basic keyword). This disassociates the Image property from the last loaded image. For our example, the code is:

```
picExample.Image = Nothing
```

- The **SizeMode** property dictates how a particular image will be displayed. There are four possible values for this property: **Normal**, **StretchImage**, **AutoSize**, **CenterImage**. The effect of each value is:

SizeMode	**Effect**
Normal	Image appears in original size. If picture box is larger than image, there will be blank space. If picture box is smaller than image, the image will be cropped.
CenterImage	Image appears in original size, centered in picture box. If picture box is larger than image, there will be blank space. If picture box is smaller than image, image is cropped.
StretchImage	Image will 'fill' picture box. If image is smaller than picture box, it will expand. If image is larger than picture box, it will scale down. Bitmap and icon files do not scale nicely, Metafiles, JPEG and GIF files do scale nicely.
AutoSize	Reverse of StretchImage - picture box will change its dimensions to match the original size of the image. Be forewarned – metafiles are usually very large!

Notice picture box dimensions remain fixed for **Normal**, **CenterImage**, and **StretchImage SizeMode** values. With **AutoSize**, the picture box will grow in size. This may cause problems at run-time if your form is not large enough to 'contain' the picture box.

Best wishes,

T. Mohamed Ezeddin
For more study materials visit my web site
http://issite.wordpress.com