Lab 1: Introducing C# and the .NET Framework

Exercise 1: Building a Simple Console Application

Task 1: Create a new Console Application project

1. Log on to the 10266A-GEN-DEV machine as **Student** with the password **Pa$$w0rd**.
2. Open Microsoft Visual Studio 2010:
   - Click **Start**, point to **All Programs**, click **Microsoft Visual Studio 2010**, and then click **Microsoft Visual Studio 2010**.
3. Create a new console application project called ConsoleApplication in the E:\Labfiles\Lab 1\Ex1\Starter folder:
   a. In Visual Studio, on the **File** menu, point to **New**, and then click **Project**.
   b. In the **New Project** dialog box, in the Installed Templates pane, expand **Visual C#**, and then click **Windows**.
   c. In the Templates pane, click **Console Application**.
   d. Specify the following values for each of the properties in the dialog box, and then click **OK**:
      - Name: **ConsoleApplication**
      - Location: **E:\Labfiles\Lab 1\Ex1\Starter**
      - Solution name: **ConsoleApplication**
      - Create directory for solution: Select the check box.

Task 2: Add code to read user input and write output to the console

1. In the **Main** method, add the statements shown in bold in the following code example, which read a line of text from the keyboard and store it in a string variable called line.

```
static void Main(string[] args)
{
    // Buffer to hold a line as it is read in
    string line;

    // Read a line of text from the keyboard
    line = Console.ReadLine();
}
```

This code uses the **Console.ReadLine** method to read the input, and includes comments with each line of code that indicates its purpose.

2. Add the statement and comment shown in bold in the following code example, which echo the text back to the console by using the **Console.WriteLine** method.

```
static void Main(string[] args)
{
    // Buffer to hold a line as it is read in
    string line;

    // Read a line of text from the keyboard
    line = Console.ReadLine();

    // Write the results out to the console window
    Console.WriteLine(line);
}
```

3. Build the application:
   - On the **Build** menu, click **Build Solution**. Correct any errors.
4. Run the application and verify that it works as expected. You should be able to enter a line of text and see that line echoed to the console:
   a. On the **Debug** menu, click **Start Without Debugging**.
   b. In the console window, type some text, and then press ENTER.
   c. Verify that the text that you typed is echoed to the console.
   d. Press ENTER to return to Visual Studio.

Task 3: Modify the program to read and echo text until end-of-file is detected

1. In the **Main** method, modify the statement and comment shown in bold in the following code example, which reads a line of text from the keyboard.

```
static void Main(string[] args)
{
    // Buffer to hold a line as it is read in
    string line;

    // Loop until no more input (Ctrl-Z in a console, or end-of-file)
    while ((line = Console.ReadLine()) != null)
    {
    }

    // Write the results out to the console window
    Console.WriteLine(line);
}
```

   This code incorporates the statement into a **while** loop that repeatedly reads text from the keyboard until the **Console.ReadLine** method returns a **null** value (this happens when the **Console.ReadLine** method detects the end of a file, or the user types CTRL+Z).

2. Move the **Console.WriteLine** statement into the body of the **while** loop as shown in bold in the following code example. This statement echoes each line of text that the user has entered.

```
static void Main(string[] args)
{
    // Buffer to hold a line as it is read in
    string line;

    // Loop until no more input (Ctrl-Z in a console, or end-of-file)
    while ((line = Console.ReadLine()) != null)
    {
        // Write the results out to the console window
        Console.WriteLine(line);
    }
}
```

3. Build the application:

   - On the **Build** menu, click **Build Solution**. Correct any errors.

4. Run the application and verify that it works as expected. You should be able to repeatedly enter lines of text and see those lines echoed to the console. The application should only stop when you press CTRL+Z:

    a. On the **Debug** menu, click **Start Without Debugging**.

    b. In the console window, type some text, and then press ENTER.

    c. Verify that the text that you typed is echoed to the console.

    d. Type some more text, and then press ENTER again.

    e. Verify that this line is also echoed to the console.

    f. Press CTRL+Z, and then verify that the application finishes.

    g. Press ENTER to return to Visual Studio.

Task 4: Add code to format the data and display it

1. In the body of the **while** loop, add the statement and comment shown in bold before the **Console.WriteLine** statement in the following code example.

```
static void Main(string[] args)
{
    // Buffer to hold a line as it is read in
    string line;

    // Loop until no more input (Ctrl-Z in a console, or end-of-file)
    while ((line = Console.ReadLine()) != null)
    {
        // Format the data
        line = line.Replace(",", " y:");

        // Write the results out to the console window
        Console.WriteLine(line);
    }
}
```

    This code replaces each occurrence of the comma character, "," in the input read from the keyboard and replaces it with the text " y:". It uses the **Replace** method of the line string variable. The code then assigns the result back to the line variable.

2. Add the statement shown in bold in the following code example to the code in the body of the **while** loop.

```
static void Main(string[] args)
{
    // Buffer to hold a line as it is read in
    string line;

    // Loop until no more input (Ctrl-Z in a console, or end-of-file)
    while ((line = Console.ReadLine()) != null)
    {
        // Format the data
        line = line.Replace(",", " y:");
        line = "x:" + line;

        // Write the results out to the console window
        Console.WriteLine(line);
    }
}
```

This code adds the prefix "x:" to the line variable by using the string concatenation operator, **+**, before the **Console.WriteLine** statement. The code then assigns the result back to the line variable.

3. Build the application:
   - On the **Build** menu, click **Build Solution**. Correct any errors.
4. Run the application and verify that it works as expected.

   The application expects input that looks like the following code example.

```
23.54367,25.6789
```

   Your code should format the output to look like the following code example.

```
x:23.54367 y:25.6789
```

   a. On the **Debug** menu, click **Start Without Debugging**.
   b. In the console window, type **23.54367,25.6789** and then press ENTER.
   c. Verify that the text **x:23.54367, y:25.6789** is displayed on the console.
   d. Type some more text that consists of pairs of numbers that are separated by a comma, and then press ENTER again.
   e. Verify that this data is correctly formatted and displayed on the console.
   f. Press CTRL+Z.
   g. Press ENTER to return to Visual Studio.

Task 5: Test the application by using a data file

1. Perform the following steps to add the DataFile.txt file that contains the sample data to the project. This file is located in the E:\Labfiles\Lab 1\Ex1\Starter folder. These steps specify that the file should be copied to the folder that holds the compiled application when the project is built:

    a. In Solution Explorer, right-click the **ConsoleApplication** project, point to **Add**, and then click **Existing Item**.

    b. In the **Add Existing Item – ConsoleApplication** dialog box, move to the **E:\Labfiles\Lab 1\Ex1\Starter** folder, select **All Files (*.*)** in the drop-down list box adjacent to the **File name** text box, click **DataFile.txt**, and then click **Add**.

    c. In Solution Explorer, select **DataFile.txt**. In the Properties window, change the **Build Action** property to **None**, and then change the **Copy to Output** property to **Copy Always**.

2. Rebuild the application:

    - On the **Build** menu, click **Rebuild Solution**.

3. Open a Visual Studio Command Prompt window, and then move to the **E:\Labfiles\Lab 1\Ex1\Starter\ConsoleApplication\bin\Debug** folder:

    a. Click **Start**, point to **All Programs**, click **Microsoft Visual Studio 2010**, click **Visual Studio Tools**, and then click **Visual Studio Command Prompt (2010)**.

    b. In the Visual Studio Command Prompt (2010) window, move to the E:\Labfiles\Lab 1\Ex1\Starter\ConsoleApplication\ConsoleApplication\bin\Debug folder.

4. Run the ConsoleApplication application and redirect input to come from DataFile.txt.

    Verify that the output that is generated looks like the following code example.

```
x:23.8976 y:12.3218
x:25.7639 y:11.9463
x:24.8293 y:12.2134
```

    In the Command Prompt window, type the command in the following code example.

```
ConsoleApplication < DataFile.txt
```

5. Close the Command Prompt window, and then return to Visual Studio.
6. Modify the project properties to redirect input from the DataFile.txt file when the project is run by using Visual Studio:
    a. In Solution Explorer, right-click the **ConsoleApplication** project, and then click **Properties**.
    b. On the **Debug** tab, in the **Command line arguments** text box, type **< DataFile.txt**
    c. On the **File** menu, click **Save All**.
    d. Close the ConsoleApplication properties window.
7. Run the application in Debug mode from Visual Studio:
    - Click **Debug**, and then click **Start Debugging**.

    The application will run, but the console window will close immediately after the output is generated. This is because Visual Studio only prompts the user to close the console window when a program is run without debugging. When a program is run in Debug mode, Visual Studio automatically closes the console window as soon as the program finishes.
8. Set a breakpoint on the closing brace at the end of the **Main** method:
    - In the Program.cs file, move the cursor to the closing brace (}) that corresponds to the end of the **Main** method, right-click, point to **Breakpoint**, and then click **Insert Breakpoint**.
9. Run the application again in Debug mode. Verify that the output that is generated is the same as the output that is generated when the program runs from the command line:
    a. Click **Debug**, and then click **Start Debugging**. The application will run, and then the program will stop when control reaches the end of the **Main** method and Visual Studio has the focus.
    b. In the Windows taskbar, click the icon for **ConsoleApplication.exe**.
    c. Verify that the output that is displayed in the ConsoleApplication.exe window is the same as before.
    d. Return to Visual Studio, and then on the **Debug** menu, click **Continue**. The application will finish.

Exercise 2: Building a WPF Application

Task 1: Create a new WPF Application project

- Create a new project called WpfApplication in the E:\Labfiles\Lab 1\Ex2 \Starter folder by using the Windows Presentation Foundation (WPF) Application template:

    a. In Visual Studio, on the **File** menu, point to **New**, and then click **Project**.

    b. In the **New Project** dialog box, in the Project Types pane, expand **Visual C#**, and then click **Windows**.

    c. In the Templates pane, click **WPF Application**.

    d. Specify the following values for each of the properties in the dialog box, and then click **OK**:

        - Name: **WpfApplication**
        - Location: **E:\Labfiles\Lab 1\Ex2\Starter**
        - Solution name: **WpfApplication**
        - Create directory for solution: Select the check box

Task 2: Create the user interface

1. Add **TextBox**, **Button**, and **TextBlock** controls to the MainWindow window. Place them anywhere in the window:

    a. Click the **Toolbox** tab.

    b. Expand the **Common WPF Controls** section of the Toolbox if it is not already open.

    c. Click the **TextBox** control, and then drag it anywhere onto the MainWindow window.

    d. Click the **Toolbox** tab.

    e. Click the **Button** control, and then drag it anywhere onto the MainWindow window.

    f. Click the **Toolbox** tab.

    g. Click the **TextBlock** control, and then drag it anywhere onto the MainWindow window.

2. Using the Properties window, set the properties of each control by using the values in the following table. Leave any other properties at their default values.

Control	Property	Value
TextBox	**Name**	**testInput**
	Height	**28**
	HorizontalAlignment	**Left**
	Margin	**12,12,0,0**
	VerticalAlignment	**Top**
	Width	**302**
Button	**Name**	**testButton**
	Content	**Format Data**
	Height	**23**
	HorizontalAlignment	**Left**
	Margin	**320,17,0,0**
	VerticalAlignment	**Top**
	Width	**80**
TextBlock	**Name**	**formattedText**
	Height	**238**
	HorizontalAlignment	**Left**
	Margin	**14,50,0,0**
	Text	***blank***
	VerticalAlignment	**Top**
	Width	**384**

a. In the MainWindow window, click the **TextBox** control.

b. In the Properties window, click the **textBox1** text adjacent to the **TextBox** prompt, and then change the name to **testInput**.

c. In the list of properties in the Properties window, locate the **Height** property, and then change it to **28**.

d. Repeat this process for the remaining properties of the **TextBox** control.

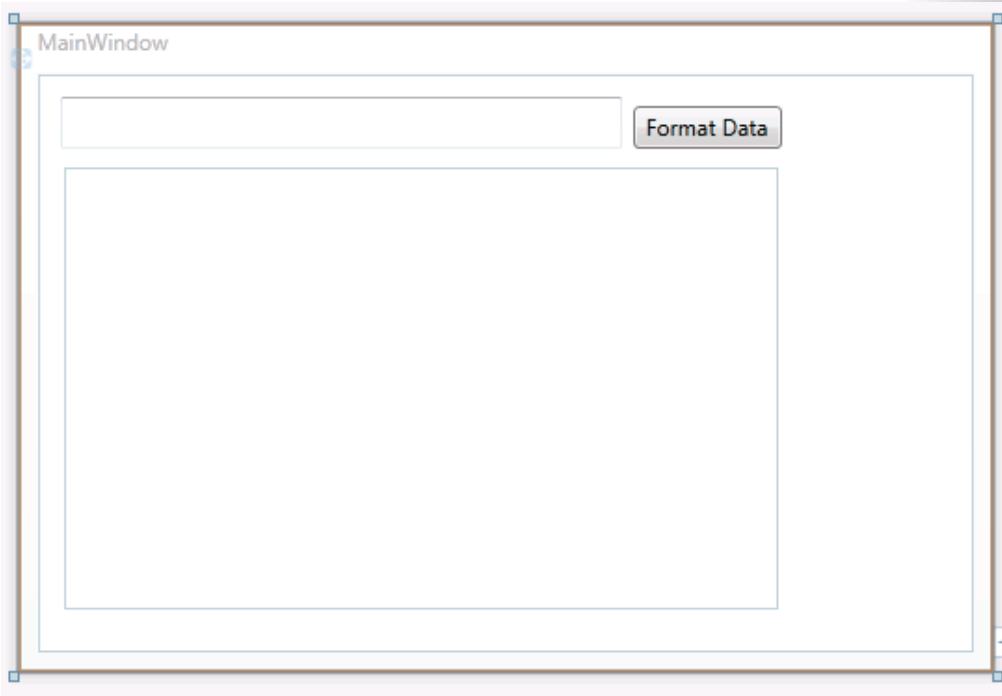
e. In the MainWindow window, click the **Button** control.

f. Follow the procedure described in steps b to e to set the specified properties for this control.

g. In the MainWindow window, click the **TextBlock** control.

h. Follow the procedure described in steps b to e to set the specified properties for this control.

The MainWindow window should look like the following screen shot.



Task 3: Add code to format the data that the user enters

1. Create an event handler for the **Click** event of the button:
   a. In the MainWindow window, click the **Button** control.
   b. In the Properties window, click the **Events** tab.
   c. In the list of events, double-click the **Click** event.
2. Add the code shown in bold in the following code example to the event-handler method.

```
private void testButton_Click(object sender, RoutedEventArgs e)
{
      // Copy the contents of the TextBox into a string
      string line = testInput.Text;

      // Format the data in the string
      line = line.Replace(",", " y:");
      line = "x:" + line;

      // Store the results in the TextBlock
      formattedText.Text = line;
}
```

   This code reads the contents of the **TextBox** control into a string variable called line, formats this string in the same way as the console application in Exercise 1, and then displays the formatted result in the **TextBlock** control. Notice that you can access the contents of a **TextBox** control and a **TextBlock** control by using the **Text** property.

3. Build the solution, and then correct any errors:
   - On the **Build** menu, click **Rebuild Solution**.
4. Run the application and verify that it works in a similar manner to the original console application in Exercise 1:
   a. On the **Debug** menu, click **Start Without Debugging**.
   b. In the MainWindow window, type **23.654,67.823** into the **TextBox** control.
   c. Click **Format Data**.
   d. Verify that **x:23.654 y:67.823** appears in the **TextBlock** control below the **TextBox** control.
5. Close the MainWindow window, and then return to Visual Studio.

Task 4: Modify the application to read data from a file

1. Create an event handler for the **Window_Loaded** event. This event occurs when the window is about to be displayed, just after the application has started up:
   a. Display the MainWindow.xaml file.
   b. Click the title bar of the MainWindow window.
   c. In the Properties window, click the **Events** tab.
   d. In the list of events, double-click the **Loaded** event.
2. In the event-handler method, add the code shown in bold in the following code example.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    // Buffer to hold a line read from the file on standard input
    string line;

    // Loop until the end of the file
    while ((line = Console.ReadLine()) != null)
    {
        // Format the data in the buffer
        line = line.Replace(",", " y:");
        line = "x:" + line + "\n";
        // Put the results into the TextBlock
        formattedText.Text += line;
    }
}
```

   This code reads text from the standard input, formats it in the same manner as Exercise 1, and then appends the results to the end of the **TextBlock** control. It continues to read all text from the standard input until end-of-file is detected.

   Notice that you can use the **+=** operator to append data to the **Text** property of a **TextBlock** control, and you can add the newline character ("\n") between lines for formatted output to ensure that each item appears on a new line in the **TextBlock** control.

3. Perform the following steps to modify the project settings to redirect standard input to come from the DataFile.txt file. A copy of this file is available in the E:\Labfiles\Lab 1\Ex2\Starter folder:
   a. In Solution Explorer, right-click the **WpfApplication** project, point to **Add**, and then click **Existing Item**.

b. In the **Add Existing Item – WpfApplication** dialog box, move to the **E:\Labfiles\Lab 1\Ex2\Starter** folder, select **All Files (*.*)** in the drop-down list box adjacent to the **File name** text box, click **DataFile.txt**, and then click **Add**.

c. In Solution Explorer, select **DataFile.txt**. In the Properties window, change the **Build Action** property to **None**, and then change the **Copy to Output** property to **Copy Always**.

d. In Solution Explorer, right-click the **WpfApplication** project, and then click **Properties**.

e. On the **Debug** tab, in the **Command line arguments**: text box, type **< DataFile.txt**

f. On the **File** menu, click **Save All**.

g. Close the WpfApplication properties window.

4. Build and run the application in Debug mode. Verify that, when the application starts, it reads the data from DataFile.txt and displays in the **TextBlock** control the results in the following code example.

```
x:23.8976 y:12.3218
x:25.7639 y:11.9463
x:24.8293 y:12.2134
```

- On the **Debug** menu, click **Start Debugging**.

5. Close the MainWindow window, and then return to Visual Studio.

Exercise 3: Verifying the Application

Task 1: Modify the data in the DataFile.txt file

- Modify the contents of the DataFile.txt file as the following code example shows.

```
1.2543,0.342
32525.7639,99811.9463
24.8293,12.2135
23.8976,12.3218
25.7639,11.9463
24.8293,12.2135
```

a. In Solution Explorer, double-click **DataFile.txt**.

b. Edit the data in the file so that it resembles the data shown.

c. On the **File** menu, click **Save All**.

d. Close the DataFile.txt window.

Note: There must be a blank line at the end of DataFile.txt.

Task 2: Step through the application by using the Visual Studio 2010 debugger

1. Set a breakpoint at the start of the **Window_Loaded** event handler:

   a. Display the MainWindow.xaml.cs file.

   b. Scroll down to the **Window_Loaded** event.

   c. Right-click the statement in the following code example, point to **Breakpoint**, and then click **Insert Breakpoint**.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
```

2. Start the application running in Debug mode:

   - On the **Debug** menu, click **Start Debugging**.

   When the application runs the **Window_Loaded** event handler, it reaches the breakpoint and drops into Visual Studio. The opening brace of the method is highlighted.

3. Step into the first statement in the **Window_Loaded** method that contains executable code:

   - On the **Debug** menu, click **Step Into**, or press F11.

   The **while** statement should be highlighted. This is because the statement that declares the line variable does not contain any executable code.

4. Examine the value of the line variable. It should be **null** because it has not yet been assigned a value:

   - In the Locals window, verify that the value of line is **null**.

5. Step into the next statement:

   - On the **Debug** menu, click **Step Into**, or press F11.

- The cursor moves to the opening brace at the start of the body of the **while** loop.

6. Examine the value of the line variable. It should be **1.2543,0.342**. This is the text from the first line of the DataFile.txt file. The **Console.ReadLine** statement in the **while** statement reads this text from the file:
   - In the Locals window, verify that the value of line is **1.2543,0.342**.
7. Step into the next statement:
   - On the **Debug** menu, click **Step Into**, or press F11.

   The cursor moves to the line in the following code example.

```
line = line.Replace(",", " y:");
```

8. Step into the next statement:
   - On the **Debug** menu, click **Step Into**, or press F11.
9. Examine the value of the line variable. It should now be **1.2543 y:0.342**. This is the result of calling the **Replace** method and assigning the result back to line:
   - In the Locals window, verify that the value of line is **1.2543 y:0.342**.
10. Step into the next statement:
    - On the **Debug** menu, click **Step Into**, or press F11.
11. Examine the value of the line variable. It should now be **x:1.2543 y:0.342\n**. This is the result of prefixing the text "x:" to line and suffixing a newline character:
    - In the Locals window, verify that the value of line is **x:1.2543 y:0.342\n**.
12. Step into the next statement:
    - On the **Debug** menu, click **Step Into**, or press F11.

    The cursor moves to the closing brace at the end of the **while** loop.
13. In the Immediate window, examine the value of the **Text** property of the **formattedText TextBlock** control. It should contain the same text as the line variable:

Note: If the Immediate window is not visible, press CTRL+ALT+I.

a. In the Immediate window, type the expression in the following code example (including the question mark), and then press ENTER.

```
?formattedText.Text
```

b. Verify that the text "x:1.2534 y:0.342\n" is displayed.

14. Set another breakpoint at the end of the **while** loop:

- Right-click the closing brace at the end of the **while** loop, point to **Breakpoint**, and then click **Insert Breakpoint**.

15. Continue the programming running for the next iteration of the **while** loop. It should stop when it reaches the breakpoint at the end of the loop:

- On the **Debug** menu, click **Continue**, or press F5.

16. Examine the value of the line variable. It should now be **x:32525.7639 y:99811.9463\n**. This is the data from the second line of DataFile.txt:

- In the Locals window, verify that the value of line is **x:32525.7639 y:99811.9463\n**.

17. In the Immediate window, examine the value of the **Text** property of the **formattedText TextBlock** control again. It should now contain the formatted results from the first two lines of DataFile.txt:

a. In the Immediate window, on a blank line after the previous results, type the expression in the following code example (including the question mark), and then press ENTER.

```
?formattedText.Text
```

b. Verify that the text "x:1.2543 y:0.342\n x:32525.7639 y:99811.9463\n" is displayed.

18. Remove the breakpoint from the end of the **while** loop:

- Right-click the closing brace at the end of the **while** loop, point to **Breakpoint**, and then click **Delete Breakpoint**.

19. Continue the programming running. The **Window_Loaded** method should now run to completion and display the MainWindow window. The **TextBlock** control should contain all of the data from DataFile.txt, formatted correctly:

a. On the **Debug** menu, click **Continue**, or press F5.

b. Verify that the **TextBlock** control displays the formatted results for every line in the DataFile.txt file.

20. Close the MainWindow window, and then return to Visual Studio.

Exercise 4: Generating Documentation for an Application

Task 1: Open the starter project

- In Visual Studio, open the WpfApplication solution located in the E:\Labfiles\Lab 1\Ex4\Starter folder. This solution is a working copy of the solution from Exercise 2:

  a. In Visual Studio, on the **File** menu, point to **Open**, and then click **Project/Solution**.

  b. Move to the **E:\Labfiles\Lab 1\Ex4 \Starter** folder, click **WpfApplication.sln**, and then click **Open**.

Task 2: Add XML comments to the application

1. Display the MainWindow.xaml.cs file:

   - In Solution Explorer, expand **MainWindow.xaml**, and then double-click **MainWindow.xaml.cs**.

2. Add the XML comment in the following code example before the **MainWindow** class declaration.

```
/// <summary>
/// WPF application to read and format data
/// </summary>
```

3. Add the XML comment in the following code example before the **MainWindow** constructor.

```
/// <summary>
/// Constructor for MainWindow
/// </summary>
```

4. Add the XML comment in the following code example before the **testButton_Click** method.

```
/// <summary>
/// Read a line of data entered by the user.
/// Format the data and display the results in the
/// formattedText TextBlock control.


/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
```

5. Add the XML comment in the following code example before the **Window_Loaded** method.

```
/// <summary>
/// After the Window has loaded, read data from the standard input.
/// Format each line and display the results in the
/// formattedText TextBlock control.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
```

6. Save MainWindow.xaml.cs.

Task 3: Generate an XML comments file

1. Set the project properties to generate an XML documentation file when the project is built:
    a. In Solution Explorer, right-click the **WpfApplication** project, and then click **Properties**.
    b. On the **Build** tab, select the **XML Documentation file** check box, and then verify that the file name is set to: **bin\Debug\comments.XML**.
    c. On the **File** menu, click **Save All**.
    d. Close the WpfApplication properties window.
2. Build the solution, and then correct any errors:
    - On the **Build** menu, click **Rebuild Solution**.
3. Verify that an XML comments file called comments.xml has been generated in the E:\Labfiles\Lab 1\Ex4\Starter\WpfApplication\bin\Debug folder, and then examine it:

a. Using Windows Explorer, move to the **E:\Labfiles\Lab 1\Ex4\Starter \WpfApplication\bin\Debug** folder.

b. Double-click the **comments.xml** file. The file should be displayed in Internet Explorer. Verify that it contains the text for the XML comments that you added to the WPF application (it will also contain other comments that Visual Studio has generated).

c. Close Internet Explorer.

4. Copy the comments.xml file to the E:\Labfiles\Lab 1\Ex4\Helpfile folder.

Task 4: Generate a .chm file

1. Open a Windows Command Prompt window as **Administrator**. The **Administrator** password is **Pa$$w0rd**:

   a. Click **Start**, point to **All Programs**, click **Accessories**, right-click **Command Prompt,** and then click **Run as administrator**.

   b. In the **User Account Control** dialog box, in the **Password** text box, type **Pa$$w0rd** and then click **Yes**.

2. Move to the **E:\Labfiles\Lab 1\Ex4\HelpFile** folder.

3. Use Notepad to edit the builddoc.cmd script, and then verify that the input variable is set to "E:\Labfiles\Lab 1\Ex4\Starter\WpfApplication\bin\Debug \WpfApplication.exe":

   a. In the Command Prompt window, type the command in the following code example.

```
notepad builddoc.cmd
```

   b. Verify that Line 13 looks like the following code example.

```
set input="E:\Labfiles\Lab
1\Ex4\Starter\WpfApplication\bin\Debug\WpfApplication.exe"
```

   c. Close Notepad.

4. Run the builddoc.cmd script:

   a. In the Command Prompt window, type the command in the following code example.

```
 builddoc.cmd
```

b. Press ENTER when the script prompts you to do so.

5. Open the test.chm file that the builddoc.cmd script generates:

   a. Using Windows Explorer, move to the **E:\Labfiles\Lab 1 \Ex4\HelpFile\Output** folder.

   b. Double-click the **test.chm** file.

6. Browse documentation that is generated for your application, and then close test.chm.