Student Guide

Information in this document is subject to change without notice. Companies, names and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Object Innovations.

Product and company names mentioned herein are the trademarks or registered trademarks of their respective owners.

Authors: Robert J. Oberg and Dana Wyatt

Copyright ©2009 Object Innovations Enterprises, LLC All rights reserved.

Object Innovations
877-558-7246
www.objectinnovations.com

Printed in the United States of America.
Table of Contents (Overview)

Chapter 1   Introduction to Windows Forms
Chapter 2   Visual Studio and the Forms Designer
Chapter 3   Using Controls
Chapter 4   List Controls
Chapter 5   Working with Dialogs
Chapter 6   Menus, Toolbars and Status Bars
Chapter 7   Advanced Windows Forms Topics
Chapter 8   Using Advanced Controls
Chapter 9   Resources
Chapter 10  Applications and Settings
Chapter 11  Data Access and Data Binding
Chapter 12  Newer Features in Windows Forms

Appendix A   Learning Resources
Directory Structure

• The course software installs to the root directory \texttt{C:\OIC\WinCs}.
  
  – Example programs for each chapter are in named subdirectories of chapter directories \texttt{Chap01}, \texttt{Chap02}, and so on.

  – The \texttt{Labs} directory contains one subdirectory for each lab, named after the lab number. Starter code is frequently supplied, and answers are provided in the chapter directories.

  – The \texttt{Demos} directory is provided for hand-on work during lectures.

• Data files install to the directory \texttt{C:\OIC\Data}. 
# Table of Contents (Detailed)

<table>
<thead>
<tr>
<th>Chapter 1</th>
<th>Introduction to Windows Forms</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>What are Windows Forms?</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Windows Forms Class Hierarchy</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Control Class</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Form Class</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Form Class Methods</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Building a Form</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Application Class</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Simple Windows Application</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Using the .NET Framework SDK</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Using Controls</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>Common Control Properties</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Ambient Control Properties</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Example: Placing a Control in a Form</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>Common Control Events</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Trapping Events</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Using the Button Control</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>HelloWorld – Step 3</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Using the Label Control</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>Using the TextBox Control</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>Example – Complete Application</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>MSDN Documentation</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>Lab 1</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td>39</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 2</th>
<th>Visual Studio and the Forms Designer</th>
<th>41</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual Studio</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>Using the Forms Designer</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Example: Creating a Windows Forms Application</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>Examining the Forms Designer Generated Code</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>Designing &quot;Pretty&quot; Forms</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>Designing &quot;Easy-to-Use&quot; Forms</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>Setting the Tab Order</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>Defining Keyboard Shortcuts</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>Defining Default and Cancel Buttons</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>Lab 2</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td>60</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 3</th>
<th>Using Controls</th>
<th>63</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controls</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>Using the TextBox – Again</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>Using the Clipboard</td>
<td>68</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 4 List Controls

Working with Lists ......................................................... 109
Using a ListBox ................................................................. 110
Selected Items ................................................................. 111
Selected Indices ............................................................... 112
Other ListBox Features .................................................... 113
Adding and Removing Items Dynamically ................................. 115
Example: Using a ListBox .................................................. 116
Using the ComboBox ......................................................... 119
ComboBox Example ........................................................... 120
Storing Objects in List Controls ............................................ 122
Lab 4A .................................................................................. 123
Using the DomainUpDown Control ......................................... 124
Example: Using the DomainUpDown ....................................... 125
Using a ListView .................................................................. 127
Adding Columns to a ListView .............................................. 129
Using a ListView .................................................................. 129
Example: Using the DomainUpDown ....................................... 80
Using the TrackBar ............................................................. 81
Example: Using the TrackBar ................................................ 82
Using the ProgressBar ....................................................... 83
Example: Using the ProgressBar ........................................... 84
Working with Dates ........................................................... 86
Using the MonthCalendar .................................................. 87
Using DateTimePicker ....................................................... 90
Example: Using the DateTimePicker ....................................... 91
Other Controls .................................................................. 94
Tracing .............................................................................. 95
Debug and Trace Classes .................................................. 96
Tracing Example .............................................................. 97
Viewing Trace Output ....................................................... 98
Debug Statements ............................................................. 99
Debug Output .................................................................. 100
WriteLine Syntax .............................................................. 101
Lab 3B .............................................................................. 102
Summary ........................................................................ 103
Context Menu Events .................................................................................................. 211
Handling Multiple Events ........................................................................................... 212
Lab 6A ........................................................................................................................ 214
Status Bars .................................................................................................................... 215
StatusStrip Example .................................................................................................... 216
StatusStrip Demo ........................................................................................................ 217
A Quick Status Bar ..................................................................................................... 221
Toolbars ...................................................................................................................... 222
ToolStrip Demo .......................................................................................................... 223
Importing Images ....................................................................................................... 225
Associating an Event Handler ..................................................................................... 226
Image and Text on Buttons ....................................................................................... 227
Lab 6B ......................................................................................................................... 228
Summary ..................................................................................................................... 229

Chapter 7  Advanced Windows Forms Topics .......................................................... 235
Forms and Controls .................................................................................................... 237
Parent/Child Relationships ......................................................................................... 238
Example: Using Parent/Child Relationships ............................................................... 240
Owner/Owned Relationships ....................................................................................... 242
Example: Using Owner/Owned Relationships ............................................................ 243
Top-Most Forms ........................................................................................................ 246
MDI Applications ....................................................................................................... 247
.NET MainMenu Class ................................................................................................. 248
Creating an MDI Parent Window ............................................................................... 249
Creating and Displaying an MDI Child Window ......................................................... 250
Arranging MDI Child Windows ................................................................................ 251
Merging the Parent and Child Menus ...................................................................... 252
Example: An MDI Request Manager .................................................................... 255
Clipboard Object ....................................................................................................... 262
Placing Data on the Clipboard .................................................................................... 263
Retrieving Data from the Clipboard ......................................................................... 264
Visual Inheritance .................................................................................................... 265
Building the Base Form ......................................................................................... 266
Example: Using Visual Inheritance .................................................................... 267
Lab 7 ........................................................................................................................... 274
Summary ..................................................................................................................... 275

Chapter 8  Using Advanced Controls ....................................................................... 281
Panel Control ............................................................................................................. 283
Panel and GroupBox Example .................................................................................. 284
TreeView Control ..................................................................................................... 285
TreeView Properties ................................................................................................. 286
TreeView Methods ................................................................................................... 287
TreeView Events ....................................................................................................... 288
TreeNode Class ......................................................................................................... 289
Chapter 12  Newer Features in Windows Forms ...................................................... 463

Evolution of Microsoft Windows ............................................................................. 465
New Controls in .NET 2.0 ......................................................................................... 466
File Browser Demo ................................................................................................. 467
ToolStrip and SplitContainer ................................................................................. 468
WebBrowser Control ............................................................................................... 473
Lab 12A ..................................................................................................................... 474
Application Settings .................................................................................................. 475
Application Settings Using Visual Studio ................................................................. 476
Application Settings Demo ....................................................................................... 477
Application Configuration File ................................................................................... 482
User Configuration File ............................................................................................. 483
Manual Application Settings ..................................................................................... 484
Default Values of Settings ......................................................................................... 488
New Data Binding ..................................................................................................... 489
BindingSource Architecture ....................................................................................... 490
Binding to a List ......................................................................................................... 491
Setting up the Bindings ............................................................................................. 492
Category Class .......................................................................................................... 493
Code in the Form ....................................................................................................... 494
Chapter 2

Visual Studio and the Forms Designer
Visual Studio and the Forms Designer

Objectives

After completing this unit you will be able to:

- Use Visual Studio to build simple Windows Forms applications.
- Use the Forms Designer to visually design forms.
- Trap events using the Forms Designer.
- Create an attractive visual design for your forms.
- Create an efficient design for your forms, including setting a tab order and implementing keyboard shortcuts.
Visual Studio

- **Visual Studio**, which we used in the first chapter to create and run simple projects, is extremely useful in developing Windows applications.

- **Visual Studio** allows us to design forms using a drag-drop interface.
  - If you are familiar with the IDEs in Visual Basic or Visual C++, the Visual Studio IDE will look very familiar!

- **The drag-drop interface is generally referred to as the Forms Designer.**
  - The Forms Designer will be available any time a Windows Forms class has been added to a project.
  - It can be opened by selecting the View Designer icon from the Solution Explorer window.
Using the Forms Designer

- The Forms Designer allows a programmer to drag and drop controls from a toolbox onto a form.
  - If the toolbox isn't visible, you can select it from the View | Toolbox menu.
Using the Forms Designer (Cont’d)

- You can modify the properties of a control using the Properties window (shown in the lower right).
  - If the Properties Window isn't visible, you can select it from the View | Properties Window menu.
  - The properties can be shown by category or alphabetically by selecting an icon from the Properties Window toolbar.

By category: | Alphabetically:
Using the Forms Designer (Cont’d)

- You can add, modify and view the event handlers for each control using the Properties window.
  - To add an event handler and associated delegate, double-click on the appropriate event from the left-hand side of the scrolling grid. Select Events by the icon.

- You can add the "default" event handler for each control by double-clicking the control in design view.
Example: Creating a Windows Forms Application

- It is easy to create a Windows Forms application using Visual Studio.
  - A copy of the application is saved in Chap02\GoCowboys directory.
  - If you want to follow along, you should do your work in the Demos directory.

1. We begin by creating a new C# Windows Forms Application project named **GoCowboys** in the Demos directory. Leave unchecked “Create directory for solution.”
Example: Creating ... (Cont’d)

2. We then rename the source file for the form to `MainForm.cs`. You will be asked if you want to rename the corresponding code elements. Say yes.

3. To verify that the required code elements have been renamed, build and run the application. You should see a bare form, which can be resized.
Example: Creating … (Cont’d)

4. We will use the toolbox to drag a button control to the form. To make everything look nifty, we will resize the form.

5. We want to make sure the following property values are set for the button and form:

<table>
<thead>
<tr>
<th>Object</th>
<th>Name Property</th>
<th>Text Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button</td>
<td>btnCheer</td>
<td>Cheer!</td>
</tr>
<tr>
<td>Form</td>
<td>MainForm</td>
<td>NFL Football</td>
</tr>
</tbody>
</table>
Example: Creating ... (Cont’d)

6. We need to trap the **Click** event for the **btnCheer** button. To do this, we can double-click on the **btnCheer** button. It will write the **Click** event handler and delegate for us and position us at the handler function in the code window.

   - We will add code to display a message box that shows the message "Go Cowboys!"

```csharp
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace GoCowboys
{
    public partial class MainForm : Form
    {
        public MainForm()
        {
            InitializeComponent();
        }

        private void btnCheer_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Go Cowboys!", "NFC");
        }
    }
}
```

- **Note the **partial** modifier on the class.**

   - This is a new feature of .NET 2.0, which enables wizard-generated code to be maintained in a separate file.
Example: Creating ... (Cont’d)

7. Finally, we can build and run the application.
Examining the Forms Designer Generated Code

- The Forms Designer generated code as we designed the form.
  - The code is in the separate file `MainForm.Designer.cs`.

```csharp
namespace GoCowboys
{
    partial class MainForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise, false.
        /// </param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
        }
    }
}
```
Designing "Pretty" Forms

- In order to make your forms look nice, you will want to:
  - Size your controls so that similar are approximately the same size.
  - Align your controls on some meaningful X and Y axis.
  - Visual Studio 2008 provides alignment lines as you drag controls, making your job easier.

- The Forms Designer allows you to use a special toolbar to perform a variety of layout tasks.
  - You can use the View | Toolbars | Layout menu option to display the toolbar.
Designing "Easy-to-Use" Forms

• In order to make your forms easy to use, you will want to:
  – Lay out the controls in a meaningful way.
  – Provide meaningful labels.

• However, there are some other steps you can take to make the form easy to use:
  – Set the tab order of the controls.
  – Define keyboard shortcuts to provide fast access to controls.
  – Define default and cancel buttons.

• Our example illustrates a form that is both “pretty” and easy to use.
  – See PrettyDialog in the current chapter directory.
Setting the Tab Order

- To set the tab order of the controls on a form, you must use the View | Tab Order menu.
  - The form must have focus for this option to be visible.
  - You must select View | Tab Order when you are done to turn off tab ordering.

- When you click on a tab number, it changes.
  - The first tab you click on becomes 0, the second tab you click on becomes 1, etc.
  - If you accidentally give a tab an incorrect number, keep clicking on the tab… the number cycles through the available number.

- You should always include your labels in the tab order directly before the control they label. The reason will be apparent on the next page!
Defining Keyboard Shortcuts

- You can define keyboard shortcuts for controls on your form.
  - These shortcuts allow the user to press Alt + shortcutKey to move focus to the control that defines the shortcut.

- To assign a shortcut to controls, you must:
  - Place an ampersand ( & ) in front of the shortcut letter in the control's Text property.
  - For example, if the Text property of an OK button is &OK, then the text will appear as OK on the screen to let the user know O is the shortcut key. When the user presses Alt + O, focus moves to the control.
  - When you use the shortcut key on a button, it invokes the click event handler for the button instead of setting focus to it.

- When a control does not have a static Text property (for example, a textbox), you must place a label that defines the key directly in front of the control—based on tab order.
  - For example, if the textbox has tab order 7, its label should have tab order 6.
  - The label's Text property must have an ampersand ( & ) in front of the shortcut letter (example: &Husband will define H as the shortcut; Hus&band will define B as the shortcut).
Defining Keyboard Shortcuts (Cont'd)

- When running the program, to use the keyboard shortcuts the user should press the Alt key.
  - Then the underlines will appear.
  - When the user enters the keyboard shortcut, focus will go to control the following the label (based on tab order) because a label does not have a tab stop.
Defining Default and Cancel Buttons

- It is common among Windows Forms applications to define default and cancel buttons for each form.

  - The default, or accept, button is one that is invoked if the user hits the Enter key in any control on the form that does not have its own `AcceptReturn` property set to True.

  - The cancel button is the one that is invoked if the user hits the Escape key in any control on the form.

- Windows Forms makes this assignment easy:

  - The form has two properties, `AcceptButton` and `CancelButton`, that can be assigned a reference to a button using a drop-down list.
Lab 2

My Calculator

In this lab you will use Visual Studio and the Forms Designer to build a simple calculator that performs addition, subtraction, multiplication and division on floating point numbers.

Detailed instructions are contained in the Lab 2 write-up at the end of the chapter.

Suggested time: 30 minutes
Summary

- Visual Studio makes it easy for programmers to build Windows Forms applications.

- The Forms Designer allows programmers to drag controls from a toolbox and visually place them on a form.

- The Properties Window can be used to specify values for form and control properties.

- The Forms Designer generates code based on the programmer's drag-drop actions and property settings.

- C# and the tools inside Visual Studio make it easy to build Windows Forms applications that look nice and are easy to use.
Lab 2

My Calculator

Introduction

In this lab you will use Visual Studio and the Forms Designer to build a simple calculator that performs addition, subtraction, multiplication and division on floating point numbers. Your form should resemble the following:

You will need to use exception handling to make sure that "garbage" data in the operand 1 and/or operand 2 textbox does not cause your program to "crash".

Suggested Time: 30 minutes

Root Directory: OIC\WinCs

Directories: Labs\Lab2 (do your work here)
OIC\Data\Graphics (contains icon files)
Chap02\MyCalculator (contains lab solution)

Instructions

1. Create a new C# Windows Forms Application named MyCalculator. Name the form class and the associated file Calculator. Save the solution.

2. Design the form similar to that shown above.

3. Copy the icon files for the four arithmetic operations, MISC18 ICO, MISC19 ICO, MISC20 ICO and MISC21 ICO, from OIC\Data\Graphics to the working directory.
4. Set the properties of each control to the following values:

<table>
<thead>
<tr>
<th>Control</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Name</td>
<td>lblOperand1</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>Operand 1:</td>
</tr>
<tr>
<td>TextBox</td>
<td>Name</td>
<td>txtOperand1</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>(blank)</td>
</tr>
<tr>
<td></td>
<td>TextAlign</td>
<td>Right</td>
</tr>
<tr>
<td>Label</td>
<td>Name</td>
<td>lblOperand2</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>Operand 2:</td>
</tr>
<tr>
<td>TextBox</td>
<td>Name</td>
<td>txtOperand2</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>(blank)</td>
</tr>
<tr>
<td></td>
<td>TextAlign</td>
<td>Right</td>
</tr>
<tr>
<td>Button</td>
<td>Name</td>
<td>btnAdd</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>(blank)</td>
</tr>
<tr>
<td></td>
<td>Image</td>
<td>MISC18.ICO</td>
</tr>
<tr>
<td>Button</td>
<td>Name</td>
<td>btnSubtract</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>(blank)</td>
</tr>
<tr>
<td></td>
<td>Image</td>
<td>MISC19.ICO</td>
</tr>
<tr>
<td>Button</td>
<td>Name</td>
<td>btnMultiply</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>(blank)</td>
</tr>
<tr>
<td></td>
<td>Image</td>
<td>MISC20.ICO</td>
</tr>
<tr>
<td>Button</td>
<td>Name</td>
<td>btnDivide</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>(blank)</td>
</tr>
<tr>
<td></td>
<td>Image</td>
<td>MISC21.ICO</td>
</tr>
<tr>
<td>Label</td>
<td>Name</td>
<td>lblAnswer</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>Answer:</td>
</tr>
<tr>
<td>TextBox</td>
<td>Name</td>
<td>txtAnswer</td>
</tr>
<tr>
<td></td>
<td>Text</td>
<td>(blank)</td>
</tr>
<tr>
<td></td>
<td>TextAlign</td>
<td>Right</td>
</tr>
</tbody>
</table>

5. Trap the Click event for each of the four buttons that specify math operations.

6. In each handler, write code to convert the string data in each textbox to a floating point value. Perform the appropriate math operation for the button. Finally, place the result back in the textbox that holds the answer. Compile and run the program.
Chapter 4

List Controls
List Controls

Objectives

After completing this unit you will be able to:

• Describe the various controls provided by Windows Forms for working with lists.

• Use the ComboBox, ListBox, DomainUpDown, and ListView controls to allow the user to interact with lists.

• Work with objects stored in list controls.
Working with Lists

- Windows Forms provides a variety of controls that allow the user to select from and/or view a list

- These controls include:
  - the ListBox control
  - the ComboBox control
  - the DomainUpDown control
  - the ListView control

- Most of these controls are very similar. They each have a collection of items, and these collections typically implement the IList interface.
  - The IList interface defines methods such as Add, Clear, Insert, Remove, and RemoveAt.
  - The IList interface also defines the Item property to provide access to an individual element in the list.
Using a ListBox

- The ListBox control is used to allow a user to enter data by selecting an item from a list.
  - The listbox can be a single-selection or multiple-selection control, and it can display multiple columns of data.

- The ListBox control manages three collections

- The *Items* property provides access to all of the items listed within the list box.
  - It has *Count* and *Item* properties, and methods such as *Add*, *AddRange*, *Clear*, *Insert*, *Remove*, and *RemoveAt*.
  - For example, to iterate through the items, you can write:

    ```csharp
    private void btnShowAll_Click(object sender, System.EventArgs e)
    {
        string show = "";
        for (int i = 0; i < lstData.Items.Count; i++)
            show += lstData.Items[i] + " ";
        MessageBox.Show(show, "All Items");
    }
    ```

- For a simple example of using a ListBox, containing the code shown above, see *ShowList\Single*.
  - The *SelectionMode* property is *One*.
Selected Items

- The `SelectedItems` property provides access the selected items in the list box.
  - It can also be used to access properties such as `Count` and `Item`.
  - For example, to iterate through the items selected, you can write:
    ```csharp
    string dataValue = "";
    for (int i=0; i<lstData.SelectedItems.Count; i++)
    {
        dataValue =
            lstData.SelectedItems[i].ToString();
        MessageBox.Show("Selected: " + dataValue);
    }
    ```

- The sample code is from the `ShowList` program.
  - See `ShowList\Multiple`.
  - The `SelectionMode` property is `MultiExtended`.

![Selected Items](image_url)
Selected Indices

- The `SelectedIndices` property provides access the indices of the selected items in the list box.

  - It can also be used to access properties such as `Count` and `Item`.
  
  - For example, to show the indices of the items selected, you can write:

    ```csharp
    private void btnShowIndices_Click(object sender, System.EventArgs e)
    {
        string show = "";
        for (int i = 0; i < lstData.SelectedIndex.Count; i++)
            show += lstData.SelectedIndex[i] + " ";
        MessageBox.Show(show, "Selected Indices");
    }
    ```
Other ListBox Features

- The ListBox class has other interesting properties and methods, including:
  - The Sorted property indicates whether the list box is sorted.
  - The SelectionMode property indicates the selection mode (single selection or multiple selection) of the list box.
  - The Text (and SelectedItem) property indicates the value of the currently selected item in the list box.
  - The SelectedIndex property indicates the index of the item currently selected in the list box.

- The ShowList\Single program illustrates a single selection list box and use of the SelectedItem and SelectedIndex properties.

- Finally, the ListBox fires several events, including:
  - The SelectedIndexChanged indicates the selected index changed.
  - The SelectedValueChanged indicates the selected text changed.
Adding and Removing Items

- Items can be added to a list box either statically, at design time, or dynamically, at run-time.

- To add the items at design time, you must use the Properties window for the list box. Click the ellipsis button for the Items property to display an editor that can be used to add items to the list box.

- Note also that the SelectionMode has been set to MultiExtended.
  - This allows you to use the Control and Shift keys to control the selection, according to standard Windows conventions.
Adding and Removing Items
Dynamically

- To add and remove items to a list box dynamically at run time, you must use methods of `Items`. Example:

```csharp
lstFriends.Items.Add("Natalie");
```

- Whereas the `Add` method adds a string to the end of the list box, the `Insert` method can specify the location of the string. (The `Insert` method is irrelevant if the `Sorted` property of the list box is true.) For example:

```csharp
lstFriends.Items.Insert(3, "Brandon");
```

- The `AddRange` method can be used to assign an entire array to the list box. For example:

```csharp
string[] people = {"Betty", "Ralph", "Mary"};
lstFriends.Items.AddRange(people);
```

- The `Remove` method can be used to remove the first occurrence of a specified string. For example:

```csharp
string sally = "Sally";
lstFriends.Items.Remove(sally);
```

- The `RemoveAt` method can be used to remove the occurrence at a specified index. For example:

```csharp
lstFriends.Items.RemoveAt(3);
```

- The `Clear` method deletes everything in a list box. For example:

```csharp
lstFriends.Items.Clear();
```
Example: Using a ListBox

- The ListOffFriends example, found in the ListOfFriends\Version1 directory, illustrates some simple list box manipulation.

- Items can be added to the listbox via the Add button.

- The listbox allows both single-selection and multi-selection deletion.
Example: Using a ListBox (Cont'd)

• The code behind the form is shown below:

```csharp
private void chkSorted_CheckedChanged(object sender, System.EventArgs e)
{
    lstFriends.Sorted = ! lstFriends.Sorted;
}
private void chkMulti_CheckedChanged(object sender, System.EventArgs e)
{
    if (chkMulti.Checked)
        lstFriends.SelectionMode =SelectionMode.MultiExtended;
    else
        lstFriends.SelectionMode =SelectionMode.One;
}
private void btnAdd_Click(object sender, System.EventArgs e)
{
    lstFriends.Items.Add(txtNewName.Text);
    txtNewName.Text = "";
}
...
```
private void btnDelete_Click(object sender, System.EventArgs e)
{
    if (!chkMulti.Checked)
    {
        if (lstFriends.SelectedIndex != -1)
        {
            lstFriends.Items.RemoveAt(lstFriends.SelectedIndex);
        }
    }
    else
    {
        for (int i =
            lstFriends.SelectedItems.Count - 1;
            i>=0; i--)
        {
            lstFriends.Items.RemoveAt(lstFriends.SelectedIndices[i]);
        }
    }
}

private void MainFrame_Load(object sender, System.EventArgs e)
{
    string[] people = {"Betty", "Ralph", "Mary"};
    lstFriends.Items.AddRange(people);
}
Using the ComboBox

• A ComboBox is very similar to a listbox, except that users can also type text into the ComboBox control as well as choose from a list.

• By default, a ComboBox displays an edit field with a hidden drop-down list.

• The DropDownStyle property controls the style of combo box to display. Values include:
  – DropDown, (default) displays an edit control with an arrow next to it that can be used to drop down the list. An item can be selected from the list, or typed into the edit control.
  – Simple, displays an edit control with a list just below it that is always visible. An item can be selected from the list, or typed into the edit control.
  – DropDownList, displays a non-editable edit control with an arrow next to it that can be used to drop down the list. An item must be selected from the list.

• The properties and methods of the ComboBox are almost exactly the same as a ListBox.
  – The primary differences are that a ComboBox is not multi-selectable and the SelectedIndex property might be –1, yet Text might still have a non-blank value.
ComboBox Example

- As an illustration, consider a simple file viewer program where the file name is entered via a ComboBox.
  - See FileViewer\Step1.
  - After a file has been successfully opened, the file name is added to the ComboBox.
ComboBox Example (Cont’d)

• Here is the code for the handler for the Open button:

```csharp
try
{
    txtData.Text = "";
    string NL = Environment.NewLine;
    StreamReader rd = File.OpenText(cmbFile.Text);
    string str = rd.ReadLine();
    while (str != null)
    {
        txtData.Text += str + NL ;
        str = rd.ReadLine();
    }
    rd.Close();
    // On successful opening, add to combobox
    if (!cmbFile.Items.Contains(cmbFile.Text))
        cmbFile.Items.Add(cmbFile.Text);
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "File Viewer");
}
```
Storing Objects in List Controls

- What is stored in all the various list controls is an object.
  - What is displayed in the dropdown list is the string obtained by applying the ToString() method to the object.
  - You can obtain the selected item in the list control by casting to the appropriate object type.

```csharp
int index = cmbAccounts.SelectedIndex;
if (index != -1)
{
    Account acc =
        (Account) cmbAccounts.Items[index];
    txtName.Text = acc.Name;
}
```

- See Accounts for an example.
Lab 4A

Storing Objects in a ComboBox

In this lab, you will implement a class of Account objects and a Windows Forms application that will store the Account objects in a ComboBox. Selecting an item from the ComboBox will cause the fields of the selected account to be displayed in textboxes. Buttons are provided to add, delete, and change an account.

Detailed instructions are contained in the Lab 4A write-up at the end of the chapter.

Suggested time: 45 minutes
Using the DomainUpDown Control

• The DomainUpDown control is a scrollable control that displays a string value from a list.
  
  – It resembles a Windows spinner control (often used for copy count in print dialogs), but contains textual entries instead of sequential numbers.
  
  – The user can also enter text in the control if its **ReadOnly** property is False, but any entry must match an item in the collection.

• Some of the more interesting properties of the DomainUpDown control include:
  
  – **Items**, which represents the collection of items in the control's list.
  
  – **SelectedIndex**, which indicates the index of the item in the list that is selected. If nothing is selected, the value is –1.
  
  – **Text**, which represents the text that is displayed in the control next to the up/down buttons.

• Events generated by the DomainUpDown control include:
  
  – **SelectedItemChanged**, which occurs when the **SelectedItem** property is changed.
  
  – **TextChanged**, which occurs when the Text property is changed.
Example: Using the DomainUpDown

- This version of the ListOfFriends example, found in the ListOfFriends\Version2 directory illustrates the use of a domain up/down control.

- The **DomainUpDown** control contains a list of acceptable association values. They were added at design time to the **Items** collection.
Example: Using the DomainUpDown (Cont'd)

- The code that references the control is shown below:

```csharp
private void btnAdd_Click(object sender, EventArgs e)
{
    Add(txtNewName.Text, dudAssociation.Text);
}

private void Add(string name, string association)
{
    string dataValue = string.Format("{0} ({1})", name, association);
    lstFriends.Items.Add(dataValue);
    txtNewName.Text = "";
}

...

private void MainForm_Load(object sender, EventArgs e)
{
    Add("Leslie", "School");
    Add("Richard", "Work");
    dudAssociation.SelectedIndex = 0;
}
```
Using a ListView

- The ListView control is a more flexible version of the ListBox control and can display its list four different ways:
  - Using large icons
  - Using small icons
  - Displaying a list
  - Displaying list details

- The right-hand pane of Windows Explorer is essentially a ListView control.
  - When displaying list details, the ListView can display multiple columns for each list item.

- The ListView has several interesting properties not inherited from the Control class, including:
  - The View property indicates the current viewing style of items in the control.
  - The Items property, which represents the items in the control.
  - The MultiSelect property, indicates whether the control supports multiple selection
  - The Sorting property indicates how items in the control are sorted.
Using a ListView (Cont'd)

- The `LargeImageList` property, which indicates the `ImageList` that contains the collection of images used when the view indicates large icons are displayed.

- The `SmallImageList` property, which indicates the `ImageList` that contains the collection of images used when the view indicates small icons are displayed.

- The `Columns` property provides access to the collection of columns.

- The `GridLines` property indicates whether grid lines are drawn between items and subitems.

- The `FocusedItem` property indicates the item with focus.

- The `FullRowSelect` property indicates whether clicking an item selects the item or the whole row.

**Methods of the ListView control include:**

- `GetItemAt`, returns the item at a specific x,y location.

- `ArrangeIcons`, arranges icons according to a specified behavior.

**Events generated by the ListView control include:**

- `ColumnClick`, which occurs when the user clicks a column

- `ItemActivate`, which occurs when an item is activated.

- `SelectedIndexChanged`, which occurs when the selection changes.
Adding Columns to a ListView

- A ListView can be designed to resemble a grid.
- To add columns to a ListView, you must:
  - Set its View property to Details.
  - Modify the Columns property to add columns for an item as needed. The Columns property may be specified at design time or by writing code.

```csharp
lvFriends.Columns.Add("Name", 200, HorizontalAlignment.Left);
lvFriends.Columns.Add("Association", 70, HorizontalAlignment.Left);
```
Adding Items to a ListView

• To add an item, typically you would begin by creating a ListViewItem. (It would be more intuitively obvious if Microsoft had called it a ListViewRow!)
  
  − Its Text property contains the text that appears in the first column of a row. If the ListView's View property is set to anything other than Details, only the value of the Text property displays.
  
  − If the ListView's View property is set Details, all subitem values associated with ListViewItem appear as subsequent columns.

• You must call the Add method of the Items collection.

```csharp
ListViewItem it = new ListViewItem();
it.Text = "Dana";
it.SubItems.Add("Instructor");
it.SubItems.Add("24 years");
it.SubItems.Add("Texas");
lvEmployees.Items.Add(it);
```

• If the View property of the ListView were SmallIcons or LargeIcons, you would typically specify:
  
  − The item's text.
  
  − An index for an image from an associated ImageList.

```csharp
// Adds a new item with ImageIndex 1
lswPeople.Items.Add("Dana", 1);
```
Adding Items to a ListView (Cont'd)

// Adds a new item with ImageIndex 1 and
// with subitems
ListViewItem it = new ListViewItem();
it.Text = "Dana";
it.ImageIndex = 1;
it.SubItems.Add("Instructor");
it.SubItems.Add("24 years");
it.SubItems.Add("Texas");
.lvEmployees.Items.Add(it);
Example: Using a ListView

- This version of the ListOfFriends example, found in the `ListOfFriends\Version3` directory, illustrates the use of a ListView.
Example: Using a ListView (Cont'd)

- The code for the example is shown below:

```csharp
private void btnAdd_Click(object sender, EventArgs e)
{
    Add(dudAssociation.Text, txtNewName.Text);
}

private void Add(string name, string association)
{
    ListViewItem it = new ListViewItem();
    it.Text = name;
    it.SubItems.Add(association);
    lvFriends.Items.Add(it);
    txtNewName.Text = "";
}

private void btnDelete_Click(object sender, EventArgs e)
{
    for (int i = lvFriends.SelectedItems.Count - 1; i >= 0; i--)
    {
        lvFriends.Items.RemoveAt(lvFriends.SelectedIndices[i]);
    }
}
```
Lab 4B

Computer Price Estimator

In this lab, you will enhance the Computer Price Estimator from the previous chapter to allow the buyer to choose an operating system and a selection of application software to go with the hardware in the new computer system.

Detailed instructions are contained in the Lab 4B write-up at the end of the chapter.

Suggested time: 30 minutes
Summary

- **ComboBox, ListBox, DomainUpDown, and ListView controls allow the user to interact with lists.**
  - Each has an **Items** collection that manages the items in their lists.

- **You can store arbitrary objects in the **Items** collection.**
  - You should provide overrides of **ToString()** and **Equals()** in a class whose objects are going to be stored in the **Items** collection of a list control.
Lab 4A

Storing Objects in a ComboBox

Introduction

In this lab, you will implement a class of Account objects and a Windows Forms application that will store the Account objects in a ComboBox. Selecting an item from the ComboBox will cause the fields of the selected account to be displayed in textboxes. Buttons are provided to add, delete, and change an account.

Suggested Time: 45 minutes
Root Directory: OIC\WinCs
Directories: Labs\Lab4A\Accounts (do your work here) Chap04\Accounts (contains lab solution)

Instructions

1. Create a new C# Windows Forms Application named Accounts. Name the form class and the associated file MainForm.

2. Design the form similar to that shown above. Make the DropDownStyle of the ComboBox to be DropDownList.

3. Add a new file Account.cs to your project that defines a class Account with the following features:

   a. A public field Name of string type.
b. A public field **Balance** of decimal type.

c. A constructor that will initialize a new account given the name and balance.

4. Create a private method **NewAccount()** that takes as arguments a name and balance and returns a Boolean. In this method construct a new **Account** object and add it to the **Items** collection of the ComboBox. For now, just return true.

5. Add a handler for the form’s **Load** event. In this handler call **NewAccount()** several times to initialize a few accounts in the ComboBox.

6. Build and run. You will see that several items have been added to the dropdown list, but what you see displayed is rather disappointing. How can you get the name associated with each account to be displayed in the dropdown list?

7. In the **Account** class add an override of **ToString()**. Return the name associated with the account.

8. Build and run. You will now see the names in the dropdown list. But no name is displayed initially when the application starts.

9. Add code to the **Load** event handler to set the Text property of the ComboBox to the first item in the collection (subscript 0). Build and run.

10. Add a handler for the **SelectedIndexChanged** event of the ComboBox. In this handler find the selected index from the **SelectedIndex** property. Extract the selected item from the **Items** collection and cast to an **Account** object. Use the **Name** and **Balance** properties to assign values to the corresponding textboxes.

11. Build and run. You should now see proper account information displayed as items are selected in the ComboBox. As you add more features to your program, continue to build and run.

12. Add a handler for the Add button. In this handler construct a new **Account** object from the account information in the textboxes and add to the ComboBox. Set the Text property of the ComboBox to show the name of the new account.

13. We want to ensure that new accounts do not have the same name as existing accounts. To facilitate being able to make this check, add an override of the **Equals()** method in the **Account** class that specifies two accounts as equal if they have the same name.

14. Now modify the **NewAccount()** method to check whether the **Items** collection already contains an account with the name to be added. If it does, return false. If it does not, go on to add the new account.

15. Modify the Add button handler to check the return of **NewAccount()** to see whether the account is already in the collection. If it is, display a suitable message.
16. Add a handler for the Delete button. Provide code to remove the account having the specified name, displaying a message if such an account is not on the list.

17. Add a handler for the Change button. Provide code to change the balance of the account with the specified name to the value entered in the textbox. You can use the `IndexOf()` method of the `Items` collection to locate the account object to be changed.

18. If you have time, make your program more robust by checking for various special conditions (for example, empty list after doing a deletion) and for erroneous data input. In particular, it would be a good idea to do your processing in the button event handlers inside try blocks.
Lab 4B

Computer Price Estimator

Introduction

In this lab, you will enhance the Computer Price Estimator from the previous chapter to allow the buyer to choose an operating system and a selection of application software to go with the hardware in the new computer system.

A ComboBox is provided for choosing an operating system, and a multiple-selection listbox for choosing application software, as shown in the enhanced form:

![Computer Price Estimator](image)

Price information for the various software components is listed below:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Miscellaneous Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Vista</td>
<td>Microsoft Office</td>
</tr>
<tr>
<td>XP Home</td>
<td>Flight Simulator</td>
</tr>
<tr>
<td>XP Professional</td>
<td>&lt;add your own list or 4-6 items&gt;</td>
</tr>
</tbody>
</table>
Your program will use the values for each of the controls to determine the price of the computer and display it just below the Calculate button.

**Suggested Time:** 30 minutes

**Root Directory:** OIC\WinCs

**Directories:**
- Labs\Lab4B\ComputerPriceEstimator (do your work here)
- Chap03\ComputerPriceEstimator\Step1 (starter code backup)
- Chap04\ComputerPriceEstimator\Step2 (solution)

**Instructions**

1. If you would like to continue with your own solution from Chapter 3, copy your code to the work area. Otherwise, use the code provided. Build and run the starter program.

2. Enhance the form as shown above to provide a ComboBox for choosing an operating system and a ListBox for choosing application software.

3. Define arrays or hash tables in your form class to hold the names and prices of each software item.

4. Write code in the form's Load event handler to initialize the arrays or hash tables and to load the OS ComboBox and the software ListBox.

5. When the user selects the Calculate button, write code to calculate the cost of the computer. Remember that lstSoftware is a multiselect list box, so the user may have selected several software items.

6. Compile and run the program.
Chapter 10

Applications and Settings
Applications and Settings

Objectives

After completing this unit you will be able to:

• Use the Application object to obtain information about the application and its environment.

• Build applications that filter messages from the message loop.

• Build a configuration file to store application-specific settings.

• Use .NET classes to read configuration files and use their settings in applications.

• Access the registry from a .NET application.
The Application Class

• The Application class, found in the System.Windows.Forms namespace, represents the class that manages a Windows Forms application.

• It has several interesting properties, including:
  − The StartupPath property, which contains the path of the .exe that started the application.
  − The ExecutablePath property, which contains the path and filename of the .exe that started the application.
  − The ProductName property, which contains the name of the application.

• It has several interesting methods, including:
  − The Run and Exit methods, which start and stop applications.
  − The ExitThread method, to close all windows running on the current thread.
  − The DoEvents method, which processes all Windows messages in the queue.

• All members of the class are static methods.
Starting and Stopping Applications

• As you have seen thus far, Windows Forms applications have a Main function that identifies the window that will appear when the application is launched.
  − The Run method starts an application message loop on the current thread and, optionally, makes a form visible.

```csharp
static void Main()
{
    ...
    Application.Run(new MainForm());
}
```

• Windows Forms applications can call the Exit method to stop a message loop, effectively terminating the application.

```csharp
private void mnuExit_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```
Life Cycle Demonstration

• A simple demo program illustrates closing an application in various ways, by buttons on the form and by using the standard “X” button.
  
  − See LifeCycle in the chapter directory.
  
  − Message boxes and simple logging to a file are used.

• Here is one scenario.
  
  − Program is started. A message box is displayed as the main form is loaded.

  ![Demo Message Box]

  − The main form is then displayed:

  ![Life Cycle Demo Form]

  − The Exit button is clicked.
Application Events

• The Application class fires several events:

  – The ApplicationExit event fires after all forms have closed and the application is about to terminate.

  – The Idle event fires when the application is entering the idle state. This occurs after the application has processed all messages in the input queue.

  – The ThreadExit event fires when a thread is about to terminate. It fires before the ApplicationExit event.

  – The ThreadException event fires when an unhandled exception occurs. It can be used to allow the application to continue executing.

• We can write an event handler for ApplicationExit:

  private static void OnExit(object sender, EventArgs e)
  {
    Log.WriteLine("OnExit called");
    MessageBox.Show("Exiting application", "Demo");
  }

• We can add the event handler in MainForm_Load:

  private void MainForm_Load(object sender, EventArgs e)
  {
    Log.Clear(); Log.WriteLine("Form loading");
    MessageBox.Show("Form loading", "Demo");
    Application.ApplicationExit += new EventHandler(OnExit);
  }
Logging to a File

• In running our little example program, we may be unsure whether the OnExit event handler was actually called, because the message box was not displayed.
  
  − There are some subtleties in Windows that may prevent a window from being displayed too late in the application shutdown process.

• Besides displaying message boxes, our LifeCycle demo program also logs to a file.

  public class Log
  {
    public static void WriteLine(string str)
    {
      StreamWriter writer =
        new StreamWriter(@"c:\OIC\log.txt", true);
      writer.WriteLine(str);
      writer.Close();
    }
  }

• The OnExit event handler contains this code:

    Log.WriteLine("OnExit called");

• The log file log.txt for the simple run shows that the event handler was indeed called.
Closing a Window

- As mentioned earlier, you should normally shut down an application more gracefully by closing its main window.
  - This is illustrated by the Close button in our example.

```csharp
private void btnClose_Click(object sender, EventArgs e)
{
    Log.WriteLine("Close clicked");
    Close();
}
```

- There are two key events associating with closing a window.
  - `FormClosing` is fired first, and the handler for the event can prevent the closing by setting a Cancel flag.
  - `FormClosed` is fired when the window is actually closed.

- You may query the user in the handler of the Closing event.

![Demo dialog](image)
Closing a Window (Cont’d)

- Here are the handlers for *FormClosing* and *FormClosed*.

```csharp
private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
{
    Log.WriteLine("Form closing");
    DialogResult status = MessageBox.Show(
        "Do you want to close?", "Demo",
        MessageBoxButtons.YesNo);
    if (status == DialogResult.No)
        e.Cancel = true;
}

private void MainForm_FormClosed(object sender, FormClosedEventArgs e)
{
    Log.WriteLine("Form closed");
}
```

- Here is the complete log file for running the application, clicking the Close button, and saying Yes to the query to close the window.

```
Form loading
Close clicked
Form closing
Form closed
OnExit called
```
Processing Windows Messages

• .NET programs that perform computationally intensive processing on their main message loop are problematic.
  
  – We have all used an application that we thought had “hung.” We click everywhere and are just about to kill it, when all of a sudden it "springs to life" and processes all of our intermediate clicks.

• We can solve this problem in one of two ways:
  
  – Use a background thread to perform computationally intensive processing—which frees the UI thread to process Windows messages.
  
  – Use DoEvents to periodically allow Windows to process queued messages when performing computationally intensive processing.

```csharp
private void mnuDoSomethingLong_Click(object sender, EventArgs e)
{
    for (i=Int32.MinValue;i<=Int32.MaxValue;i++)
    {
        // do something computationally intensive
        if ((Math.Abs(i) % 10000) == 0)
        {
            Application.DoEvents();
        }
    }
}
```
Filtering Messages

• .NET allows Windows Forms programmers to add a message filter to the application message pump to monitor Windows messages.
  – You should be quite familiar with Windows SDK programming before attempting this.

• You begin by defining a class that implements the IMessageFilter interface.
  – This class can view the messages before they are processed, potentially stopping an event from being processed.
  – The IMessageFilter interface defines the method PreFilterMessage. It returns true to block the message from being processed.

```csharp
// From Windows SDK file winuser.h
// #define WM_RBUTTONDOWN                  0x0204
class FilterMouseMessages : IMessageFilter
{
    public bool PreFilterMessage(ref Message m)
    {
        // Filter right mouse button clicks
        if (m.Msg == 0x204)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```
Filtering Messages (Cont'd)

- You must then call the `AddMessageFilter` method and pass it a reference to the class that implements `IMessageFilter`.
  
  - You can remove the filter by calling `RemoveMessageFilter`.

```csharp
private IMessageFilter msgFilter = null;

private void btnFilter_Click(object sender, EventArgs e)
{
    // Toggle filter
    if (msgFilter == null)
    {
        msgFilter = new FilterMouseMessages();
        btnFilter.Text = "Turn Filter Off";
        Application.AddMessageFilter(msgFilter);
    }
    else
    {
        Application.RemoveMessageFilter(msgFilter);
        msgFilter = null;
        btnFilter.Text = "Turn Filter On";
    }
}
```
Example: Using Application Class

- The example program ApplicationDemo demonstrates the use of the application class.

- The App Properties button displays some of the Application properties in a separate window.

- The Message Filter button toggles filtering out right mouse button clicks on the form.
  - Before the button is pressed, the form displays a message for left and right mouse clicks. After the button is pressed, it displays a message only when the left button is clicked.

- The Close button calls Close().
Configuration Files

- Configuration files are XML files that provide configuration parameters to applications.
  - They can be changed without having to recompile the application.

- Several configuration files exist:
  - Each application can have a .config file named applicationName.config which is located in the application's directory. For example, if the application were called Notepad.exe, the config file would be Notepad.exe.config.
  - The machine has a .config file named machine.config which is located in the directory Windows\Microsoft.NET\Framework\vx.y.zzzz\Config (where x.y.zzzz is the version number of .NET)

- Microsoft suggests all application-specific configuration settings be stored as key/value pairs in the <appSettings> section of the application's config file.
• For example, a config file that specifies a default user name and connection string for database access might resemble:

```xml
<configuration>
  <appSettings>
    <add key="Default User" value="BWW" />
    <add key="Connection String" value="Data Source=(local);Initial Catalog=pubs" />
  </appSettings>
</configuration>
```

• **NOTE:** Important new features pertaining to configuration files were introduced in .NET 2.0.
  
  – See Chapter 12.
Reading Configuration Files

- The System.Configuration namespace contains classes which can be used to read the .config file, including:
  - The ConfigurationManager\(^1\) class provides access to the AppSettings or user-defined sections of a .config file.

- The class provides access to the key/value pairs via a NameValueCollection object. This type includes:
  - The Count property, which identifies the number of key/value pairs.
  - The AllKeys property, which returns an array of strings representing each key.
  - The Get method, which accepts a key and returns the value associated with that key.

---

\(^1\) The ConfigurationManager class supersedes the ConfigurationSettings class, which is now obsolete.
Example: Using Config Files

- In the example program ConfigFiles, we will read initialization parameters from a config file and display them in a message box.

  - The application's name is ConfigFiles.exe, therefore the .config file is named ConfigFiles.exe.config and must reside in the same directory as the application.

- The code that read the config file is shown below:

```csharp
private void btnRetrieve_Click(object sender, EventArgs e)
{
    NameValueCollection parms;
    parms = ConfigurationManager.AppSettings;
    string msg = "";
    foreach (string key in parms.AllKeys)
    {
        msg += string.Format(
            "Key: {0} Value: {1}\n", key, parms.Get(key));
    }
    MessageBox.Show(msg, "Info");
}
```
Example: Using Config Files (Cont'd)

- In order for the code shown above to work, we must have two using statements:

```csharp
using System.Collections.Specialized;
using System.Configuration;
```

- Your project also needs a reference to `System.Configuration`.

- To run the example, you should copy the config file down to the `bin\Debug (or bin\Release)` directory.
Configuration File and Visual Studio

- To conveniently work with a configuration file in Visual Studio, name it `App.config` and add it to your project.

- When you build the project, Visual Studio will copy the configuration file to `bin\Debug` (or `bin\Release`) and rename it based on the name of the assembly.

- See the example project `ConfigFilesVs` for this chapter.
  - The configuration file is renamed to `ConfigFiles.exe.config` when the project is built.
Accessing the Registry

• .NET allows you to access the Windows system registry via classes in the Microsoft.Win32 namespace.
  – These are platform-specific classes and are not available under the System namespace.
  – Their use limits the portability of the application.

• However, if you are committed to the Win32 platform, you may need to access settings stored in the Windows registry.

• The Registry class exposes six read-only static properties that return a RegistryKey reference to a registry hive.
  – The properties are: ClassesRoot, CurrentConfig, CurrentUser, DynData, LocalMachine, PerformanceData and Users.

    RegistryKey reg = Registry.ClassesRoot;

• The RegistryKey class has three instance properties:
  – The Name property is the name of the key.
  – The ValueCount property is the count of the values for the key.
  – The SubKeyCount, if greater than 0, is the number of subkeys for this key.
Accessing the Registry (Cont'd)

• The RegistryKey class also contains several methods, including:
  – OpenSubKey, which returns a reference to a subkey of the instance key.
  – GetValueNames, which returns a list of names for the values associated with the key.
  – GetValue, which returns a value for a key.
  – Close, which closes the key.

• There are many other methods to manipulate the registry that you should research using MSDN.
Example: Manipulating the Registry

- The example *ManipulatingTheRegistry* illustrates the use of the *Registry* and *RegistryKey* classes.

  - This example checks the registry to determine if Microsoft Word is installed. It also displays the CLSID for Word if it is installed.

    ```csharp
    private void btnWord_Click(object sender, EventArgs e)
    {
        // Get a reference to HKEY_CLASSES_ROOT
        RegistryKey hive = Registry.ClassesRoot;

        // Lookup the ProgID Word.Application
        RegistryKey wordProgID = hive.OpenSubKey("Word.Application");
    }
    ```
Example: Manipulating the Registry
(Cont'd)

if (wordProgID != null)
{
    string msg = "Word is installed";
    ShowWordInformation(msg, wordProgID);
    wordProgID.Close();
}
else
{
    MessageBox.Show("Word is not installed");
}

private void ShowWordInformation(string msg, 
RegistryKey progID)
{
    // Get a reference to HKEY_CLASSES_ROOT
    RegistryKey hive = Registry.ClassesRoot;

    // Lookup the value of CLSID
    RegistryKey clsid = progID.OpenSubKey("CLSID");
    string strCLSID = (string) clsid.GetValue(""); 
    clsid.Close();

    MessageBox.Show(msg + "\nCLSID: " + strCLSID);
}
Lab 10

Checking Your System

In this lab, you will use a config file to specify a set of application ProgIDs that should be loaded into a listbox. The program will allow the user to select a ProgID from the listbox and check to see if the corresponding application is loaded on the system.

Detailed instructions are contained in the Lab 10 write-up at the end of the chapter.

Suggested time: 45 minutes
Summary

- The Application class represents the class that manages a Windows Forms application.
  - It contains methods to start and stop a Windows Forms application.

- .NET applications use configuration files are encode using XML.
  - The ConfigurationManager class provides access to the AppSettings or user-defined sections of a .config file.

- .NET allows you to access the Windows system registry via classes in the Microsoft.Win32 namespace.
  - These are platform-specific classes and are not available under the System namespace.
  - Their use limits the portability of the application.

- The Registry class provides access to the registry hives.

- The RegistryKey class contains properties and methods that can be used to navigate the registry.
Lab 10

Checking Your System

Introduction

In this lab, you will use a config file to specify a set of application ProgIDs that should be loaded into a listbox. The program will allow the user to select a ProgID from the listbox and check to see if the corresponding application is loaded on the system.

Suggested Time: 45 minutes

Root Directory: OIC\WinCs

Directories: Labs\Lab10\CheckingYourSystem (do your work here)

Chap10\CheckingYourSystem (contains lab solution)

Instructions

1. Create a new Windows Forms Application named CheckingYourSystem. Name the form class and the associated file MainForm.

2. Build a config file that resembles the following. Name the file App.config, save it in the source directory, and add it to your Visual Studio project.

```
<configuration>
    <appSettings>
        <add key="Count" value="3" />
        <add key="App1" value="Word.Application" />
        <add key="App2" value="Excel.Application" />
    </appSettings>
</configuration>
```
3. Design your main form to resemble that shown above.

4. In your form's **Load** event handler, write code to read the config file and load the ProgIDs into the listbox.

5. Write code in the **Click** event of your Check button to use the ProgID of the selected listbox item and test to see if it is listed in the registry as an installed application. If it is, it will be listed under the HKEY_CLASSES_ROOT hive.

6. Compile and run your program.
This page has been deliberately left blank.