Visual Programming IS202

Chapter Two

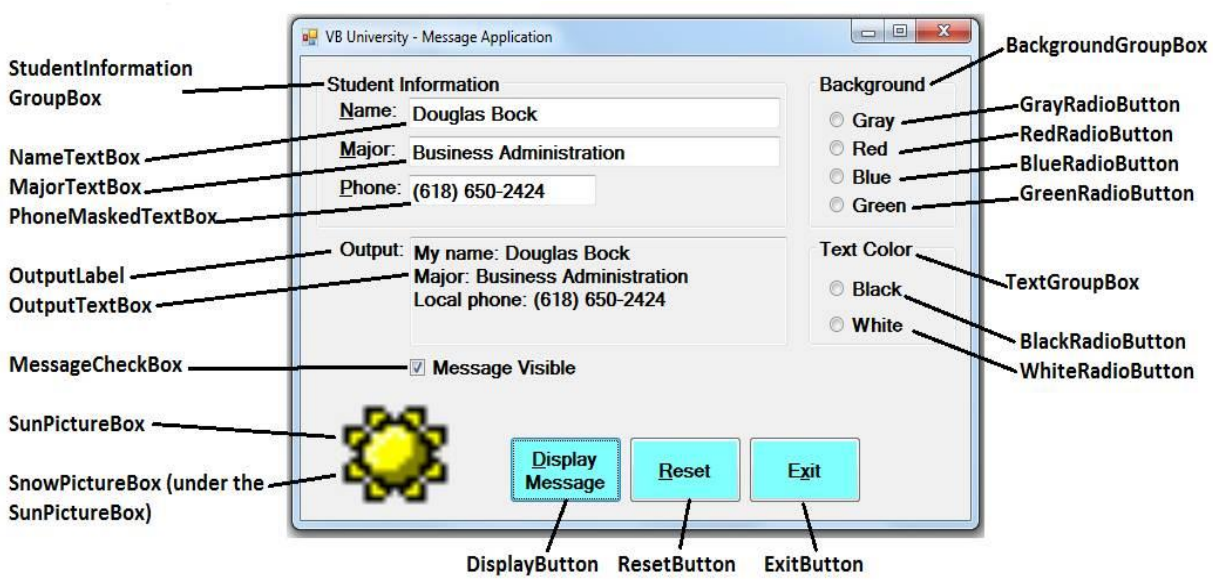
User Interface Design

INTRODUCTION

Chapter 2

In this chapter uses numerous controls. You will build a project form that is similar to that shown below. It will include the following new controls:

- TextBox
- MaskedTextBox
- GroupBox
- RadioButton
- CheckBox
- PictureBox



Control tool	Name	Text	Other Properties
Form1	StudentInfo	VB University – Message Application	StartPosition=CenterScreen
GroupBox1, GroupBox2, GroupBox3	GroupBox1, GroupBox2, GroupBox3	Student Information, Background, Text Color	
Label1, Label2, Label3, Label4	Label1, Label2, Label3, Label4	Name Major	

		Phone Output	
TextBox1, TextBox2, MaskedTextBox1, RichTextBox	NametextBox, MajorTextBox, PhoneMaskedTextBox, OutputTextBox		Readonly=True, Tabstop=False, Forecolor, Backcolor, TextAlign, WordWrap, BorderStyle, TabIndex. Mask. outputRichTextBox has Readonly=True, WordWrap, MultyLine=True
RadioButton1, RadioButton2,RadioButton3, RadioButton4,RadioButton5, RadioButton6	GrayRadioButton, RedRadioButton, BlueRadioButton, GreenRadioButton, BlackRadioButton, WhiteRadioButton,	Gray, Red, Blue, Green, Black, White	Checked=True or False
CheckBox1	MessageCheckBox	Message Vissible	
PictureBox1, PictureBox2	SunPictureBox, SnowPictureBox		Choose Image, Size Mode, Visible, BackgroundImageLayout
Button1, Button2, Button3	DissplayButton, ResetButton, ExitButton	Display Message, Reset, Exit	ForeColor, Backcolor

GroupBox Control

A **GroupBox** control is used to group or contain other controls. It is used to organize a form into different sections and this can make the form easier for an application user to use.

- **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 – if you were to name the control, a name such as **grpbxStdInfo** would be appropriate.
- **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.

The form above has three GroupBox controls with **Text** property settings of **Student Information**, **BackGround**, and **Text Color**.

TextBox and Label Controls

TextBox controls can be used for:

- **Data input** – this use requires no special property settings.
- **Data output** display.

- Set the **ReadOnly** property to **True** and the TextBox looks like a Label.
- You can also use a label to display output, but a TextBox a control is easier to use.
- Set **TabStop** = **False** to keep from tabbing to a read only TextBox.

The form in the figure shown above has three TextBox controls – two of them are accompanied by two Label controls inside of the **Name Information GroupBox** control.

- The label controls display the words "**Name:"** and "**Major:"** – these are **prompts** to tell the system user what type of information is to be typed into the TextBox controls.
- Don't bother naming Label controls – you will not refer to them later in programming code.
- Set the **Text** property to the Label control to the prompt to be displayed – "**Name:"** and "**Major:"**.

The first two TextBox controls are next to the labels.

- Data entered into a TextBox control is saved to the **Text** property of the control.
- Assign the **Name** property a meaningful name to each TextBox – **NameTextBox** and **MajorTextBox** or (txtName, txtDept).

The third TextBox is used to display output and has an accompanying label with the **Text** property = **Output:** and the **Name** property = **OutputLabel**.

- **Name** property for= **OutputTextBox** or (txtOutput).
- **ReadOnly** property = **True**.
- **MultiLine** property = **True** – the size of the TextBox is then stretched to allow for multiple output lines.

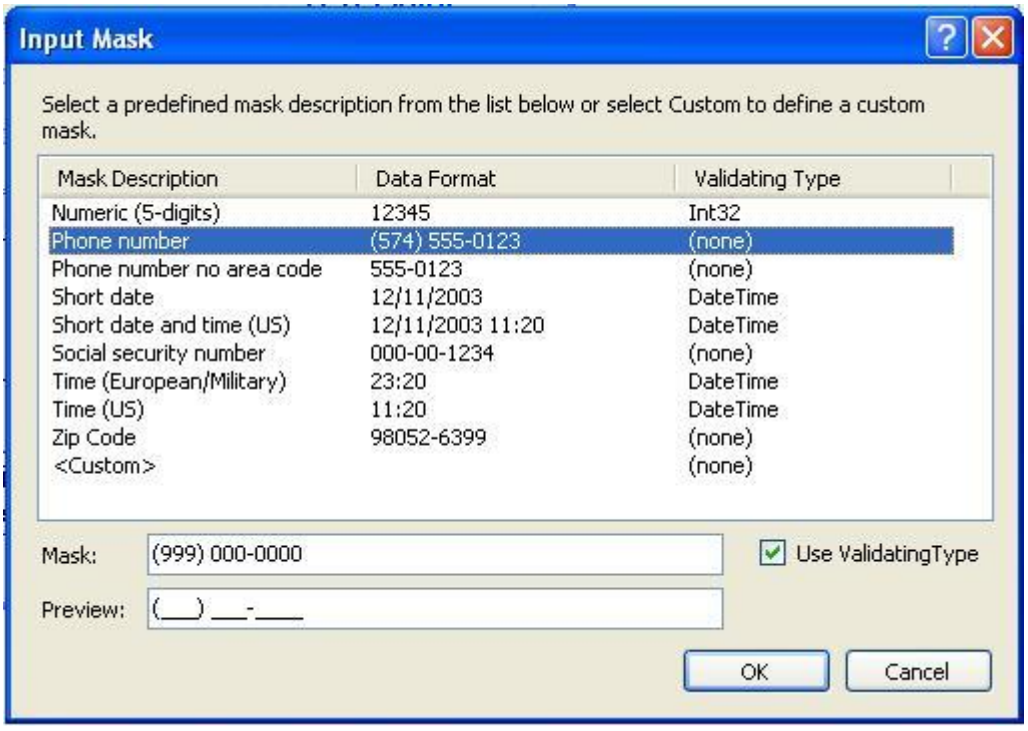
Other TextBox control properties include:

- **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**.
- **ReadOnly** – set to **True** if the TextBox will only display output – the default is **False** so you can type text into the TextBox.

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** or (txtPhoneNo).
- **Mask** property – set to different input mask values. Click the ellipse button on the property to open up the **Input Mask** window shown in the figure below.



- In the figure above, the **Phone number** mask description is selected. Note the display of the example mask.
- In the figure showing the form design, the MaskedTextBox control is accompanied by a label with a **Text** value of "**Phone:**" that provides a prompt.

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 or (RchtxtOutput)**.
- **WordWrap** and **Multiline** properties – apply to the RichTextBox just as they do to a regular TextBox.

This figure illustrates use of a RichTextBox to display text loaded from a "RTF (Rich Text File)" file by use of this coding segment (the file is stored in the project's \bin\debug folder):

```
'Display information from Rich TextBox Contents.rtf file to the RichTextBox

RichTextBox1.LoadFile("RichTextBoxContents.rtf")
```

Start a New Project

Start VB.NET. Create a new project named **Ch02VBUniversity**.

Form property settings:

- **FileName** – set to **StudentInfo.vb**.
- **Text** property to read **VB University – Message Application**.
- **StartPosition** property – set to **CenterScreen**.

Add a GroupBox control to store student information.

- **Text** property – as shown in the figure above.
- You can name the GroupBox control, but it is not necessary as you will not refer to its name when writing code.

Add three Label controls and two TextBox controls to the GroupBox as shown in the figure.

- **Name** property of the Label controls – leave as the default value.
- **Name** property of the TextBox controls – **NameTextBox** and **MajorTextBox**.

Add a MaskedTextBox control to the GroupBox as shown in the figure.

- **Name** property – **PhoneMaskedTextBox**.
- **Mask** property – **Phone number**.

Add the output TextBox control and accompanying Label control below the GroupBox control as shown in the figure.

- Label property settings:
    - **Name** property – **OutputLabel**.
    - **Text** property– **Output:**.
- TextBox property settings:
    - **Name** property – **OutputTextBox**.
    - **ReadOnly** property – **True**.
    - **TabStop** property – **False**.
    - **MultiLine** property – **True**.

Save the project and run it. Attempt to enter data into the TextBox and MaskedTextBox controls to ensure the controls work properly.

RadioButton Control

RadioButton controls are used to restrict the selection of values from a defined set of alternatives.

- **Name** property – use a name such as **GrayRadioButton** or **RedRadioButton** or (rdButGray, rdbutRed).
- **Checked** property – only one RadioButton control in a group can be the default – specify this by setting its **Checked** property to **True** – the default is **False**.
- Only one RadioButton in a group can be selected at a time – selecting a new RadioButton causes the other RadioButtons to be unselected.
- **Text** property – displays next to the RadioButton, e.g., **Black**, **Red**, **Blue**, etc.
- RadioButtons that are to function as a group of RadioButtons should be placed inside a GroupBox control. RadioButton controls not in a group that are just on a form function as a group, but this keeps you from creating more than one group.
- **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.

Two different GroupBox controls with two sets of RadioButton controls.

Note that each group can have a default RadioButton that is already checked.

Continue In-Class Exercise

Add two GroupBox controls and set the following properties:

- **Text** property – as shown in the figure.
- Add RadioButton controls inside each GroupBox and set their properties:
    - **Text** – **Gray**, **Red**, **Blue**, and **Green** in one and **Black** and **White** in the other.
    - **Checked** – set to **True** for the Gray and the Black RadioButtons.
    - **Name** – name each RadioButton an appropriate name, e.g., **GrayRadioButton**, **RedRadioButton**, etc.

Save the project and run it. Select among the RadioButton controls. You should find that when you select one control, the previously checked control within the group is unchecked.

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.

Our program only has a single CheckBox control, although you can always use as many as are necessary.

- **Name** property – use a name such as **MessageCheckBox**.

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

Continue In-Class Exercise

Add a CheckBox control and set the properties as indicated:

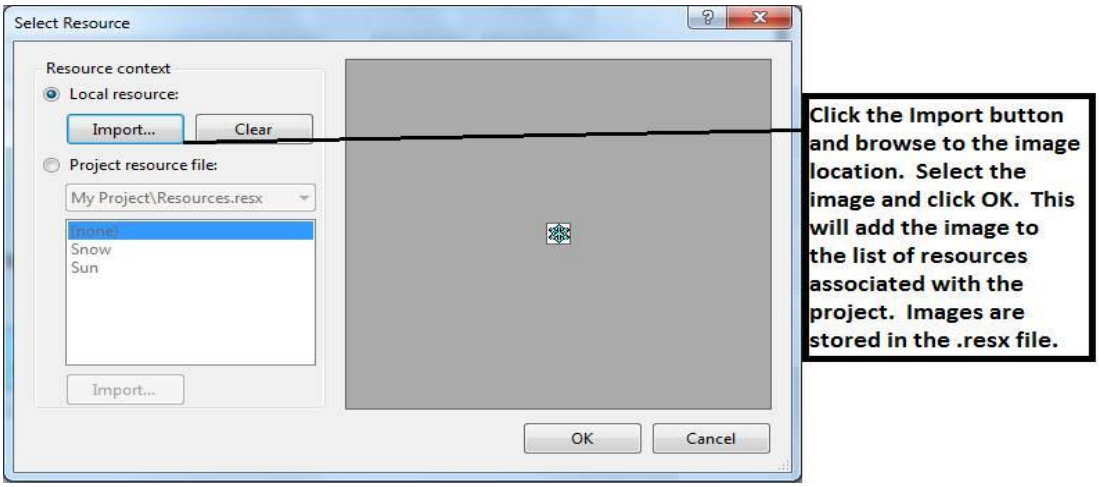
- **Name** property – set to **MessageCheckBox**.
- **Text** property – set to **Message Visible** as shown in the figure.

Save the project and run it. You should be able to check and uncheck the CheckBox, but this doesn't have any effect on the program interface yet – later you'll write code to make the CheckBox do something.

PictureBox Control

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

- Put the control on the form – stretch it to the size desired – notice that the control is empty.
- **Name** property – use a name such as **SnowPictureBox**.
- **Image** property – access this property to display the **Select Resource** dialog box as shown in the figure below to add an image to the PictureBox – any kind of image will generally do – ico (icon), bitmap, jpg, gif, etc.



- **SizeMode** property – set this to the **StretchImage** value in order to cause an icon file to fill the entire PictureBox control space.
- **Visible** property – set to **True** to make an image visible; **False** to make an image invisible.

This code segment demonstrates how to make a PictureBox control appear or disappear by setting the **Visible** property with program code at run time.

```
'Turn on the sun and turn off the snow

SunPictureBox.Visible = True

SnowPictureBox.Visible = False
```

Other Image Facts

- You can set the **BackgroundImage** property of a form or control to create an effect with a background image.
- You can alter the appearance of an image with the **BackgroundImageLayout** property -- setting it to Tile, Center, Stretch, or Zoom.
- Buttons and other controls have an **Image** property that will cause the display of a graphic on the control.

Continue In-Class Exercise

Add two PictureBox controls to your project form. Place these side by side and use the **Format** menu to size them to identical sizes. Set the properties as follows:

- **Name** property – **SunPictureBox** and **SnowPictureBox**.
- **Image** property – import two images of the sun and snow to the resource listing.
- **SizeMode** property – set both to a **StretchImage** value to stretch the image to fill a PictureBox control.
- **Visible** property – set the **SunPictureBox** to **True** (the default), and the **SnowPictureBox** to **False**.

Note: To add icon files with the **.ico** filename extension, select the **All Files** for the **Files of type** option in the **Open** dialog box.

Run your project to confirm that the sun displays and the snow doesn't – later you will write program code to make the images appear and disappear.

Application Design

Adding a Professional Touch

BorderStyle property – Labels, TextBox and PictureBox controls all have a **BorderStyle** property – this property makes controls appear as either flat or three-dimensional.

- **BorderStyle** property -- set to an appropriate value to enhance the appearance of a form and add a professional touch to a project.
- **BorderStyle** property values:
    - **None** – flat appearance with no border.
    - **FixedSingle** – a flat appearance with black border.
    - **Fixed3D** – for a TextBox, this looks about like FixedSingle. For a Label control, the appearance is a three-dimensional, recessed appearance.
- The TextBox control default value for **BorderStyle** is **Fixed3D**.

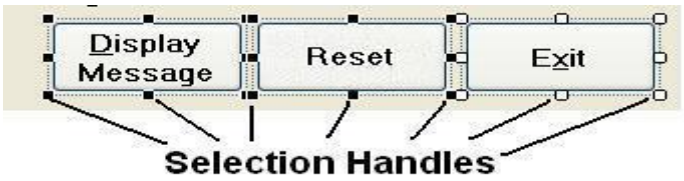
- The Label and PictureBox controls default value for **BorderStyle** is **None**.

Multiple Controls

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.

Methods to select more than one control at a time:

- **Lasso** method – if the controls are located on a form near to each other, simply place the mouse pointer at one corner of an imaginary box around the controls, hold down the left mouse button and drag the mouse toward the opposite corner. You'll notice that a "dashed line" selection box forms around the controls, and when you release the mouse button, each selected control will be highlighted with selection handles like those shown around the buttons in this figure.

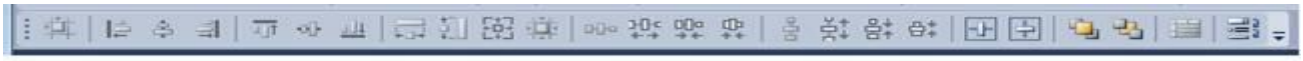


- **Shift** and **Ctrl** (Control) key method – hold down either the **Shift** or **Ctrl** key and single-click the other controls that you wish to select. This approach works best when selecting controls that are not next to one another or controls located inside a GroupBox control.
- **Edit menu, Select All option** – this selects all controls on a form.

To "**unselect**" a group of controls, click away from the controls someplace on the form.

Once selected, you can:

- move all controls selected as a group by dragging them with the mouse button.
- set properties of multiple controls that are common properties. For example, you can select the three command buttons on your in-class exercise form and set the **BackColor** property to **red** for all three buttons at the same time. This is faster than setting them one at a time.
- align and resize controls by using the buttons on the **Layout Toolbar** shown in the figure below. If this toolbar is not visible, select the **View menu**, **Toolbars** option and select the **Layout Toolbar**. The toolbar buttons are disabled (grayed out) until you select more than one control. Practice with this tool bar now.



Layout Toolbar - display by selecting the View menu, Toolbars option.

User Interface Features

The form you design should be easy to use. The more intuitive the form is in terms of system user's understanding how to use it, the less training that will be required to teach system users how to be productive with the computer applications that you program. Follow these guidelines.

Color

- 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. Using different colors will keep application users from trying to type data into Labels and read-only TextBoxes where data cannot be entered, and will make the areas on a form can be used for data entry more obvious.

Grouping and BorderStyle

- Use GroupBoxes to group controls to aid the system user in organizing how information on a form is presented and is to be used.
- If Labels display messages such as prompts to enter data into an accompanying TextBox, leave the Labels with a flat **BorderStyle** property.

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.
- Do not use bold or italic fonts except to highlight select information.

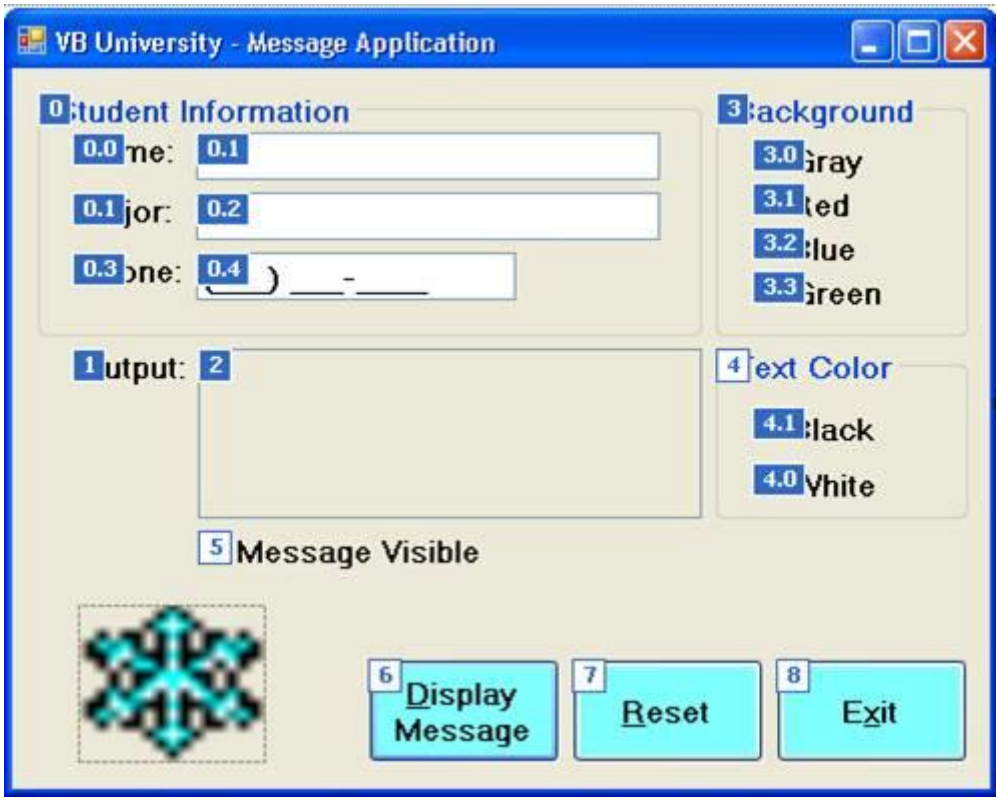
Form's AcceptButton and CancelButton Properties

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.

Tab Order

- Select the **View** menu, **Tab Order** option to display the tab order for a form as shown in this figure.
- Change the order numbering by mouse-clicking on the numbers.
- Tabbing should be left to right, top to bottom.



-
- **TabIndex** property – changing the tab order sets the **TabIndex** property value (a numeric value) for a control – it is the order of the numbers that actually controls the tabbing during run time.

    - **TabIndex** values start at **zero** (0) and increase one unit at a time – most controls are assigned a TabIndex value (PictureBox controls are not).

    - A control with **TabIndex = 0** has the **focus** on startup of an application.

- Select the **View** menu, **Tab Order** option again to turn off the display of the tab order.
- **TabStop** property – set this to **False** if you do not want a control to be in the tab order.
- Labels – even though labels have a tab order number, they cannot receive the focus so you cannot tab to a label.
- GroupBox and RadioButtons/CheckBoxes – you can tab to a GroupBox, but you must use the up/down arrow keys to select among RadioButtons or CheckBoxes.

Set the tab order, and then examine the **TabIndex** property of each control.

Keyboard Access Keys

- Define keyboard access keys (**hot keys**) for buttons and other controls in case the computer mouse quits working – also this provides shortcuts favored by some individuals.
- Access keys are letters in text that are underlined.
- Type an **ampersand (&)** just before the intended hot key character in the Text property of the button. For example, to display **Exit**, type **E&xit** – this will make the **Alt-X key** combination the hot keys for the Exit button.
- Do not use the same hot key more than once on a form.

- When grouping a Label and TextBox control, you can use the **&** symbol to create a hot key for the Label—since a Label control cannot get focus, the focus will shift to the next control in the tab order which will be the associated TextBox control.
- **Windows Operating System Alt Key** – if the underline does not appear on a control at run time, you can press the **Alt** key to initiate the use of the hot keys.

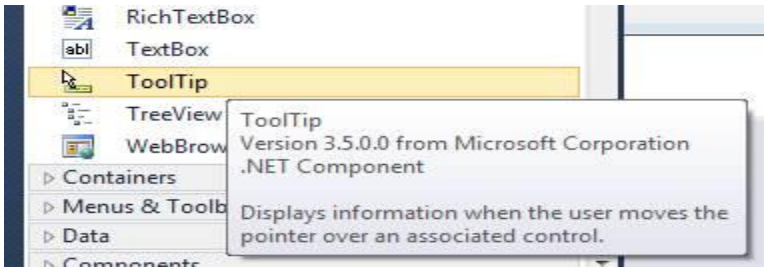
Form's StartPosition Property

- **StartPosition** property – as you learned in chapter 1, set to **CenterScreen** to display a form centered on the display at run time.

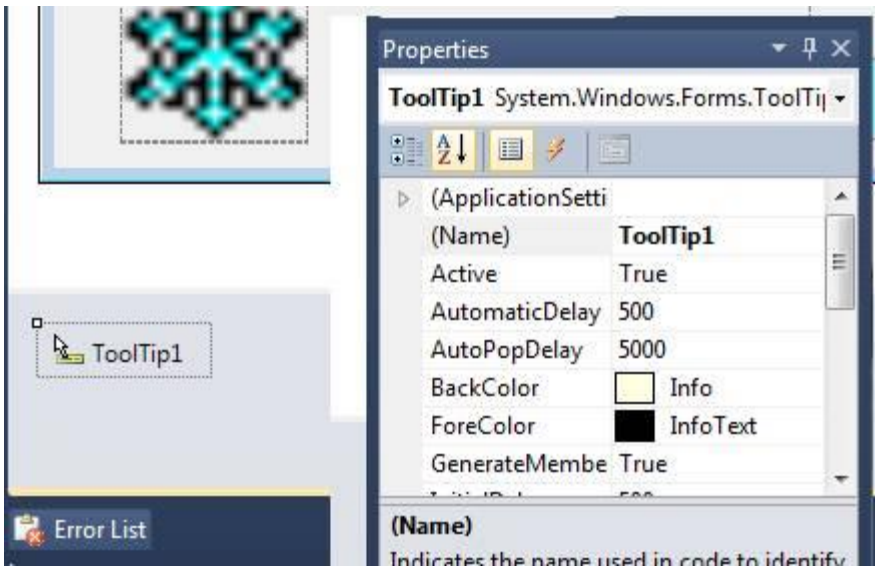
Tool Tips

Tool tips are small labels that display whenever an application user pauses the mouse pointer over a toolbar button or a form control.

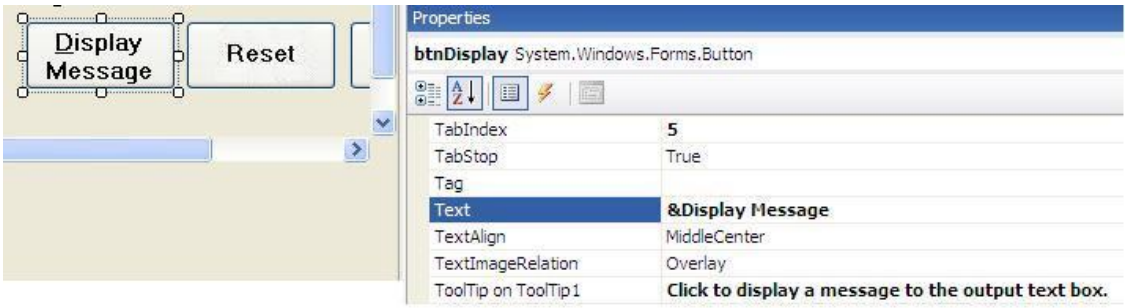
- Each form needs only one **ToolTip** control.
- Access the toolbox and add a **ToolTip** control (component) by double-clicking it as shown in this figure.



- 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:
    - o A single **ToolTip** control on a form causes each control on the form acquires a property with this name when you add a tool tip control to the project.
    - o Type the text of the tip to display into the **ToolTip on ToolTip1** property of a control such as a button as shown in this figure.
    - o At run time, pausing the mouse over a control will cause the tool tip to display.



Continue In-Class Exercise

Modify the form to make the appearance and use of the form more professional.

- Use the **Layout Toolbar** to align all controls on the form that might not be aligned properly.
- Set the form's **AcceptButton** and **CancelButton** properties to map the **DisplayButton** button to the **Enter** key and the **ResetButton** button to the **ESC** key.
- Select all three buttons at the same time – set the **BackColor** property of all three buttons to light blue.
- Define hot keys for the three buttons as shown in the above figure.
- Define hot keys for the Name:, Major:, and Phone: Label controls.
- Check the tab order of the controls on your project. Ensure that they tab from top to bottom, left to right.
- Add a **ToolTip** control to the project. Add appropriate tip messages for the input TextBox, MaskedTextBox, and Button controls.

Run the project to check that the modifications made display properly during project execution.

Program Coding

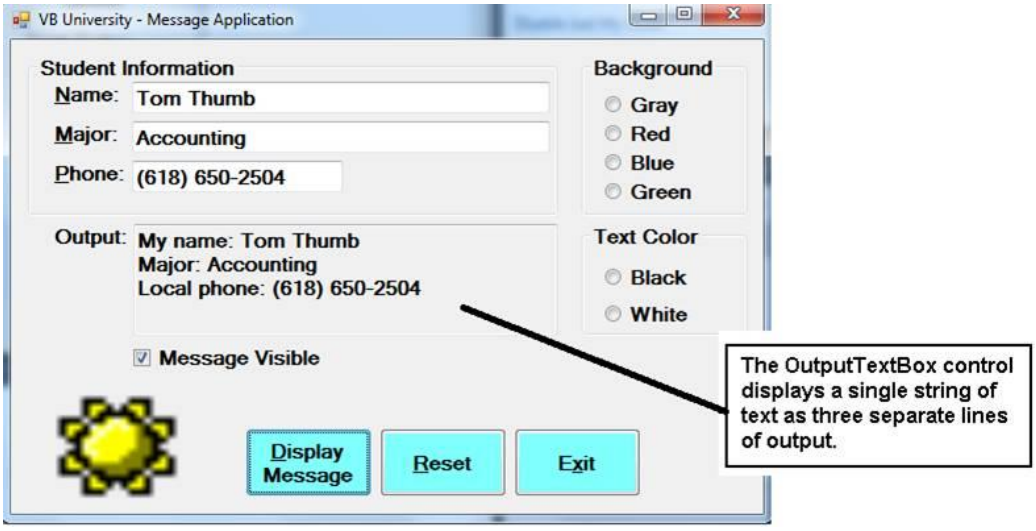
Exit Button – Click Event

From your study of Chapter 1 you should recall how to use the **Close** method for the form (use the **Me** keyword) to code the Exit Button's click event.

```
    Private Sub ExitButton_Click(ByVal sender As System.Object, ByVal e As Syste
m.EventArgs) Handles ExitButton.Click

        'Exit the form

        Me.Close()

    End Sub
```

Display Message Button – Click Event

Clicking the Display Message Button control should cause the Output TextBox control to display three separate lines of output as shown in this figure. This display is actually one long string of text – this is termed string or text **concatenation** that is the addition of different strings of text together.



The **concatenation operator** is the ampersand (**&**) – you can also use the plus symbol (**+**) – the result is to simply add 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**

"42" & " " & "16" results in a string of **42 16** -- Note the blank space that has been concatenated (added) to the middle of the string of characters.

"42" & ControlChars.NewLine & "16" results in output on two lines like this:

42

16

The **ControlChars.NewLine** is a value of the **ControlChars** (control characters) VB **enumeration** of values used to control the output display of information (your textbook also notes that you can use the **Environment.NewLine** constant as well).

Sometimes you will need to display text typed into a TextBox control to another TextBox or label control. This is easily accomplished with an assignment statement like the one shown here. This code would be placed inside a **Click** event for a button control.

```
OutputTextBox.Text = "My name: " & NameTextBox.Text
```

You may need to display information from two or more TextBox controls to a single TextBox or label control.

- Use the concatenation operator (the ampersand **&** symbol) to concatenate information (means to add two or more strings together) from the Text properties of the controls and store the single resulting string to the Text property of the receiving control.
- This code produces the output shown in the above figure.

```
    Private Sub DisplayButton_Click(ByVal sender As System.Object, ByVal e As Sy
stem.EventArgs) Handles DisplayButton.Click

        'Store output message to the output TextBox

        OutputTextBox.Text = "My name: " & NameTextBox.Text &
ControlChars.NewLine & "Major: " & MajorTextBox.Text & ControlChars.NewLine
& "Local phone: " & PhoneMaskedTextBox.Text

    End Sub
```

You may find the above code difficult to read because the code wraps around multiple display lines. You can break the code up into readable segments by using the **underscore** (**_**) character as a continuation character for a line of VB code. You'll also notice that we indented the continued line 4 spaces. This is a normal coding procedure to make the code easier to read.

```
    Private Sub DisplayButton_Click(ByVal sender As System.Object, ByVal e As Sy
stem.EventArgs) Handles DisplayButton.Click

        'Store output message to the output TextBox

        OutputTextBox.Text = "My name: " _

            & NameTextBox.Text _

            & ControlChars.NewLine _

            & "Major: " _

            & MajorTextBox.Text _

            & ControlChars.NewLine _

            & "Local phone: " _
```

```
            & PhoneMaskedTextBox.Text

    End Sub
```

You can break a line of code any place except in the middle of a string of text that is within double-quote marks. This is equivalent to the above.

```
    Private Sub DisplayButton_Click(ByVal sender As System.Object, ByVal e As Sy
stem.EventArgs) Handles DisplayButton.Click

        'Store output message to the output TextBox

        OutputTextBox.Text = "My name: " & NameTextBox.Text _

            & ControlChars.NewLine & "Major: " _

            & MajorTextBox.Text & ControlChars.NewLine _

            & "Local phone: " & PhoneMaskedTextBox.Text

    End Sub
```

You can also use implicit line continuation. VB 10 allows you to continue a line just by pressing the Enter key to break the line into two or more lines, but the line break has to come:

- after a comma, or
- after an opening parenthesis or before a closing parenthesis, or
- after a concatenation operator (&), or
- after the equal sign in an assignment statement.

Valid Examples (valid break after the equal sign and the concatenation operator):

```
        OutputTextBox.Text =

            "My name: " &

            NameTextBox.Text

        OutputTextBox.Text =

            "My name: " & NameTextBox.Text & ControlChars.NewLine &

            "Major: " & MajorTextBox.Text & ControlChars.NewLine &

            "Local phone: " & PhoneMaskedTextBox.Text
```

Invalid Example (break at the wrong place for both the equal sign and the concatenation operator)

```
        OutputTextBox.Text

            = "My name: "

            & NameTextBox.Text
```

Continue In-Class Exercise

Add code for the **Click** event for both the **Display Message** and **Exit** buttons.

Test the display of output.

RadioButtons – CheckedChanged Event

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.
- Each RadioButton control can have its own **CheckedChanged** event procedure.
- You will use the **CheckedChanged** event to modify the color properties of different controls on the form.

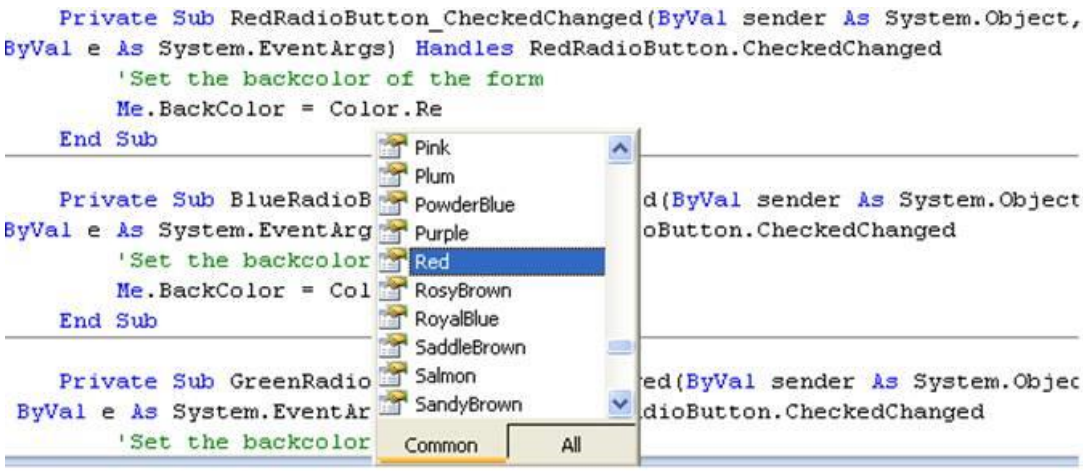
Setting Control Colors

Most controls have both **ForeColor** and **BackColor** properties. To review earlier material:

- **ForeColor** property – sets the color of the text display.
- **BackColor** property – sets the background color of a control.

Colors are defined in VB as an **enumeration** of values – 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**. Intellisense will list enumerated color values by their name.

Colors are set by use of an assignment statement along with the Intellisense popup window's list of enumerated values as shown in this figure and the examples below the figure.



Intellisense popup window displays defined values of the Color enumeration.

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
 'Set the background color of the form to blue
Me.ForeColor = Color.Blue
```

This sub procedure is the code for the **Red** RadioButton control's **CheckedChanged** event.

- The code sets the **BackColor** property of the form (referenced by keyword Me) to red.

```
    Private Sub RedRadioButton_CheckedChanged(ByVal sender As System.Object, ByV
al e As System.EventArgs) HandlesRedRadioButton.CheckedChanged
        'Set the backcolor of the form
        Me.BackColor = Color.Red
    End Sub
```

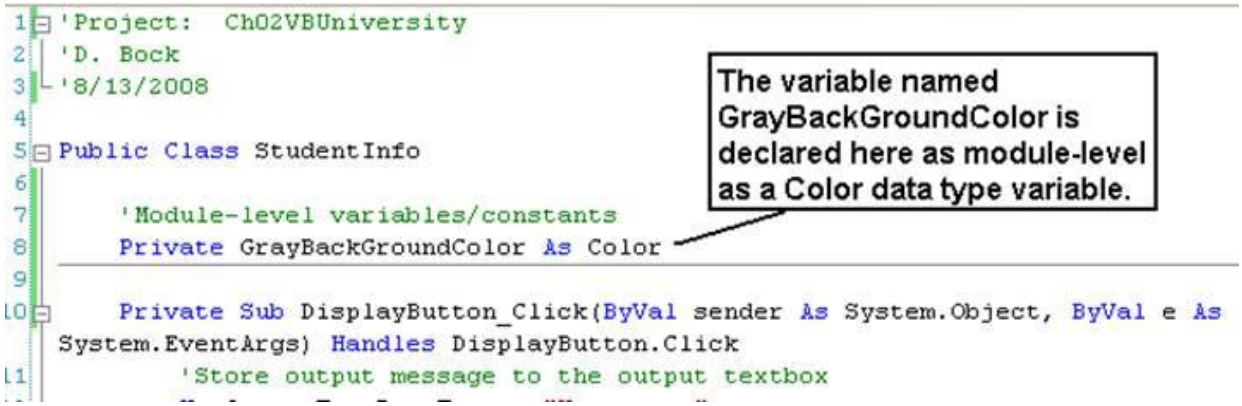
This sub procedure changes the **ForeColor** property of the form changing the text from the default color black to the color white.

```
    Private Sub WhiteRadioButton_CheckedChanged(ByVal sender As System.Object, B
yVal e As System.EventArgs) HandlesWhiteRadioButton.CheckedChanged
        'Set form's text color
        Me.ForeColor = Color.White
    End Sub
```

Saving a Color to a Variable

In Chapter 3 you will learn about variables in detail, but basically a variable is a storage location in memory and a variable can store a lot of different types of data. You will learn to use a variable to store the default background color of a form (the default color is control gray).

- Variables can be declared using the keyword Private.
- Variables declared at module-level as shown in the figure below are accessible from any sub procedure within an application.



In chapter 7 you will learn about a Form's **Load** event in detail – here you are introduced to this event.

- Click the Form's title bar to generate the Load event sub procedure.
- The Load event executes when the form loads from disk storage into computer memory.
- Here the form's **BackColor** value (the standard default gray color) is stored to the module-level **GrayBackGroundColor** variable declared above. This code executes when the form initially loads from disk into memory at the beginning of program execution.

```
    Private Sub StudentInfo_Load(ByVal sender As System.Object, ByVal e As Syste
m.EventArgs) Handles MyBase.Load

        'Assign background color to module-level variable

        GrayBackGroundColor = Me.BackColor

    End Sub
```

This sub procedure is the **CheckedChanged** event for the **GrayRadioButton** control. Note how the **GrayBackGroundColor** variable that stores gray color is used to reset the **BackColor** property of the form.

```
    Private Sub GrayRadioButton_CheckedChanged(ByVal sender As System.Object, By
Val e As System.EventArgs) HandlesGrayRadioButton.CheckedChanged

        'Set the backcolor of the form to the default gray

        Me.BackColor = GrayBackGroundColor

    End Sub
```

CheckBox – CheckedChanged Event

CheckBox controls also exhibit a **CheckedChanged** event when their **Checked** property changes value. This code causes the **OutputTextBox** control to appear and disappear depending on whether the **MessageCheckBox** control is checked or not.

```
    Private Sub MessageCheckBox_CheckedChanged(ByVal sender As System.Object, By
Val e As System.EventArgs) HandlesMessageCheckBox.CheckedChanged

        'Make the OutputTextBox and OutputLabel invisible
```

```
        OutputTextBox.Visible = MessageCheckBox.Checked

        OutputLabel.Visible = MessageCheckBox.Checked

    End Sub
```

Since both the **Visible** and **Checked** properties of controls are Boolean (both store **True** or **False**), you can store the value of a **Checked** property to the **Visible** property of another control and cause it to appear/disappear.

Continue In-Class Exercise

Add code to the project as follows:

- Code the **CheckedChanged** events for all RadioButtons so that they function as described above.
- Add a module-level variable to store the Form's default background color.
- Code the form's Load event as shown above.
- Add code for the **CheckedChanged** event for the CheckBox control on the form.

Test the display of output.

PictureBox Control – Click Event

A PictureBox control's **Click** event is accessed by double-clicking on the control.

The effect you want is to turn the sun into the snow, and back into the sun again when one of the PictureBox control images is clicked. This is coded with the **Visible**property by setting the property to **True** (displays the image) or **False** (makes the image invisible).

Code for the **Click** event sub procedure for the **SunPictureBox** control is:

```
    Private Sub SunPictureBox_Click(ByVal sender As System.Object, ByVal e As Sy
stem.EventArgs) Handles SunPictureBox.Click

        'Turn the sun into the snow

        SnowPictureBox.Visible = True

        SunPictureBox.Visible = False

    End Sub
```

Code for the Click event sub procedure for the SnowPictureBox control is:

```
    Private Sub SnowPictureBox_Click(ByVal sender As System.Object, ByVal e As System.
EventArgs) Handles SnowPictureBox.Click

        'Turn the snow into the sun

        SnowPictureBox.Visible = False
```

```
        SunPictureBox.Visible = True

    End Sub
```

Continue In-Class Exercise

Add code for the **Click** events for both of the PictureBox controls and test their functionality. Drag one PictureBox control on top of the other one.

Test the display of output – it should appear as if the snow turns into the sun and back into the snow again.

Reset Button – Click Event

The Reset button control is used to reset the form to its original condition so that it is prepared for data entry for another student.

- This requires a number of different coding statements.
- A good way to organize the logic for this event is to first type in remarks to outline the task requirements as shown in this sub procedure outline.

```
    Private Sub ResetButton_Click(ByVal sender As System.Object, ByVal e As Syst
em.EventArgs) Handles ResetButton.Click

        'Clear the input and output TextBox and MaskedTextBox controls -

        'This code shows use of the Clear method, String.Empty value, and

        'empty double quote marks


        'Reset the form's BackColor to gray

        'and ForeColor to black


        'Reset the message display CheckBox control to checked


        'Turn on the sun and turn off the snow


        'Set focus to first TextBox

    End Sub
```

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()

MajorTextBox.Clear()

PhoneMaskedTextBox.Clear()
```

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.

Setting the Focus

If you wish to set the focus of the cursor to a specific control, such as the first TextBox on a form, you use the **Focus** method. Example:

```
NameTextBox.Focus()
```

The completed **Click** event sub procedure for the Reset button is:

```
    Private Sub ResetButton_Click(ByVal sender As System.Object, ByVal e As Syst
em.EventArgs) Handles ResetButton.Click
```

```
        'Clear the input and output TextBox and MaskedTextBox controls -
        'This code shows use of the Clear method, String.Empty value, and
        'empty double quote marks
        NameTextBox.Clear()
        MajorTextBox.Text = String.Empty
        PhoneMaskedTextBox.Clear()
        OutputTextBox.Text = ""

        'Reset the form's BackColor to gray
        'and ForeColor to black
        GrayRadioButton.Checked = True
        BlackRadioButton.Checked = True

        'Reset the message display CheckBox control to checked
        MessageCheckBox.Checked = True

        'Turn on the sun and turn off the snow
        SunPictureBox.Visible = True
        SnowPictureBox.Visible = False

        'Set focus to first TextBox
        NameTextBox.Focus()
    End Sub
```

Continue In-Class Exercise

Code the Reset button control's **Click** event sub procedure.

Run the project and test the click events for the button.

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.

The code shown here 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
```

You'll notice that the code inside the **With** block is indented four spaces. This is normal coding procedure to make the code easier to read – Visual Basic will automatically indent the code.

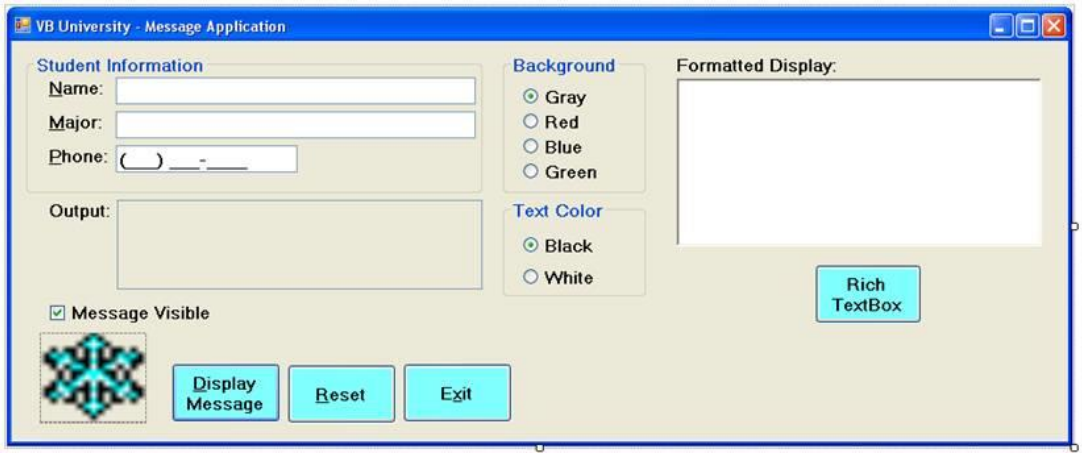
This completes the notes for this chapter. Please take time to rework the In-Class Exercise so that you learn the material before proceeding to your next lab assignment.

Optional Exercise – Loading a RichTextBox

In this exercise you will practice loading the information from a RTF (Rich Text File) into a RichTextBox.

1. Copy the RTF file named **RichTextBoxContents.rtf** into your project folder. Store the file into the **\bin\Debug** folder. Note that you can create your own file by using Microsoft Word and saving the document using file type of "rtf".

2. Alter the form layout as shown in this figure.



- Add a **Label** – set the text to display the words **Formatted Display:.**
- Add a **RichTextBox** – use the default name of **RichTextBox1**.
- Add a **Button** control – name it **RichButton**.
- Add code to the Button control's click event to load data into the RichTextBox from the file

```
    Private Sub RichButton_Click(ByVal sender As System.Object, ByVal e As Syst
em.EventArgs) Handles RichButton.Click

        'Display information from Rich TextBox Contents.rtf file to the
RichTextBox
```

```
        RichTextBox1.LoadFile("RichTextBoxContents.rtf")

    End Sub
```

Test the application. The information from the file should display in the RichTextBox.

Solution to In-Class Exercise

```
'Project:  Ch02VBUniversity
'D. Bock
'Today's Date
 Public Class StudentInfo
     'Module-level variables/constants
    Private GrayBackGroundColor As Color


    Private Sub DisplayButton_Click(ByVal sender As System.Object, ByVal e As Sy
stem.EventArgs) Handles DisplayButton.Click
        'Store output message to the output textbox using implicit line
continuation
        OutputTextBox.Text =
            "My name: " & NameTextBox.Text & ControlChars.NewLine &
            "Major: " & MajorTextBox.Text & ControlChars.NewLine &
            "Local phone: " & PhoneMaskedTextBox.Text
    End Sub


    Private Sub ResetButton_Click(ByVal sender As System.Object, ByVal e As Syst
em.EventArgs) Handles ResetButton.Click
```

```
        'Clear the input and output TextBox and MaskedTextBox controls -
        'This code shows use of the Clear method, String.Empty value, and
        'empty double quote marks
        NameTextBox.Clear()
        MajorTextBox.Text = String.Empty
        PhoneMaskedTextBox.Clear()
        OutputTextBox.Text = ""

        'Reset the form's BackColor to gray
        'and ForeColor to black
        GrayRadioButton.Checked = True
        BlackRadioButton.Checked = True

        'Reset the message display CheckBox control to checked
        MessageCheckBox.Checked = True

        'Turn on the sun and turn off the snow
        SunPictureBox.Visible = True
        SnowPictureBox.Visible = False

        'Set focus to first TextBox
        NameTextBox.Focus()
    End Sub

    Private Sub ExitButton_Click(ByVal sender As System.Object, ByVal e As Syste
m.EventArgs) Handles ExitButton.Click
        'Exit the form
        Me.Close()
    End Sub

    Private Sub SnowPictureBox_Click(ByVal sender As System.Object, ByVal e As S
ystem.EventArgs) Handles SnowPictureBox.Click
        'Turn the snow into the sun
```

```
        SnowPictureBox.Visible = False

        SunPictureBox.Visible = True

    End Sub


    Private Sub SunPictureBox_Click(ByVal sender As System.Object, ByVal e As Sy
stem.EventArgs) Handles SunPictureBox.Click

        'Turn the sun into the snow

        SnowPictureBox.Visible = True

        SunPictureBox.Visible = False

    End Sub


    Private Sub GrayRadioButton_CheckedChanged(ByVal sender As System.Object, By
Val e As System.EventArgs) Handles GrayRadioButton.CheckedChanged

        'Set the backcolor of the form to the default gray

        Me.BackColor = GrayBackGroundColor

    End Sub


    Private Sub RedRadioButton_CheckedChanged(ByVal sender As System.Object, ByV
al e As System.EventArgs) Handles RedRadioButton.CheckedChanged

        'Set the backcolor of the form

        Me.BackColor = Color.Red

    End Sub


    Private Sub BlueRadioButton_CheckedChanged(ByVal sender As System.Object, By
Val e As System.EventArgs) Handles BlueRadioButton.CheckedChanged

        'Set the backcolor of the form

        Me.BackColor = Color.Blue

    End Sub


    Private Sub GreenRadioButton_CheckedChanged(ByVal sender As System.Object, B
yVal e As System.EventArgs) HandlesGreenRadioButton.CheckedChanged

        'Set the backcolor of the form

        Me.BackColor = Color.Green

    End Sub
```

```
    Private Sub WhiteRadioButton_CheckedChanged(ByVal sender As System.Object, B
yVal e As System.EventArgs) HandlesWhiteRadioButton.CheckedChanged

        'Set form's text color

        Me.ForeColor = Color.White

    End Sub


    Private Sub BlackRadioButton_CheckedChanged(ByVal sender As System.Object, B
yVal e As System.EventArgs) HandlesBlackRadioButton.CheckedChanged

        'Set form's text color

        Me.ForeColor = Color.Black

    End Sub


    Private Sub MessageCheckBox_CheckedChanged(ByVal sender As System.Object, By
Val e As System.EventArgs) Handles MessageCheckBox.CheckedChanged

        'Make the OutputTextBox and OutputLabel invisible

        OutputTextBox.Visible = MessageCheckBox.Checked

        OutputLabel.Visible = MessageCheckBox.Checked

    End Sub


    Private Sub StudentInfo_Load(ByVal sender As System.Object, ByVal e As Syste
m.EventArgs) Handles MyBase.Load

        'Assign background color to module-level variable

        GrayBackGroundColor = Me.BackColor

    End Sub


    Private Sub RichButton_Click(ByVal sender As System.Object, ByVal e As Syste
m.EventArgs) Handles RichButton.Click

        'Display information from Rich TextBox Contents.rtf file to the
RichTextBox

        RichTextBox1.LoadFile("RichTextBoxContents.rtf")

    End Sub


End Class
```