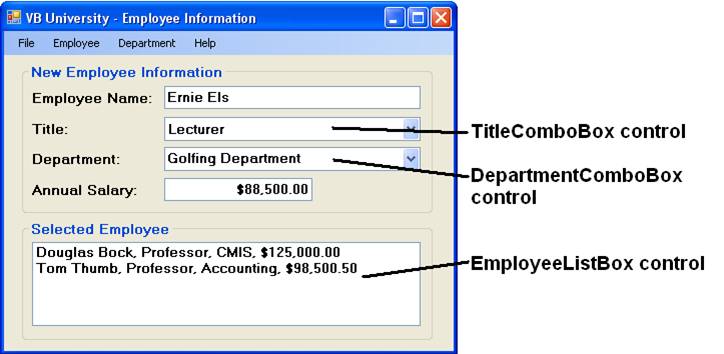
**Chapter 6**

**Lists and Loops**

**Coverage:**

This chapter teaches you to use ListBox and ComboBox controls in designing forms. Additionally, you will learn to code loop structures, both condition-controlled (Do loops) and count-controlled loops (For...Next loops).

In this chapter you will develop a project for the VB University that can be used for data entry of employee information. The form will make use of a new type of control – a list control. This form includes both a ListBox and two ComboBox controls. List controls allow an application user to select from a list of items.

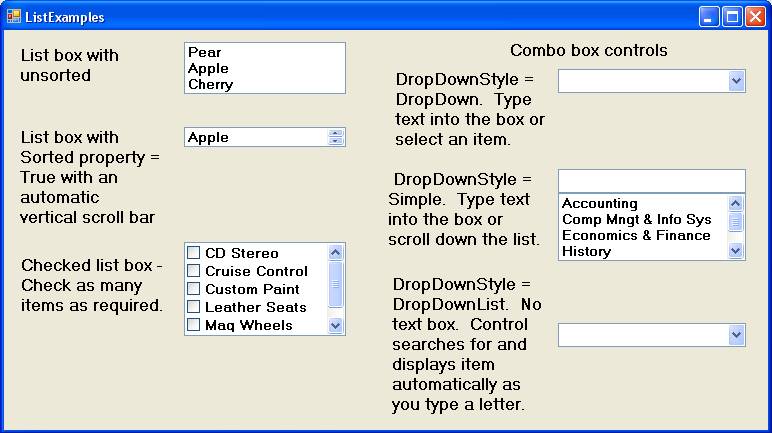


**ListBox, CheckedListBox, and ComboBox**

**Controls to List Items**

Visual basic provides three controls to enable an application user to select from a list of items: ListBox, CheckedListBox, and ComboBox.

This figure (a VB project form) displays different styles for these controls. To see the figure, run the **Ch07VBUniversity** project you copied earlier – display this form by setting the **Startup Form property** inside the **My Project** dialog box.



* **ListBox** – size the control as desired; application user selects from the list – new items cannot be typed into the list.
* **CheckedListBox** – has a list of checkboxes inside a ListBox.
* **ComboBox** – has a **DropDownStyle** property that affects how the control works.
  + **DropDownStyle** = **DropDown** – includes a TextBox and ListBox together, you can type new items into the TextBox portion of the control, and the control does not consume very much space on the form. The list drops down when you click the drop-down arrow.
  + **DropDownStyle** = **Simple** – you can type text into the TextBox at the top of the ComboBox control – the list displays in a ListBox format and the you can scroll up and down the listing.
  + **DropDownStyle** = **DropDownList** – a ComboBox with no textbox into which to type new items. As you type a letter, the control automatically searches for the first item in the listing that begins with that letter. The listing drops down when you click the drop-down arrow.

ListBox, CheckedListBox, and ComboBox controls:

* Automatically have a **vertical scroll bar** added by VB when the list is too large to display all items on the form.
* Have a **Sorted** property that will automatically sort the list when set to **True**.

ListBox controls display the **Name** property at design time – you cannot delete it, but the name does not appear at run time. Also, if items are added to the control the name property disappears at design time.

ComboBox controls display the **Text** property. At run time they display the appropriate values so don't worry about what they display at design time.

**The Items Collection**

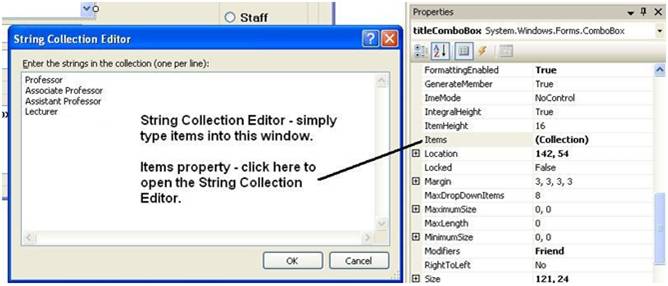
The list of items that display one of these controls is called a **collection**.

* The collection is accessed through the **Items** property.
* The Items collection has properties and methods to make it easy to add items, remove items, and refer to individual items.
* The items in the list are **numbered** beginning with the number **0**. A list of **10** items has the items numbered **0** through **9**.

**Filling the List**

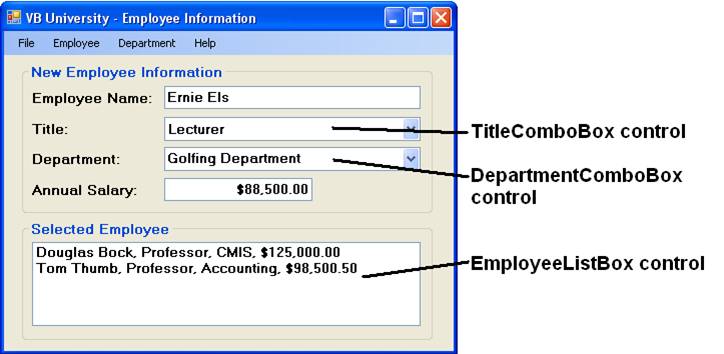
If the list of items varies, the best way to fill a list is to read the list into the control's Items collection from a database table – you will learn to do this in a later chapter in your studies.

If the list of items never changes or does not change much, you can add items at design type by clicking on the **Items** property shown in this figure – a **String Collection Editor** window will open and you type the items into the editor window.



Start VB and open the **Ch07VBUniversity** project that you copied earlier. Set the **Startup Form** in **MyProject** to **Employee** and open the form.

Note that the form is missing two **ComboBox** and one **ListBox** control – add the controls with control names as indicated.



Property settings for the controls:

* **TitleComboBox**:
  + **DropDownStyle** – **DropDownList**
  + **Items** **Collection** list items:
    - Professor
    - Associate Professor
    - Assistant Professor
    - Lecturer
  + **Sorted** property – **True**
* **DepartmentComboBox**:
  + DropDownStyle - **DropDown**
  + Items Collection list items:
    - Accounting
    - Economics & Finance
    - CMIS
    - Management & Marketing
  + **Sorted** property – **True**
* **EmployeeListBox**:
  + **Sorted** property – **True**

The form has a menu strip control with the following menu entries.

|  |  |  |  |
| --- | --- | --- | --- |
| File  Count  Exit | Employee  Add Employee  Reset Employee  ----------------------------------  Clear Employee List  Display Selected Employee | Department  Add Department  Count Departments  Remove Department  Remove At Department | Help  About |

Check and set the form's **Tab Order**.

Run the project – test that the ComboBox controls have appropriate lists as specified by you at design time.

**Coding ListBox and ComboBox Controls**

**Items.Add Method**

The **Items.Add** method is used to add an item to a list or ComboBox at run time.

Example commands to add a new faculty listing to the **EmployeeListBox** and new departments to the **DepartmentComboBox** controls are shown below:

**Dim NewEmployeeNameString As String = "Mary Sumner"**

**Dim NewDepartmentNameString As String = "Chemistry"**

**'This adds a literal value.**

**EmployeeListBox.Items.Add("Mary Sumner")**

**'This adds a value from a TextBox control.**

**EmployeeListBox.Items.Add(EmployeeNameTextBox.Text)**

**'This adds a value from a string memory variable.**

**EmployeeListBox.Items.Add(NewEmployeeNameString )**

**'The Items.Add method works for ComboBoxes in the same fashion.**

**'This adds a new department name using a literal value.**

**DepartmentComboBox.Items.Add("Chemistry")**

**'This adds a value from the text property of the ComboBox control.**

**DepartmentComboBox.Items.Add(DepartmentComboBox.Text)**

**'This adds a value from a string memory variable.**

**DepartmentComboBox.Items.Add(NewDepartmentNameString)**

**Task #1:** Code the Click event sub procedure for the **Add Department** menu item.

* Add the value typed into the **DepartmentComboBox** control's **Text** property to the **Items** collection of the ComboBox control.
* Check to ensure that the **Text** property is not the empty string.

**Private Sub AddDepartmentToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AddDepartmentToolStripMenuItem.Click**

**'Add department text property to ComboBox listing**

**'if not the empty string**

**If DepartmentComboBox.Text <> String.Empty Then**

**'Ok to add**

**DepartmentComboBox.Items.Add(DepartmentComboBox.Text)**

**Else**

**'Cannot add**

**MessageBox.Show("You must type a new department name.", "Name Missing Error", MessageBoxButtons.OK, MessageBoxIcon.Error)**

**DepartmentComboBox.Focus()**

**End If**

**End Sub**

In the above, if the **Text** property is not the **empty string**, the **Text** property value is added to the Items collection. The code does not handle the insertion of duplicate department name items – you will learn to write code to prevent this error later in this chapter.

**Task #2:** Code the Click event sub procedure for the **Add Employee** menu item.

* Create a string variable named **NewEmployeeString**.
* Store to the string variable the concatenated value of the **EmployeeNameTextBox.Text** & **TitleComboBox.Text** & **DepartmentComboBox.Text** & **SalaryTextBox.Text** – separate each entry with a comma and blank space.
* Add the string value to the **EmployeeListBox** control.
* Clear the rest of the form and set the focus to the **EmployeeNameTextBox** in order to prepare for entering the next employee’s information

**Private Sub AddEmployeeToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AddEmployeeToolStripMenuItem.Click**

**'Build the string to add**

**Dim NewEmployeeString As String = EmployeeNameTextBox.Text & ", " & TitleComboBox.Text & ", " & DepartmentComboBox.Text & ", " & SalaryTextBox.Text**

**'Add string to ListBox**

**EmployeeListBox.Items.Add(NewEmployeeString)**

**'Clear the form - ready to add another employee**

**EmployeeNameTextBox.Clear()**

**TitleComboBox.SelectedIndex = -1**

**DepartmentComboBox.SelectedIndex = -1**

**SalaryTextBox.Clear()**

**'Set focus**

**EmployeeNameTextBox.Focus()**

**End Sub**

**Items.Insert Method**

If you do not sort a list, you can use the **Items.Insert** method to specify exactly where within an Items collection to add a new item. You do this by specifying the number position in the collection – this is also called the index position of the new item.

This will add the item **Chemistry** as the very first item in the department ComboBox control; however, if the **Sorted** property is **True**, then specifying the location will not have any effect.

**DepartmentComboBox.Items.Insert(0, "Chemistry")**

If the **ListBox** or **ComboBox** control is **sorted**, then the **Insert** method appears to work exactly as the **Add** method – the new item is added at the location specified, but then control then immediately **resorts** the listing.

**Items.Clear Method**

The **Items.Clear** method will remove the contents of a list or ComboBox.

**DepartmentComboBox.Items.Clear()**

**Task:** Code the Click event sub procedure for the **Clear Employee List** menu item.

* Declare a dialog result variable and display a message box that asks the system user if they want to "**Clear employee listing?**" – capture their response to the variable.
* The message box will display **Yes** and **No** buttons – make **No** the default button with the **MessageBoxDefaultButton.Button2** parameter.
* If the response is **Yes**, then clear the ListBox control.

**Private Sub ClearToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ClearToolStripMenuItem.Click**

**'Use response variable to capture user response**

**Dim ResponseDialogResult As DialogResult = MessageBox.Show("Clear employee listing Y/N?", "Yes or No", MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2)**

**'Test to confirm whether to clear listing**

**If ResponseDialogResult = Windows.Forms.DialogResult.Yes Then**

**EmployeeListBox.Items.Clear()**

**End If**

**End Sub**

Run the project, add two faculty members, then test the ability to clear the listing.

**SelectedIndex and SelectedItem Properties**

When you select an item from a list, the index number associated with the item is stored to the **SelectedIndex** property.

* When no item is selected, **SelectedIndex** equals **-1**.
* You can select (highlight) an item in a list by storing a numeric value to the **SelectedIndex** property.

**'This selects the 4th department item**

**DepartmentComboBox.SelectedIndex = 3**

**'This unselects any selected (highlighted) item**

**DepartmentComboBox.SelectedIndex = -1**

When you select an item from a list, the item selected is stored to the **SelectedItem** property. This property can be used to display the selection elsewhere, for example, in a message box or to a TextBox control.

**'Displays the selected current faculty member to a message box**

**MessageBox.Show(EmployeeListBox.SelectedItem.ToString)**

Another way, though more cryptic, to display the selection elsewhere is to use the Items collection Index value.

To display an item in a list to another control, such as a textbox named **EmployeeInformationTextBox**, the command is shown here.

* The **ToString** method ensures the item (that is stored in the list as an **object**) converts to string for storage to the Text property of the TextBox control.
* The **ToString** method is optional, but MUST be used if **Option Strict On** is set.

**'Displays first item to the textbox**

**EmployeeInformationTextBox.Text = EmployeeListBox.Items(0).ToString()**

The **Reset Employee** menu item Click event sub procedure demonstrates setting the **SelectedIndex** property to **-1** to unselect items in two ComboBox controls.

**Private Sub ResetEmployeeToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ResetEmployeeToolStripMenuItem.Click**

**'Clear TextBox controls**

**EmployeeNameTextBox.Clear()**

**SalaryTextBox.Clear()**

**'Unselect title and department ComboBox controls and**

**'set Text property to empty string**

**TitleComboBox.SelectedIndex = -1**

**DepartmentComboBox.SelectedIndex = -1**

**TitleComboBox.Text = String.Empty**

**DepartmentComboBox.Text = String.Empty**

**'Unselect the currently selected faculty member in the Listbox**

**EmployeeListBox.SelectedIndex = -1**

**'Reset focus to the employee name TextBox**

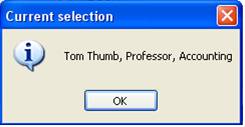
**EmployeeNameTextBox.Focus()**

**End Sub**

**In-Class Exercise**

**Task #1:** Code the Click event sub procedure for the **Display Selected Faculty** menu item.

* Use a message box to display the current selected item from the **EmployeeListBox** control.



**Private Sub DisplaySelectedToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles DisplaySelectedToolStripMenuItem.Click**

**'Use messagebox to display selected employee**

**If EmployeeListBox.SelectedIndex = -1 Then**

**MessageBox.Show("You must select an employee member to display.", "No Selection Error", MessageBoxButtons.OK, MessageBoxIcon.Error)**

**Else**

**MessageBox.Show(EmployeeListBox.SelectedItem.ToString, "Current selection", MessageBoxButtons.OK, MessageBoxIcon.Information)**

**End If**

**End Sub**

**Task #2:** Code the Click event sub procedure for the **Reset Employee** menu item as shown earlier.

**Items.Count Property**

The **Items.Count** property stores a number equal to the number of items in a list or ComboBox Items collection.

* The value of **Items.Count** is always 1 more than the maximum allowable **SelectedIndex** value.
* If there are 5 items in a list, then **SelectedIndex** possible values are 0 through 4, but **Items.Count** equals 5.

Use this property to display the number of items in a list.

**Private Sub CountDepartmentsToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CountDepartmentsToolStripMenuItem.Click**

**'Display a message box with the count of the number of departments**

**Dim MessageString As String = "Number of departments: " & DepartmentComboBox.Items.Count.ToString()**

**Dim TitleString As String = "Count of Departments"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Information)**

**End Sub**

**Items.RemoveAt and Items.Remove Methods**

To remove an individual item from a list, specify either the index or text of the item.

* **Items.RemoveAt** method – removes an item by specifying the index position.
* **Items.Remove** method – removes an item by specifying the item name.

The **RemoveAt** method fails if an item has not been selected for removal because the value of **SelectedIndex = -1** and you cannot remove a non-existent item – this throws an exception: **ArgumentOutOfRangeException**.

When the **Remove** method is used and an item has not been selected for removal, the method simply does nothing.

**'Removes the first item from the list**

**DepartmentComboBox.Items.RemoveAt(0)**

**'Removes the item according to the value of IndexInteger**

**DepartmentComboBox.Items.RemoveAt(IndexInteger)**

**'Remove the currently selected item**

**DepartmentComboBox.Items.RemoveAt(DepartmentComboBox.SelectedIndex)**

**'Remove item by name where the item name is stored**

**'to the SelectedItem property of the ComboBox**

**DepartmentComboBox.Items.Remove(DepartmentComboBox.SelectedItem)**

**In-Class Exercise**

**Task #1:** Code the Click event sub procedure for the **Department** menu, **Count Departments** menu item.

* Display the current number of departments to a message box.



**Private Sub CountDepartmentsToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CountDepartmentsToolStripMenuItem.Click**

**'Display a message box with the count of the number of departments**

**Dim MessageString As String = "Number of departments: " & DepartmentComboBox.Items.Count.ToString()**

**Dim TitleString As String = "Count of Departments"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Information)**

**End Sub**

**Task #2:** Code the Click event sub procedure for the **Department** menu, **Remove Department** menu item.

* Use the Items collection **Remove** method.
* Use a message box to ask the system user to confirm to remove the selected department as shown here.



**Private Sub RemoveDepartmentToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RemoveDepartmentToolStripMenuItem.Click**

**'Declare dialog result variable**

**Dim ResponseDialogResult As DialogResult = MessageBox.Show("Remove the selected department?", "Remove ?",** MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2)

If ResponseDialogResult = Windows.Forms.DialogResult.Yes Then

'Remove the selected departme**nt from the listing**

**DepartmentComboBox.Items.Remove(DepartmentComboBox.SelectedItem)**

**'This next line of code clears the Text**

**'property of the ComboBox – this is automatic**

**'for the Remove method, but not for the last**

**'item in a list.**

**DepartmentComboBox.Text = String.Empty**

**End If**

**End Sub**

**Task #3:** Code for the Click event sub procedure for the **Department** menu, **Remove At Department** menu item.

* Use the Items collection **RemoveAt** method.
* Use a message box to ask the system user to confirm to remove the selected department.
* Use a Try-Catch block to trap the possible **ArgumentOutOfRangeException** that will be thrown if an item is not selected. Display a message box like the one shown in this figure.



**Private Sub RemoveAtDepartmentToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RemoveAtDepartmentToolStripMenuItem.Click**

**'Try to remove the department if one is selected**

**Try**

**'Declare dialog result variable**

**Dim ResponseDialogResult As DialogResult = MessageBox.Show("Remove the selected department?", "Remove ?", MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2)**

**If ResponseDialogResult = Windows.Forms.DialogResult.Yes Then**

**'Remove the selected department from the listing**

**DepartmentComboBox.Items.RemoveAt(DepartmentComboBox.SelectedIndex)**

**'This next line of code clears the Text**

**'property of the ComboBox – this is automatic**

**'for the Remove method, but not for the last**

**'item in a list.**

**DepartmentComboBox.Text = String.Empty**

**End If**

**Catch ex As ArgumentOutOfRangeException**

**MessageBox.Show("You must select a department to remove.", "No Selection Was Made", MessageBoxButtons.OK, MessageBoxIcon.Error)**

**End Try**

**End Sub**

**ListBox and ComboBox Events**

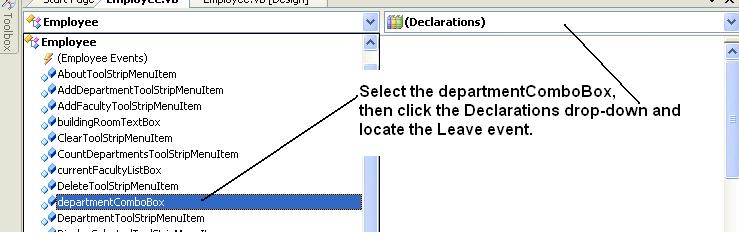
Code can be written for several events of ListBoxes and ComboBoxes. These include:

* **SelectedIndexChanged Event**. Occurs when a system user makes a new selection for a list or ComboBox.
* **TextChanged Event**. As a system user types characters into the TextBox portion of a ComboBox, this event is triggered for each character typed. ListBoxes do not have this event.
* **Enter** **Event**. When a list or ComboBox receives focus, this event occurs.
* **Leave** **Event**. When a list or ComboBox loses focus, this event occurs.

**In-Class Exercise**

**Task:** Code the **Leave** event.

* In the object drop-down ComboBox, select **DepartmentComboBox** as shown in the figure below.
* Next select the **Leave** event from the Declarations (property, method and event) drop-down ComboBox.



* Code the **Leave** event to add the value that is typed into the **Text** property of **DepartmentComboBox** to the Items collection of **DepartmentComboBox**.

**Private Sub DepartmentComboBox\_Leave(ByVal sender As Object, ByVal e As System.EventArgs) Handles DepartmentComboBox.Leave**

**'Store ComboBox Text property value to Items collection**

**DepartmentComboBox.Items.Add(DepartmentComboBox.Text)**

**End Sub**

* Note that this code enables adding **duplicate values** as well as blank values. Later you will learn how to prevent this from happening.
* Test the code.
* After testing, delete the Leave event code.

**Loops**

**Loop** structures enable you to execute a series of VB statements over and over. An **iteration** is a single execution of the loop.

There are two types of loops:

* **Condition-controlled**. In VB, these are **Do Loops** – loops that execute over and over until a specific condition occurs. In VB.NET, these are **Do Loops**.
* **Count-controlled**. In VB, these are **For…Next Loops** – loops that execute for a specified number of times.

**The Boolean Data Type**

A **Boolean** variable has a value of either **True** or **False**.

* Use a Boolean variable to control the stopping condition for a loop.
* The **default value** for a declared Boolean variable is **False**. Some programmers explicitly set the variable to **False** when it is declared to make it clear that the starting value for a loop search is initially false.
* An example of declaring a Boolean is:

**Dim FoundBoolean As Boolean = False**

**Do Loops**

**Do Loops** can execute the coding statements within the body of the loop either:

* until a condition is met that causes the loop to terminate (stop executing),
* while a condition is met that causes a loop to continue executing.

**Do Loops** can have the **condition test** (termed the **stopping condition**) at either the top or bottom of a loop, and can use either a **While** or **Until** option to test the condition.

**Condition at Top of Loop – While Logic.** The following loop executes while the stopping condition **FoundBoolean** is **False**.

**Dim FoundBoolean As Boolean = False**

**Do While FoundBoolean = False**

**'Coding statements to execute.**

**'Code to search for some value**

**If ValueIsFound = True Then**

**FoundBoolean = True**

**Else**

**IndexInteger += 1**

**End If**

**Loop**

**'Control transfers here after the loop finishes.**

**Condition at Top of Loop – Until Logic.** This loop works like the one above, but uses **Until** logic. The loop executes until the stopping condition **FoundBoolean** is **True**.  These are the mirror image of each other. Use the approach that you like best.

**Dim FoundBoolean As Boolean = False**

**Do Until FoundBoolean = True**

**'Coding statements to execute.**

**'Code to search for some value**

**If ValueIsFound = True Then**

**FoundBoolean = True**

**Else**

**IndexInteger += 1**

**End If**

**Loop**

**'Control transfers here after the loop finishes.**

**Condition at Bottom of Loop – While Logic.** This next loop will always execute the body of the loop at least one time because the stopping condition is not tested until the end of the loop.

**Dim FoundBoolean As Boolean = False**

**Do**

**'Coding statements to execute.**

**'Code to search for some value**

**If ValueIsFound = True Then**

**FoundBoolean = True**

**Else**

**IndexInteger += 1**

**End If**

**Loop While FoundBoolean = False**

**'Control transfers here after the loop finishes.**

You can also test for an **Until** condition at the bottom of a Do Loop.

**Searching a ListBox or ComboBox**

As was noted earlier, one of the problems with adding new departments to the **DepartmentComboBox** control is that you can end up with duplicate departments – Duplicates can occur when adding items to any ComboBox.

Search coding procedures requires the following concepts:

* When adding items, **ACCOUNTING** is not equal to **Accounting** or **accounting** – you can work around this potential problem so that you can search for duplicates by adding the **.ToUpper** method to search items – this causes VB to treat all values as if they are typed in capital letters.
* Assume the department to be added is **NOT** in the items collection by setting **FoundBoolean** = **False**.
* Start the search with the first department in the ComboBox by setting **IndexInteger** = **0**.
* Use a Do Loop is used to examine each department in the **Items** collection starting at item **0** until there are no more items.
* If a department in the collection matches the Text property value, then set **FoundBoolean** = **True** and stop the search.
* If no department matches the Text property value, the item index variable **IndexInteger** will eventually grow in value until it equals **DepartmentComboBox.Count** – in this situation, you can add the department to the ComboBox Items Collection property.
* Note the use of multiple methods **ToString**, **Trim**, and **ToUpper** with the **DepartmentComboBox** to convert the value to a string, trim off any leading/trailing blank spaces, and treat the value as if it were entered in all capital letters.

**Private Sub AddDepartmentToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AddDepartmentToolStripMenuItem.Click**

**'Add department text property to ComboBox listing**

**Dim FoundBoolean As Boolean = False**

**Dim IndexInteger As Integer = 0**

**Do Until FoundBoolean = True Or IndexInteger = DepartmentComboBox.Items.Count**

**If DepartmentComboBox.Text.Trim.ToUpper = DepartmentComboBox.Items(IndexInteger).ToString.Trim.ToUpper Then**

**FoundBoolean = True**

**Else**

**'Add 1 to index value**

**IndexInteger += 1**

**End If**

**Loop 'Loop back and try again**

**'Now decide whether to add item or not**

**If FoundBoolean = False And DepartmentComboBox.Text.Trim <> String.Empty Then**

**DepartmentComboBox.Items.Add(DepartmentComboBox.Text.Trim)**

**Else**

**MessageBox.Show("Duplicate or Invalid Department Name", "Duplicate Data Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)**

**DepartmentComboBox.Focus()**

**DepartmentComboBox.SelectAll()**

**End If**

**End Sub**

**In-Class Exercise**

Replace the code for the **Add Department** menu item's click event with the code shown above.

Run the project and test the system by adding a new department (Philosophy), a duplicate department (Accounting), and a blank department value.

**For…Next Loops**

A **For…Next** loop is a **count-controlled** loop – the code executes a set number of times based on a **loop index**.

* A **loop index** is a numeric variable (usually integer or single).
* The loop index is initialized to a **starting value**.  The starting value in the code below is **1**.
* After each execution of the loop, the loop index is changed (made larger or smaller) by the **step** value.  The step value in the example below is **2**. The Step value is optional - if it is not specified, it is automatically assumed to be **1**.
* The loop terminates when the loop index **exceeds** the **ending value**. The **ending value** below is **5**.

This example merely illustrates the mechanics of coding a count-controlled loop.

**Private Sub CountToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CountToolStripMenuItem.Click**

**Dim IndexInteger As Integer**

**For IndexInteger = 1 To 5 Step 2**

**'Print some output so you can see the**

**'loop counting as it loops around.**

**CountListBox.Items.Add(IndexInteger.ToString)**

**Next IndexInteger**

**'Control transfers to here when the loop ends**

**MessageBox.Show("Finished Counting")**

**End Sub**

Other example **For…Next** statements:

**For IndexInteger = 2 To 100 Step 4**

**For IndexInteger = StartValueInteger To EndValueInteger Step IncrementInteger**

**For IndexInteger = 0 To (DepartmentComboBox.Items.Count - 1)**

**For InterestRateDecimal = 0.05D To 0.25D Step 0.05D**

**For BackwardsIndexInteger = 10 To 0 Step -1**

**In-Class Exercise**

**Task 1. Counting Forward**

* Add a ListBox to the form named **CountListBox**, and a menu item under the File menu named **Count**.
* Add the code to the **Count** menu item's Click event shown in the above example.
* Try different values for the ending value and step increment.

**Task 2. Counting Backward**

Modify the code for the File-Count menu to count backwards from 10 to 1 by 1. The solution given here stops when **IndexInteger** is less than 1.

**For IndexInteger = 10 To 1 Step -1**

**'Print some output so you can see the**

**'loop counting as it loops around.**

**CountListBox.Items.Add(IndexInteger.ToString)**

**Next IndexInteger**

**A Loop That Will Not Execute**

A loop condition is always checked the first time that a **For...Next** statement executes. In this example, **IndexInteger** is already larger than the **ending value** for the loop (**FinalValueInteger**).

**Dim FinalValueInteger As Integer = 5**

**For IndexInteger = 6 To FinalValueInteger**

**'Nothing happens – the final value is less than**

**'the starting value**

**Next IndexInteger**

**An Endless Loop**

This loop executes endlessly because **IndexInteger** is reset to the starting value every time the loop executes.

**For IndexInteger = 1 To 5**

**'loop control index is reset**

**IndexInteger = 1**

**Next IndexInteger**

Once a **For...Next** loop begins to execute, the values of **starting value**, **ending value**, and **step** **increment** are set – changing these values within the loop **CAN** affect execution of the loop.

**Nested Loops – A Loop Inside a Loop**

This example shows a **For...Next** loop inside another **For...Next** loop. The inner loop must be completely contained within the outer loop. The inner loop executes a "normal number" of times for each execution of the outer loop.

**Dim TotalLong As Long**

**Dim OuterIndexInteger As Integer**

**Dim InnerIndexInteger As Integer**

**'Start of the outer loop**

**For OuterIndexInteger = 1 To 10**

**'Start of the inner loop – each time we get here the**

**'inner loop starts all over again**

**For InnerIndexInteger = 1 To 10**

**TotalLong += 1**

**CountListBox.Items.Add(TotalLong.ToString)**

**Next InnerIndexInteger**

**'Exited inner loop – control transfers to outer loop**

**Next OuterIndexInteger**

**'Control transfers here when the outer loop ends**

**Exit Statement**

The **Exit For** statement is used to exit a **For...Next** loop before the ending value is reached. Example:

**For IndexInteger = 1 To 10**

**If InputTextBox.Text = String.Empty Then 'Blank input**

**MessageBox.Show("Input is invalid")**

**Exit For**

**End If**

**'Other statements to process if the input is valid**

**Next IndexInteger**

There are also **Exit Do** and **Exit While** and **Exit If** statements. For the most part, using any **Exit** statement is unnecessary if your code is constructed properly.

**Selecting Control Entries**

**Selecting a TextBox Entry**

This technique enables you to highlight all data typed into a TextBox control whenever the application user tabs to the control or selects the control with the computer mouse. The TextBox **Enter** event combined with the **SelectAll** method highlights text.

**Private Sub EmployeeNameTextBox\_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles EmployeeNameTextBox.Enter**

**'Select existing text values**

**EmployeeNameTextBox.SelectAll()**

**End Sub**

**Selecting a ListBox or ComboBox Entry**

You can highlight text in a list or ComboBox. We've seen this already. Simply store the numeric index value of the desired item to the **SelectedIndex** property of the list or ComboBox.

**EmployeeListBox.SelectedIndex = IndexInteger**

**Coding a TextChanged Event**

Expand the size of the form for your Ch07 project – you will find that the form already has a TextBox named **FruitTextBox** and a ListBox named **FruitListBox** as shown in this figure.



This sub procedure shows code for the **TextChanged** event for the **FruitTextBox** control – whenever a character is typed into the TextBox, it triggers a search for a matching value in the **FruitListBox**.

**Private Sub FruitTextBox\_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles FruitTextBox.TextChanged**

**'Initialize variables to control the search**

**Dim IndexInteger As Integer = 0**

**Dim FoundBoolean As Boolean = False**

**Dim ListItemString As String**

**'Start the search**

**Do While Not FoundBoolean And IndexInteger < FruitListBox.Items.Count**

**'Compare the value in the TextBox to the ListBox Items**

**ListItemString = FruitListBox.Items(IndexInteger).ToString.ToUpper**

**If ListItemString.StartsWith(FruitTextBox.Text.ToUpper) Then**

**'found a fruit name that starts with the Text typed thus far**

**FruitListBox.SelectedIndex = IndexInteger**

**FoundBoolean = True**

**Else**

**IndexInteger += 1**

**End If**

**Loop**

**If FoundBoolean = False Then**

**MessageBox.Show("That fruit is not in the list.", "Not Found", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)**

**End If**

**End Sub**

**In-Class Exercise**

**Task #1:** Selecting a TextBox Entry

* Add a TextBox control named **FruitTextBox** and a ListBox control named **FruitListBox** to the form.
* Set the ListBox **Sorted** property to **True**.
* Add some fruit items to the Items collection for the fruit ListBox.
* Use the **SelectAll** method as shown earlier to test your ability to highlight text in the fruit ListBox control.
* Type a value in the TextBox, then tab around the form until the TextBox is again selected – it should highlight the text in the control.

**Private Sub FruitTextBox\_Enter(ByVal sender As Object, ByVal e As System.EventArgs) Handles FruitTextBox.Enter**

**'Select existing text values**

**FruitTextBox.SelectAll()**

**End Sub**

**Task #2:** Selecting a List or ComboBox Entry

* Use the code shown above to search for a value in the **FruitListBox** as you type the name of a type of fruit into the **FruitTextBox**.
* Test the code.

**ValidData Function**

When an application uses menu controls, you will find that menu items do not have a **CausesValidation** property. An alternative approach to validating data is to code a function that returns a value of **True** if data is valid or a value of **False** if data is not valid.

**Coding ValidData Function**

The **ValidData** function shown here demonstrates how to enforce business rules about the data. Typical rules include:

* A data value cannot be missing from a TextBox control.
* An item must be selected from a ComboBox control (with **DropDownStyle** = **DropDownList**).
* An item must be selected or a data value entered for a ComboBox control (with **DropDownStyle** = **DropDown**).
* A TextBox that must store numeric data contains numeric data.
* A value in a TextBox that is numeric is within a specific legal range of values, such as greater than zero.

**Private Function ValidData() As Boolean**

**'Assume data is not valid**

**ValidData = False**

**Dim MessageString, TitleString As String**

**If EmployeeNameTextBox.Text.Trim = String.Empty Then**

**'Business rule #1 - Name cannot be missing**

**MessageString = "Employee name is missing."**

**TitleString = "Employee name error"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Error)**

**EmployeeNameTextBox.Focus()**

**EmployeeNameTextBox.SelectAll()**

**ElseIf TitleComboBox.SelectedIndex = -1 Then**

**'Business rule #2 - Title must be selected**

**'from the list provided in the ComboBox**

**MessageString = "Title was not selected."**

**TitleString = "Title error"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Error)**

**TitleComboBox.Focus()**

**ElseIf DepartmentComboBox.Text.Trim = String.Empty Then**

**'Business rule #3 - Department must be selected**

**'or typed into the Text property of the ComboBox**

**MessageString = "Department name was not selected or entered."**

**TitleString = "Department Name error"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Error)**

**DepartmentComboBox.Focus()**

**ElseIf IsNumeric(SalaryTextBox.Text) = False OrElse Decimal.Parse(SalaryTextBox.Text, Globalization.NumberStyles.Currency) <= 0D Then**

**'Business rule #4 - Salary must be valid number > 0**

**MessageString = "Salary is missing or invalid-must be greater than zero."**

**TitleString = "Salary error"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Error)**

**SalaryTextBox.Focus()**

**SalaryTextBox.SelectAll()**

**Else**

**'Data passes all business rules**

**ValidData = True**

**End If**

**End Function**

* The function is declared **Boolean** because it returns either a value of **True** or **False**.
* Begin by assuming that one or more business rule will be violated by storing a value of False to the ValidData function name.
* Use an **If-ElseIf-Else-End If** structure to code a series of business rules. Here the rules are:
  + Business rule #1: Name cannot be missing.
  + Business rule #2: Title must be selected from the TitleComboBox (**DropDownStyle** = **DropDownList**).
  + Business rule #3: Department must be either entered as Text or selected from the DepartmentComboBox (**DropDownStyle** = **DropDown**).
  + Business rule #4 & #5: Coded as a single ElseIf, the SalaryTextBox must contain a numeric value and it must be greater than zero.
* If all business rules are satisfied, the **Else** branch sets **ValidData** = **True**.
* Remember, storing a value to the name of the function causes this value to be returned at the End Function statement.
* Only one business rule will be tested at a time – the first one violated displays a MessageBox and then exits the function.
* There is no reasonable limit to the number of business rules tested.

**Coding ValidData Function**

By itself the **ValidData** function does nothing – it must be called in order to execute.

* This sub procedure illustrates calling ValidData when adding a new employee to the ListBox control.
* The call is in the highlighted If statement – control passes to ValidData and on return, the value is either True or False. If the value returned is True, then code inside the If statement executes to add employee information to the ListBox control.

**Private Sub AddEmployeeToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AddEmployeeToolStripMenuItem.Click**

**'Add a new employee to the ListBox if Valid**

**If ValidData Then**

**'Build the string to add**

**Dim NewEmployeeString As String = EmployeeNameTextBox.Text & ", " & TitleComboBox.Text & ", " & DepartmentComboBox.Text & ", " & SalaryTextBox.Text**

**'Add string to ListBox**

**EmployeeListBox.Items.Add(NewEmployeeString)**

**'Clear the form - ready to add another employee**

**EmployeeNameTextBox.Clear()**

**TitleComboBox.SelectedIndex = -1**

**DepartmentComboBox.SelectedIndex = -1**

**SalaryTextBox.Clear()**

**'Set focus**

**EmployeeNameTextBox.Focus()**

**End If**

**End Sub**

**In-Class Exercise**

**Task #1:** Add the **ValidData** function to your program.

Task #2: Modify the Click event for the **AddEmployeeToolStripMenuItem** control to call the **ValidData** function.

Test the application one business rule at a time.

**Solution to In-Class Exercise**

This solution shows the **Ch07VBUniversity** project code.

**'Project: Ch07VBUniversity (Solution)**

**‘CS2**

**Option Strict On**

**Public Class Employee**

**Private Sub AddDepartmentToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AddDepartmentToolStripMenuItem.Click**

**'Add department text property to combo box listing**

**Dim FoundBoolean As Boolean = False**

**Dim IndexInteger As Integer = 0**

**Do Until FoundBoolean = True Or IndexInteger = DepartmentComboBox.Items.Count**

**If DepartmentComboBox.Text.Trim.ToUpper = DepartmentComboBox.Items(IndexInteger).ToString.Trim.ToUpper Then**

**FoundBoolean = True**

**Else**

**'Add 1 to index value**

**IndexInteger += 1**

**End If**

**Loop 'Loop back and try again**

**'Now decide whether to add item or not**

**If FoundBoolean = False And DepartmentComboBox.Text.Trim <> String.Empty Then**

**DepartmentComboBox.Items.Add(DepartmentComboBox.Text.Trim)**

**Else**

**MessageBox.Show("Duplicate or Invalid Department Name", "Duplicate Data Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)**

**DepartmentComboBox.Focus()**

**DepartmentComboBox.SelectAll()**

**End If**

**End Sub**

**Private Sub AddEmployeeToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AddEmployeeToolStripMenuItem.Click**

**'Add a new employee to the ListBox if Valid**

**If ValidData Then**

**'Build the string to add**

**Dim NewEmployeeString As String = EmployeeNameTextBox.Text & ", " & TitleComboBox.Text & ", " & DepartmentComboBox.Text & ", " & SalaryTextBox.Text**

**'Add string to ListBox**

**EmployeeListBox.Items.Add(NewEmployeeString)**

**'Clear the form - ready to add another employee**

**EmployeeNameTextBox.Clear()**

**TitleComboBox.SelectedIndex = -1**

**DepartmentComboBox.SelectedIndex = -1**

**SalaryTextBox.Clear()**

**'Set focus**

**EmployeeNameTextBox.Focus()**

**End If**

**End Sub**

**Private Sub ClearEmployeeListToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ClearEmployeeListToolStripMenuItem.Click**

**'Use response variable to capture user response**

**Dim ResponseDialogResult As DialogResult = MessageBox.Show("Clear employee listing Y/N?", "Yes or No", MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2)**

**'Test to confirm whether to clear listing**

**If ResponseDialogResult = Windows.Forms.DialogResult.Yes Then**

**EmployeeListBox.Items.Clear()**

**End If**

**End Sub**

**Private Sub ResetEmployeeToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ResetEmployeeToolStripMenuItem.Click**

**'Clear text box controls**

**EmployeeNameTextBox.Clear()**

**'Unselect title and department combo box controls and**

**'set Text property to empty string**

**TitleComboBox.SelectedIndex = -1**

**DepartmentComboBox.SelectedIndex = -1**

**TitleComboBox.Text = String.Empty**

**DepartmentComboBox.Text = String.Empty**

**'Unselect the currently selected faculty member in the Listbox**

**EmployeeListBox.SelectedIndex = -1**

**'Reset focus to the employee name text box**

**EmployeeNameTextBox.Focus()**

**End Sub**

**Private Sub DisplaySelectedToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles DisplaySelectedToolStripMenuItem.Click**

**'Use messagebox to display selected employee**

**If EmployeeListBox.SelectedIndex = -1 Then**

**MessageBox.Show("You must select an employee member to display.", "No Selection Error", MessageBoxButtons.OK, MessageBoxIcon.Error)**

**Else**

**MessageBox.Show(EmployeeListBox.SelectedItem.ToString, "Current selection", MessageBoxButtons.OK, MessageBoxIcon.Information)**

**End If**

**End Sub**

**Private Sub CountDepartmentsToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CountDepartmentsToolStripMenuItem.Click**

**'Display a message box with the count of the number of departments**

**Dim MessageString As String = "Number of departments: " & DepartmentComboBox.Items.Count.ToString()**

**Dim TitleString As String = "Count of Departments"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Information)**

**End Sub**

**Private Sub RemoveDepartmentToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RemoveDepartmentToolStripMenuItem.Click**

**'Declare dialog result variable**

**Dim ResponseDialogResult As DialogResult = MessageBox.Show("Remove the selected department?", "Remove ?", MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2)**

**If ResponseDialogResult = Windows.Forms.DialogResult.Yes Then**

**'Remove the selected department from the listing**

**DepartmentComboBox.Items.Remove(DepartmentComboBox.SelectedItem)**

**'This next line of code clears the Text**

**'property of the ComboBox – this is automatic**

**'for the Remove method, but not for the last**

**'item in a list.**

**DepartmentComboBox.Text = String.Empty**

**End If**

**End Sub**

**Private Sub RemoveAtDepartmentToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles RemoveAtDepartmentToolStripMenuItem.Click**

**'Try to remove the department if one is selected**

**Try**

**'Declare dialog result variable**

**Dim ResponseDialogResult As DialogResult = MessageBox.Show("Remove the selected department?", "Remove ?", MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Button2)**

**If ResponseDialogResult = Windows.Forms.DialogResult.Yes Then**

**'Remove the selected department from the listing**

**DepartmentComboBox.Items.RemoveAt(DepartmentComboBox.SelectedIndex)**

**'This next line of code clears the Text**

**'property of the ComboBox – this is automatic**

**'for the Remove method, but not for the last**

**'item in a list.**

**DepartmentComboBox.Text = String.Empty**

**End If**

**Catch ex As ArgumentOutOfRangeException**

**MessageBox.Show("You must select a department to remove.", "No Selection Was Made", MessageBoxButtons.OK, MessageBoxIcon.Error)**

**End Try**

**End Sub**

**Private Sub CountToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles CountToolStripMenuItem.Click**

**'Demonstrates using nested For...Next loops.**

**Dim TotalLong As Long**

**Dim OuterIndexInteger As Integer**

**Dim InnerIndexInteger As Integer**

**'Start of the outer loop**

**For OuterIndexInteger = 1 To 3**

**'Start of the inner loop – each time we get here the**

**'inner loop starts all over again**

**For InnerIndexInteger = 1 To 6**

**TotalLong += 1**

**CountListBox.Items.Add(TotalLong.ToString)**

**Next InnerIndexInteger**

**'Exited inner loop – control transfers to outer loop**

**Next OuterIndexInteger**

**MessageBox.Show("Finished Counting.")**

**End Sub**

**Private Sub FruitTextBox\_TextChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles FruitTextBox.TextChanged**

**'Initialize variables to control the search**

**Dim IndexInteger As Integer = 0**

**Dim FoundBoolean As Boolean = False**

**Dim ListItemString As String**

**'Start the search**

**Do While Not FoundBoolean And IndexInteger < FruitListBox.Items.Count**

**'Compare the value in the TextBox to the ListBox Items**

**ListItemString = FruitListBox.Items(IndexInteger).ToString.ToUpper**

**If ListItemString.StartsWith(FruitTextBox.Text.ToUpper) Then**

**'found a fruit name that starts with the Text typed thus far**

**FruitListBox.SelectedIndex = IndexInteger**

**FoundBoolean = True**

**Else**

**IndexInteger += 1**

**End If**

**Loop**

**If FoundBoolean = False Then**

**MessageBox.Show("That fruit is not in the list.", "Not Found", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)**

**End If**

**End Sub**

**Private Sub HelpToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles HelpToolStripMenuItem.Click**

**MessageBox.Show("Programmed by D. Bock", "About Me", MessageBoxButtons.OK, MessageBoxIcon.Information)**

**End Sub**

**Private Sub AboutToolStripMenuItem\_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles AboutToolStripMenuItem.Click**

**'Display about message**

**Dim MessageString As String = "Programmed by D. Bock" & ControlChars.NewLine & "Today's date and time: " & Date.Now**

**Dim TitleString As String = "About Ch07 VB University"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Information)**

**End Sub**

**Private Function ValidData() As Boolean**

**'Assume data is not valid**

**ValidData = False**

**Dim MessageString, TitleString As String**

**If EmployeeNameTextBox.Text.Trim = String.Empty Then**

**'Business rule #1 - Name cannot be missing**

**MessageString = "Employee name is missing."**

**TitleString = "Employee name error"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Error)**

**EmployeeNameTextBox.Focus()**

**EmployeeNameTextBox.SelectAll()**

**ElseIf TitleComboBox.SelectedIndex = -1 Then**

**'Business rule #2 - Title must be selected**

**'from the list provided in the ComboBox**

**MessageString = "Title was not selected."**

**TitleString = "Title error"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Error)**

**TitleComboBox.Focus()**

**ElseIf DepartmentComboBox.Text.Trim = String.Empty Then**

**'Business rule #3 - Department must be selected**

**'or typed into the Text property of the ComboBox**

**MessageString = "Department name was not selected or entered."**

**TitleString = "Department Name error"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Error)**

**DepartmentComboBox.Focus()**

**ElseIf IsNumeric(SalaryTextBox.Text) = False OrElse Decimal.Parse(SalaryTextBox.Text, Globalization.NumberStyles.Currency) <= 0D Then**

**'Business rule #4 - Salary must be valid number > 0**

**MessageString = "Salary is missing or invalid-must be greater than zero."**

**TitleString = "Salary error"**

**MessageBox.Show(MessageString, TitleString, MessageBoxButtons.OK, MessageBoxIcon.Error)**

**SalaryTextBox.Focus()**

**SalaryTextBox.SelectAll()**

**Else**

**'Data passes all business rules**

**ValidData = True**

**End If**

**End Function**

**End Class**

**End of Notes**