USER INTERFACE

The user interface is the link between the user and capabilities of the application. A well-designed user interface makes easy for the users to learn and use the application.

User interfaces can be roughly categorized into two types

- Command-line interfaces use textual input and output i.e. the end user interacts with an application by typing commands
- Most Windows user interfaces are form-based visual interfaces i.e. the end user interacts with an application through its visual elements

Principles of a User Interface

i) Control – The end user should control the application
ii) User-friendliness – The interface should help the end user accomplish tasks
iii) Intuitiveness – The interface should follow a direct style that proceeds logically (left to right; up and down?)
iv) Consistency – The user interface should have consistent fonts and shapes
v) Feedback – The interface should provide clear and immediate feedback
vi) Graphics – Avoid use of unnecessary graphics
vii) Color – Pleasant but not overdone
viii) Input – Minimize transitions between the keyboard and mouse where possible
ix) User protection – prevent bad input data
x) Screen resolution – The user interface should adapt to different screen resolutions i.e. users may be visually impaired, requiring larger fonts

Designing a User Interface

A user interface should be designed before it is implemented

- Design user interface using a tool such as Visio or pencil and paper.
- Name your controls so that you know that you have at all times and be consistent with user names

Principles of Control Design

i) Alignment – Align control instances vertically or horizontally
ii) Balance – Distribute control instances evenly about the form
iii) Color – Use soft colors with adequate contrast between foreground and background colors. Use predominantly gray colors to avoid problems for people who are color blind. Use white backgrounds for TextBoxes and gray backgrounds for Labels. The background for read-only TextBoxes is also gray.
iv) Function grouping – Group control instances based on their function
v) Consistent sizing – Control instances should have the same size

Elements of the Graphical User Interface:

A user interface consists of various elements with which the user can interact and control the application.
The first element is the FORM and it acts as a container object for all elements on the interface (controls).

Form Object

The Form is where the user interface is drawn. It is central to the development of Visual Basic applications, whether for databases or other uses.

Some of Form Properties:

- BackColor - controls the background color of the whole screen
- Text - the text that appears in the form's title bar
- WindowState - controls how big the window is when the program is run
  0 Normal 1 Minimized 2 Maximized (best)
- Name - the name of the form
  eg. change Form1 to TitleScreenForm
- FormBorderStyle - controls the appearance of the form. It also determines whether the user can resize the form

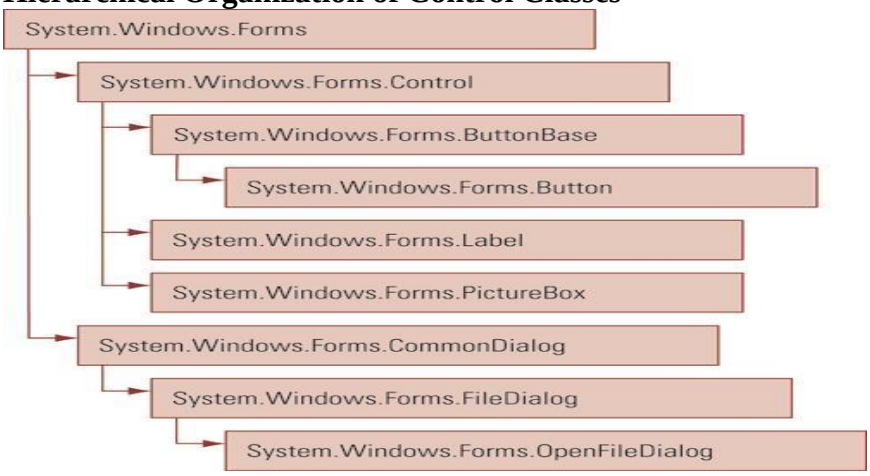
- The **Width** and **Height** properties define the form's size (x length and y length)
- BackgroundImage property- used to create an effect with a background image. You can alter the appearance of an image with the b**ackgroundImageLayout** property -- setting it to Tile, Center, Stretch, or Zoom.
- StartPosition property – set to **CenterScreen** to display a form centered on the display at run time.
- Fonts - Use a **MS Sans Serif** font for most information on a form as this font is easiest for most people to read. Do not use large fonts except for a limited number of items, and do not use bold or italic fonts except to highlight select information.

Controls

Controls are all instances of control classes and the collection of control classes are arranged in a class hierarchy i.e. **System.Windows.Forms.Form.Control** Class

- This class is the base class from which all visible controls are derived
- Dialog boxes derived from the **CommonDialog** class

Hierarchical Organization of Control Classes



1). GroupBox Control

A **GroupBox** control is used to group or contain other controls i.e. it is a container control that is used to group control instances together. It is used to organize a form into different sections and this can make the form easier for an application user to use. All group controls move with the group box

- **Name** property – most of the time you will not bother to name a GroupBox control because they are rarely referred to when you write programming code.
- **Text** property – displays the words in the upper left corner of the GroupBox control. If you leave the Text property blank, the GroupBox looks like a rectangle.

2). Label Control

A label is a graphical control used to display text. The user cannot edit the text in a label. The most common use for a Label control is to provide prompt for other controls, such as the TextBox control. You can also use the Label control to display text such as status messages and other program information. The Labels **BorderStyle** property should be left flat.

3). TextBox

You use a TextBox control to obtain information from the user or to display information provided by the application. Unlike information displayed in a label, the user can change information displayed in a text box. Data entered into a TextBox control is saved to the **Text** property of the control.

Some properties of text box include: -

- **ForeColor** – color of the text in the control.
- **BackColor** – color of the background – white is the default.
- **TextAlign** – your options are to display the text **left**, **right**, or **center** justified. **Left** is the default.

- **Multiline** – set to **True** if the TextBox will display more than one line of text – the default is **False**.
- **WordWrap** – this property determines if the contents of a TextBox should wrap to a second line (or subsequent lines) if the output will not fit on a single line – the default is **True**.
- **BorderStyle** property – this property makes controls appear as either flat or three-dimensional.
- **ReadOnly** – set to True if the TextBox will only display output – the default is False so you can type text into the TextBox.

Setting Control Colors
Colors are defined in VB as an **enumeration** of values i.e. an enumeration is a list of predefined values. The enumeration name for colors is simply **Color** and is accessed by typing the word **Color** followed by the **dot**.

Examples:
```
'Set the text color of the NameTextBox to black
NameTextBox.ForeColor = Color.Black

'Set the color of the text of the for the form to white
NameTextBox.BackColor = Color.White
```

Variables can be declared at module-level to hold colors as shown below
```
Private GrayBackGroundColor AS Color
```

Setting the Focus
The **Focus** method is used to set the focus of the cursor to a specific control.
Example:
```
NameTextBox.Focus()
```

4). MaskedTextBox Control
The MaskedTextBox control is a special form of TextBox control with special properties to format the data entered by an application user.
- **Name** property – use a name such as **PhoneMaskedTextBox**.
- **Mask** property – set to different input mask values. Click the ellipse button on the property to open up the **Input Mask** window.

Clearing TextBox, MaskedTextBox, and Label Contents
Use an assignment statement to clear the contents of a TextBox, MaskedTextBox, or label control.
- Assign a value of the **empty string** to the **Text** property of these controls. The empty string is denoted by typing a set of two double-quote marks together with no space between them.
- The empty string is also defined by the **String.Empty** defined VB enumerated value. Examples:
```
NameTextBox.Text = ""
MajorTextBox.Text = String.Empty
```

However, the **Clear** method is used most often to clear a TextBox or MaskedTextBox. Example:
```
NameTextBox.Clear()
PhoneMaskedTextBox.Clear()
```

5). RichTextBox Control
The RichTextBox control is a special form of TextBox control with special properties to enable applying different character and paragraph formatting, as with word processor software package.
- Name property – use a name such as **OutputRichTextBox**.
- WordWrap and Multiline properties – apply to the RichTextBox just as they do to a regular TextBox.

6). RadioButton Control

RadioButton controls are used to restrict the selection of values from a defined set of alternatives that are mutually exclusive i.e. only one RadioButton in a group can be selected at a time – selecting a new RadioButton causes the other RadioButtons to be unselected.
RadioButtons that are to function as a group of RadioButtons should be placed inside a GroupBox control. RadioButton controls directly on a form function as one group.
The primary event used for RadioButton controls is NOT the **Click** event – it is the **CheckedChanged** event.

- **CheckedChanged** fires whenever a RadioButton's **Checked** property value changes i.e. the **CheckedChanged** event fires when the button is clicked.
- Each RadioButton control can have its own **CheckedChanged** event procedure.

Properties:

- C**hecked** property – defines whether the **RadioButton** is selected and only one RadioButton control in a group can be the default – specify this by setting its **Checked** property to **True** – the default is **False**.
- **Text** property – displays next to the RadioButton, e.g., **Black**, **Red**, **Blue**, etc.

Note: Format menu – use this to align/size the controls and set vertical/horizontal spacing. Select the controls by holding down CTRL or SHIFT keys and using the mouse to click each one in order. The first control selected is the base control which all of the other controls will mirror.

7). CheckBox Control

CheckBox controls are similar to RadioButton controls except that they are used whenever the system user can select zero, one, or more than one of the options presented by CheckBoxes.

- **Checked** property – works like the RadioButton – stores **True** or **False**, depending on whether the CheckBox is checked.

Selecting and Unselecting RadioButtons and CheckBoxes

You can select and unselect these controls in program code by setting the **Checked** property to either **True** (to select) or **False** (to unselect). Examples:

```
 'Set the black RadioButton to be checked
BlackRadioButton.Checked = True

'This unchecks a RadioButton control – not used very often
BlueRadioButton.Checked = False

'This checks a CheckBox control
MessageCheckBox.Checked = True
```

To reset a group of RadioButtons so that one is checked and the others are not checked, you only need to set the **Checked** property to **True** for one of the controls – the others will automatically be set to **False**. For CheckBox controls, you must set each of the control's **Checked** property to the desired value.

8). PictureBox Control

The **PictureBox** control is used to display images on a form.

- **Image** property – access this property to display the **Select Resource** dialog box to add an image to the PictureBox – any kind of image will generally do – ico (icon), bitmap, jpg, gif, etc.
- The **ImageLocation** property contains the disk file name where the image is stored
- The **SizeMode** property defines how the image appears (scaled) inside of the **PictureBox**
- **BorderStyle** property – this property makes controls appear as either flat or three-dimensional.
- **Visible** property – When set to **True** (displays the image) or **False** (makes the image invisible).

Reading an Image
Use the FromFile method of the System.Drawing.Image class. The method accepts a file name as the argument

Example
To read the Image named "C:\House1.jpg" into the **PictureBox** control instance named picDemo

```
picDemo.Image = _
    System.Drawing.Image.FromFile( _
    "C:\House1.jpg")
```

Note: Buttons and other controls have an **Image** property that will cause the display of a graphic on the control.

9). Button Control
The **Button** control is derived from the **ButtonBase** class and supplies a clickable button. It is used to perform a task when the user clicks the button and can be used to begin, interrupt or end a process. The **Click** event fires when the end user clicks the button

Form's AcceptButton and CancelButton Properties
Set these properties of the Form control to map keyboard keys to Button controls on a form.
- **AcceptButton** property – maps the keyboard **Enter** key to the specified button on the form – makes the Enter key act like you've clicked the button with the mouse.
- **CancelButton** property – maps the keyboard **ESC** key to a specified button on the form.
  Note: It is a good idea to place buttons horizontally along the bottom of the form or vertically along the right side of the form.

Working with Multiple Control Instances
You may need to modify several controls of the same type, for example, several Buttons or CheckBoxes or Labels to select and set properties that they have in common such as the **BorderStyle**. This will also enable you to align controls on a form more rapidly.

10). Tool Tips
ToolTips appear as pop-up windows when the mouse hovers over a control instance and is used to display informational messages to the end user
The **ToolTip** control is a **provider control**, meaning that it works with other control instances
- Each form needs only one **ToolTip** control.
- The **ToolTip** control displays in the **component tray** at the bottom of the IDE below the form. The default **Name** property value is **ToolTip1**.

ToolTip on ToolTip1 property:
- A single **ToolTip** control on form causes each control on the form acquires a property with this name when you add a tool tip control to the project.
- Type the text of the tip to display into the **ToolTip on ToolTip1** property of a control such as a button.

Properties
- **InitialDelay** property controls the amount of time the mouse must hover before displaying the ToolTip
- **AutomaticDelay** property controls the length of time the ToolTip appears

Concatenation Operator

The **concatenation operator** is the ampersand (**&**) or plus symbol (**+**) is used to join two strings of text or characters together to form a longer, single string.

Example of string concatenation:

`"42" & "16"` is NOT equal to **`58`** – the correct answer is the string of characters **`4216`**

ControlChars

The **ControlChars.NewLine** is a value of the **ControlChars** (control characters) VB **enumeration** of values used to control the output display of information

The WITH and END WITH Statements

If you need to set several properties for an individual control, you can use the **With** and **End With** statements to shorten the coding, and to make your code easier to read. Additionally, programs written in this fashion with these **With** blocks will execute a little bit faster than those that do not use this approach.

Example

The code below will make the **NameTextBox** control visible, set the text colors for foreground and background, and set the focus to this control.

```
With NameTextBox
    .Visible = True
    .ForeColor = Color.Black
    .BackColor = Color.White
    .Focus()
End With
```