

An Introduction to C# and the Visual C#.NET IDE

“Adding the Tools of the Trade to your OBP Toolbox”

Walk&Talk Activity 3

Creating a Windows Presentation Foundation (WPF) Application

The Visual Studio programming environment contains everything you need to create

graphical Windows-based applications and allows you to design the form-based user interface of a Windows-based application interactively

Visual Studio 2012 will generate the program statements to implement the user interface you’ve designed

Visual Studio 2012 provides you with two views of a graphical application: the design view and the code view

You use the Code and Text Editor window to modify and maintain the code and logic for a graphical application, and you use the Design View window to lay out your user interface

You can switch between the two views whenever you want

This introduction will display a simple form containing a text box where you

can enter your name and a button that displays a personalized greeting in a message box when you click the button

Perform the following operations to create a new graphical application:

Having launched Visual Studio 2012,

1. On the File menu, point to New, and then click Project

The New Project dialog box opens

2. In the Project Types pane, click Visual C#

3. In the Templates pane, click the WPF Application icon

4. Ensure that the Location field refers to a personal folder of your choice

5. In the Name field, type WPFintroduction

6. Ensure Create directory for solution is selected

This action creates a new solution for holding the project

Ignore the Add to source control tick box

(to be discussed, along with adding this project to an existing solution)

7. Click OK

Visual Studio 2012 creates the new WPF application

It displays an empty WPF form in the Design View window, together with another window containing an XAML description of the form

XAML stands for Extensible Application Markup Language and is an XML-like language used by WPF applications to define the layout of a form and its contents

If you have knowledge of XML, XAML will look familiar

You can actually define a WPF form completely by writing in XAML

In the walkthru, you’ll use the Design View window to add three controls to the

Windows form and examine some of the C# code automatically generated by Visual Studio 2012 to implement these controls

Firstly, let’s create the user interface

1. Click the Toolbox tab that appears to the left of the form in the Design View window

The Toolbox appears, partially obscuring the form, and displaying the various components and controls that you can place on a Windows form

The Common section displays a list of controls that are used by most WPF applications

The Controls section displays a more extensive list of controls

2. In the Common section, click Label, and then click the visible part of the form

A label control is added to the form (you will move it to its correct location in a moment)

Tip: If you want the Toolbox to remain visible but not to hide any part of the form, click the Auto Hide button to the right in the Toolbox title bar (it looks like a pin)

The Toolbox appears permanently on the left side of the Visual Studio 2012 window, and the Design View window shrinks to accommodate it

Clicking the Auto Hide button once more causes the Toolbox to disappear again

3. The label control on the form is probably not exactly where you want it to be

You can click and drag the controls you have added to a form to reposition them

Using this technique, move the label control so that it is positioned toward the upper-left corner of the form

Note: The XAML description of the form in the lower pane now includes the label control, together with properties such as its location on the form, governed by the Margin property

The Margin property consists of four numbers indicating the distance of each edge of the label from the edges of the form

If you move the control around the form, the value of the Margin property changes

If the form is resized, the controls anchored to the form’s edges that move are resized to preserve their margin values

You can prevent this by setting the Margin values to zero

4. On the View menu, click Properties Window

The Properties window appears on the lower-right side of the screen, under Solution

Explorer (if it was not already displayed)

The Properties window provides another way for you to modify the properties for items on a form, as well as other items in a project

It is context sensitive in that it displays the properties for the currently selected item

If you click the title bar of the form displayed in the Design View window, you can see that the Properties window displays the properties for the form itself

If you click the label control, the window displays the properties for the label instead

If you click anywhere else on the form, the Properties window displays the properties for a mysterious item called a grid

A grid acts as a container for items on a WPF form, and you can use the grid, among other things, to indicate how items on the form should be aligned and grouped together

5. Click the label control on the form

In the Properties window, locate the Text section

By using the properties in this section, you can specify the font and font size for the

label but not the actual text that the label displays

6. Change the FontSize property to 20, and then click the title bar of the form

The size of the text in the label changes, although the label is no longer big enough to display the text

Change the FontSize property back to 12

Note: The text displayed in the label might not resize itself immediately in the Design

View window

It will correct itself when you build and run the application, or if you close and open the form in the Design View window

7. Scroll the XAML description of the form in the lower pane to the right, and examine the properties of the label control

The label control consists of a <Label> tag containing property values, followed by the text for the label itself (“Label”), followed by a closing </Label> tag

8. Change the text Label ( just before the closing tag) to Please enter your name

Notice that the text displayed in the label on the form changes, although the label is

still too small to display it correctly

9. Click the form in the Design View window, and then display the Toolbox again

Note: If you don’t click the form in the Design View window, the Toolbox displays the message “There are no usable controls in this group”

10. In the Toolbox, click TextBox, and then click the form

A text box control is added to the form

Move the text box control so that it is directly underneath the label control

Tip: When you drag a control on a form, alignment indicators appear automatically when the control becomes aligned vertically or horizontally with other controls

This gives you a quick visual cue for making sure that controls are lined up neatly

11. While the text box control is selected, in the Properties window, change the value of the Name property displayed at the top of the window to userName

12. Display the Toolbox again, click Button, and then click the form

Drag the button control to the right of the text box control on the form so that the bottom of the button is aligned horizontally with the bottom of the text box

13. Using the Properties window, change the Name property of the button control to ok

14. In the XAML description of the form, scroll the text to the right to display the caption displayed by the button, and change it from Button to OK

Verify that the caption of the button control on the form changes

15. Click the title bar of the MainWindow.xaml form in the Design View window

In the Properties window, change the Title property to Hello

16. In the Design View window, notice that a resize handle (a small square) appears on the lower right-hand corner of the form when it is selected

Move the mouse pointer over the resize handle

When the pointer changes to a diagonal double-headed arrow, click and drag the pointer to resize the form

Stop dragging and release the mouse button when the spacing around the controls is roughly equal

Important - click the title bar of the form and not the outline of the grid inside the form before resizing it

If you select the grid, you will modify the layout of the controls on the form but not the size of the form itself

Note If you make the form narrower, the OK button remains a fixed distance from the right-hand edge of the form, determined by its Margin property

If you make the form too narrow, the OK button will overwrite the text box control

The right-hand margin of the label is also fixed, and the text for the label will start to disappear when the label shrinks as the form becomes narrower

17. On the Build menu, click Build Solution, and verify that the project builds successfully

18. On the Debug menu, click Start Without Debugging

The application should run and display your form

You can type your name in the text box and click OK, but nothing happens yet

You need to add some code to process the Click event for the OK button, which is what you will do next

19. Click the Close button (the X in the upper-right corner of the form) to close the form and return to Visual Studio

You have managed to create a graphical application without writing a single line of C# code

It does not do much yet (you will have to write some code soon), but Visual Studio actually generates a lot of code for you that handles routine tasks that all graphical applications must perform, such as starting up and displaying a form

Before adding your own code to the application, it helps to have an understanding of what Visual Studio has generated for you

In Solution Explorer, expand the MainWindow.xaml node

The file MainWindow.xaml.cs appears

Double-click the file MainWindow.xaml.cs

The code for the form is displayed in the Code and Text Editor window

It looks like this:

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**using System.Windows;**

**using System.Windows.Controls;**

**using System.Windows.Data;**

**using System.Windows.Documents;**

**using System.Windows.Input;**

**using System.Windows.Media;**

**using System.Windows.Media.Imaging;**

**using System.Windows.Navigation;**

**using System.Windows.Shapes;**

**namespace WPFintroduction**

**{**

**/// <summary>**

**/// Interaction logic for MainWindow.xaml**

**/// </summary>**

**public partial class MainWindow : Window**

**{**

**public MainWindow()**

**{**

**InitializeComponent();**

**}**

**}**

**}**

Apart from a good number of using statements bringing into scope some namespaces that most WPF applications use, the file contains the definition of a class called MainWindow but not much else

There is a little bit of code for the MainWindow class known as a constructor that calls

a method called InitializeComponent, but that is all

A constructor is a special method with the same name as the class

It is executed when an instance of the class is created and can contain code to initialize the instance

You will learn more about constructors in subsequent lectures and activities

In fact, the application contains a lot more code, but most of it is generated automatically based on the XAML description of the form, and it is hidden from you

This hidden code performs operations such as creating and displaying the form, and creating and positioning the various controls on the form

The purpose of the code that you can see in this class is so that you can add your own methods to handle the logic for your application, such as what happens when the user clicks the OK button

Tip: You can also display the C# code file for a WPF form by right-clicking anywhere in the Design View window and then clicking View Code

At this point you might well be wondering where the Main method is and how the form gets displayed when the application runs; remember that Main defines the point at which the program starts

In Solution Explorer, you should notice another source file called App.xaml

If you double-click this file, the XAML description of this item appears

One property in theXAML code is called StartupUri, and it refers to the MainWindow.xaml file, as shown

**<Application x:Class="WPFintroduction.App"**

**xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"**

**xmlns:x="http://schemas.microsoft.com.winfx/2006/xaml"**

**StartupUri="MainWindow.xaml">**

**<Application.Resources>**

**</Application.Resources>**

**</Application>**

If you click the Design tab at the bottom of the XAML pane, the Design View window for App. xaml appears and displays the text “Intentionally left blank. The document root element is not supported by the visual designer”

This occurs because you cannot use the Design View window to modify the App.xaml file

Click the XAML tab to return to the XAML pane

If you expand the App.xaml node in Solution Explorer, you will see that there is also an Application.xaml.cs file

If you double-click this file, you will find it contains the following code:

**using System;**

**using System.Collections.Generic;**

**using System.Configuration;**

**using System.Data;**

**using System.Linq;**

**using System.Windows;**

**namespace WPFintroduction**

**{**

**/// <summary>**

**/// Interaction logic for App.xaml**

**/// </summary>**

**public partial class App : Application**

**{**

**}**

**}**

Once again, there are a number of using statements, but not a lot else, not even a Main method

In fact, Main is there, but it is also hidden

The code for Main is generated based on the settings in the App.xaml file; in particular, Main will create and display the form specified by the StartupUri property

If you want to display a different form, you edit the App.xaml file

Time to write some code for yourself …….write the code for the OK button

1. Click the MainWindow.xaml tab above the Code and Text Editor window to display

MainWindow in the Design View window

2. Double-click the OK button on the form

The MainWindow.xaml.cs file appears in the Code and Text Editor window, but a new

method has been added called OK\_Click

Visual Studio automatically generates code to call this method whenever the user clicks the OK button

This is an example of an event (we will learn more about events as we progress)

3. Add the code shown in bold type to the OK\_Click method:

**void OK\_Click(object sender, RoutedEventArgs e)**

**{**

**MessageBox.Show(“Hello “ + userName.Text);**

**}**

This is the code that will run when the user clicks the OK button

Do not worry too much about the syntax of this code just yet (just make sure you copy it exactly as shown) because you will learn all about methods in subsequent lectures and activities

4. Click the MainWindow.xaml tab above the Code and Text Editor window to display

MainWindow in the Design View window again

5. In the lower pane displaying the XAML description of the form, examine the Button element, but be careful not to change anything

Notice that it contains an element called Click that refers to the OK\_Click method:

<Button Height=”23” … Click=”OK\_Click”>OK</Button>

And finally, test your WPF application

1. On the Debug menu, click Start Without Debugging

2. When the form appears, type your name in the text box, and then click OK

A message box appears, welcoming you by name

3. Click OK in the message box (the message box closes)

4. Close the form and save your project