LAB 4

User Interface Design

This lab contains the following exercises and activities:

- Exercise 4-1: Adding error handling
- Exercise 4-1: Improving the user interface
- Exercise 4-3: Using Data Binding
- Exercise 4-4: Databinding with Lists

SCENARIO

During these practical sessions you will take the role of a developer creating applications for the Windows Phone platform.

After completing this lab you will be able to:

- Add error handling and presentation to a Windows Phone application
- Use events and data binding to improve a Windows Phone application
- Create a multi-page Windows Phone application

Estimated lesson time: 45 minutes

EXERCISE 4-1: ADDING ERROR HANDLING

Estimated completion time: 25 minutes

In this exercise you will improve the error handling provided by a TimeTracker application.

Note: In the following text things like **File>New>Project** mean "Open the **File** Menu on the top of Visual Studio, Select **New** and then select **Project**."

The present version of the code works well if the values are sensible, but it will fail if the user enters values which are out of range or types text instead of numbers. You will need to add extra code to the program to deal with this. The things you should address are:

- Entering text instead of numbers into the text boxes
- Entering hour values which are greater than 23 or less than 0
- Entering minute values which are greater than 59 or less than 0
- Entering a start time which is after the end time

In this exercise you will improve a version of the Time Calculator that another programmer has created for you.

1. Use Visual Studio to open the **TimeCalculator** project in the **Lab 4.1 Working TimeCalculator** folder.
2. Run the program. Enter the values **Start: 0a:00** and **End 00:00**
3. Press the calc button.
4. The application will throw an exception as the value 0a can't be converted into a number.

Using the example code from the lectures and notes to make the following changes to the code.

- Use the TryParse method to convert from text to number so that the program can detect invalid number entries
- Display invalid items in Red text and valid ones in the default text colour
- Display a message box if the start and end times are invalid

EXERCISE 4-2: IMPROVE THE USER INTERFACE

Estimated completion time: 20 minutes

In this exercise you will remove the need for the Calculate button so that the display updates automatically. You will base this work on the solution that you created in 4.1 so that errors are automatically handled as well.

TextChanged events

The TextBoxes in the application can be made to generate TextChanged events when the user types in new values. Code that is bound to these events can automatically update the values on the display.

1. Add handling for TextChanged events so that there is no need for the calculate button.
2. Remove the calculate button.

The best way to organise this is to have a single calculate method which is called as a result of changes to any of the text boxes. You can even bind the changed events from each display TextBox to the same method to make the code even simpler.

EXERCISE 4-3: USING DATA BINDING

Estimated completion time: 25 minutes

In this exercise you will add data binding to a version of the Time Calculator program.

The TimeClass

Another programmer has created a TimeClass for the application. This has five properties:

1. StartHour
2. StartMinute
3. EndHour
4. EndMinute
5. MinuteDifference

You have been employed to perform the data binding that will link the display to these values. The first four items will use Two Way binding, with the fifth being an output value which is used to display the result.

Adding the TimeClass as a resource

The first step is to add the TimeClass resource to our the project so that it can be used by Silverlight elements

1. Use Visual Studio to open the **TimeCalculator** project in the **Lab 4.3 Data Binding Time Calculator** folder.
2. Open the **MainPage.xaml** page.
3. The first thing you need to do is add the namespace to the MainPage resources.
4. Locate the line that starts:

```
xmlns:mc="http://schemas.openxmlformats.....
```

5. Add the following line beneath it.

```
xmlns:local="clr-namespace:TimeCalculator"
```

6. Now you need to add the binding to the class we want to use. Locate the line:

```
shell:SystemTray.IsVisible="True">
```

7. Now add the following lines beneath it.

```
<phone:PhoneApplicationPage.Resources>
    <local:TimeClass x:Key="TimeClass" />
</phone:PhoneApplicationPage.Resources>
```

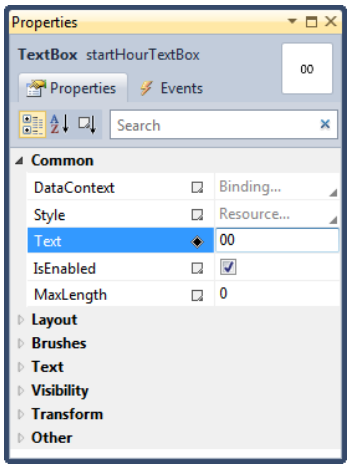
8. Now you can add the resource to the grid that contains the Silverlight elements in our user interface. Locate the Grid item with the name LayoutRoot and add the following to the item as a new attribute:

```
DataContext="{StaticResource TimeClass}"
```

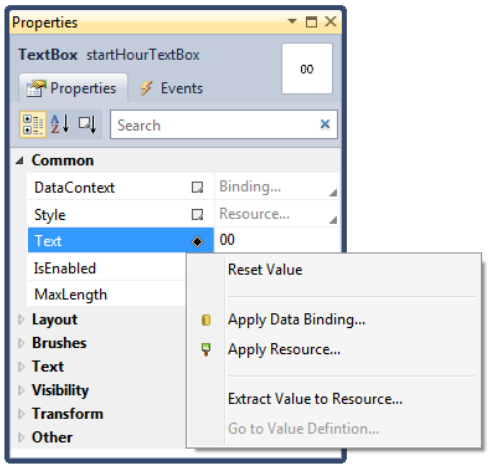
Databinding the Properties in the Controls

Now that we have the DataContext of the TimeClass as part of our page we can bind properties of elements on the display to these.

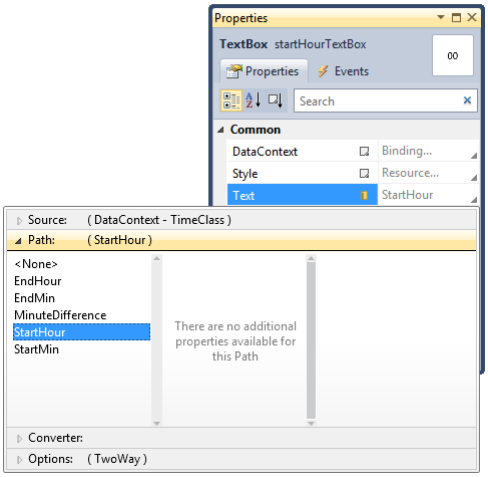
1. Click on **startHourTextBox** in the editor so that its properties are shown in the Properties pane.



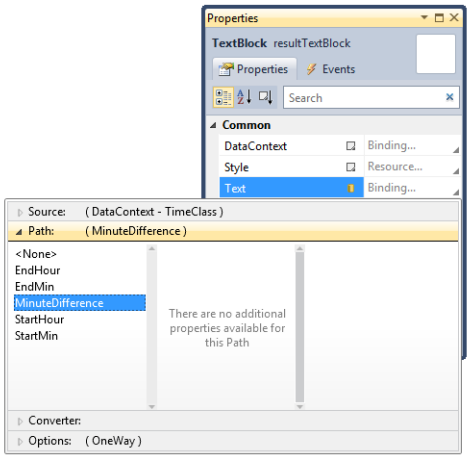
2. Find the Text property and click on the black diamond as shown above. This will cause the settings for this property to appear as shown below.



3. Select Apply Data Binding from the menu. This should cause a list of available properties to appear as shown below. Double click the StartHour item for this binding. Make sure that the options are set to TwoWay.



4. Repeat this for StartMin, EndHour and Endmin.
5. The final item to bind is the Result item to the MinuteDifference value as show below.



6. Note that for this property the binding is only OneWay, as the result is a TextBlock and not a TextBox. Bind the property.

7. Run the program and enter values into the EndTime values to test that the value updates. Note that the updates only occur when the user navigates away from the text boxes.

The program now uses data binding, but some of the error handling has been removed. You might like to add some error management to the solution. The best way to do this would be to have the TimeClass expose some error message properties that can be bound to display items. Note that if you do this you will not need to add the class again, the new properties will just be picked up when the list of properties is generated.

EXERCISE 4-4: DATA BINDING WITH LISTS

Estimated completion time: 25 minutes

In this exercise you will explore data binding and use it to display a list of session log values.

Your boss has seen the Customer Manager application that has been used in this course to demonstrate how a list of customers can be managed on a Windows Phone. She wants the application extended to allow it to display a list of sessions for each customer. A programmer has done some work on the project but has not completed it. You will have to find out how the program works and add the code to finish it off.

The Session Class

This contains a text description of the session (for example "Takeover Litigation Meeting") and the length of the session, in minutes.

```
public class Session
{
    public string Description { get; set; }
    public int LengthInMins { get; set; }

    public Session(string inDescription, int inLength)
    {
        Description = inDescription;
        LengthInMins = inLength;
    }
}
```

The customer class contains a list of sessions for that customer.

```
public List<Session> Sessions;
```

The program creates a set of test session data for each customer, but your boss doesn't think it creates enough.

Increasing the Amount of Test Data

The first step is to edit the **MakeTestCustomers** method in the **Customers** class to create much more test data.

1. Use Visual Studio to open the **CustomerManager** project in the **Lab 4.4 Customer Time Logger** folder.
2. Open the **Customers.cs** source file.
3. Find the **MakeTestCustomers** method in the **Customers** class.

4. This method creates a set of test customers, and for each customer it creates a number of test sessions. The number of sessions for each customer is determined randomly, but the range of values is too small.
5. Find the statement:

```
int noOfSessions = sessionRand.Next(1,4);
```

6. The Next method will return a value in the range 1 to 3, giving no more than 3 test sessions per customer. (Note that the upper limit of the Next method is exclusive)
7. Change this to a more appropriate value, so that each customer will have between 20 and 30 sessions. This will allow us to make sure that the customer list display scrolls correctly.
8. Leave Visual Studio open for the next task.

Add the Page Navigation to the Session Button

The Customer information now contains a Sessions button that can be used to view the sessions that have taken place for that customer.

At the moment it is not connected to an event handler, and so nothing happens when it is selected. The next thing to do is create an event handler for this button and add the code to navigate to the Sessions display.

The first step is to edit the **MakeTestCustomers** method in the **Customers** class to create much more test data.

1. Use Visual Studio to open the **CustomerDetailPage.xaml** page in the design editor.
2. We are going to add an event handler for the sessions button. Double click on the Sessions button to create the event handler and navigate to the code in the file **CustomerDetailPage.xaml.cs**.
3. At the moment the **sessionButton_Click** method is empty. Add the following code to perform the page navigation.

```
// Navigate to the detail page
NavigationService.Navigate(new Uri("/SessionDetailPage.xaml",
                    UriKind.RelativeOrAbsolute));
```

4. Run the program. Select a customer and then press the **Sessions** button. The program will navigate to the sessions page but nothing

will be displayed there. This is because there is no data binding on that page at the moment.

5. Stop the program, but leave Visual Studio open for the next step.

Adding the Data Binding to the Sessions display page

The program now contains the navigation behaviour but does not display the results. This is because the display components in the **SessionDetailPage** have not been bound to the objects to be displayed. Now we are going to make that final connection and create a working program.

1. Use Visual Studio to open the **SessionDetailPage.xaml.cs** program source. Open the source code, not the design.
2. At the moment there is nothing much in this class apart from the constructor. We are going to add a method that will run when the page is navigated to and set the display up.
3. Position the cursor in the class directly underneath the closing brace of the SessionDetailsPage constructor. Start typing word **override**.
4. Select the word when it appears in the IntelliSense popup.
5. Intellisense will display a list of methods that can be overridden in this class. We want to override the **OnNavigatedTo** method, so select that from the list and press Enter. Make sure you override the correct method, otherwise the program will not work correctly.
6. We now need to insert the text that will find the current customer and then bind the customer name and the customer list to the two display elements on the page. Add the following code into the **OnNavigatedTo** method.

```
// Get the parent application that contains the active custom
App thisApp = Application.Current as App;

CustomerName.DataContext = thisApp.ActiveCustomer;
SessionList.ItemsSource = thisApp.ActiveCustomer.Sessions;
```

7. The first statement obtains a reference to the currently executing application. This contains the variable **ActiveCustomer** which is the customer currently selected. The second statement sets the name of the customer to the customer display at the top of the page. The third sets the session list to the sessions for this customer.
8. Run the program. Select a customer and then press the **Sessions** button for that customer. You should see a list of sessions for that customer. If you press the **Back** key you will be sent back to the customer. Pressing **Back** key again will take you back to the customer list from which you can select another customer and view their settings.

The data binding in the session list is presently one way, in that if we add or edit session information this will not be reflected in the display. You might like to find out what you would need to do to allow changes in the sessions to update the list.