



silverlight for windows phone

learn & practise



Puja Pramudya

FOREWORD

All the praises and gratitude to Allah SWT for the chance and the strength to compile and finish this e-book titled “Silverlight for Windows Phone: LEARN & PRACTICE”. It has been a enjoyable journey because Windows Phone is Microsoft’s latest mobile platform that is fascinating for end users, and for developers as well. I also like to express my gratitude to Ronald Rajagukguk, for his introduction to the community, and for being an inspiration to keep learning, and always look for opportunities of self-improvement.

WHAT WE CAN LEARN

- ✓ Windows Phone Overview
- ✓ Using Windows Phone Development Tools
- ✓ Silverlight on Windows Phone
- ✓ Specific Features on Windows Phone
- ✓ Developing a Simple Windows Phone Application

TARGET READER

This e-book is written for those who want to get to know, use, and develop applications for Windows Phone, Microsoft’s latest mobile platform. Of course, it would be naive to consider that this e-book covers the topic about Windows Phone entirely, but it can undoubtedly give you a good basic to learn. In this e-book you will not find topics that require advanced hardware supports such as multi-touch or FM, because this e-book is written based on the available emulator.

The readers are assumed to at least understand the C# programming language. Readers are also expected to have used Visual Studio. An understanding in Silverlight is also advisable.

Wise men say, “you bind knowledge by writing them”; therefore this e-book is dedicated to community members, and hopefully will be of use for us all.

Enjoy! :)

Puja Pramudya

puja.pramudya@gmail.com

<http://geeks.netindonesia.net/blogs/poedia>

Microsoft Innovation Center
Bandung Institute of Technology
Indonesia

CONTENTS

Foreword.....	i
What We Can Learn	ii
Target Reader.....	ii
PART I OVERVIEW.....	1
Windows Phone	2
Silverlight and Windows Phone.....	3
Application Life Cycle	4
Security.....	7
Development Requirements	9
System Requirements.....	9
Windows Phone Emulator Requirements	9
PART II LEARN.....	10
You Had Me At “Hello World”	11
Navigations on Windows Phone.....	14
Navigating Between Pages	14
Passing Parameters Between Pages	16
Pivot and Panorama	18
Dealing with Page Orientations	26
Application Bar.....	30
Global Application Bar.....	30
Local Application Bar	32
Local Application Bar (Programmatic Approach).....	34
Inserting Event Handler	35
Web Service Consumption	38
Access via Generated Class.....	38
Using Standard HTTP Request	46
Working with Data	48
Using Isolated Storage.....	54
SIP Layout.....	60
Getting to Know Web Browser.....	64
Globalization & Localization	68
Globalization	68

Localization.....	71
Location Based System.....	79
Getting to Know Accelerometer.....	84
Bing Maps Control for Windows Phone	91
Registering Bing Maps Account.....	91
Using Bing Maps Control	92
PART III PRACTICE.....	96
#1 - Unit Converter.....	97
Preparing the Main Interface	97
Converting.....	101
Adding Culture List	104
Saving User Preferences.....	107
#2 – Stock Screen	110
Preparing Data for Company Stock Values	110
Preparing Company Stock List Page.....	119
Company Stock Detail Navigation	126
Preparing the Stock Transaction Signal Page.....	133
Creating Application Navigation using Application Bar	137
Stock Transaction Detail Page	138
Adding Company List.....	143
CLOSING	iv
REFERENCES	5

PART I

OVERVIEW

WINDOWS PHONE

The year 2010 may be a milestone for Microsoft, and also mobile platform. In the computing industry, what Microsoft had done is called reboot strategy. Microsoft refers to Windows Phone as 'a revolutionary new platform'. Microsoft rebuilt the whole thing from the beginning, with a fresher, cleaner user interface. Using a design philosophy they call Metro, inspired by signs displayed in metro subways, Microsoft's interface shows distinctive characteristics, retrieves information easily, is intuitive, and uses user friendly symbols. Its integration with service available in Microsoft's cloud—Bing, Xbox Live, Push Notification, and Office to name a few—and third party service has given a unique appeal, something Microsoft should have started long ago.

From the development platform point of view, Windows Phone offers an interesting developing experience for developers. A Windows Phone is bound to have 800x480 WGA or 480x320 HVGA resolution, touch screen, GPS sensor, accelerometer, compass, light, camera, multimedia, GPU with DirectX9, and three hardware buttons. As a developer, it is guaranteed that the whole specifications will be available in any devices that support Windows Phone. Each and every device driver is created directly by Microsoft to ensure consistency. To develop application over a Windows Phone platform, you have two popular and modern options: Silverlight and XNA.

Silverlight is known to enable web developers to create stunning interfaces with the combination of controls, text, vector graphic, media, animation, and data binding that can run on a number of platforms and browsers. Meanwhile SNA is a gaming platform that supports 2D and 3D games meant for Xbox 360, console, and PC.

Now, all that we need is the apps :)

SILVERLIGHT AND WINDOWS PHONE

In developing a Windows Phone application, we can select one of the two options, which are Silverlight and XNA. Silverlight for Windows Phone is similar to Silverlight 3 that has been released for web developments. Here are some important points regarding Silverlight in Windows Phone:

- ✓ Uses the same base class library
- ✓ Has been modified for performance
- ✓ Integrated with the hardware
- ✓ Integrated with the operating system
- ✓ Specific API for the device (accelerometer, GPS, etc.)
- ✓ Uses out-of-browser model

In the template provided for application development using Silverlight platform, there are five types of project we can choose, depending on what we need:

- ✓ Windows Phone Application, which provides an empty page with no control at all
- ✓ Windows Phone List Application, which provides a sample scenario for master-detail data application
- ✓ Windows Phone Panorama Application, which provides a sample usage of panorama navigation in an application
- ✓ Windows Phone Pivot Application, which provides a sample usage of pivot navigation in an application
- ✓ Windows Phone Class Library, to build components that can be reused in other projects

By default, a project will consist of these files:

Item	Description
App.xaml/App.xaml.cs	The application's entry point which initializes resources and layouts of the application
MainPage.xaml/MainPage.xaml.cs	Defines a page with interface in the application
Background.png	A graphic file which shows as the application's icon in the applications list. This icon can be replaced
SplashScreenImage.jpg	A graphic file that is displayed when application is launched. Splash screen is designed to give fast response to users while the application's initial page loads
Properties\AppManifest.xml	Manifest file for application package generation purposes
Properties\AssemblyInfo.cs	Assembly file that contains information regarding the name and version of metadata attached to the assembly that is generated
Properties\WMAppManifest.xml	Manifest file with specific metadata regarding Windows Phone application that defines icon name, initial page, etc.

Another point to consider is that applications using Silverlight in Windows Phone fully apply navigation techniques in Silverlight 3. Using frame container, navigation can naturally be easy to handle, and the navigation to go back is integrated to the button on the hardware. No need to override the method :).

APPLICATION LIFE CYCLE

Model execution on a Windows Phone has a complete cycle, from when the application is launched until it is deactivated. This execution model is designed to provide a fast, responsive experience at all times. This causes the Windows Phone to only be able to run one application at a time. This is to prevent the device from being slow or unresponsive due to the existence of background applications.

Several terminologies we should get familiar with in order to understand the aspects of execution model on Windows Phone application

Term	Description
Tombstoning	A procedure in which the operating system deactivates the application process as users exits the application. Operating system preserves any information about the application's state. When the application is re-launched, the operating system restarts the process and sends the last known state from before the application was turned off
Page State	A state regarding the application page. It includes scroll positions or text field contents. Modifications to this state is done by overriding OnNavigatedTo or OnNavigatedFrom methods
Application State	An application's condition in which there are no specific associations to any page. This condition can be modified using PhoneApplicationService class
Persistent Data	Data shared by application. This data is stored and retrieved from isolated storage. Application setting is one example of persistent data
Transient State	Transient data are those related to an instance of the application. Transient data is stored in state dictionary provided by PhoneApplicationService. An application in tombstoned state will return to transient condition when application is reactivated. An example of transient state is web service query

Now let us move on to a short journey about application lifecycle in Windows Phone.

- **Launching**

A Windows Phone application is launched when it is called either because the user pressed the Back button to said application, selected from application list, or from tiles in the main screen. Regardless of the way it is called, an instance of the application will be created, and as the application starts running, **Launching** event is started. The application preferably should not retrieve any data from

isolated storage. Since the event is generated before the application is active or displayed, doing tasks that consume time, such as accessing isolated storage, may cause unwanted user experience because it slows down the application's launch time. Accessing isolated storage, or calling network related actions, should be done asynchronously when application has been loaded. It is also not advisable for the application to call transient state from its previous instance. When an application is launched, it should look like an entirely new instance.

- **Running**

After launching event is handled, the application will start running. In this condition, the application defines its conditions when the user, for example, navigates his way through the application's pages. The only activity that can happen is application incrementally stores data or settings in order to reduce the amount of data to be stored as the application's state changes. For applications using small amount of data, this becomes ignorable.

- **Closing**

A sample scenario that starts closing event is when the user presses the device's Back button on an application's initial page. Application has to store persistent data into isolated storage. It is not necessary to store transient data, or data related only to one instance of application, because the only way for a user to return to application after it is deactivated is by re-launching the application, and as stated above, it will be an entirely new instance.

- **Deactivating**

When a running application is replaced by another, the previous one will be deactivated. There are several scenarios as to how this event is started. One is by pressing the Start button or due to timeout when the main screen is locked. An application can also be deactivated by the invocation of a Launcher or Chooser—default applications that enables users to do common tasks on a mobile device, such as taking pictures or sending emails. In those cases, the running application will start Deactivated event and enter deactivating condition. Unlike when it is closed, an application that launches Deactivating event will enter tombstoned condition. This means that the application is no longer running, but the operating system records the application's conditions and stores several data related to it. It is very likely for users to return to the application, and when this happens, the application enters reactivated condition.

In event Deactivated condition, an application should store information regarding the current conditions using State property on PhoneApplicationService. Data stored into the dictionary are transient data which will restore the application to its condition before it is deactivated. Since there is no guarantee that applications that enter tombstoned condition will be reactivated, applications should also store data into isolated storage. The whole actions has to be finished within 10 seconds, otherwise the operating system will not terminate the application. For this reason, an application that uses large amount of data is advised to store its data incrementally while the application is running.

This is the list of actions that will cause an application to enter tombstoned condition:

- WebBrowserTask
- MarketplaceDetailTask
- MarketplaceHubTask

- SaveEmailAdressTask
- SavePhoneNumberTask
- SearchTask
- SmsComposeTask

The following actions will not automatically cause application to enter tombstoned condition, and thus should be handled:

- PhotoChooserTask
- CameraCaptureTask
- MediaPlayerLauncher
- EmailAddressChooserTask
- PhoneNumberChooserTask

- **Activating**

After an application is deactivated and enters tombstoned condition, it is very likely to be reactivated. Applications can be invoked as a new application instance from **Start**. Users may also start application from another application, causing the tombstoned application to never be launched again. When Launcher or Chooser is the cause of deactivation, users can finish tasks related to the plug-in then return to the application's tombstoned condition. When this happens, application will be reactivated or Activated event will be called. Application should load data stored in PhoneApplicationService dictionary to restore last known condition of the application. Similar to the handling of Launching event, application should not access resources from network or isolated storage to avoid slowing the process down.

This knowledge regarding execution model is absolutely necessary in order to preserve consistency and provide a consistent user experience. Microsoft team has issued the best practice guide regarding execution models in Windows Phone applications, as can be seen [here](#). The following image elaborates the application's workflow for better understanding.

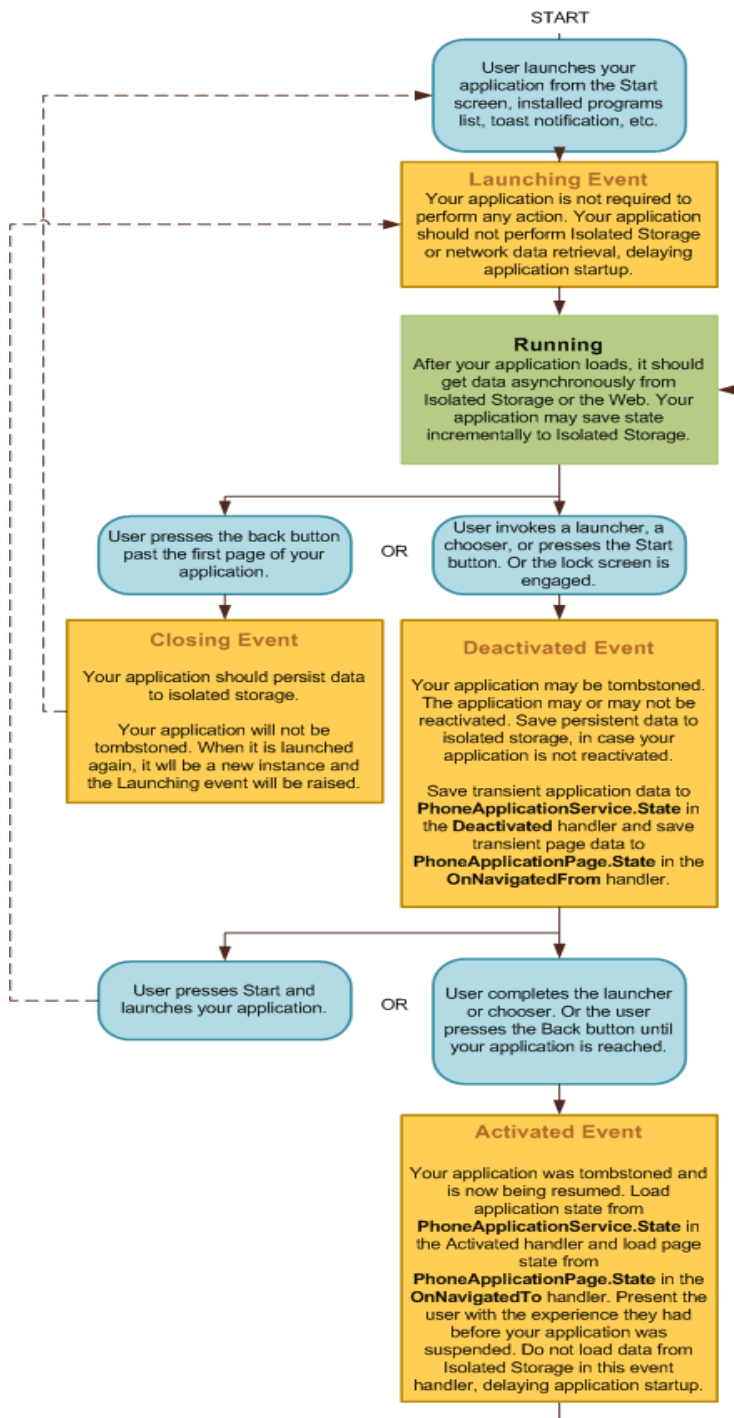


FIGURE 1 EXECUTION MODEL[4]

SECURITY

Security has become a certain issue in Windows Phone application development. This aspect directly affects developers in building the application. There are several built-in features provided in Windows Phone, and these affect what we should do and how we should code. If our application sends or receives sensitive information through the internet we also have to secure the data. In this case Silverlight for Windows Phone provides several classes that can be used.

Silverlight for Windows Phone is designed with several built-in features to support security aspects. Windows Phone applications run in limited environment, Sandbox, thus limiting their access to filesystem or

other application files like any other .NET applications. This assures that applications will not affect operating system or certain features in the device, such as camera or email. From the development point of view, this means that developers only need to know how to call tasks related to the operating system or those features through managed-code, because they cannot directly invoke the features. And this is where Launchers and Choosers come in. Since applications may not access filesystem whilst several scenarios require data storage, Silverlight provides isolated storage, in which we can store data. It is isolated because an application may only access its own isolated storage. This very much simplifies codes regarding to data storage in our application.

The feature mentioned above is already provided, and we only have to learn how to use it. On the other side, Silverlight for Windows Phone also provides some tools for your application's security. Sending sensitive data through internet is one scenario where we want to secure the data. For this purpose, there are a number of namespace that can be applied:

- System.Security.Principal gives information regarding user management and role management
- System.Security.Permissions exposes features on access to certain resources
- System.Security.Cryptography provides encryption and hash functions: AES, SHA1, SHA256, and HMAC

DEVELOPMENT REQUIREMENTS

To begin development and learn how to build Windows Phone applications, we need Windows Phone Developer Tools set. It includes Visual Studio 2010 Express for Windows Phone, Windows Phone Emulator, XNA Game Studio, Expression Blend for Windows Phone, samples, and documentations. If you have already installed Visual Studio Professional or later versions, an additional Add-In for Visual Studio will automatically be installed. It has reached RTW version on September 16th 2010 and can be obtained [here](#).

SYSTEM REQUIREMENTS

- Operating System: Windows 7 and Windows Vista
 - Windows Vista (x86 and x64) ENU Service Pack 2 all editions other than Starter
 - Windows 7 (x86 and x64) ENU – all editions other than Starter
- Hard disk with a minimum 3GB free space
- Recommend 2 GB of memory
- Graphic card that supports DirectX 10 with WDDM 1.1 driver

WINDOWS PHONE EMULATOR REQUIREMENTS

Running the emulator requires system configurations as in system requirements and more attention to these points:

- .xap packet no more than 400MB
- No GPU usage
- Only supports VC-1 encoding, no support for blur and drop-shadow
- Data in isolated storage will be stored in emulator until activated
- Does not support multi touch simulation using mouse; only devices that actually have multi touch feature may support the simulation
- Accelerometer, GPS, and camera cannot be used as in the real device

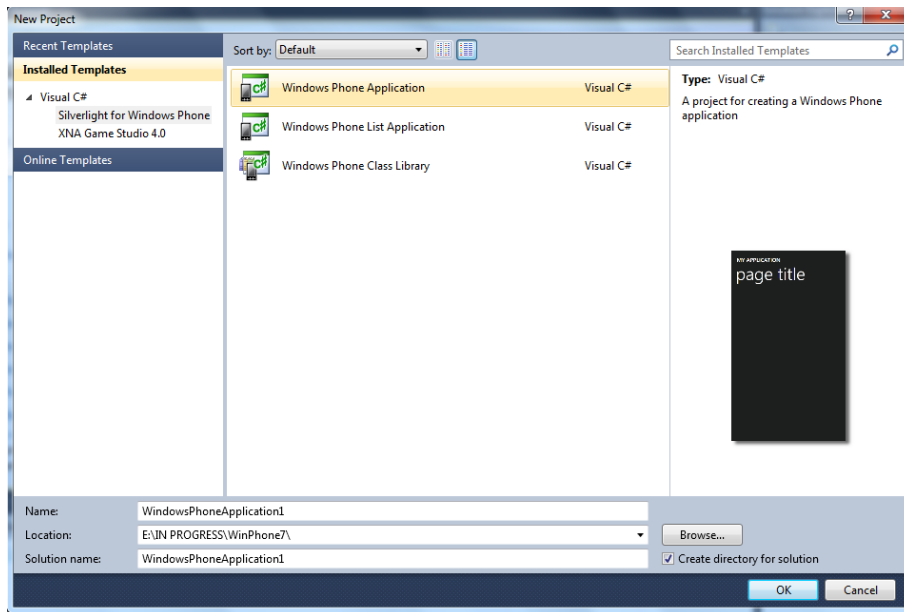
PART II

LEARN

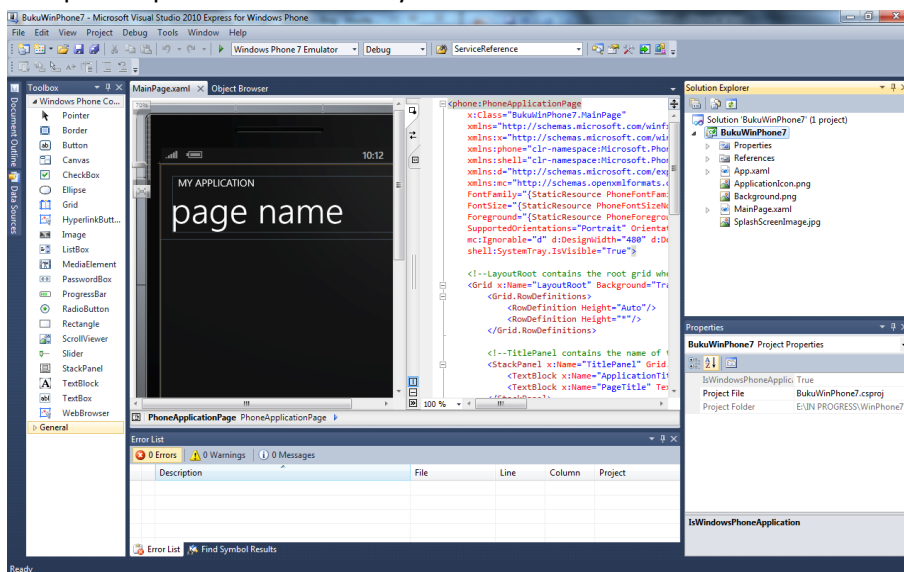
YOU HAD ME AT “HELLO WORLD”

Any programming journey begins by the time coders write their first line of hello world. The first lesson in this Windows Phone e-book should be no different. To keep up with that tradition, here are the steps to make your Windows Phone say hello:

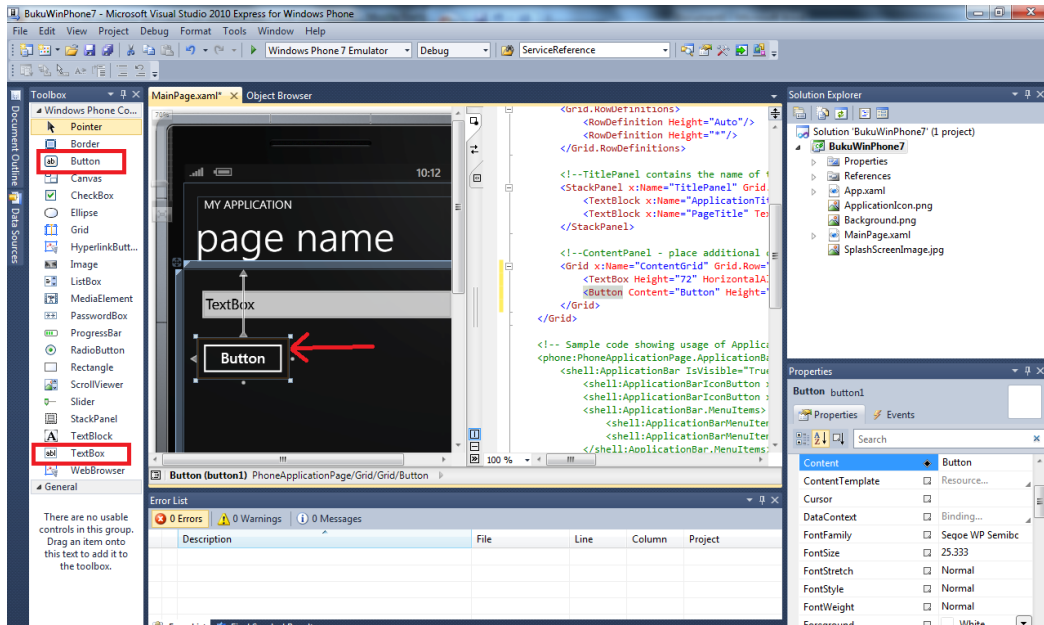
1. Open *Visual Studio Express for Windows Phone*. Select **File -> New Project**. Choose the Visual C# Silverlight for Windows Phone template. Select Windows Phone Application and name the project to your liking.



2. After the project is created, the screen will show design and XAML markup codes. Design view shows the phone interface which enables us to see how our program looks like during the development. For those of you who are already familiar with Visual Studio, then the Tool Box panel, Solution Explorer, and Properties pane will be around your main view.

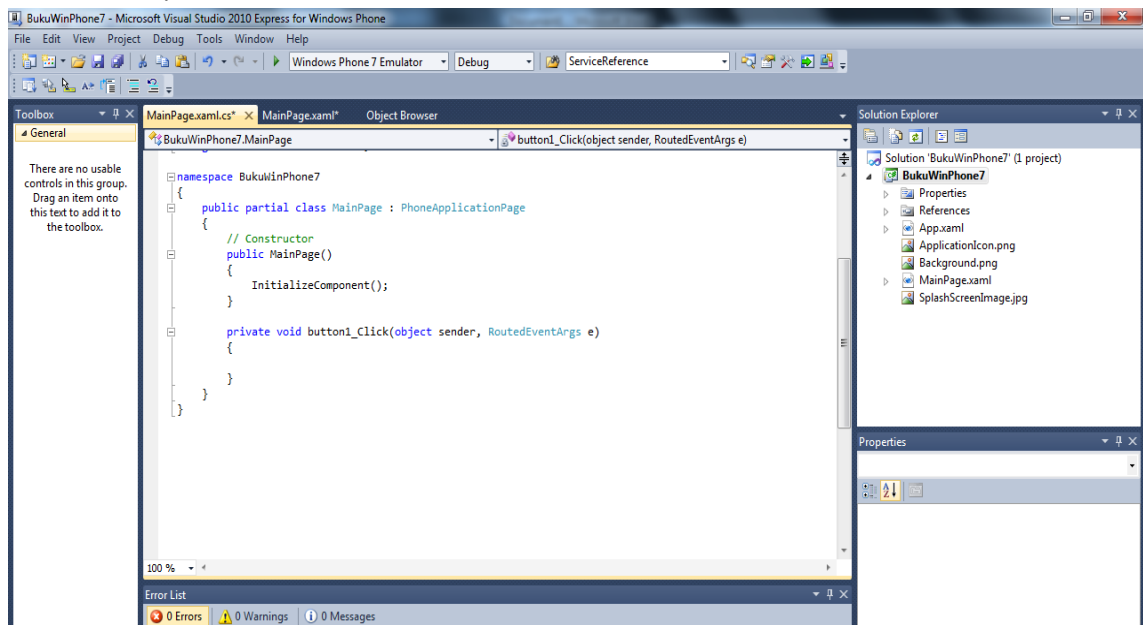


3. Add a **TextBox** and a **Button** from the **ToolBox**. Note that we get the same experience in developing Silverlight for web applications. We can easily alter the interface from the **Properties** pane.



When a Button is selected, we can see that the button is highlighted with a box outside the Button's border. The box indicates the Button's touch area. This property is owned by every control. Change the title text in XAML into "Hello World".

4. Double-clicking the Button will show codes behind the active page. Add a function to change Title into "Hello +" input from available TextBox.

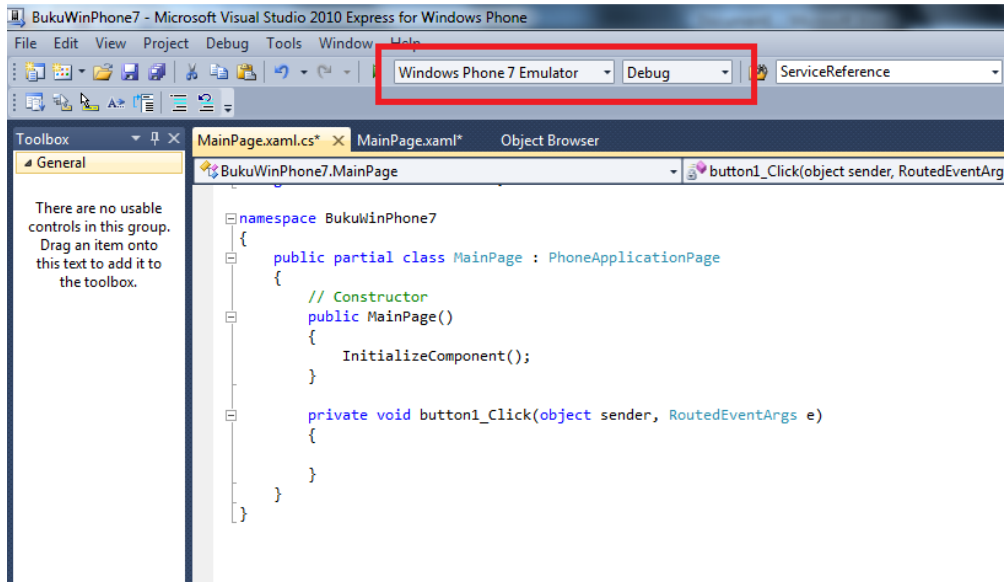


5. Type this code for the Button's event handler,

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    PageTitle.Text = "Hello " + textBox1.Text;
}
```


}

- Now we can test the simple application we have made. To deploy and launch the application we can choose between running in an emulator or available Windows Phone device. Since there are not many deployable Windows Phone yet, let's use the emulator to run this simple application. Press F5 and see the result.



First-time deployment may take some time to process. However, the next deployments will take less time, provided we don't shut down the emulator. Type any text in the TextBox and press the button. The title in the page will change according to the text inserted into TextBox. This is your first Windows Phone application. Congratulations! :)



NAVIGATIONS ON WINDOWS PHONE

As explained in the previous part, navigations on Windows Phone use the same navigation introduced in Silverlight 3. There are two important elements in the application level which are Frame and Page, and one important element in device level.

FRAME

Frame is integrated with the whole layout of a Windows Phone application, and only one frame can be used throughout the application. Several characteristics related to frame are the properties that can be used (full screen, orientation), the ability to expose page areas in it and provide a location for system tray and application bar. System tray is an area in which system status, such as battery, signal, etc. are displayed. Application bar, on the other hand, provides space for frequently used tasks.

PAGE

A page fills the whole content of a frame. Main characteristics of a page are title and the ability to show application bar specifically on certain pages.

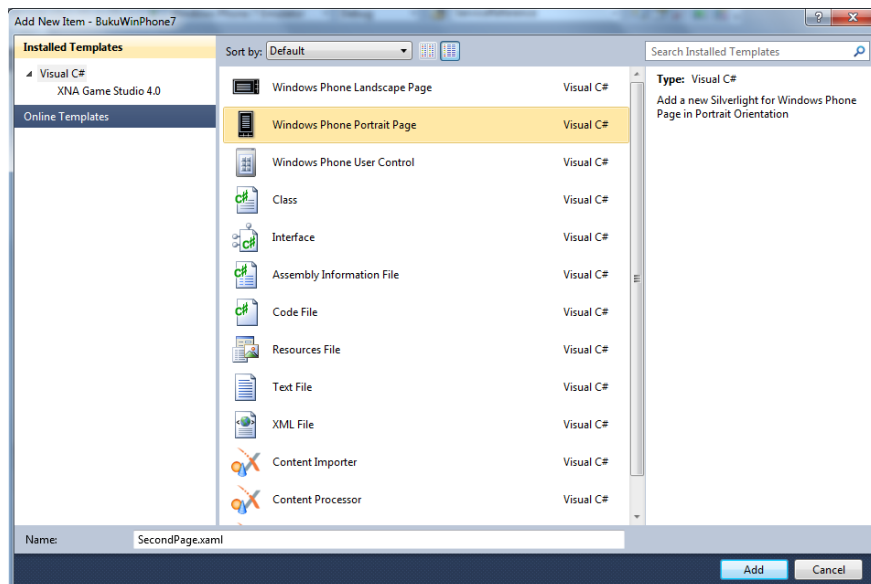
BACK BUTTON

One important element that has become a standard in every Windows Phone device is a “Back” button. This button is used to move one page backwards. With this button present, developers are advised not to add any back button in their applications, unless absolutely necessary. By default the back button will also close any pop-up menu displayed and bring users back to the previous screen.

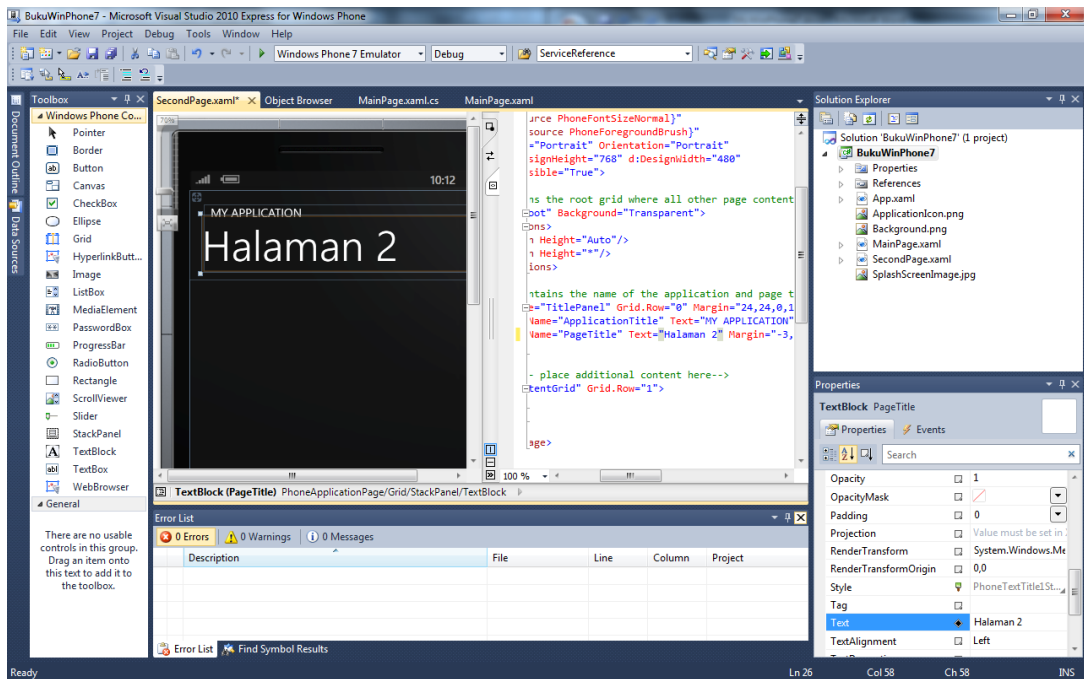
NAVIGATING BETWEEN PAGES

Now we will learn how to navigate between pages in a Windows Phone application. Follow these steps:

1. Use the project we have created in the previous exercise. Add a page; in **Solution Explorer**, select **Add > New Item**.
2. Select **Windows Phone Portrait Page** and rename the file as you like, in this example **SecondPage.xaml**, then select **Add**.



3. Change the page title into “Page 2” from the XAML code



4. On **MainPage.xaml.cs**, change the codes in the Button's click event handler into the following:

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Hello World
    // PageTitle.Text = "Hello " + textBox1.Text;
    NavigationService.Navigate(new Uri("/SecondPage.xaml", UriKind.Relative));
}
```

Navigate is a static function from NavigationService which is used to navigate to desired pages using target URI as the parameter.

5. Press F5 and see the result. Click on the Button to go forward to the next page.



6. We can use the Back button to return to the previous page. Additionally, if we want to go one page backwards using custom button, we can do so by typing the following code into an event handler

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    NavigationService.GoBack();
}
```

In the next part, we will see how to do a parameter passing while navigating between pages.

PASSING PARAMETERS BETWEEN PAGES

1. Use the previously developed project. The scenario we will use in this exercise is passing a string typed into the TextBox.
2. Type the following code into the Button's event handler

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //Hello World
    // PageTitle.Text = "Hello " + textBox1.Text;
    NavigationService.Navigate(new
Uri("/SecondPage.xaml?msg="+textBox1.Text,UriKind.Relative));
}
```

3. On the second page, we will try to retrieve the sent string and show it on the page title. Type the following code to do so:

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    string msg = "";

    if (NavigationContext.QueryString.TryGetValue("msg", out msg))

        PageTitle.Text = msg;
}
```

4. Press F5 and see the result. Type any string into TextBox then press Button. The title on the second page will change according to the input text.



PIVOT AND PANORAMA

Navigation is an aspect that can be found very open for exploration, especially to build a better user experience; and Microsoft seems to be concerned about the topic. In the latest release, developers have the advantage with two complete look and feel experiences available, along with the built-in controls and navigations. Let us get to know Pivot and Panorama.

PANORAMA

Panorama is designed to fit into the device's main screen limitations. The panorama application offers a unique way to show controls, data, and services on a horizontal canvas which size extends beyond the device's display. The dynamic view uses layer animations that give fun parallax effects. Panorama can be used for application with non task-oriented page browsing and hub with a lot of information.



FIGURE 2 PANORAMA VIEW [5]

Panorama view supports touch interaction for its navigations. We don't need to re-implement. Among the supported interactions are:

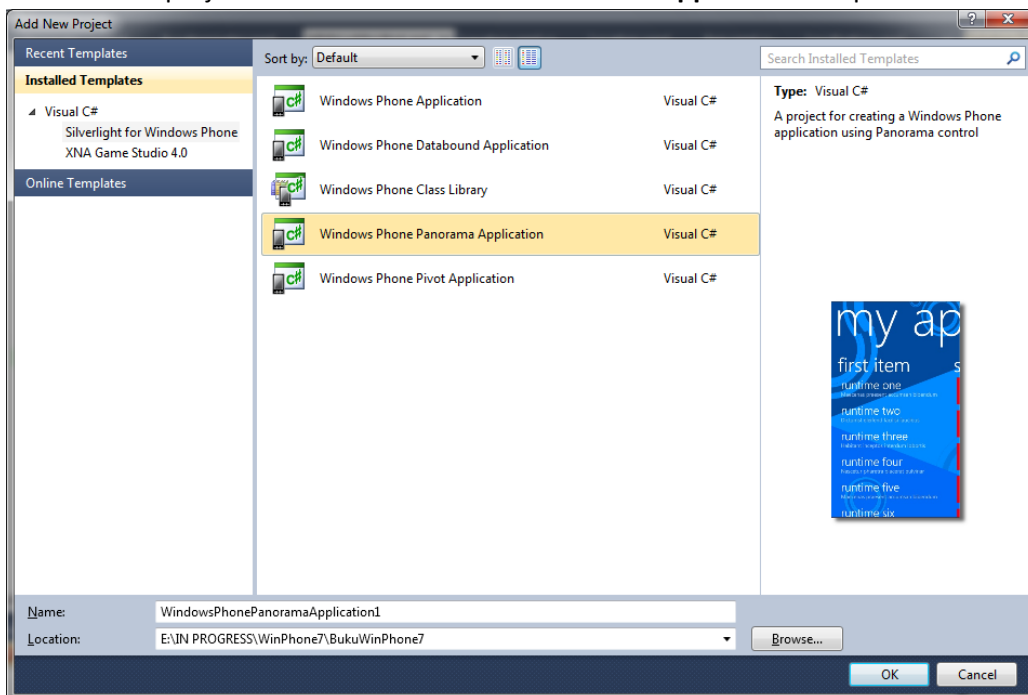
1. Horizontal pan (press and drag left or right)
2. Horizontal flick (press and slide quickly left or right)
3. Navigation hosted controls

Several guidelines to develop application using Panorama:

- Make sure to limit the number of sections used to a maximum of 4 sections. If the contents are too tight or a lot of the application's sections use hosted controls, then use less than 4 sections.
- Hide the parts where data is nonexistent.
- Sections can be extended beyond the display width by using Horizontal property.
- Use suitable background color or a wide picture background all through the Panorama control.
- The recommended size is 2000px (width) x 800px (height)
- Avoid drop-shadow effects.
- Make sure that title does not depend on the background.
- Avoid animations on Panorama titles.

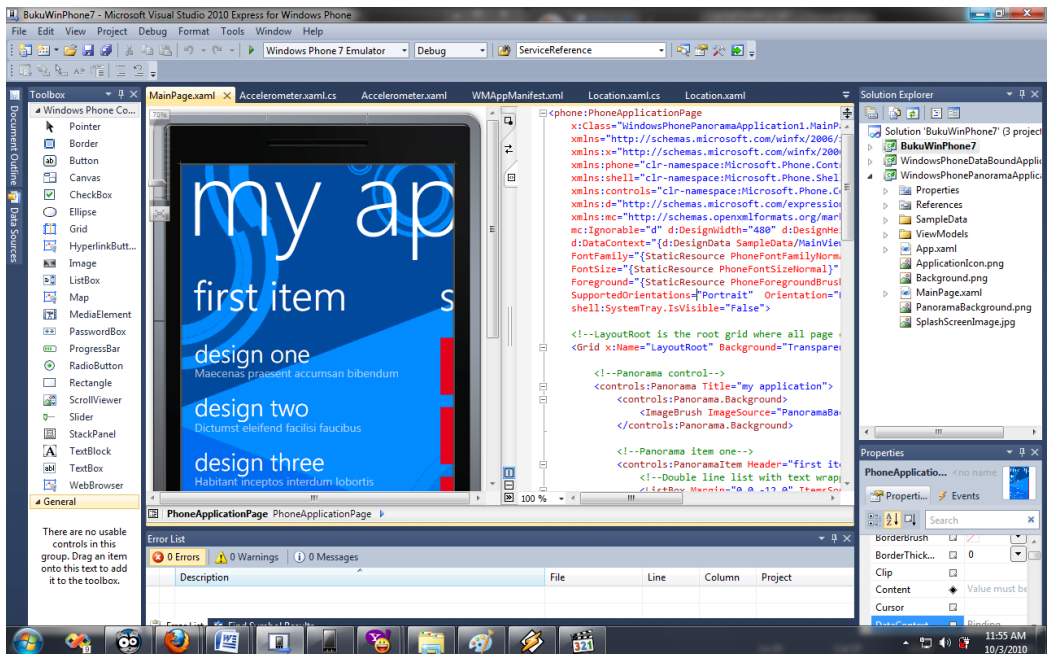
Now let's practice making an application using Panorama view on Windows Phone.

1. Create a new project from **Windows Phone Panorama Application** template.



Note:

If you want to continue from the previous exercises, right click on the project and select **Add Windows Phone Panorama Page**. Or if you want to add Panorama control on existing page, drag and drop from the toolbox to your page.



2. We will only use the template that is generated automatically while creating the application. Press F5 and see the results. Do a flick to slide to next section.



3. Now let's review the codes to understand application structures of a Panorama view.

```

<!--Panorama control-->
<controls:Panorama Title="my application">
<controls:Panorama.Background>
  <ImageBrush ImageSource="PanoramaBackground.png"/>
</controls:Panorama.Background>

<!--Panorama item one-->
<controls:PanoramaItem Header="first item">
  <!--Double line list with text wrapping-->
  <ListBox Margin="0,0,-12,0" ItemsSource="{Binding Items}">

```



```

        <ListBox.ItemTemplate>
            <DataTemplate>
                <StackPanel Margin="0,0,0,17" Width="432">
                    <TextBlock Text="{Binding LineOne}"
TextWrapping="Wrap" Style="{StaticResource PhoneTextExtraLargeStyle}"/>
                    <TextBlock Text="{Binding LineTwo}"
TextWrapping="Wrap" Margin="12,-6,12,0" Style="{StaticResource
PhoneTextSubtleStyle}"/>
                </StackPanel>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
</controls:PanoramaItem>
.....
<!--Panorama item two-->
<!--Use 'Orientation="Horizontal"' to enable a panel that lays out
horizontally-->
</controls:Panorama>

```

The main control is Panorama, which consists of a number of **PanoramaItem**. Consider **PanoramaItem** as a layout, like canvas or grid, where we can put any control in it. To add an item, all we have to do is add a panorama item within the Panorama control's range.

4. Add another PanoramaItem. We do the following in XAML code, adding the panorama item count on the application to 3 items.

```

<controls:PanoramaItem Header="third item">
    <ListBox FontSize="{StaticResource PhoneFontSizeLarge}">
        <sys:String>This</sys:String>
        <sys:String>item</sys:String>
        <sys:String>has</sys:String>
        <sys:String>a</sys:String>
        <sys:String>short</sys:String>
        <sys:String>list</sys:String>
        <sys:String>of</sys:String>
        <sys:String>strings</sys:String>
        <sys:String>that</sys:String>
        <sys:String>you</sys:String>
        <sys:String>can</sys:String>
        <sys:String>scroll</sys:String>
        <sys:String>up</sys:String>
        <sys:String>and</sys:String>
        <sys:String>down</sys:String>
        <sys:String>and</sys:String>
        <sys:String>back</sys:String>
        <sys:String>again.</sys:String>
    </ListBox>
</controls:PanoramaItem>

```

Don't forget to add this reference:

```
xmlns:sys="clr-namespace:System;assembly=mscorlib"
```

5. Press F5 and see the result. Adding items on Panorama is an easy matter because the project template has already given the big picture as how to use the control.



PIVOT

Pivot is designed to show a number of data and enable selections, and view items based on a certain category. Pivot is used to manage application views that have several layouts or pages with built-in navigations that support, such as:

1. Horizontal pan (press and drag left or right)
2. Horizontal flick (press and slide quickly left or right)
3. Navigation hosted controls

A sample of pivot implementation is the following figure:

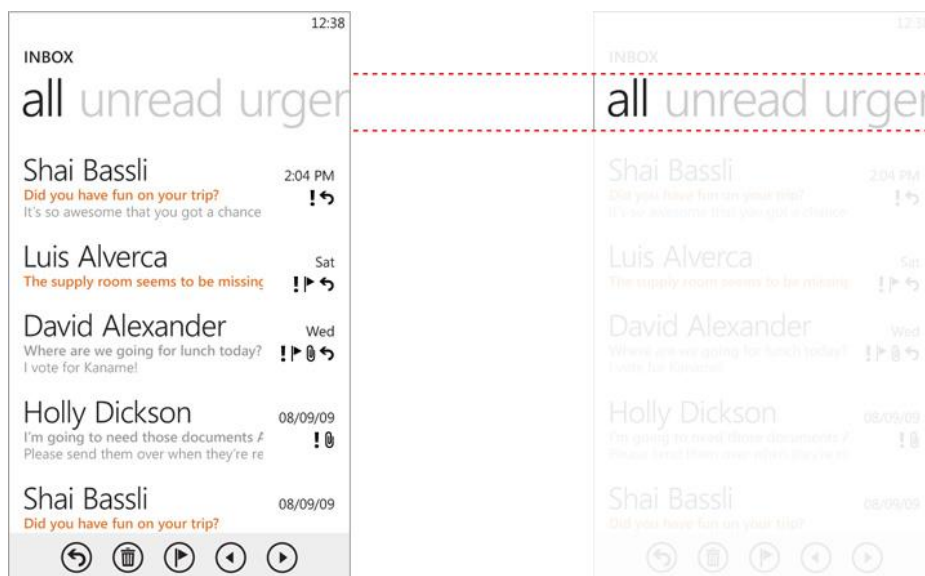


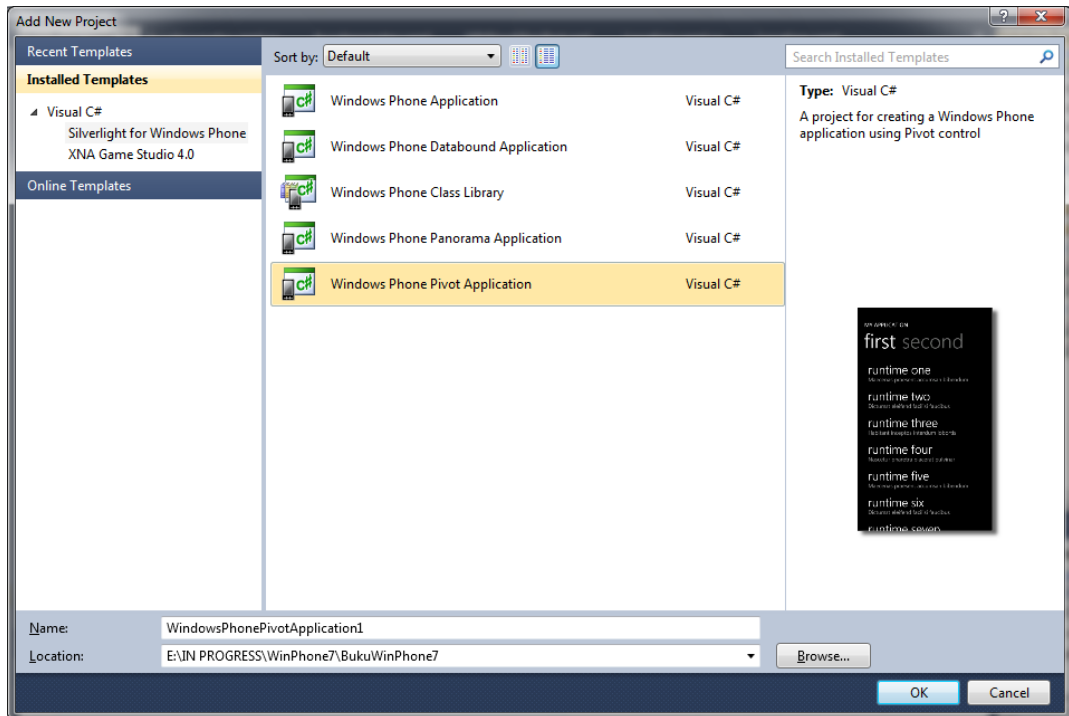
FIGURE 3 PIVOT VIEW [6]

Some best practices that can be put into account in developing application using Pivot controls are:

- Reduce the number of pages in Pivot control for performance reasons.
- Boost application performance by displaying data on-demand as opposed to loading all of them at once in the beginning.
- Make sure that each item displayed to users is of the same type.
- Pivot control should only be used if it suits the desired user experience.

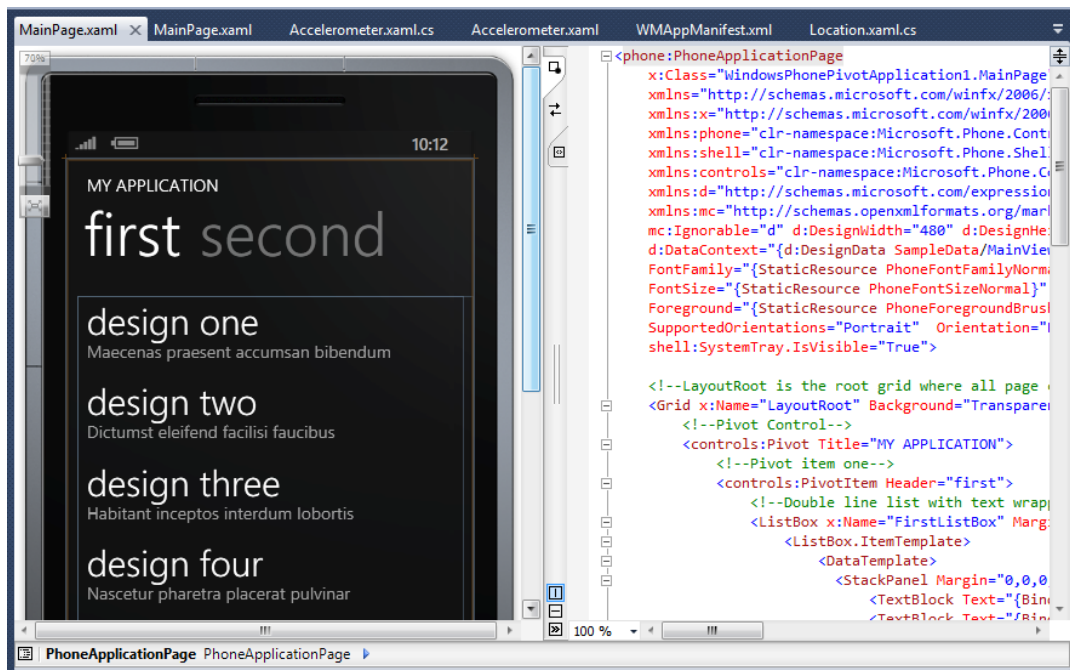
Now let us learn how to use Pivot control in an application.

1. Create a new **Windows Phone Pivot Application**.

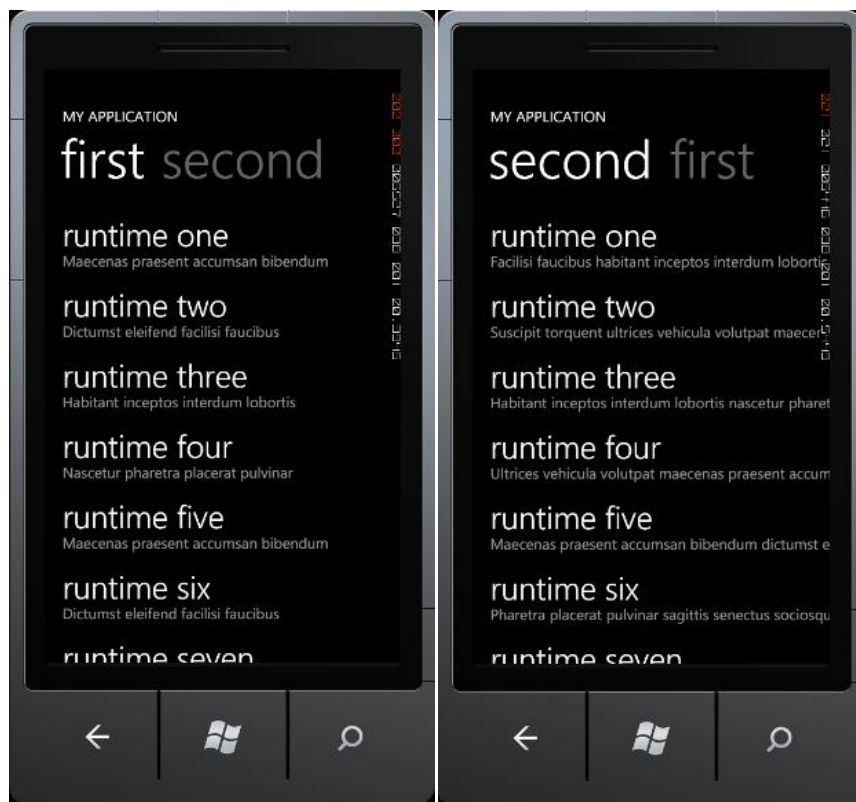


Note:

To continue from the previous exercise, right click on the project and select **Add Windows Phone Pivot Page**. Or if you want to add Pivot control on existing page, drag and drop from the toolbox to your page.



2. We will only use the template that is generated automatically while creating the application. Press F5 and see the results.



3. Now let's review the code in the main page. This is what it looks like:

```

<!--Pivot Control-->
    <controls:Pivot Title="MY APPLICATION">
        <!--Pivot item one-->
        <controls:PivotItem Header="first">

```

```

        <!--Double line list with text wrapping-->
        <ListBox x:Name="FirstListBox" Margin="0,0,-12,0"
ItemsSource="{Binding Items}">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Margin="0,0,0,17" Width="432">
                        <TextBlock Text="{Binding LineOne}"
TextWrapping="Wrap" Style="{StaticResource PhoneTextExtraLargeStyle}"/>
                        <TextBlock Text="{Binding LineTwo}"
TextWrapping="Wrap" Margin="12,-6,12,0" Style="{StaticResource
PhoneTextSubtleStyle}"/>
                    </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </controls:PivotItem>
    ...
    <!--Pivot item two-->
    ....
</controls:Pivot>
</Grid>

```

Examine that it is basically similar to Panorama View's schematics. In Pivot control there is one main container which includes several **PivotItem**. PivotItem can be used as containers equal to other containers such as grid or canvas in which we can place other controls.

4. To add a new item, here is the example:

```

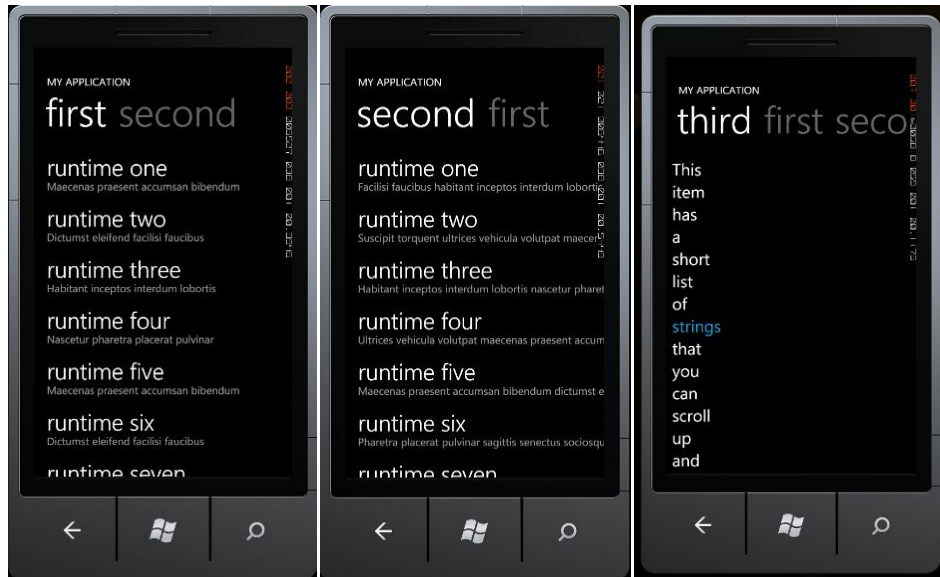
<controls:PivotItem Header="third">>
    <Grid>
        <ListBox FontSize="{StaticResource PhoneFontSizeLarge}">
            <sys:String>This</sys:String>
            <sys:String>item</sys:String>
            <sys:String>has</sys:String>
            <sys:String>a</sys:String>
            <sys:String>short</sys:String>
            <sys:String>list</sys:String>
            <sys:String>of</sys:String>
            <sys:String>strings</sys:String>
            <sys:String>that</sys:String>
            <sys:String>you</sys:String>
            <sys:String>can</sys:String>
            <sys:String>scroll</sys:String>
            <sys:String>up</sys:String>
            <sys:String>and</sys:String>
            <sys:String>down</sys:String>
            <sys:String>and</sys:String>
            <sys:String>back</sys:String>
            <sys:String>again.</sys:String>
        </ListBox>
    </Grid>
</controls:PivotItem>

```

Don't forget to add the following reference in the class declaration:

```
xmlns:sys="clr-namespace:System;assembly=mscorlib"
```

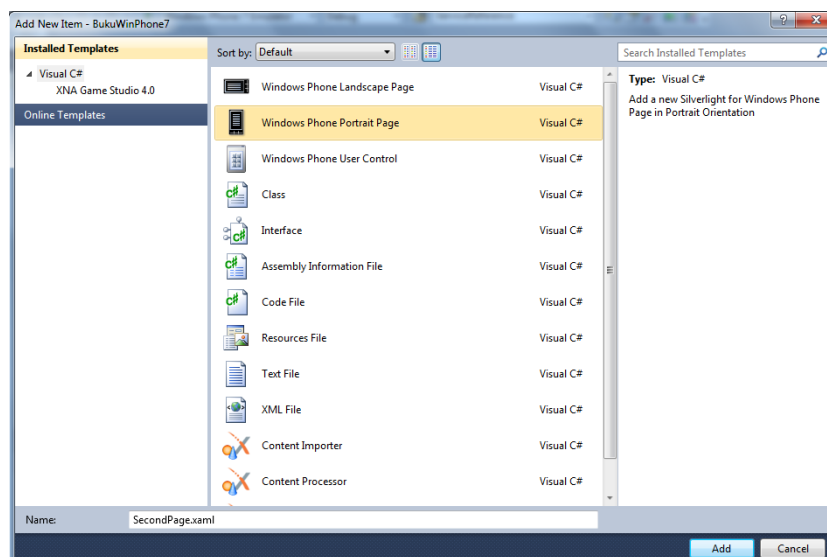
5. Press F5 for results.



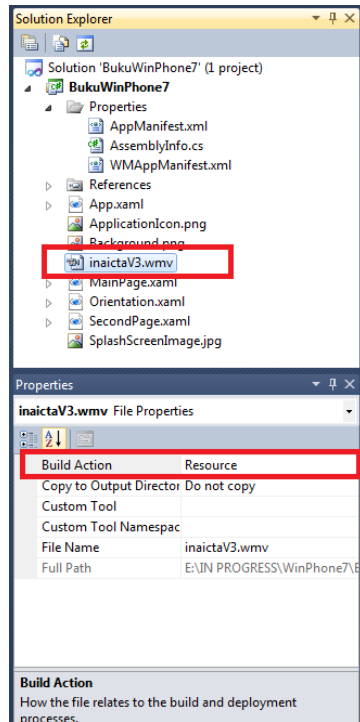
DEALING WITH PAGE ORIENTATIONS

In this part, we will learn how to handle switching between page orientations based on the device's position. There are two types of page orientation: portrait or landscape. Let us follow the steps below:

1. Create a new project or use any previously created project. Insert a new page; on **Solution Explorer** select **Add**, then **New Item**.
2. Select **Windows Phone Portrait Page** and rename the file to your liking, in this example, **Orientation.xaml**, then select **Add**.



3. Insert a video file into your project and set its Build Action property as "Resource".



4. Add a **MediaElement** control and place it inside content grid. Then set the source property to refer to the previously added video file.

```

<phone:PhoneApplicationPage
    x:Class="BukuWinPhone7.Orientation"
    ..... >

    <!--LayoutRoot contains the root grid where all other page content is
    placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*/>
        </Grid.RowDefinitions>

        <!--TitlePanel contains the name of the application and page title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="24,24,0,12">
            <TextBlock x:Name="ApplicationTitle" Text="MY APPLICATION"
            Style="{StaticResource PhoneTextNormalStyle}"/>
            <TextBlock x:Name="PageTitle" Text="page name" Margin="-3,-
            8,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
        </StackPanel>

        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentGrid" Grid.Row="1">
            <MediaElement Stretch="UniformToFill" Source="inaictaV3.wmv"/>
        </Grid>
    </Grid>
</phone:PhoneApplicationPage>

```

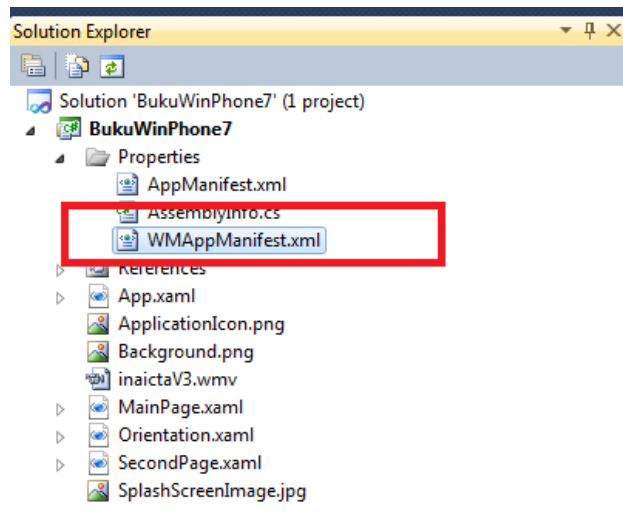
- In order to make the page support both portrait and landscape orientation, insert the following code into the page's PhoneApplicationPage definition.

```

<phone:PhoneApplicationPage
    x:Class="BukuWinPhone7.Orientation"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-
namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
    SupportedOrientations="PortraitOrLandscape" Orientation="Portrait"
    mc:Ignorable="d" d:DesignHeight="768" d:DesignWidth="480"
    shell:SystemTray.IsVisible="True">

```

- Now set **Orientation.xaml** as the application's initial page by changing the property of WMApManifest manifest file.



On the Task section, change the value of NavigationPage into **Orientation.xaml**

```

</Capabilities>
<Tasks>
    <DefaultTask Name="_default" NavigationPage="Orientation.xaml"/>
</Tasks>
<Tokens>

```

- Then on the **Orientation.xaml** page source code, add an event handler to handle the page's orientation switch. Type in the code below:

```

public Orientation()
{
    InitializeComponent();
}

```



```

        this.OrientationChanged += new
EventHandler<OrientationChangedEventArgs>(Orientation_OrientationChanged);
    }

    void Orientation_OrientationChanged(object sender, OrientationChangedEventArgs
e)
    {
        if (e.Orientation == PageOrientation.Landscape ||
            e.Orientation == PageOrientation.LandscapeLeft ||
            e.Orientation == PageOrientation.LandscapeRight)
        {
            TitlePanel.Visibility = System.Windows.Visibility.Collapsed;
            ContentGrid.SetValue(Grid.RowSpanProperty, 2);
            ContentGrid.SetValue(Grid.RowProperty, 0);
        }
        else
        {
            TitlePanel.Visibility = System.Windows.Visibility.Visible;
            ContentGrid.SetValue(Grid.RowSpanProperty, 1);
            ContentGrid.SetValue(Grid.RowProperty, 1);
        }
    }
}

```

8. Press F5 and change the emulator's orientation by pressing the orientation button on the top right corner of the emulator. When the orientation changes to landscape, the video will be shown in full screen mode and the page title will disappear.



The handling of page orientation alteration can increase the Windows Phone application's user experience. It is of course your option to develop an application that can adapt to the way users hold their devices.

APPLICATION BAR

Application bar is a control system that can be used to build a toolbar on a Windows Phone application. Application bar can be considered as the main option to develop a fast and consistent navigation. There are two types of application bar that we can use, icon button based and text menu based. Both types can also be combined. Icon bars are usually used for main, frequently used activities. Application bar can be defined for the whole application (global) or only on certain pages (local).

According to best practices written in MSDN, there are a couple things to be considered:

- If a task can be represented clearly using an icon, then use icon button. Otherwise, use text menu.
- Use application bar to make sure system menus are consistent with the user experience in every applications on the device.
- The recommended opacity are 0 (not shown, content page fills the display screen), 0.5, and 1 (shows on the screen).

To use icon button, things to be considered are:

- Use images with white foreground and alpha channel transparency.
- No need to manually add a circle on the icon border, because it is automatically added.
- Use a 48 x 48 image with a main icon sized 26 x 26 placed in the center of it.
- Do not use icon button to navigate back, because the hardware has already provided it.
- Use icon buttons for important tasks in the application.
- Icon samples can be downloaded here: [Microsoft Download Center](#)
- Avoid using more than 5 icon buttons.

GLOBAL APPLICATION BAR

If you want an application with many different pages (XAML files) that has only one application bar for all or most pages, then global application bar is the perfect choice. To add a global application bar, what we have to do is add the application bar's definition in **App.xaml**. Don't forget to add a unique key in resource application bar so that it can be used in other XAML files.

Now let's start making our first application bar.

1. You can continue from the previous projects or make a new one. In this example I will use a previously made project.
2. Open the **App.xaml** file, add the following code:

```
<Application
  x:Class="BukuWinPhone7.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
  xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone">
```

```

<!--Application Resources-->
<Application.Resources>

    <shell:ApplicationBar x:Name="globalApplicationBar"
x:Key="globalApplicationBar" Opacity="0.7">
        <shell:ApplicationBar.MenuItems>
            <shell:ApplicationBarMenuItem Text="Home" />
            <shell:ApplicationBarMenuItem Text="Help" />
            <shell:ApplicationBarMenuItem Text="About" />
        </shell:ApplicationBar.MenuItems>
    </shell:ApplicationBar>

</Application.Resources>

```

In the above example, we will add three menus on the application bar. Next, the pages which will use the application bar can refer to the previously defined key.

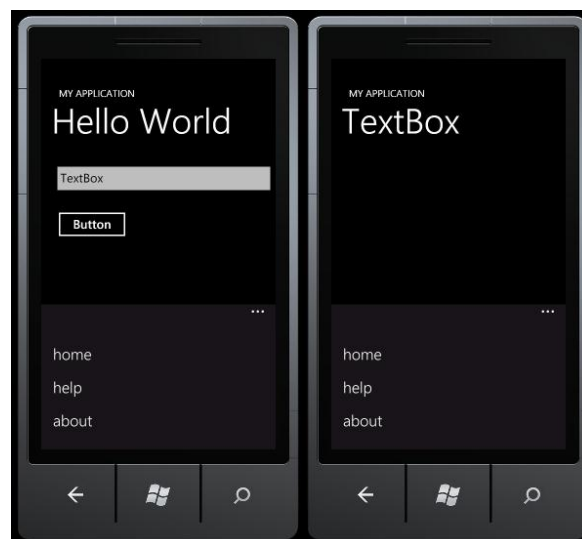
3. Open the pages to which the application bar will be added, and type the code below on each page:

```

<phone:PhoneApplicationPage
    x:Class="BukuWinPhone7.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-
namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    .....
    SupportedOrientations="Portrait" Orientation="Portrait"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    shell:SystemTray.IsVisible="True"
    ApplicationBar="{StaticResource globalApplicationBar}">

```

4. Press F5 for results.



For the pages in the application (in this example, **MainPage.xaml** and **SecondPage.xaml**) to which the application bar's definition has been added, a menu list will be visible on the bottom of the main screen. The next section will explain the use of application bar in only one certain page.

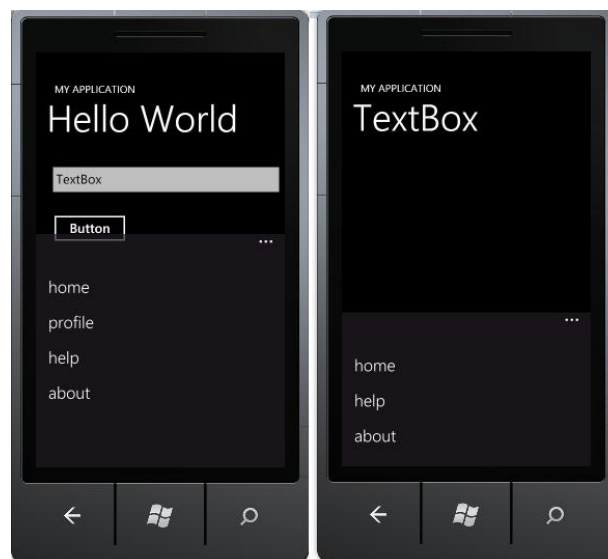
LOCAL APPLICATION BAR

Creating a local application bar can be done in two ways: using codes or XAML. Make sure that the XAML page doesn't already have a global application bar declaration. In this example we will use **MainPage.xaml** file again.

1. Open your XAML page and add the code below under the root container:

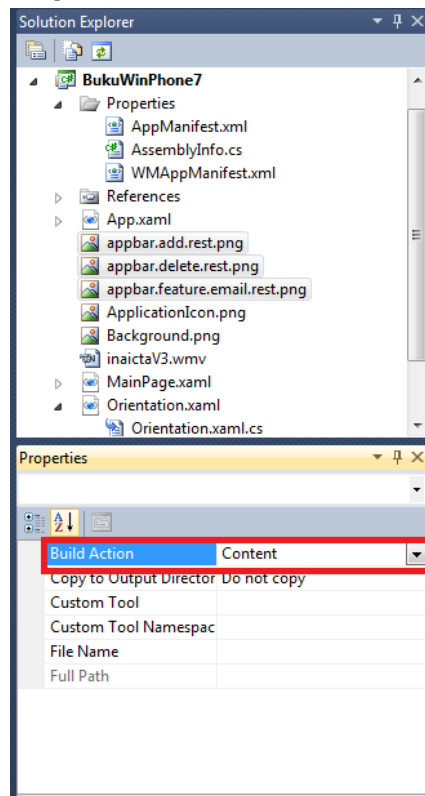
```
<phone:PhoneApplicationPage.ApplicationBar>
  <shell:ApplicationBar x:Name="globalApplicationBar"
x:Key="globalApplicationBar" Opacity="0.7">
  <shell:ApplicationBar.MenuItems>
    <shell:ApplicationBarMenuItem Text="Home" />
    <shell:ApplicationBarMenuItem Text="Profile" />
    <shell:ApplicationBarMenuItem Text="Help" />
    <shell:ApplicationBarMenuItem Text="About" />
  </shell:ApplicationBar.MenuItems>
</shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

2. Press F5 for results. In this example, you can see the difference between the application bar in **MainPage.xaml** and the one in **SecondPage.xaml**.



3. We used menu item in the previous example, now let's try using icon for our application bar. We should prepare the required icons before starting. Icons can be downloaded from [Microsoft Download Center](#).

4. Add a couple of icons into the project. Right click on the project, select **Add Existing Item**. Set *Build Action* property of the images to content.



5. Change the application bar codes into the following:

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar x:Name="globalApplicationBar"
IsMenuEnabled="True" Opacity="0.7">
        <shell:ApplicationBarIconButton Text="Add"
IconUri="appbar.add.rest.png"/>
        <shell:ApplicationBarIconButton Text="Message"
IconUri="appbar.feature.email.rest.png"/>
        <shell:ApplicationBarIconButton Text="Delete"
IconUri="appbar.delete.rest.png"/>
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

6. Press F5 and see the results. Now the application bar consisting of three buttons is ready to use.



LOCAL APPLICATION BAR (PROGRAMMATIC APPROACH)

In this section we will see how application bar can be created programmatically with codes instead of XAML file declaration. Follow the steps below:

1. Make sure that the page doesn't already have either global or local application bar defined.
2. Add a reference to the following *dll*
`using Microsoft.Phone.Shell;`
3. Add the following code in the code-behind file:

```
public MainPage()
{
    InitializeComponent();
    ApplicationBar appBar = new ApplicationBar();
    appBar.IsMenuEnabled = true;
    appBar.Buttons.Add(new ApplicationBarIconButton(){ Text="Add", IconUri=new
Uri("appbar.add.rest.png",UriKind.Relative)});
    appBar.Buttons.Add(new ApplicationBarIconButton() { Text = "Message",
IconUri = new Uri("appbar.feature.email.rest.png", UriKind.Relative) });
    appBar.Buttons.Add(new ApplicationBarIconButton() { Text = "Delete",
IconUri = new Uri("appbar.delete.rest.png", UriKind.Relative) });

    this.ApplicationBar = appBar;
}
```

4. Press F5 for results.



Exactly the same, aren't they? :)

INSERTING EVENT HANDLER

To implement functionality into application bar items, we need to assign a click event for each item (regardless icon based or menu based). This can be done by using XAML or code. We can add a click event on XAML by declaring a function inside the item's property.

For the exercise we've done in creating icon bar, we only have to add the following declaration:

```
<phone:PhoneApplicationPage.ApplicationBar>
  <shell:ApplicationBar x:Name="globalApplicationBar" IsMenuEnabled="True"
Opacity="0.7">
    <shell:ApplicationBarIconButton Text="Add"
Click="ApplicationBarIconButton_Click" IconUri="appbar.add.rest.png"/>
    <shell:ApplicationBarIconButton Text="Message"
IconUri="appbar.feature.email.rest.png"/>
    <shell:ApplicationBarIconButton Text="Delete"
IconUri="appbar.delete.rest.png"/>
  </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

Or if you want it done programmatically, here is the code:

```
...
    InitializeComponent();
    AppBar appBar = new AppBar();
    appBar.IsMenuEnabled = true;

    AppBarIconButton appIcon = new AppBarIconButton() { Text =
"Add", IconUri = new Uri("appbar.add.rest.png", UriKind.Relative) };
    appBar.Click += new EventHandler(AppBarIconButton_Click);

    appBar.Buttons.Add(appIcon);
    appBar.Buttons.Add(new AppBarIconButton() { Text = "Message",
IconUri = new Uri("appbar.feature.email.rest.png", UriKind.Relative) });
    appBar.Buttons.Add(new AppBarIconButton() { Text = "Delete",
IconUri = new Uri("appbar.delete.rest.png", UriKind.Relative) });

    this.AppBar = appBar;
....
```

After the declaration, add the desired functionality inside the method (in this example it's `AppBarIconButton_Click`). To give an illustration, the following code will show a message box on button clicked.

```
private void AppBarIconButton_Click(object sender, EventArgs e)
{
    //add your functionality
    MessageBox.Show((sender as AppBarIconButton).Text);
}
```

Press F5 for results. When icon button on application bar is pressed, a message box will appear with the icon text as its content.



WEB SERVICE CONSUMPTION

Web service has become a standard when it comes to using a predefined web function in our application. In this section we will learn how to consume web service on Windows Phone. Web services consumable by Windows Phone can be in the form of SOAP (built in WCF or other technologies), plain HTTP, or REST. Explanations regarding the said terms will not be discussed here; if you are not familiar with the terms you can look it up in your preferred search engine.

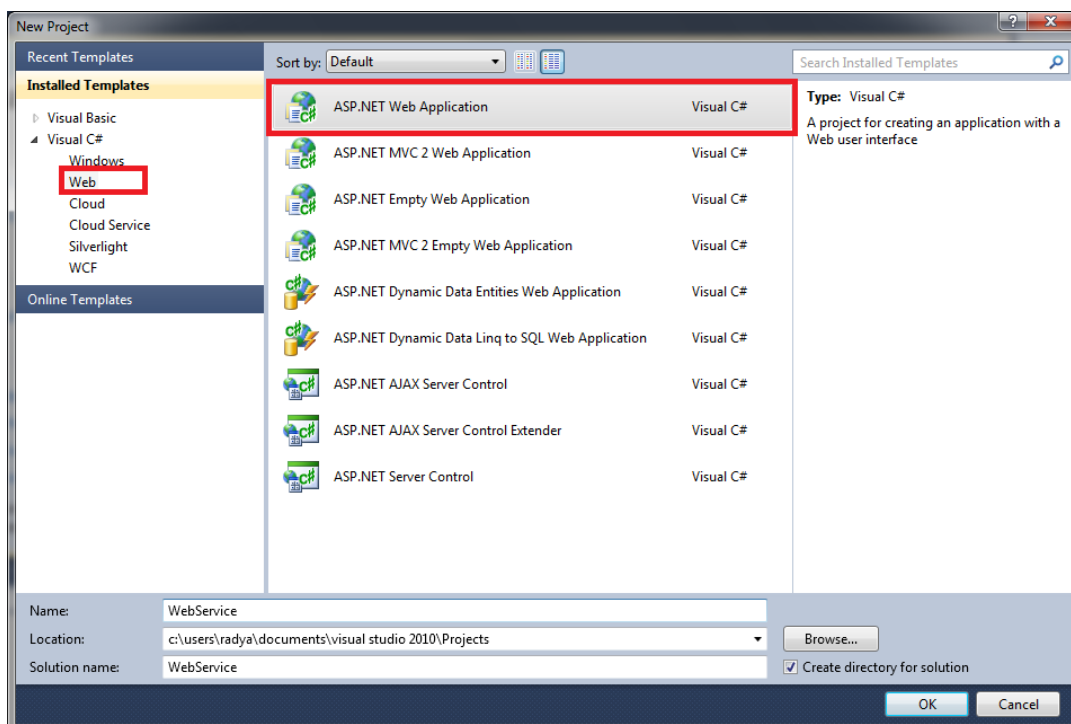
Windows Phone applications can access these web services either directly or through a proxy class that is automatically generated from metadata attached to a service. A service can be in a form of user defined service that you place in your server, or a third party server, for example Facebook, Twitter, and other services. Silverlight can work with many different data format, such as XML, JSON, RSS, or ATOM, and data access can be done under numerous scenarios, such as serialization, LINQ to XML, LINQ to JSON, or syndication. It has unlimited combinations, and can be implemented based on your need.

ACCESS VIA GENERATED CLASS

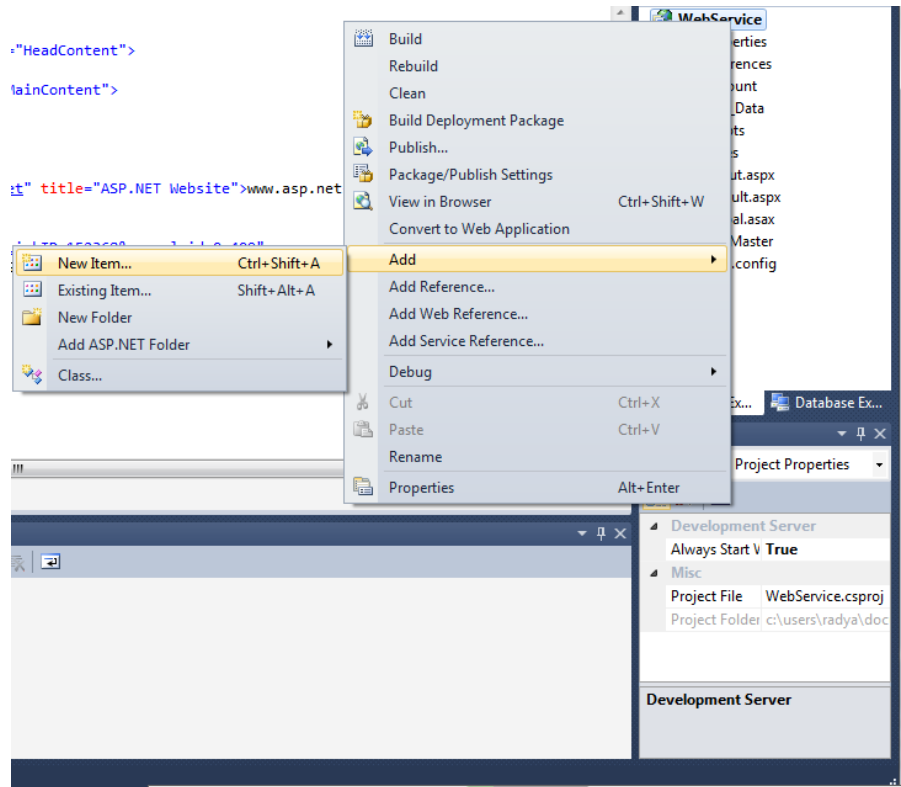
Accessing a service through a proxy class generated from metadata can improve development speed. This section will discuss an example of web service access using .NET technology with auto-generated proxy class.

CREATING WEB SERVICES

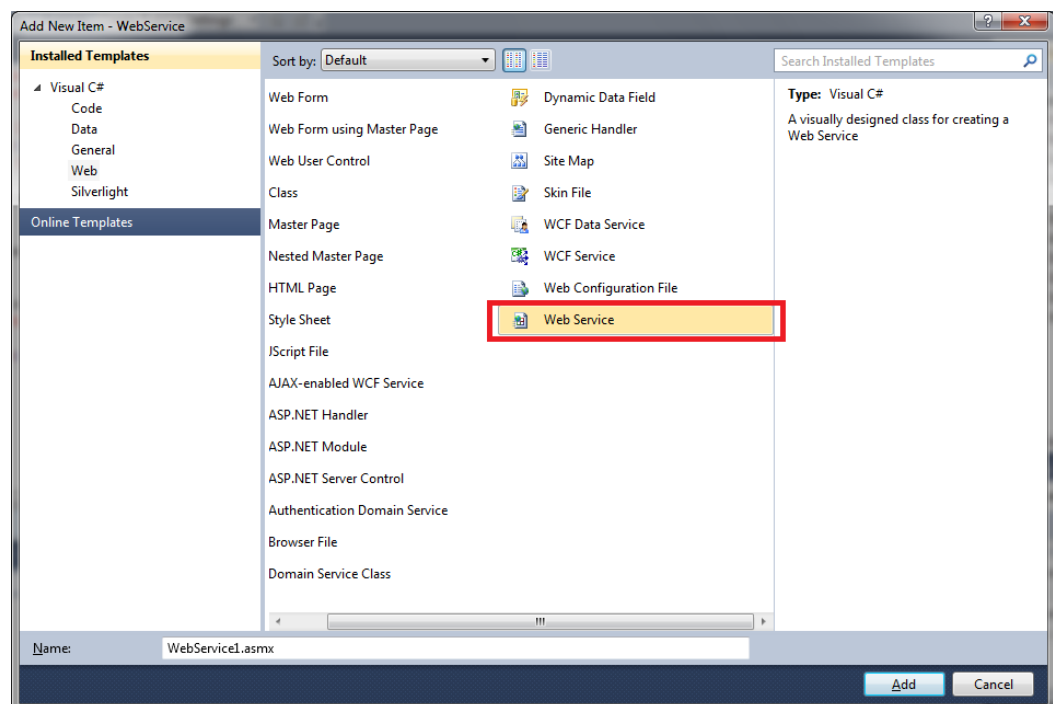
1. Run a Visual Studio program that support Web Application project creation (in this case I use Visual Studio 2010 Web Developer Express), then create a new Web ASP.NET Web Application project.



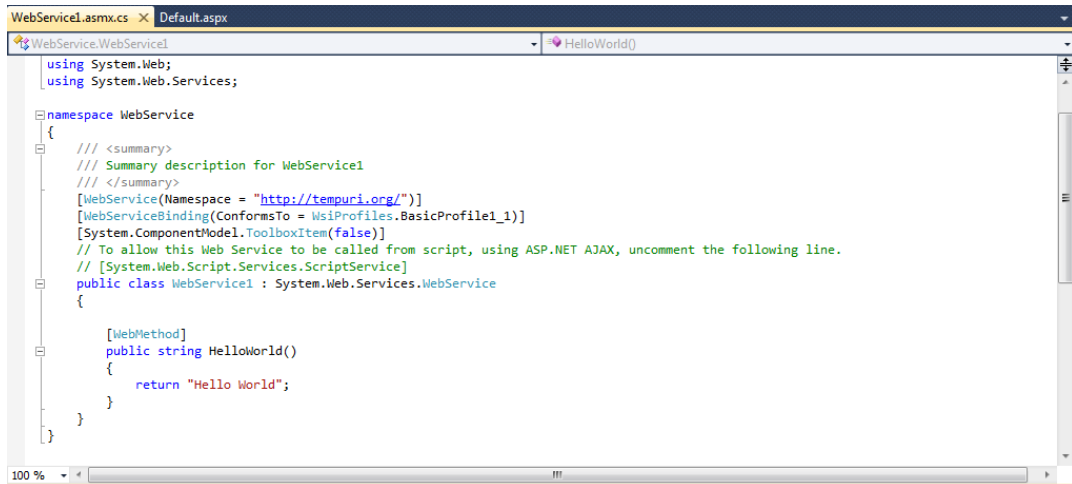
- On the Solution Explorer window, add a web service file by right clicking on Project, then select **Add > Add New Item**.



- Select **Web Service**, then click **Add**.



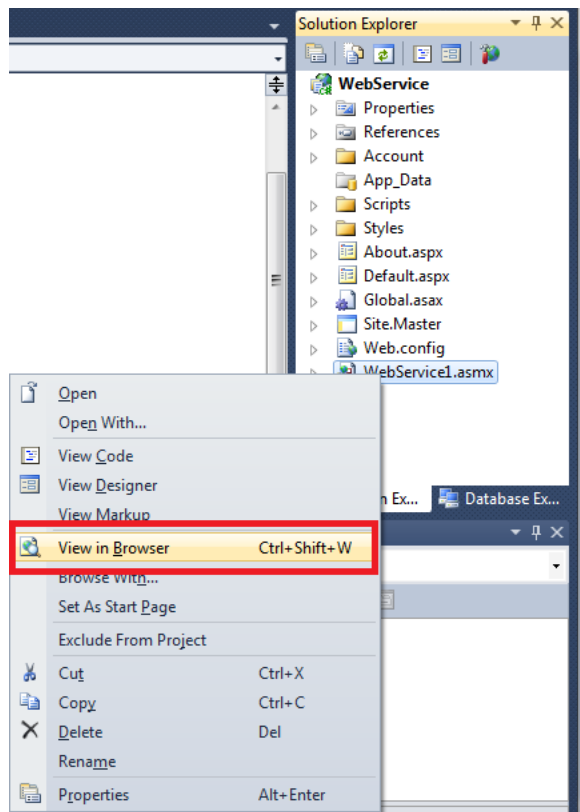
- We will only use the automatically generated function, which is a web service method that returns a “Hello World” string. Further explanations regarding web service will not be discussed, and we will focus to the Windows Phone access aspect.



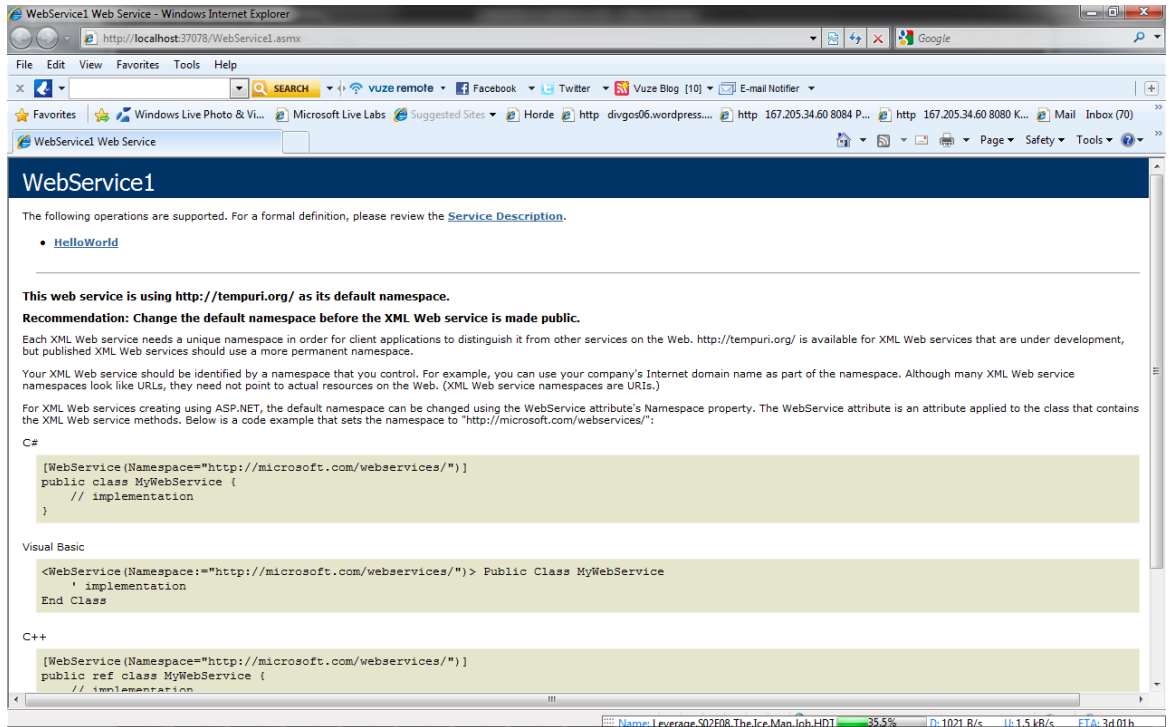
```
using System.Web;
using System.Web.Services;

namespace WebService
{
    /// <summary>
    /// Summary description for WebService1
    /// </summary>
    [WebService(Namespace = "http://tempuri.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    // To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the following line.
    // [System.Web.Script.Services.ScriptService]
    public class WebService1 : System.Web.Services.WebService
    {
        [WebMethod]
        public string HelloWorld()
        {
            return "Hello World";
        }
    }
}
```

- Right click on the web service file and select **View in Browser** to test the function.

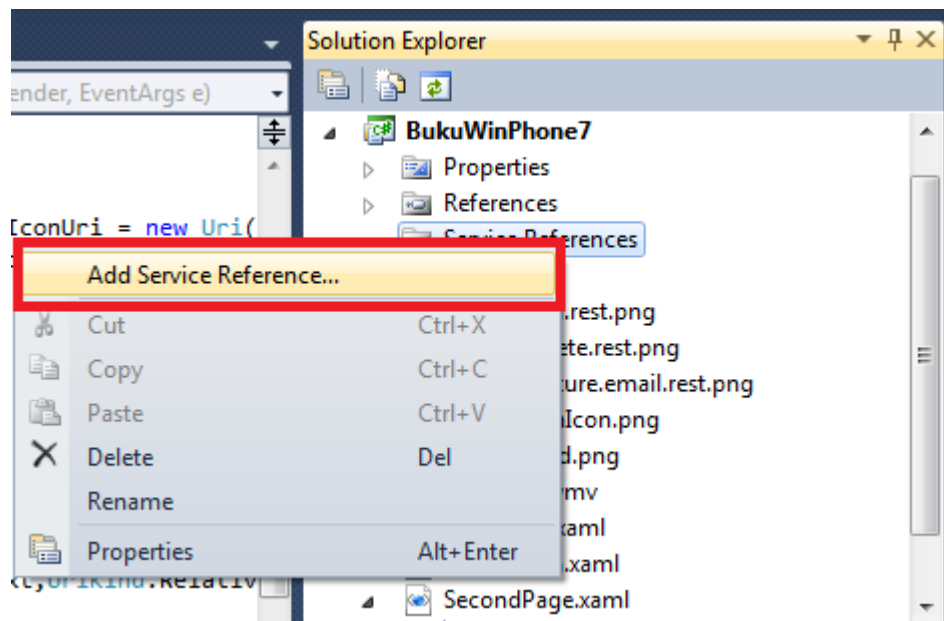


6. Your sample web service is ready for consumption. Do not turn off your browser.

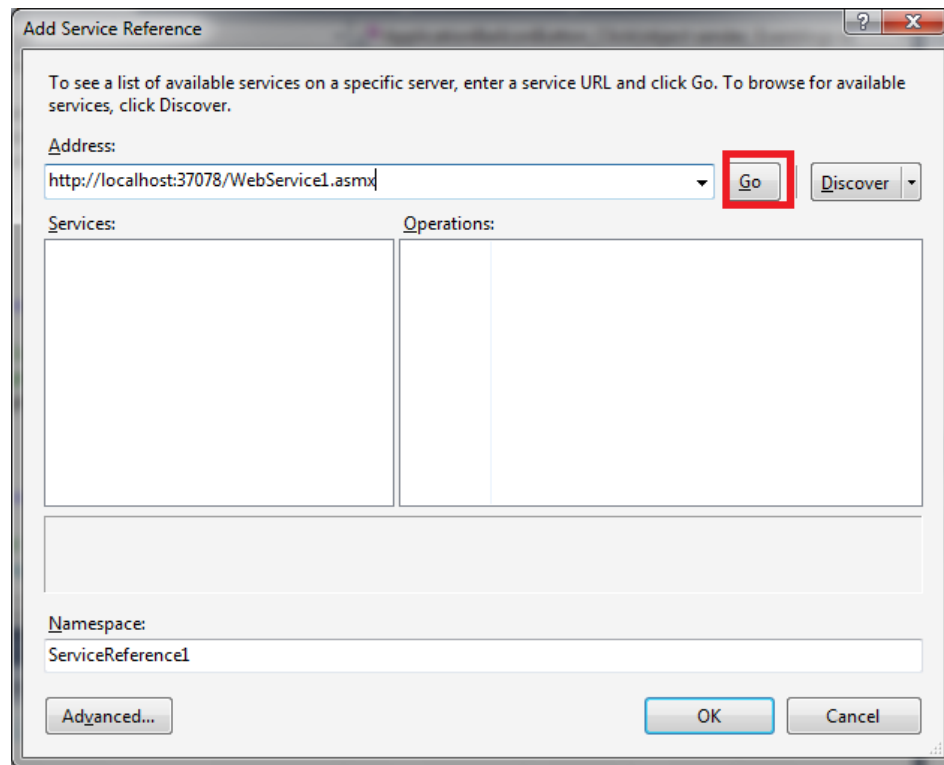


ADDING WEB SERVICE REFERENCE

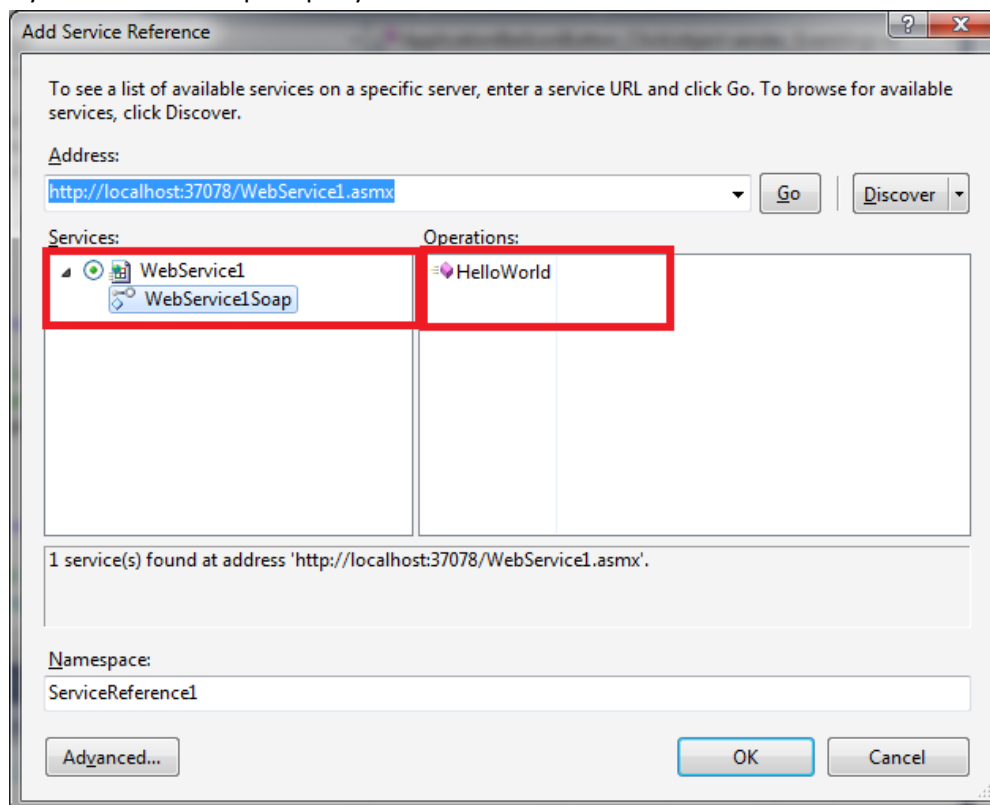
1. Open your Windows Phone project. On the Solution Explorer windows, right click on **References**, then select **Add Service Reference**.



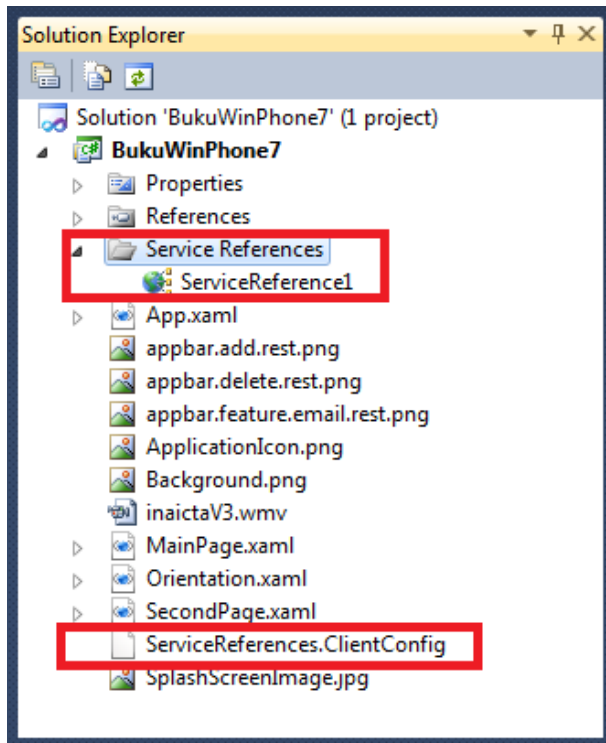
- Copy and paste the access address for the web service that we created. This address can be retrieved from the URL address on the browser. Click **Go**.



- If the web service is found and accessible, a list of services and available operations will be displayed. Give a namespace per your need then click **OK**.



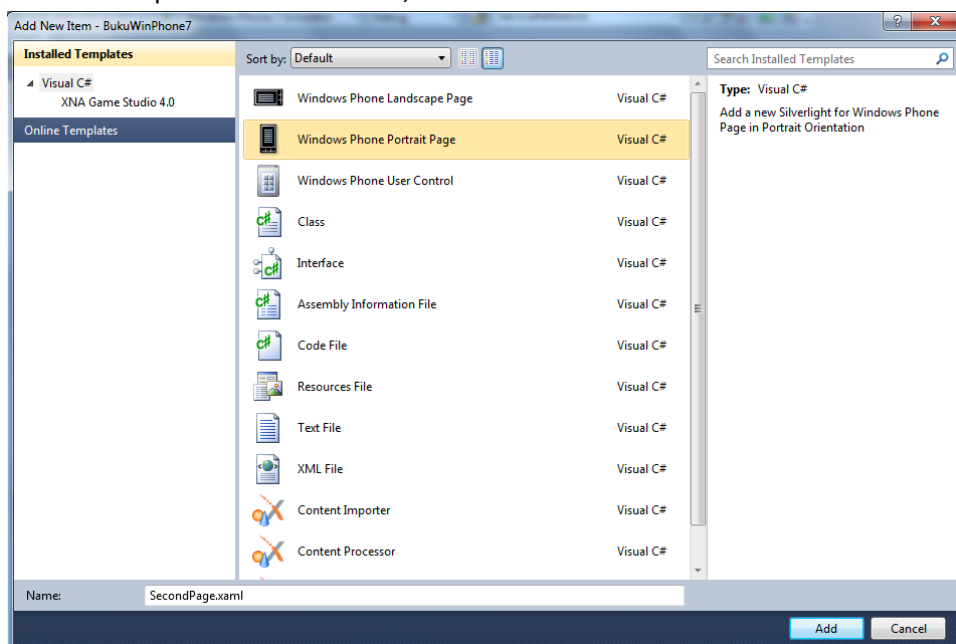
4. If the addition is successful, on the Solution Explorer window we can see a configuration file and a reference file for the web service we created.



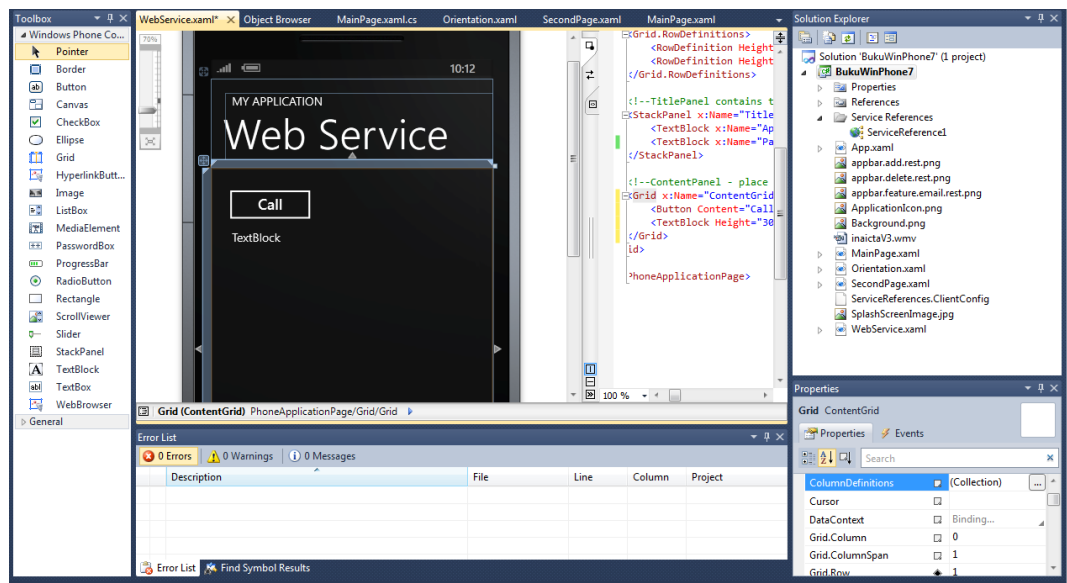
CONSUMING WEB SERVICE

To consume a web service, we will create a new page so that the **MainPage.xaml** will not be overcrowded.

1. Right click on the project, **Add New Item** and select **Windows Phone Portrait Page**. Rename the file as you wish, in this example **WebService.xaml**, then click **Add**.



2. Insert a **Button** and a **TextBox**. The scenario is that when the button is pressed, we will call for a web service via an auto-generated proxy class.



3. Double-click on the **button** to add *event handler*. Type in the following code:

```

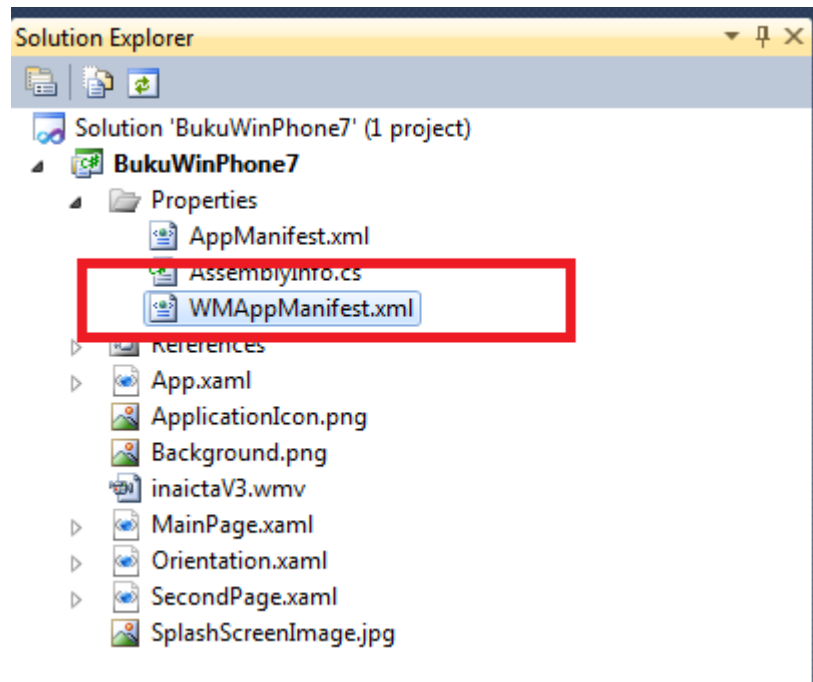
private void button1_Click(object sender, RoutedEventArgs e)
{
    ServiceReference1.WebService1SoapClient proxy = new
ServiceReference1.WebService1SoapClient();
    proxy.HelloWorldCompleted += new
EventHandler<ServiceReference1.HelloWorldCompletedEventArgs>(proxy_HelloWorldCompleted
);
    proxy.HelloWorldAsync();
}

void proxy_HelloWorldCompleted(object sender,
ServiceReference1.HelloWorldCompletedEventArgs e)
{
    if (e.Error == null)
    {
        textBlock1.Text = e.Result;
    }
}

```

First we initialize proxy from the service. Define handler when request is done, then call the function that will be used. On the above example, when the web service request is done, the content of the TextBox will become “HelloWorld”.

- Now set Orientation.xaml as the initial page for the application by changing the property on WMApManifest manifest file.



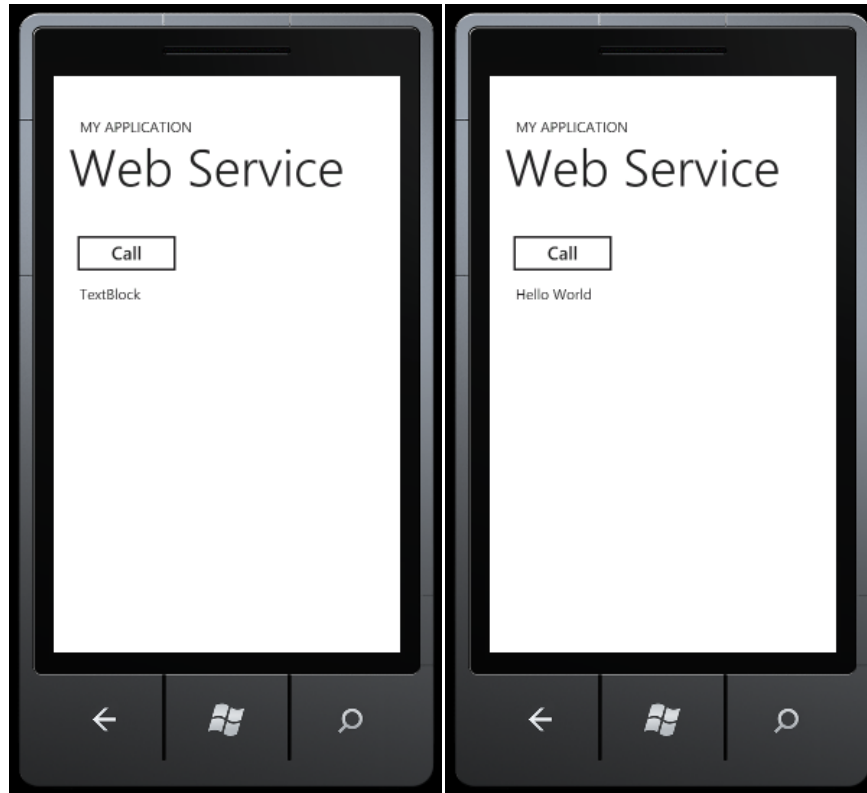
On Task section, change the value of NavigationPage into **WebService.xaml**

```
</Capabilities>  
<Tasks>  
  <DefaultTask Name = "_default" NavigationPage="WebService.xaml"/>  
</Tasks>  
<Tokens>
```

- Press F5 to see results.

Note:

Don't forget to set this page as the first page your application goes to if you continue from the previous project. Learn how to do so [here](#).



USING STANDARD HTTP REQUEST

In this section we will see how to consume a service in form of plain HTTP or RESTful. Silverlight and Windows Phone SDK have a System.Net standard package to send request through ftp and http. The scenario used in this case is to consume a real live service, which is service from a website that provides weather information.

This below is the URL for the service with <city_name> as input parameter.

<http://www.google.com/ig/api?weather=jakarta>

1. If you are still using the project from the previous section, let's add a little modification on the **WebService.xaml** file. If you are new to this, then create a new file by following [these](#) steps.
2. Insert the code below into the button's event handler

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    //ServiceReference1.WebService1SoapClient proxy = new
    ServiceReference1.WebService1SoapClient();
    //proxy.HelloWorldCompleted += new
    EventHandler<ServiceReference1.HelloWorldCompletedEventArgs>(proxy_HelloWorldCompleted
);
    //proxy.HelloWorldAsync();
    WebClient wc = new WebClient();
    wc.DownloadStringAsync(new
    Uri("http://www.google.com/ig/api?weather=jakarta"));
}
```

```

        wc.DownloadStringCompleted += new
DownloadStringCompletedEventHandler(wc_DownloadStringCompleted);
    }

    void wc_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
    {
        if (e.Error==null)
        {
            textBlock1.Text = e.Result;
        }
    }
}

```

There are two things done here. First, create a request by entering a URL address in WebClient class. Request will later be done asynchronously so that it will not disturb the application's responsiveness. The second part is what to do after the request is done. In the above example, the result retrieved from the request is written into a TextBlock.

3. Press F5 and see the results. Press button to call the service.



Download speed will be very dependent to your PC connection. The same thing applies in the real device. You can see that the service data can be retrieved successfully. Data can then be processed further before being displayed to users.

WORKING WITH DATA

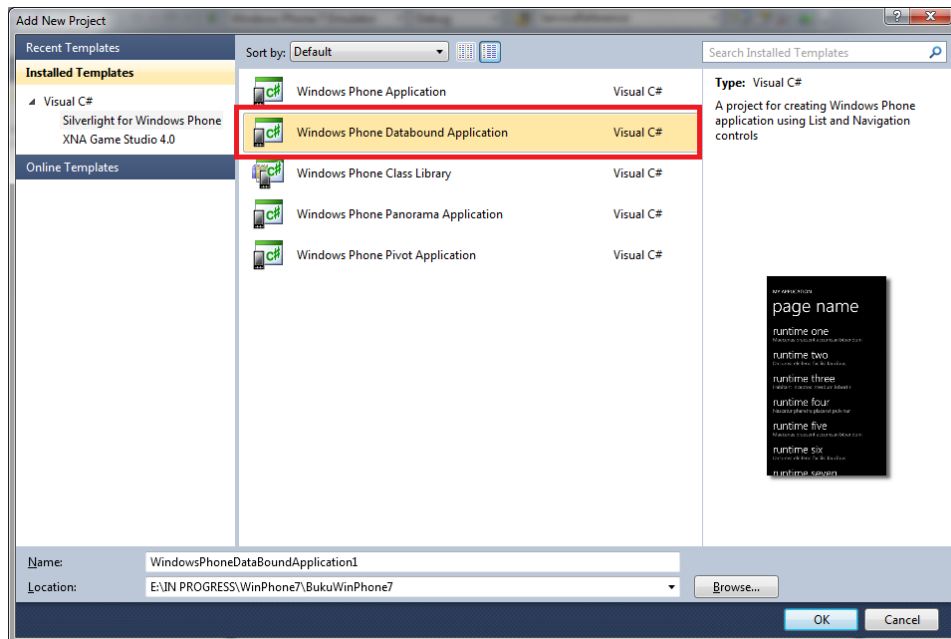
Windows Phone which uses Silverlight as a development platform surely inherits the beauty of interacting with data using DataBinding. Binding data to UI Control means that any alteration we do in UI will cause a change in the data bound to it and vice versa. DataBinding makes it easier for us to display data by trimming down handling on code-behind, thus making the code much simpler.

One scenario commonly used is working with a ListBox to show a number of data. The part highlighted in yellow in the box below shows XAML code that declares the binding of a ListBox to a collection named "items".

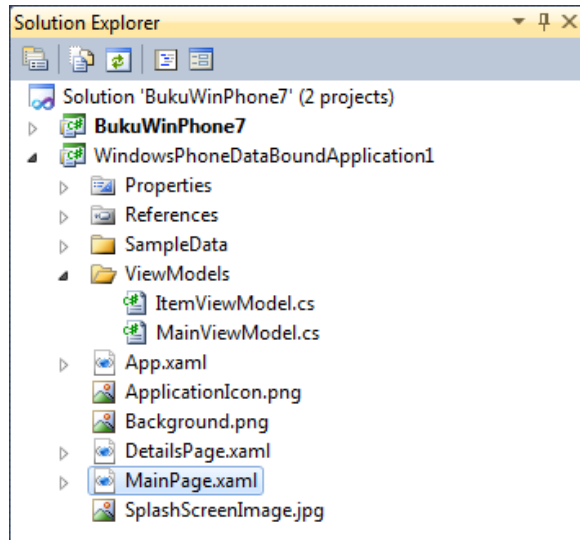
```
<ListBox x:Name="MainListBox" Margin="0,0,-12,0" ItemsSource="{Binding  
Items}">  
    ...  
    <TextBlock Text="{Binding LineOne}" />  
    ...  
</ListBox>
```

The part highlighted in green shows that the text value of *TextBlock* will refer to the property *LineOne*, regardless of its value, from the "Items" collection. To learn about it further, we will take a look at a standard template Windows Phone Tools has provided, Windows Phone Databound Application, which provides a comprehensible general illustration of databinding usage to display data collection.

1. Create a new Windows Phone Databound Application project. Right click on the solution on **Solution Explorer** window, select **Add New Project** and choose **Windows Phone Databound Application**.

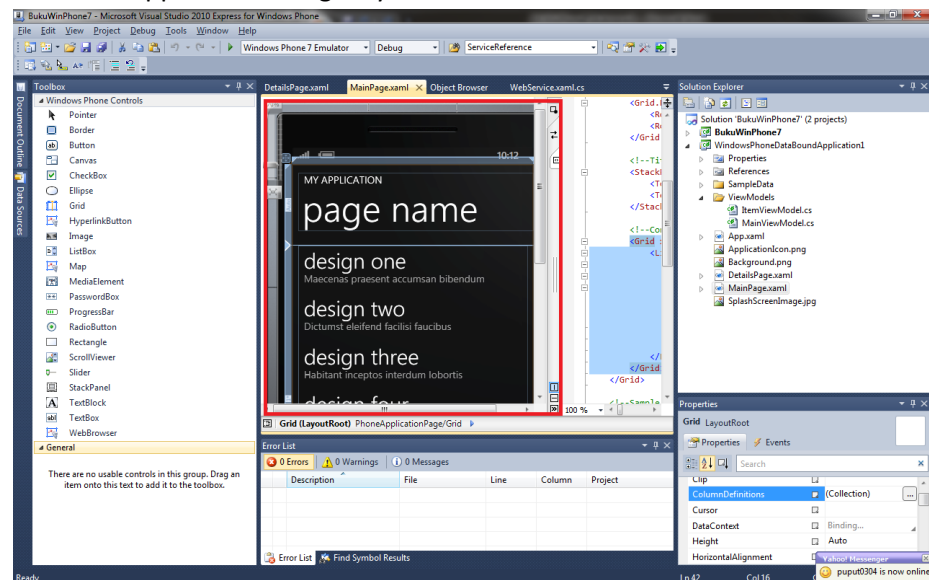


- Several files will be automatically created, and we will review them one by one.



Item	Description
SampleData/MainViewModelSample Data.xaml	This file consists of sample data which will be the input data during design process
ItemViewModel.cs	This file declares the view model for each item on the list
MainViewModel.cs	This file declares the main view for the application's main page
App.xaml	The main file, consists of resources usable throughout the application and events to handle application states
DetailsPage.xaml	This file displays detailed data for every item on the list
MainPage.xaml	This file displays list from data collections using ListBox control

- Now take notice on the application design layout.



The main page already has a layout containing item design one, design two, and so on. The data is retrieved from SampleData and displayed on design time with data context declaration on the page.

```
<phone:PhoneApplicationPage
  x:Class="WindowsPhoneDataBoundApplication1.MainPage"
  .....
  d:DataContext="{d:DesignData SampleData/MainViewModelSampleData.xaml}"
  .....>
```

4. Open ItemViewModel.cs file and let's observe it.

```
public class ItemViewModel : INotifyPropertyChanged
{
    private string _lineOne;
    /// <summary>
    /// Sample ViewModel property; this property is used in the view to display
    its value using a Binding.
    /// </summary>
    /// <returns></returns>
    public string LineOne
    {
        get
        {
            return _lineOne;
        }
        set
        {
            if (value != _lineOne)
            {
                _lineOne = value;
                NotifyPropertyChanged("LineOne");
            }
        }
    }
}

....

public event PropertyChangedEventHandler PropertyChanged;
private void NotifyPropertyChanged(String propertyName)
{
    PropertyChangedEventHandler handler = PropertyChanged;
    if (null != handler)
    {
        handler(this, new PropertyChangedEventArgs(propertyName));
    }
}
}
```

The class includes declarations for properties to be displayed, on the example above consists of 3 properties: a string, PropertyChanged property, and event handler. The last item is used to give notification when your data is changed.

5. Now open MainViewModel.cs file and observe the contents

```
public MainViewModel()
{
    this.Items = new ObservableCollection<ItemViewModel>();
}

/// <summary>
/// A collection for ItemViewModel objects.
/// </summary>
public ObservableCollection<ItemViewModel> Items { get; private set; }
```

We can see a declaration of a collection called Items, which consists of ItemVlewModel. This collection will later be bound with ListBox UI Control. Next we have Load data mechanism, in this part data is inserted into items. In reality, data source can be a service or a database.

```
public void LoadData()
{
    // Sample data; replace with real data
    this.Items.Add(new ItemViewModel() { LineOne = "runtime one", LineTwo =
"Maecenas praesent accumsan bibendum", LineThree = "Facilisi faucibus habitant
inceptos interdum lobortis nascetur pharetra placerat pulvinar sagittis senectus
sociosqu" });
    this.Items.Add(new ItemViewModel() { LineOne = "runtime two", LineTwo =
"Dictumst eleifend facilisi faucibus", LineThree = "Suscipit torquent ultrices
vehicula volutpat maecenas praesent accumsan bibendum dictumst eleifend facilisi
faucibus" });
    this.Items.Add(new ItemViewModel() { LineOne = "runtime three", LineTwo =
"Habitant inceptos interdum lobortis", LineThree = "Habitant inceptos interdum
lobortis nascetur pharetra placerat pulvinar sagittis senectus sociosqu suscipit
torquent" });
    .....
    this.IsDataLoaded = true;
}
```

6. Now let's open MainPage.xaml.cs file

```
// Constructor
public MainPage()
{
    InitializeComponent();

    // Set the data context of the ListBox control to the sample data
    DataContext = App.ViewModel;
    this.Loaded += new RoutedEventHandler(MainPage_Loaded);
}

// Handle selection changed on ListBox
private void MainListBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    // If selected index is -1 (no selection) do nothing
}
```

```

        if (MainListBox.SelectedIndex == -1)
            return;

        // Navigate to the new page
        NavigationService.Navigate(new Uri("/DetailsPage.xaml?selectedItem=" +
MainListBox.SelectedIndex, UriKind.Relative));

        // Reset selected index to -1 (no selection)
        MainListBox.SelectedIndex = -1;
    }

    // Load data for the ViewModel Items
    private void MainPage_Loaded(object sender, RoutedEventArgs e)
    {
        if (!App.ViewModel.IsDataLoaded)
        {
            App.ViewModel.LoadData();
        }
    }
}

```

The part highlighted in yellow is how you can set MainPage's DataContext with App.ViewModel that refers to an instance of MainViewModel in App.xaml.cs file. With that declaration, MainViewModel automatically becomes the data source on MainPage.xaml page. There is also a navigation mechanism in Section Changed event handler from ListBox. It means that if one item in the ListBox is clicked, it will navigate to DetailPage and display details of the clicked item. This navigation has been discussed in [Navigations on Windows Phone](#), along with passing parameters to target page. Now open MainPage.xaml file and observe the code in ListBox.

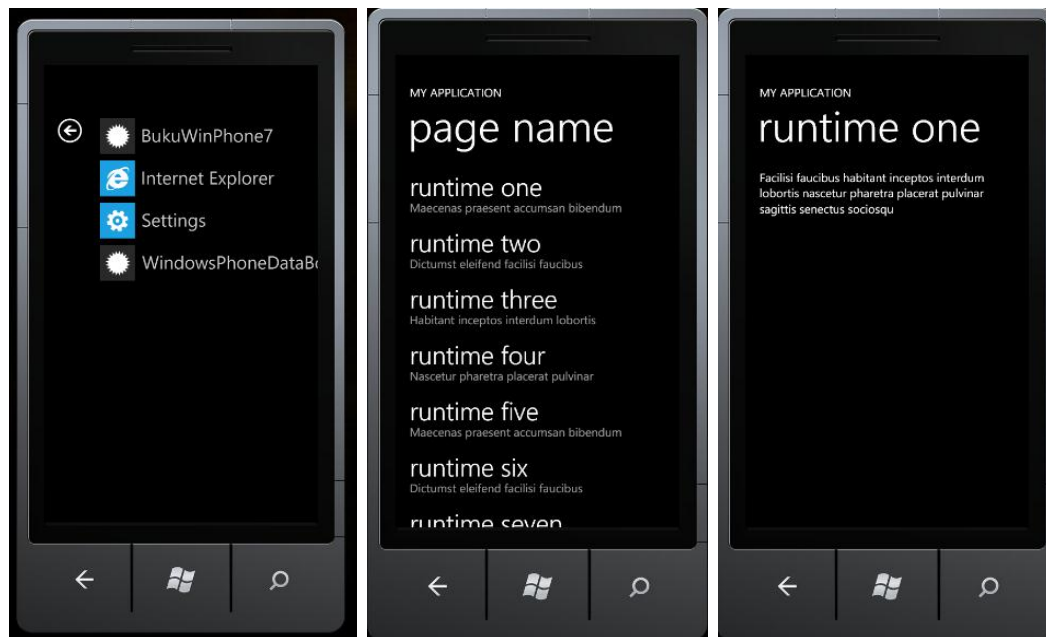
```

<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <ListBox x:Name="MainListBox" Margin="0,0,-12,0" ItemsSource="{Binding
Items}" SelectionChanged="MainListBox_SelectionChanged">
        <ListBox.ItemTemplate>
            <DataTemplate>
                <StackPanel Margin="0,0,0,17" Width="432">
                    <TextBlock Text="{Binding LineOne}" TextWrapping="Wrap"
Style="{StaticResource PhoneTextExtraLargeStyle}"/>
                    <TextBlock Text="{Binding LineTwo}" TextWrapping="Wrap"
Margin="12,-6,12,0" Style="{StaticResource PhoneTextSubtleStyle}"/>
                </StackPanel>
            </DataTemplate>
        </ListBox.ItemTemplate>
    </ListBox>
</Grid>
</Grid>

```

Notice that data source of ListBox, the ItemsSource property is Items, an ItemViewModel collection, and because data context is set to ViewModel then data will automatically be retrieved from ViewModel. Also notice that inside the ListBox, two TextBlocks are placed each of their values, in that order, bound to LineOne and LineTwo property of ItemViewModel in ViewModel collection. Therefore the values are automatically assigned to suitable data.

7. Now press F5 and see how the application works.



We can see that emulator displays several data runtime one, two, and so on. When we click an item we will be directed to the details of the data. This scenario is also known as Master/Detail scenario and can be applied further to make menu list, display dynamic data, et cetera. With databinding, working with data is made easier, and the code is cleaner. Furthermore Windows Phone Databound Application implements the use of [MVVM \(Model-View-ViewMode\) Design Pattern](#) which actuality derives from DataBinding practice. For applications related to data and MVVM view, maybe you should know that the concept will not be discussed further here. Search engines should be your good friend :)

USING ISOLATED STORAGE

Isolated storage is a local storage that can be used for Windows Phone application data storage needs. When an application is running, it works with a number of data, some of which is temporary and only required for certain sessions, therefore storing them will not be necessary. This is where isolated storage comes in handy.

The whole IO operation has to use `IsolatedStorage` and for security reasons applications cannot access operating system storage and other application's space. The figure below shows the storage structure in Windows Phone applications.

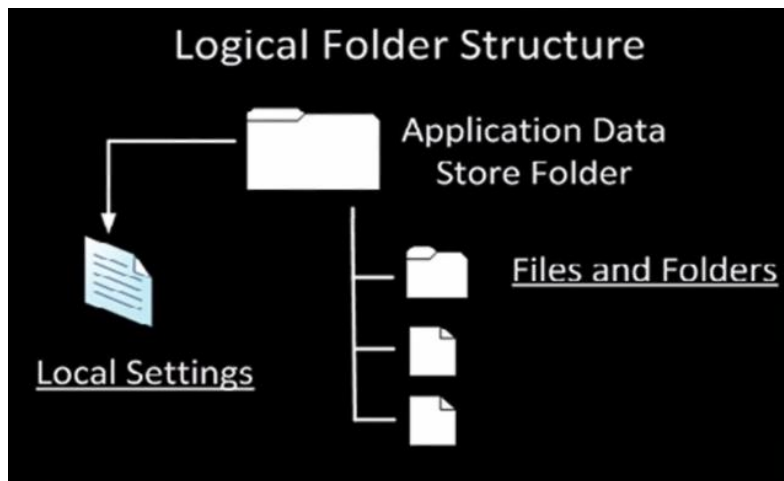


FIGURE 4 LOGICAL FOLDER STRUCTURE

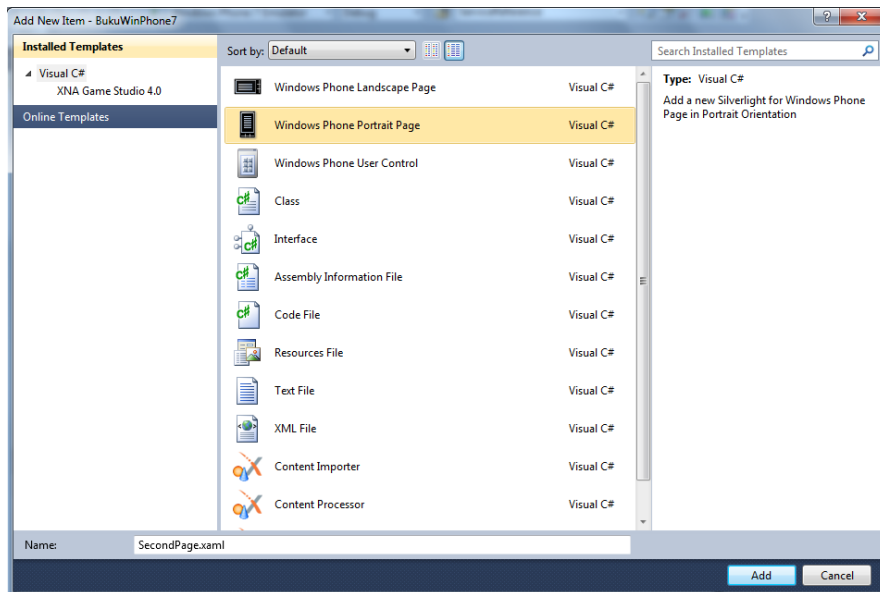
There are two parts of storage, which are standard file folder and a particular folder to store application settings.

We should put into account that storage space in mobile devices is very limited. However Windows Phone does not give quota to applications in order to provide high flexibility for developers. Nevertheless, as a developer we should consider the space storage usage responsibly. Every temporary file should immediately be deleted, and users should be given the liberty to delete what they store. It would be better to give transparency to users, letting them know how much space is used in the mobile device. We can also consider the option of storing in cloud storage, whether it's in our own application server or in a third party server. Make sure applications on device are thin, making them comfortable to use. Would users use applications that take up a lot of space in their mobile device? That's how smart phones lose their smartness :)

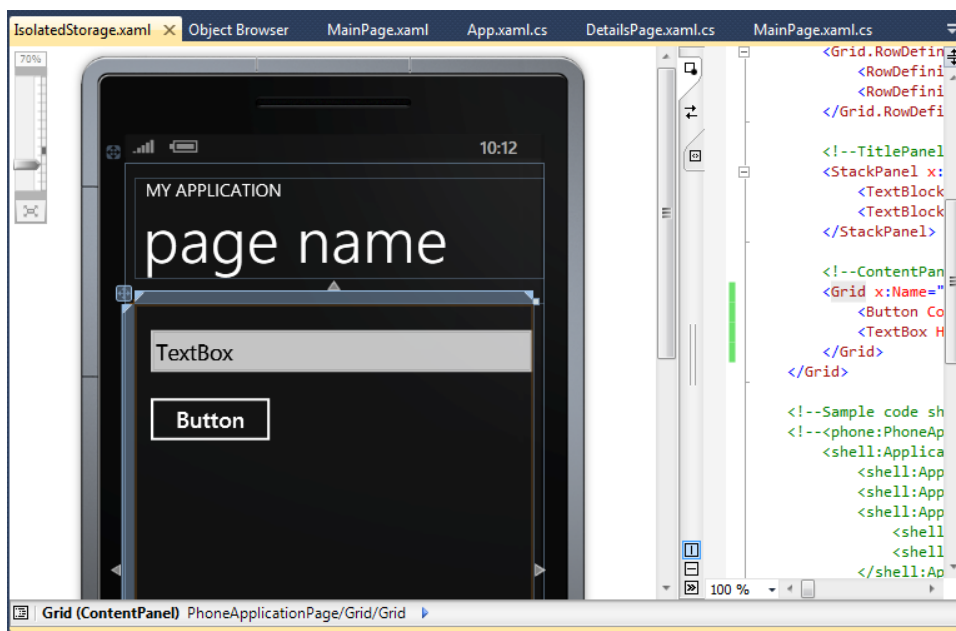
ISOLATED STORAGE FOR FILES

If you continue from the previously made project, then we need a new page to learn about isolated storage, but if you don't, create a new project for this matter. If you have learnt everything up to this page that should be relatively easy to do.

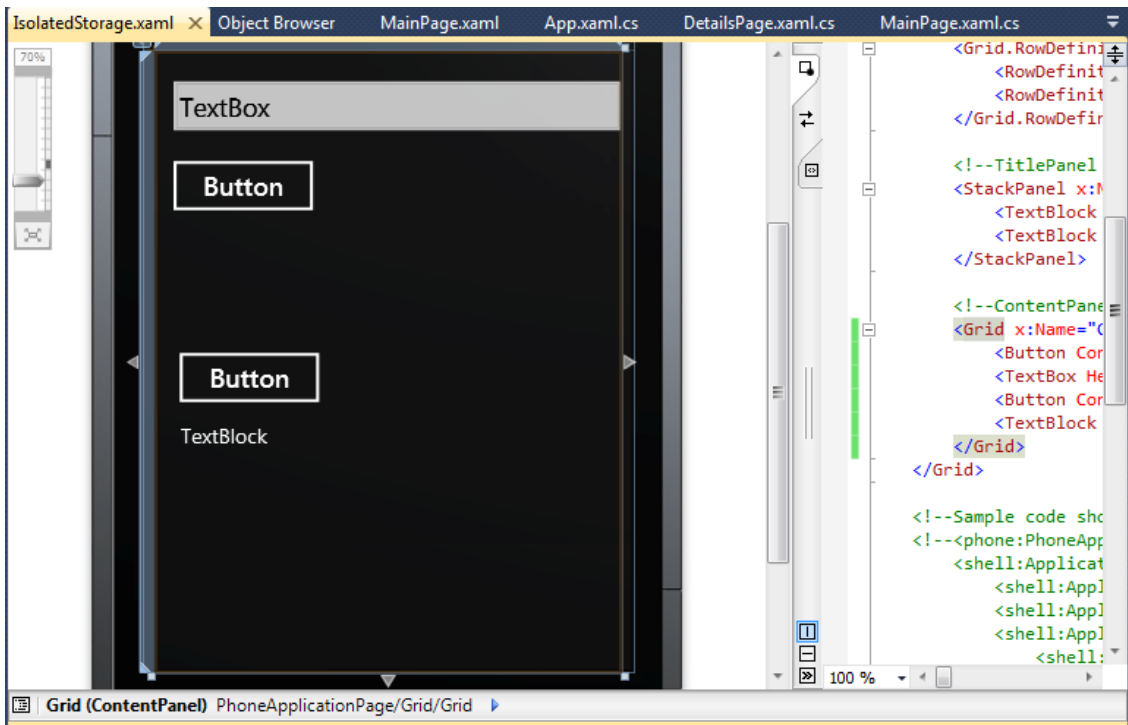
1. Right click on Project, **Add New Item** and select **Windows Phone Portrait Page**. Rename the file as you like, in this example it's called **IsolatedStorage.xaml**, then choose **Add**.



2. Insert a button and a TextBox. Click event button will later be set so that it will store the string inside the TextBox into isolated storage.



3. Next, insert a TextBlock and a button. By clicking the second button, we will call any string stored in isolated storage.



- Now double click on the first button to handle storing data into isolated storage. Add these namespaces:

```
using System.IO;
using System.IO.IsolatedStorage;
```

And type in the following code:

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();
    isf.CreateDirectory("Data");
    StreamWriter sw = new StreamWriter(new
IsolatedStorageFileStream("Data\\myfile.txt", FileMode.Create, isf));
    sw.WriteLine(textBox1.Text);
    sw.Close();
}
```

What the above code does is calling an application specific isolated storage then creates a folder called Data. The folder creation is meant for data organization simplicity. The code on the next line creates a stream writer with an isolated storage file as an input then stores the value from textBox1. Pretty straightforward.

- Next, double click on the second button to handle loading data from isolatedstorage. Type in the code below:

```
private void button2_Click(object sender, RoutedEventArgs e)
```

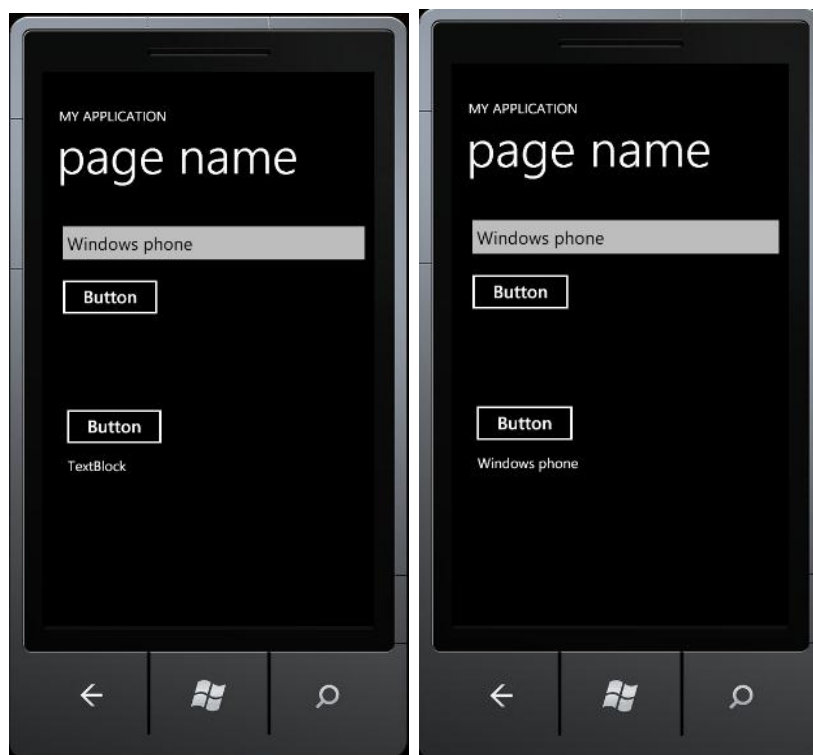
```

    {
        IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();
        StreamReader sr = null;

        try
        {
            sr = new StreamReader(new
IsolatedStorageFileStream("Data\\myfile.text", FileMode.Open, isf));
            textBlock1.Text = sr.ReadLine();
            sr.Close();
        }
        catch (Exception ex)
        {
            MessageBox.Show("error");
        }
    }
}

```

6. Press F5 and let's see how the application works.



ISOLATEDSTORAGE FOR APPLICATION SETTING

Some applications have the need to store user inputs to be used later when the application is restarted. This scenario is useful for data such as user preferences, URL, or common information. Like other .NET applications, Window Phone also supports application setting storage. For this purpose we can use **IsolatedStorageSettings.ApplicationSettings**. Application Setting itself is an instance of **IEnumerable**.

1. There is no need for a new page, let's just continue from **IsolatedStorage.xaml** for this purpose. On the first button's event handler, change the code into the following:

```
IsolatedStorageSettings settings = IsolatedStorageSettings.ApplicationSettings;
settings.Add("duration", "daily");
settings.Save();
```

The code above is pretty straightforward: we call for an instance of ApplicationSetting, insert a value and a key then store it.

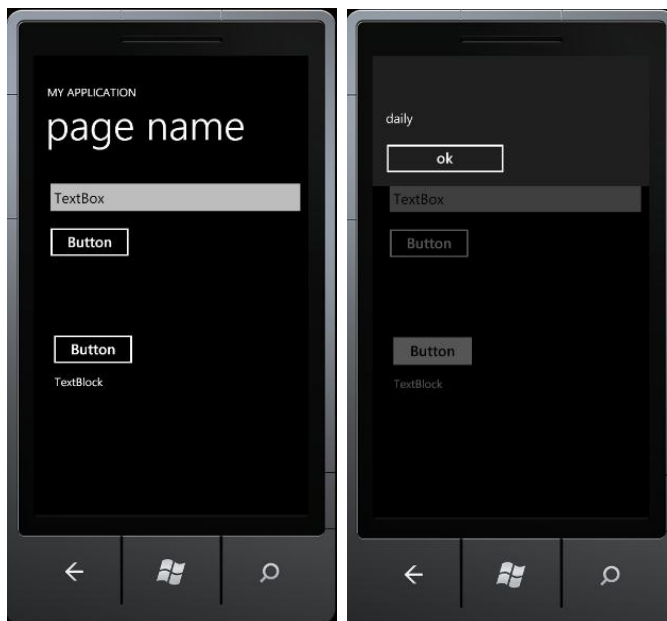
2. On the second button's event handler, change the code into the following:

```
IsolatedStorageSettings settings = IsolatedStorageSettings.ApplicationSettings;
string value = "";

try
{
    settings.TryGetValue("duration", out value);
    MessageBox.Show(value.ToString());
}
catch
{
    MessageBox.Show("error");
}
```

The code above will call an instance of ApplicationSetting, try to fetch the value of duration, and display it with a MessageBox.

3. Press F5 for results.



Press the first button to store the application's setting. Nothing seems to happen, but do believe that the setting has been saved. To prove this, press the second button. A MessageBox will appear, showing the value stored in the application's setting.

To delete the setting, it is also quite simple.

```
IsolatedStorageSettings settings = IsolatedStorageSettings.ApplicationSettings;  
settings.Remove("duration");  
settings.Save();
```

SOFT INPUT PANEL LAYOUT

On Windows Phone devices, with the absence of physical keyboard, you probably would have guessed that we have to interact with the device using on-screen keyboard. SIP or Soft Input Panel is the name given for Windows Phone's on-screen keyboard. One of the common scenarios in which SIP will appear is when we interact with a TextBox.



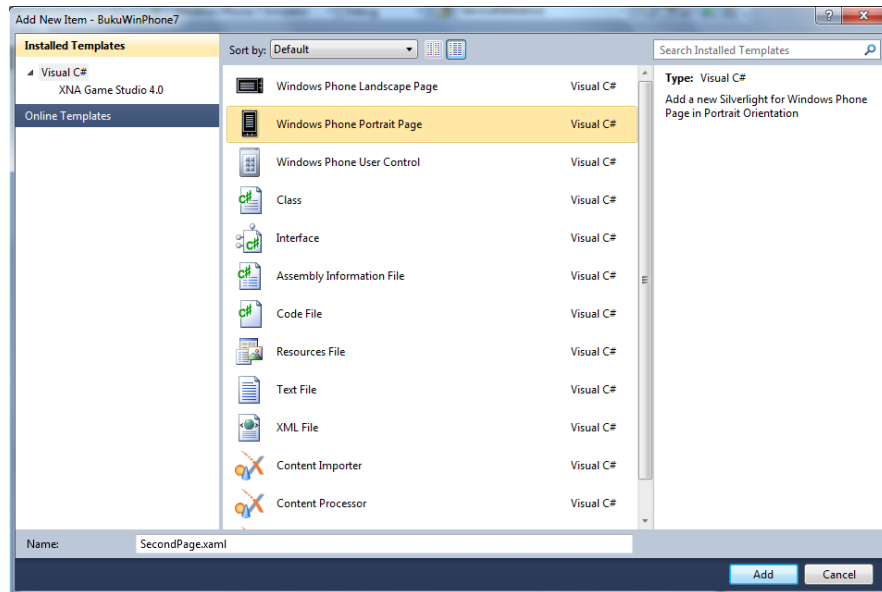
A standard SIP layout is QWERTY panel with alphabets as the main display. To view the numbers, we press the digit button on the lower left corner of the keyboard. But there may be certain scenarios in which you'd want the SIP to only display numbers when users use it to input a value. This can also be a mechanism to prevent invalid user inputs.

Such configuration can be easily done using `LayoutOptions` property for SIP. The supported layouts are: Default, Text, Digits, Web, and Email Address. Each of the layouts has its own unique characteristic. Text for example, has a similar layout to the default layout but with the addition of autocorrect and text suggestion features.

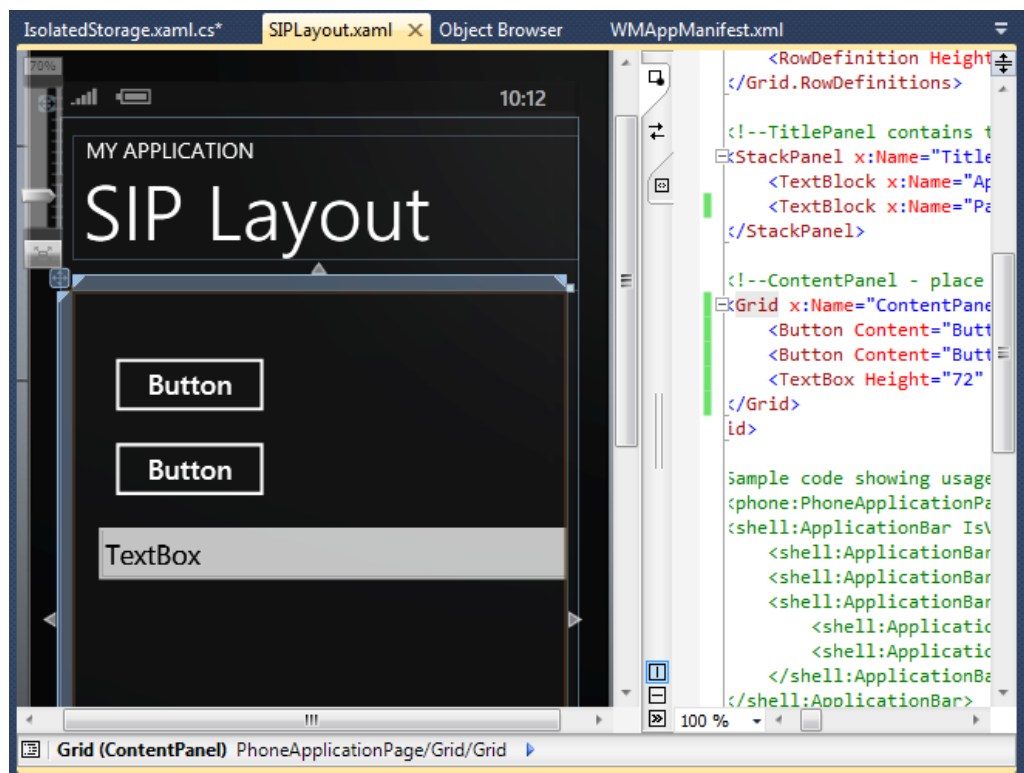
To learn about this, let's follow the steps below:

1. If you continue from the previously made project, then add a page to learn about SIP Layout. Otherwise, create a new project for this purpose. Having been doing exercises up to this page, it should be fairly easy to do. The following example uses a previously existing project. Right click

on the project, **Add New Item**, select **Windows Phone Portrait Page** and rename the file, in this example **SIPLayout.xaml** then select **Add**.



2. Add two buttons and a TextBox.



For this example, if the first button is pressed, it will show the Text layout, while pressing the second button will show the Email layout.

3. Double click on the first button and type in the following code:

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    textBox1.InputScope = new InputScope()
    {
        Names = { new InputScopeName() { NameValue = InputScopeNameValue.Text } }
    };
}
```

4. Double click on the second button and type in the following code:

```
private void button2_Click(object sender, RoutedEventArgs e)
{
    textBox1.InputScope = new InputScope()
    {
        Names = { new InputScopeName() { NameValue = InputScopeNameValue.EmailSmtpAddress } }
    };
}
```

5. Press F5 to see results.



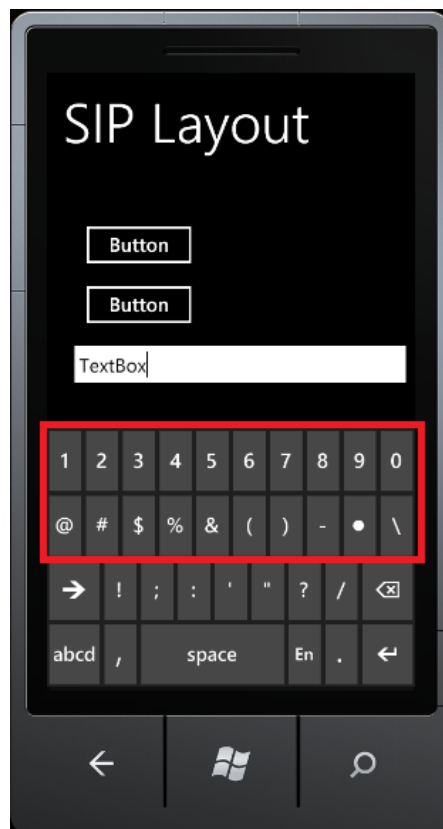
Click on the TextBox first to see the Default layout. Then click on the first button then click on the TextBox again. Now your keyboard shows the Text layout. When you type in 'Fr' for example, several word suggestions that you can pick from will appear. Now click on the second button and click the TextBox again. Although there aren't any significant differences, now on the bottom part of the

keyboard you can see two additional characters, which are @ and .com that we often use to input email addresses.

6. Other than using code, keyboard layout can also be configured directly with XAML in TextBox control. Observe the following code:

```
<TextBox Height="72" HorizontalAlignment="Left" Margin="12,207,0,0" Name="textBox1"
Text="TextBox" VerticalAlignment="Top" Width="460">
  <TextBox.InputScope>
    <InputScope>
      <InputScopeName NameValue="Digits"/>
    </InputScope>
  </TextBox.InputScope>
</TextBox>
```

7. Press F5 for results. Click on the TextBox and now the SIP layout will display the number panels as its main display.

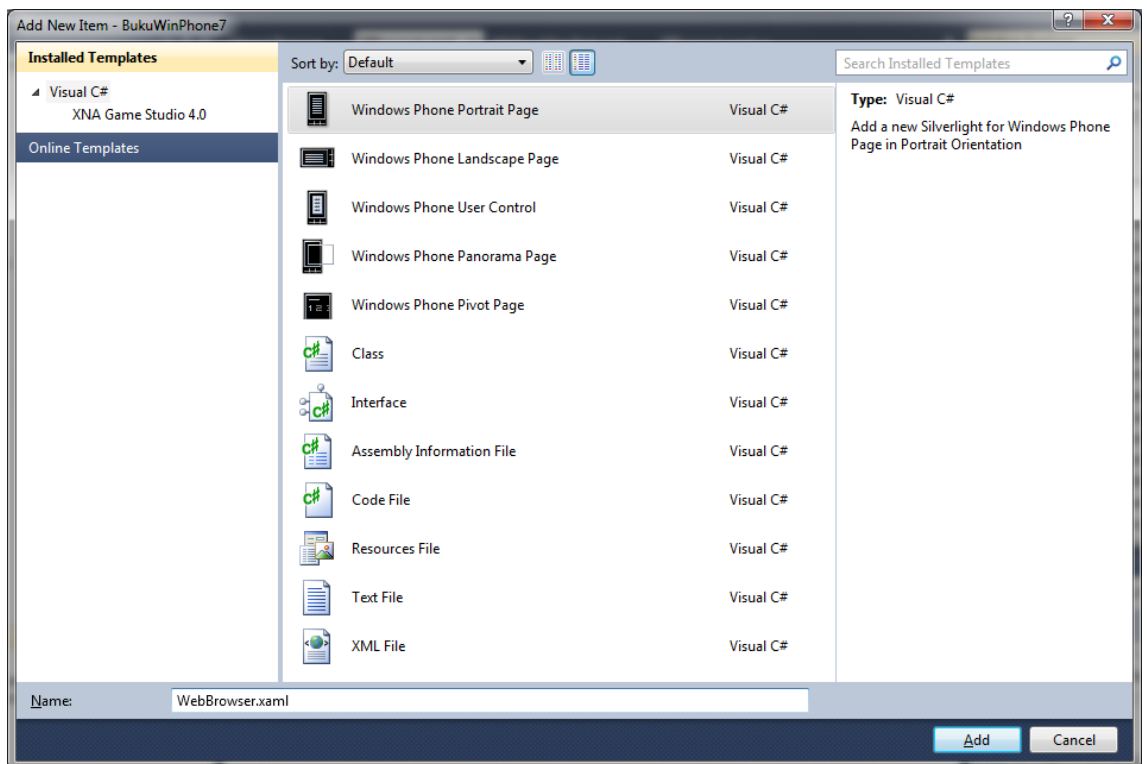


Using SIP Layout we can freely configure keyboard display. This should be used to our advantage. A better implementation of this feature will give the application a more professional feel to it, don't you think?

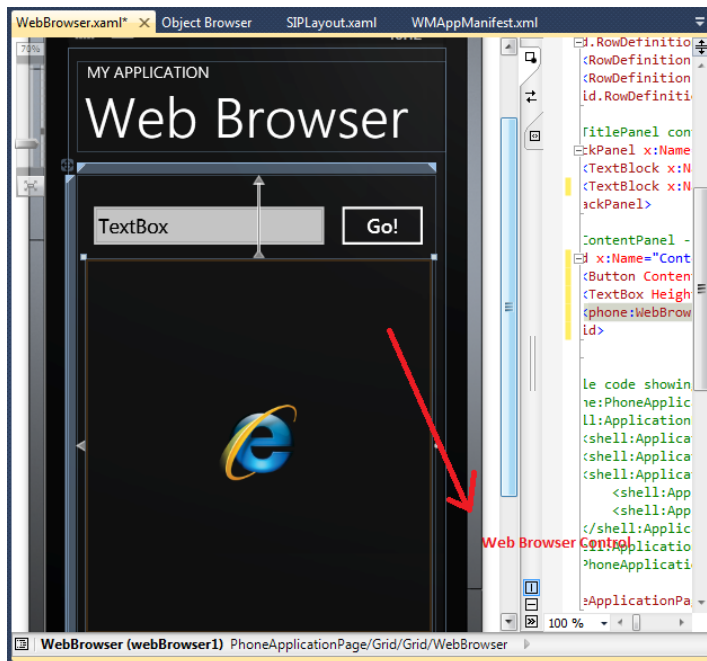
GETTING TO KNOW WEB BROWSER

Say you want a scenario in which you need to display a webpage but you don't want to use your device's built-in browser. Web Browser control is an option for this. WebBrowser control is a control that can be used to display contents in the form of a web page, whether it is a locally stored file or dynamically generated from a code.

1. If you continue from the previously made project, then add a page to learn about WebBrowser. Otherwise, create a new project for this purpose. Having been doing exercises up to this page, it should be fairly easy to do. The following example uses a previously existing project. Right click on the project, **Add New Item**, select **Windows Phone Portrait Page** and rename the file, in this example **WebBrowser.xaml** then select **Add**.



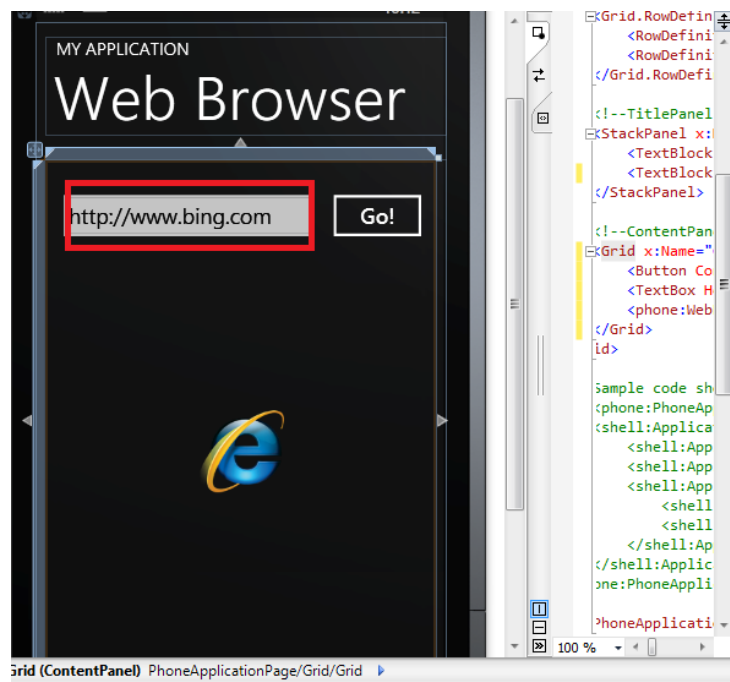
2. Change the page title into WebBrowser, add a TextBox, a button, and WebBrowser from the toolbox like the figure below.



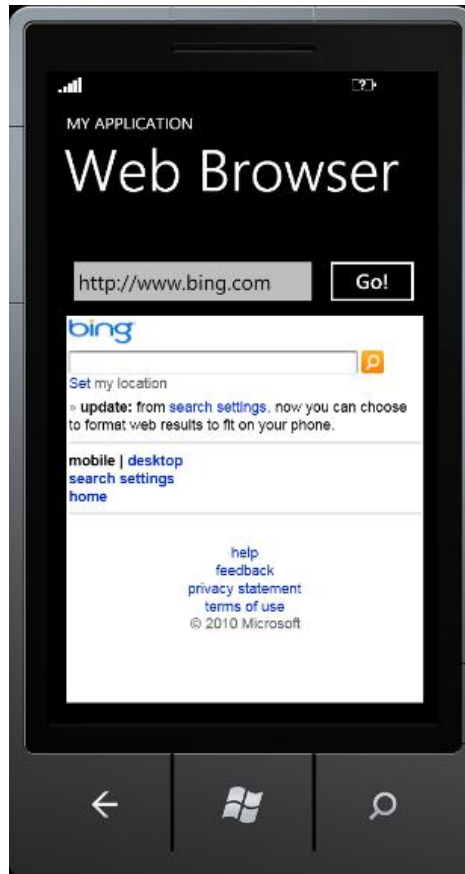
3. Double click on the button and type in the following code:

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    webBrowser1.Navigate(new Uri(textBox1.Text));
}
```

To retrieve the web content, call for Navigate method with a URI as input parameter. For starting point, change the TextBox value into a web address.



4. Press F5 for results. This of course is very dependent to your connection speed.



5. Now let's try to dynamically display web content from a code. Type in the code below in GO! Button's event handler.

```
string html = "<html><body bgcolor='white' text='red'><h1>Hello  
World</h1></body></html>";  
//webBrowser1.Navigate(new Uri(textBox1.Text));  
  
webBrowser1.NavigateToString(html);
```

In this case, use *NavigateToString* method with an HTML string as input parameter. This string will be rendered by WebBrowser and be displayed just like any webpage.

6. Press F5 and click on the button. See the results.



A file containing HTML declaration can of course be made and stored in local storage. We have learned about using IsolatedStorage in [this](#) part.

GLOBALIZATION & LOCALIZATION

Speaking of applications, especially mobile applications, as a developer you surely are not aiming to make an application just for yourself. You build the application so that it is usable for as many people as possible, maybe even for users from different countries.

Globalization is used in order to accommodate different cultures, so that applications can be distributed more broadly. Users often expect to easily adapt to applications they use daily. Examples on the matter are the language used by labels and information in the application, time format, currency, or calendar format.

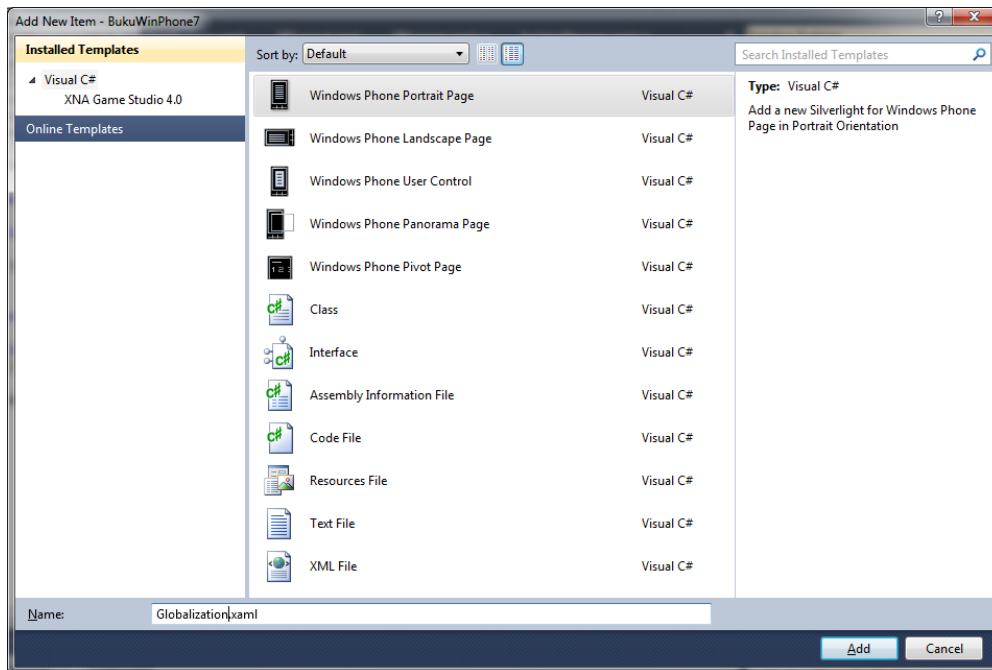
If you are experienced in using culture code in other .NET applications, then doing this shouldn't be any different in Windows Phone. Setting is done by declaring the culture type targeted to certain users in the format of 2 lower case letters, representing the language used, and 2 upper case letters, representing the name of the country, in double quotes.

"en-US" English United States	"fr-FR" French France
"en-CA" English Canada	"fr-CA" French Canada

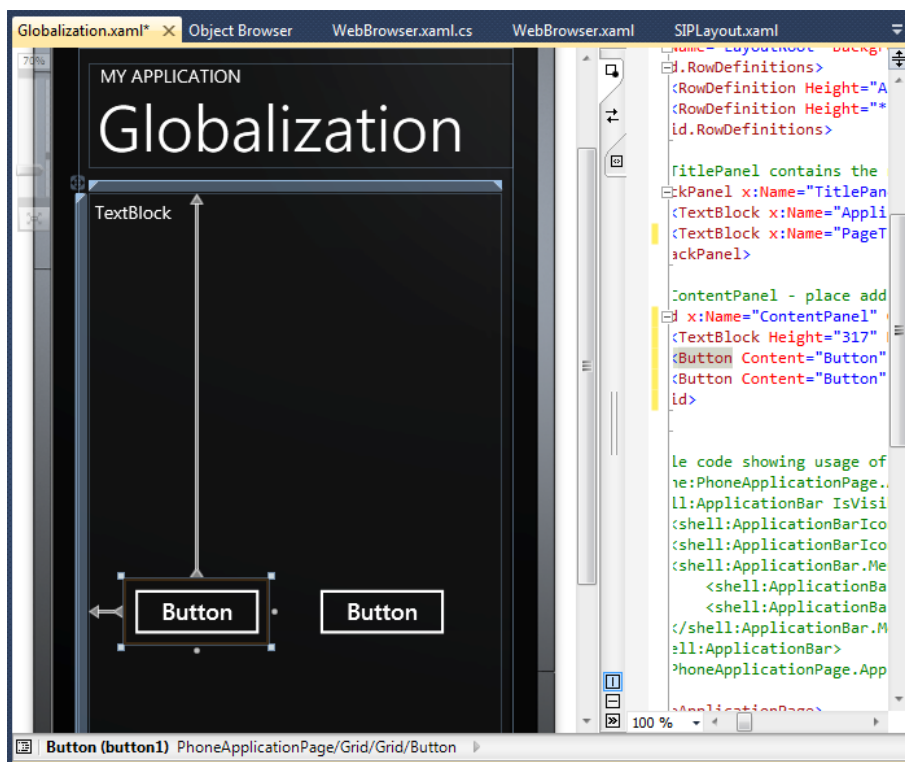
Localization, on the other hand, is used so that applications can adapt into certain culture. Besides formatting, it also deals with text translations and other configurations. To do this, we need a separated resource for the supported cultures, then the applications' code only have to refer to that certain resource. Note that separating resource from the code makes a cleaner and more maintainable application.

GLOBALIZATION

1. If you continue from the previously made project, then add a page to learn about Globalization. Otherwise, create a new project for this purpose. Having been doing exercises up to this page, it should be fairly easy to do. The following example uses a previously existing project. Right click on the project, **Add New Item**, select **Windows Phone Portrait Page** and rename the file, in this example **Globalization.xaml** then select **Add**.



2. Insert a TextBlock and two buttons like the figure below:



3. Double-click on the first button and type in the following code:

```

CultureInfo cult = new CultureInfo("en-US");
Thread.CurrentThread.CurrentCulture = cult;

textBlock1.Text = "Culture : " + Environment.NewLine + cult.NativeName;
textBlock1.Text += Environment.NewLine + DateTime.Now.ToString("d");

```

```
Int32 currency = 12500;
textBlock1.Text += Environment.NewLine + currency.ToString("C");
```

Don't forget to add the following namespaces so that Visual Studio recognizes CultureInfo and Thread classes.

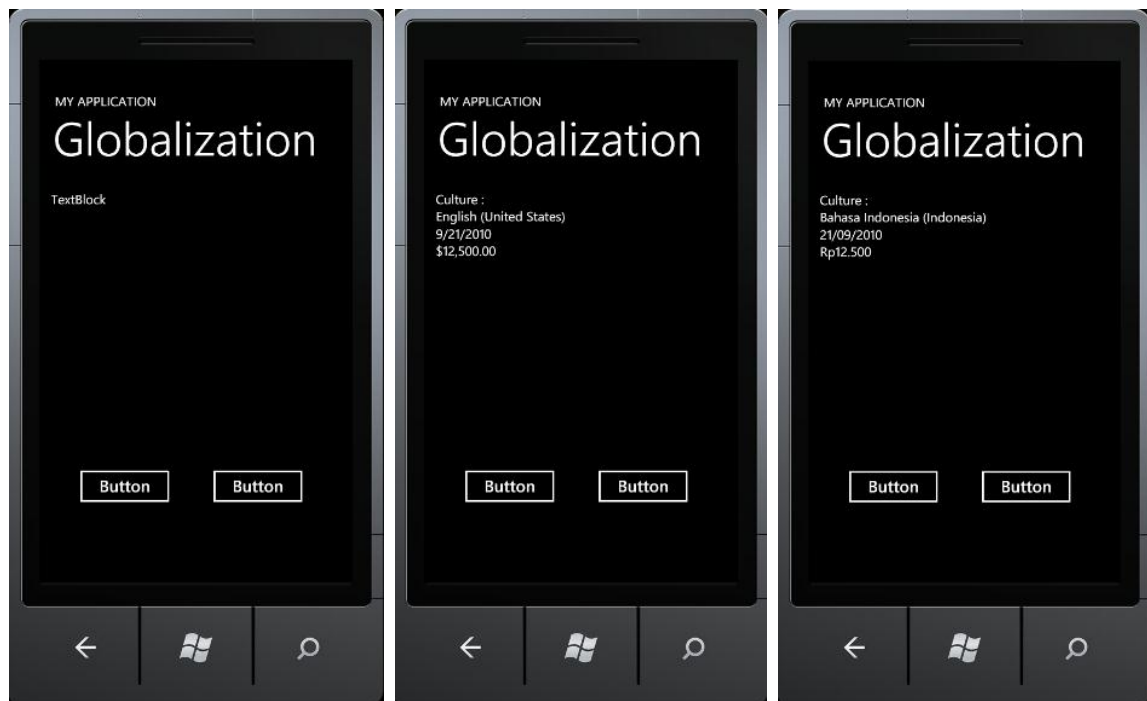
```
using System.Globalization;
using System.Threading;
```

Double click on the second button and add the following code:

```
CultureInfo cult = new CultureInfo("id-ID");
Thread.CurrentThread.CurrentCulture = cult;

textBlock1.Text = "Culture : " + Environment.NewLine + cult.NativeName;
textBlock1.Text += Environment.NewLine + DateTime.Now.ToString("d");
Int32 currency = 12500;
textBlock1.Text += Environment.NewLine + currency.ToString("C");
```

4. Press F5 to see how this application works.

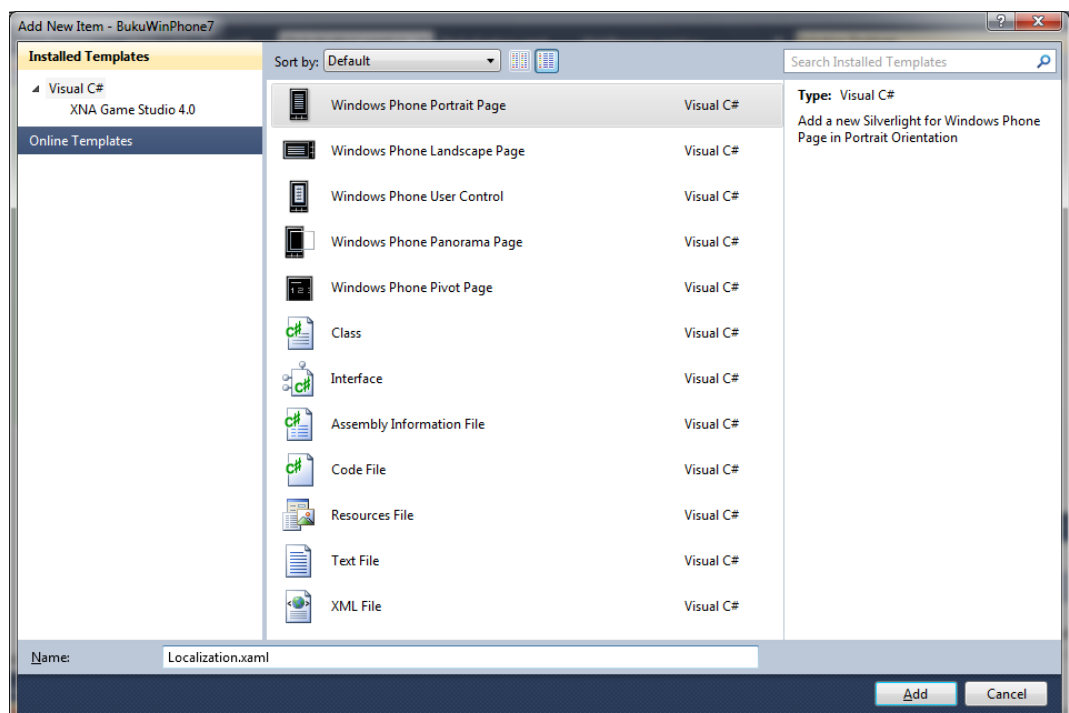


Press the first button, then the second button, and see the difference. What we just did was displaying the name of the currently used culture and displaying date and currency in a format suitable to the culture. See the formatting differences between United States and Indonesia.

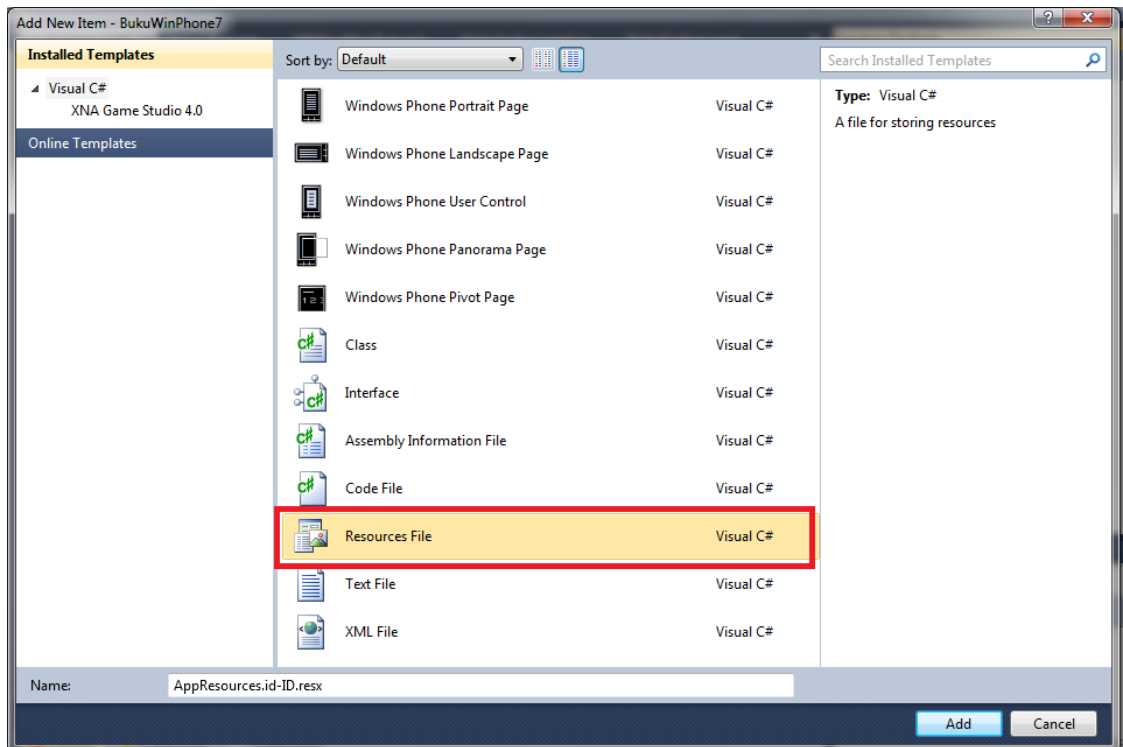
LOCALIZATION

A compiled Windows Phone application will include a number of standard resource and assembly files added for each localized languages. The language used by Windows Phone UI depends on the culture setting of the device. Take for example an application that has resources in English and Indonesian. If the device's culture setting is en-US, then the application will display the English resource. During compilation, Visual Studio will generate a standard culture used in main assembly and automatically generate a different assembly for other language resource that the developer created.

1. If you continue from the previously made project, then add a page to learn about Localization. Otherwise, create a new project for this purpose. Having been doing exercises up to this page, it should be fairly easy to do. The following example uses a previously existing project. Right click on the project, **Add New Item**, select **Windows Phone Portrait Page** and rename the file, in this example **Localization.xaml** then select **Add**.

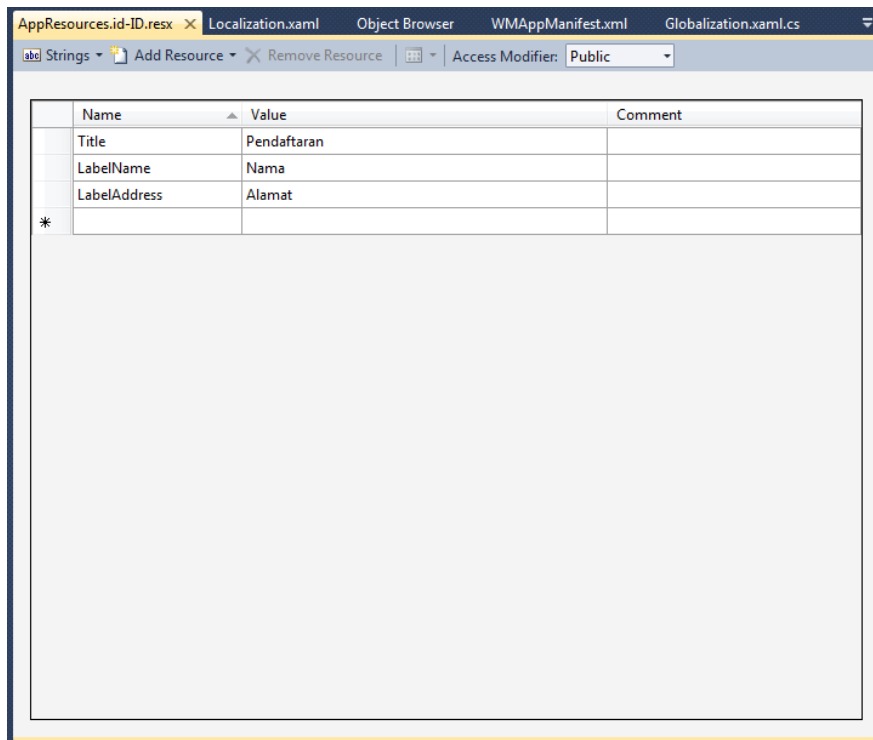


2. On Solution Explorer window, right click on project, then select **Add > New Item**. On the dialog box, select Resource File, and rename it accordingly, for example AppResources.resx. This is the file that will store a different language for the application.



- List all the strings in the application and add them inside the resource file. Each string consists of name, value, and optional comment. The name should be descriptive and unique. Values are in the form of string and will be displayed to users. For example the file is **AppResources.resx**, and this file will contain default values of the application.

Add a resource file for each language your application supports. The example below is resource file for Indonesian.

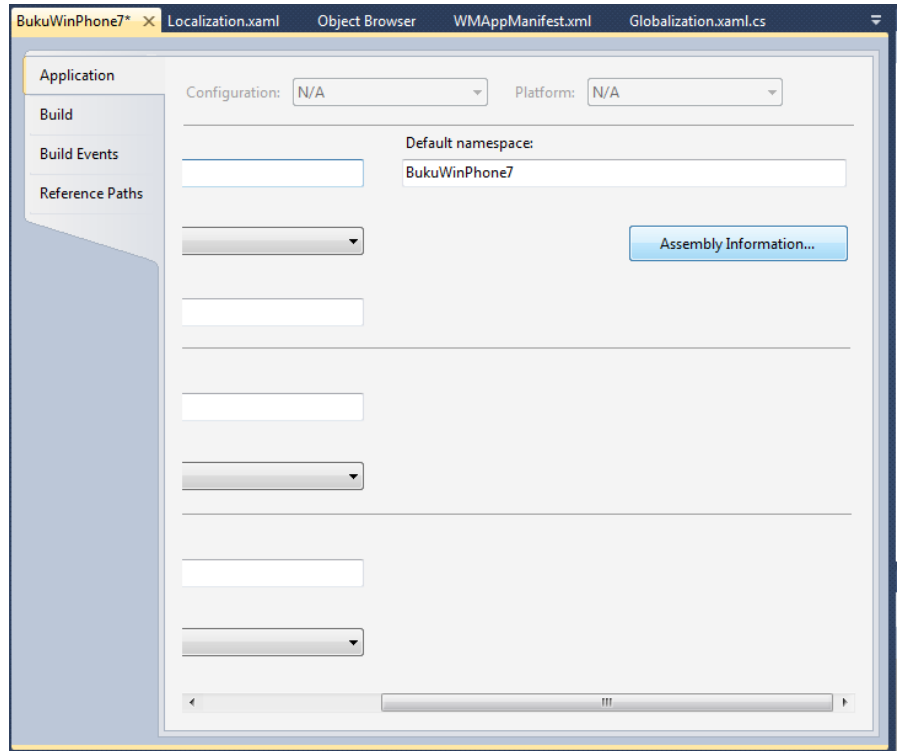


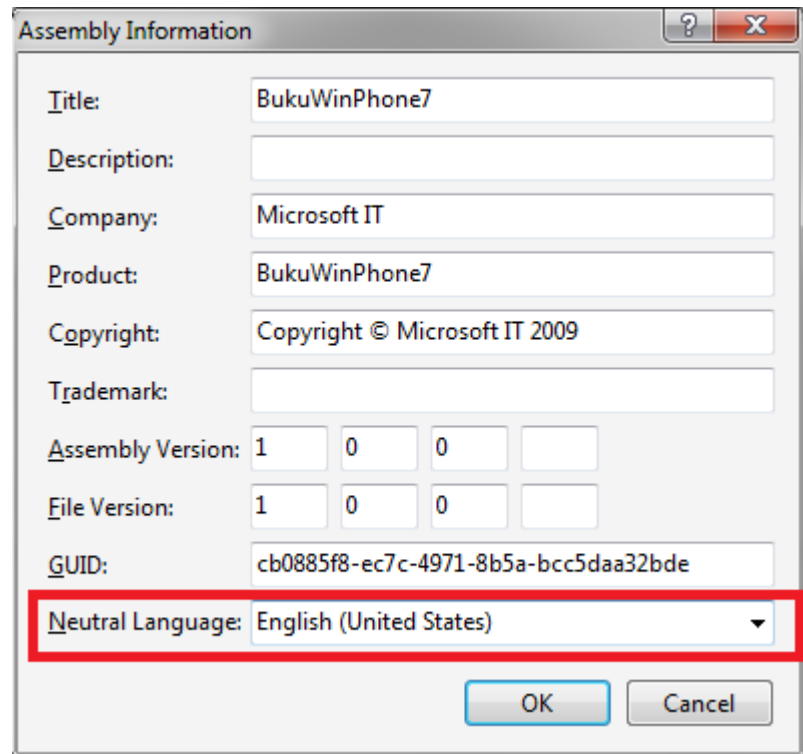
Each resource should obey the following name convention:

<default resource filename>.<culture name> where culture name is derived from CultureInfo, for example :

AppResources.id-ID.resx and **AppResources.en-US.resx**

4. Define standard culture which the application supports. Do this by right clicking on project name and select **Properties**. On **Application** tab, click the **Assembly Information** button. On **Neutral Language**, select default culture. This choice will identify the language used as standard language.

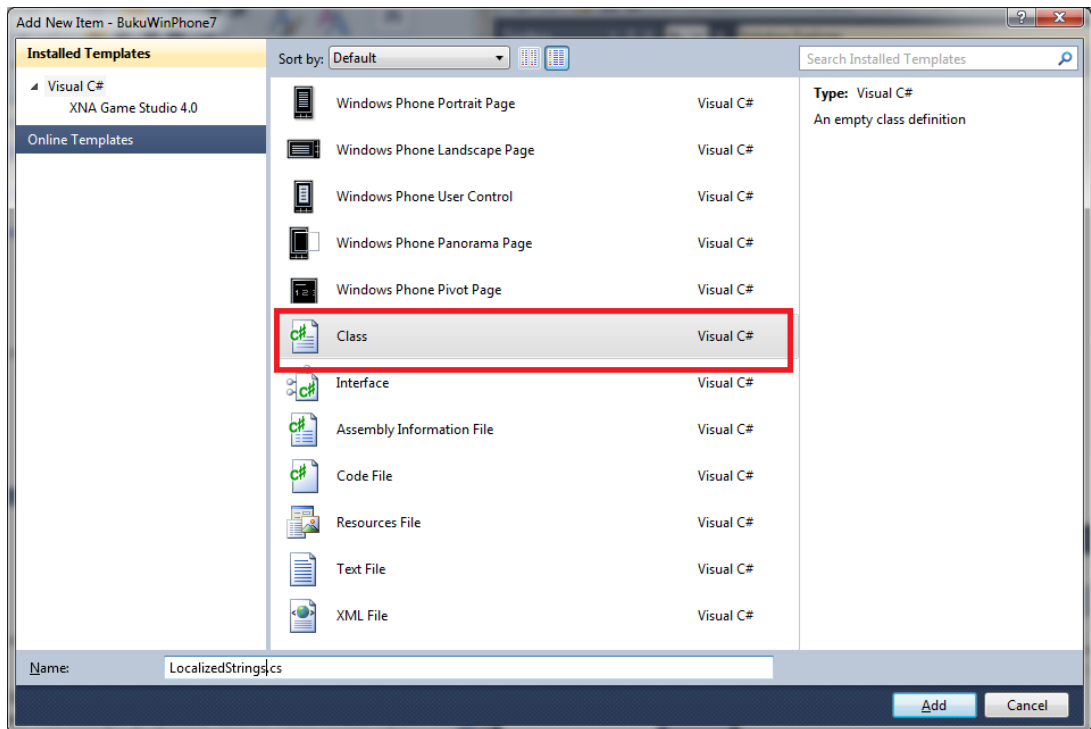




5. Close the project and open (<projectname>.csproj) file using a text editor. Find <SupportedCultures> tag and add cultures that are supported by the application. Separate each language using a semicolon. It is not necessary to add default UI, this means that if the application supports English, United States with Indonesian, Indonesia as an alternative, the tag will look like this:

<SupportedCultures>id-ID; SupportedCultures>

6. Close the text editor and reopen the project using Visual Studio. Add a class into which we will add a property that refers to the previously made resource.



Type in the following code:

```
public class LocalizedStrings
{
    private static BukuWinPhone7.AppResources localizedresources = new AppResources();

    public BukuWinPhone7.AppResources Localizedresources
    {
        get { return localizedresources; }
    }
}
```

7. Open **App.xaml** file and add a reference to the resources file that we've made

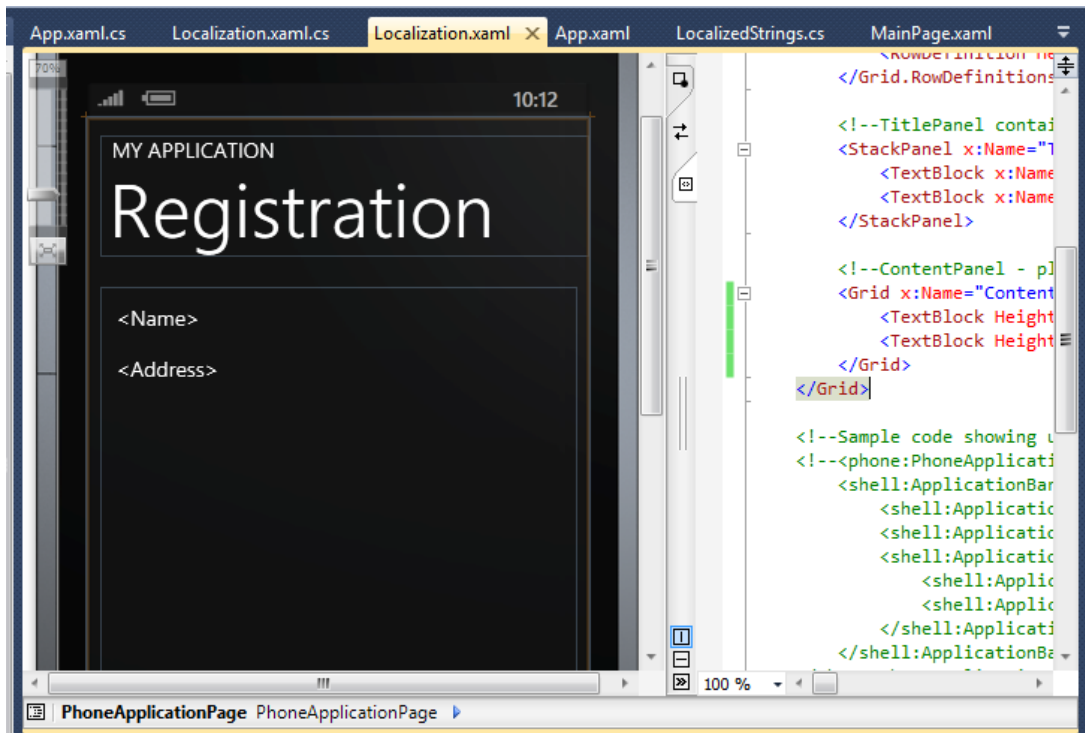
```
<Application
    x:Class="BukuWinPhone7.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    .....
    xmlns:local="clr-namespace:BukuWinPhone7"
    >

    <!--Application Resources-->
    <Application.Resources>
        <local:LocalizedStrings x:Key="LocalizedStrings"></local:LocalizedStrings>
        ...
    </Application.Resources>
</Application>
```

```
</Application.Resources>
```

Key value is used to call the resource later on.

8. Open **Localization.xaml** file and insert several TextBox like the figure below:



9. Now we will call resource to fill the value of page title, name, and address. What we will do is basically using databinding on Silverlight so that the code can be clean. Observe how to do it in XAML file:

```
<Grid x:Name="LayoutRoot" Background="Transparent">

    <!--TitlePanel contains the name of the application and page title-->
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="MY APPLICATION"
Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="{Binding
Path=Localizedresources.Title, Source={StaticResource LocalizedStrings}}" Margin="9,-
7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

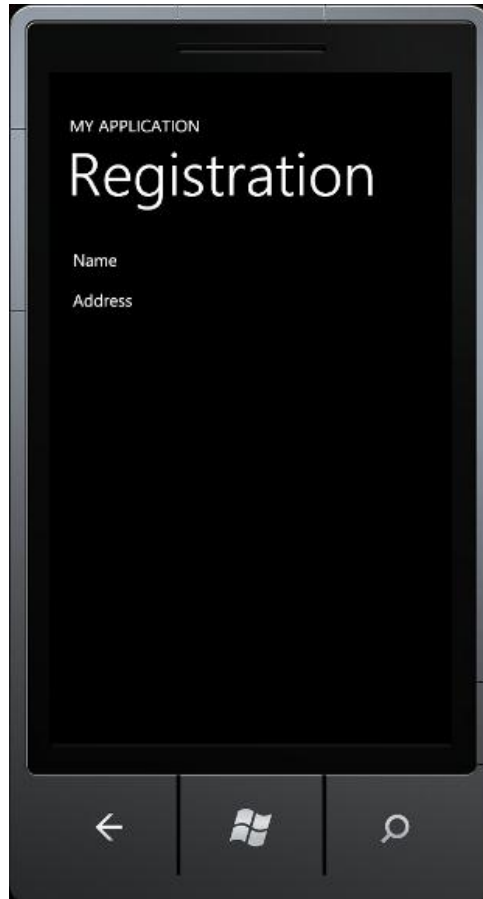
    <!--ContentPanel - place additional content here-->
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <TextBlock Height="30" HorizontalAlignment="Left" Margin="16,16,0,0"
Name="textBlock1" Text="{Binding Path=Localizedresources.LabelName,
Source={StaticResource LocalizedStrings}}" VerticalAlignment="Top" />
        <TextBlock Height="30" HorizontalAlignment="Left" Margin="16,64,0,0"
Name="textBlock2" Text="{Binding Path=Localizedresources.LabelAddress,
Source={StaticResource LocalizedStrings}}" VerticalAlignment="Top" />
    </Grid>
</Grid>
```



```
</Grid>  
</Grid>
```

The parts highlighted in yellow are ways to call resources we previously created. Property path contains the name of a string that is automatically bound to a suitable value, while source defines where the resource data comes from. On the above example it is named **LocalizedStrings** which refers to **ApplicationResources** in **App.xaml** file

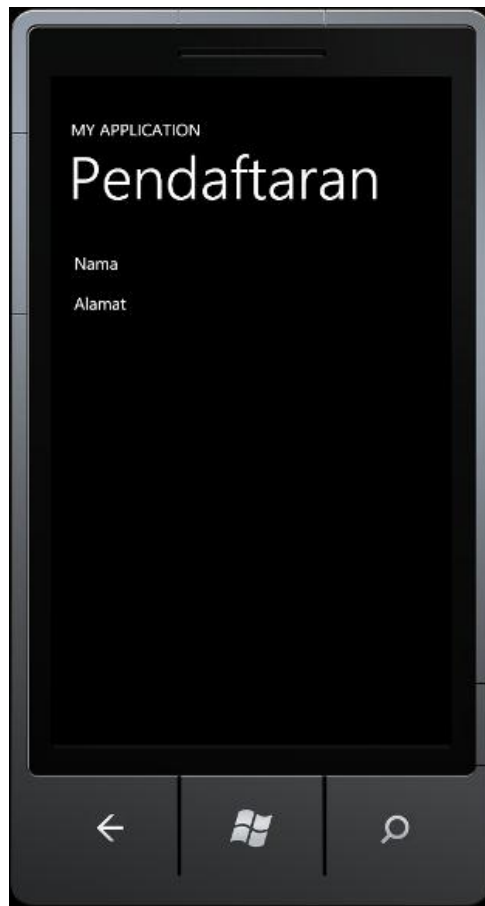
10. Press F5 and see the results



11. Stop the debugging process and add the following code into **App.xaml.cs**.

```
// Code to execute when the application is launching (eg, from Start)  
// This code will not execute when the application is reactivated  
private void Application_Launching(object sender, LaunchingEventArgs e)  
{  
    CultureInfo cul = new CultureInfo("id-ID");  
    Thread.CurrentThread.CurrentCulture = cul;  
    Thread.CurrentThread.CurrentUICulture = cul;  
}
```

We are changing the culture used so that it follows the resource file we have prepared for Indonesian. Press F5 to see results.



As expected, the application can adapt with the selected culture and automatically load the resource we've created. Using localization we can create an application that can adapt to its users. We also have the advantage of having a cleaner, more maintainable code thus making it easier to develop further.

LOCATION BASED SYSTEM

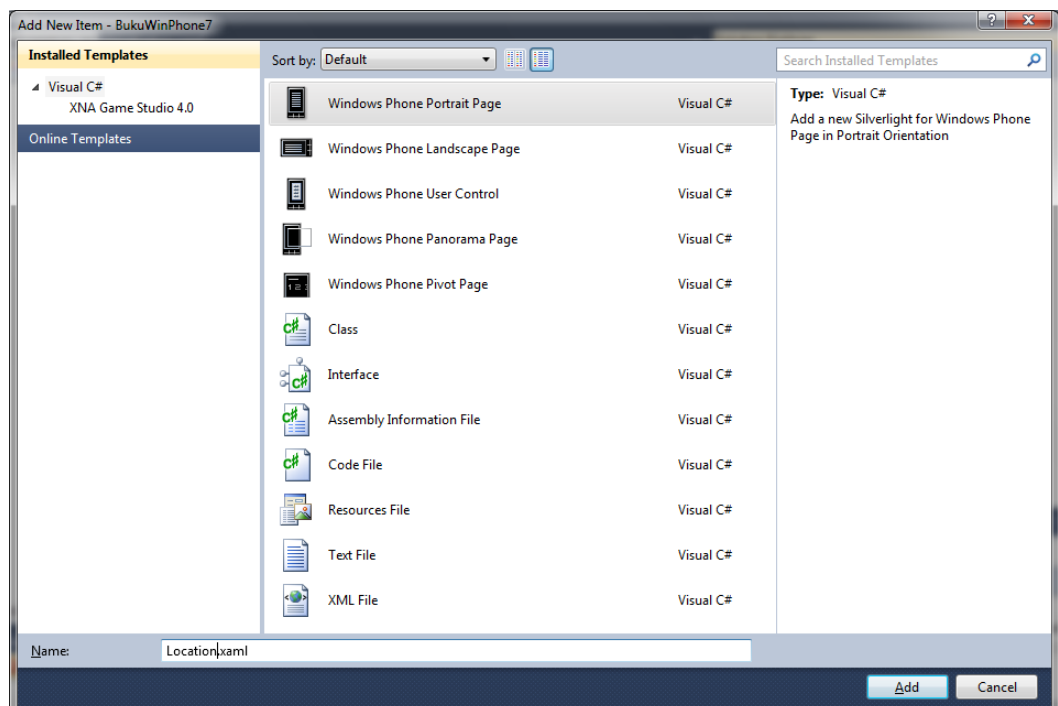
Location based system has become a certain trend in 2010. We can see services such as Gowalla, Foursquare, and even Facebook offering location features in their applications. The ability to retrieve locations gives developers an opportunity to give a unique user experience.

Any Windows Phone device manufacturer are obliged to include in the device a sensor that can fetch the device's location at a given time. Furthermore the location service from Microsoft enables us to develop location-aware applications on Windows Phone. This service can fetch location data from GPS, Wi-Fi, or cellular towers. All three data source can be retrieved using managed code.

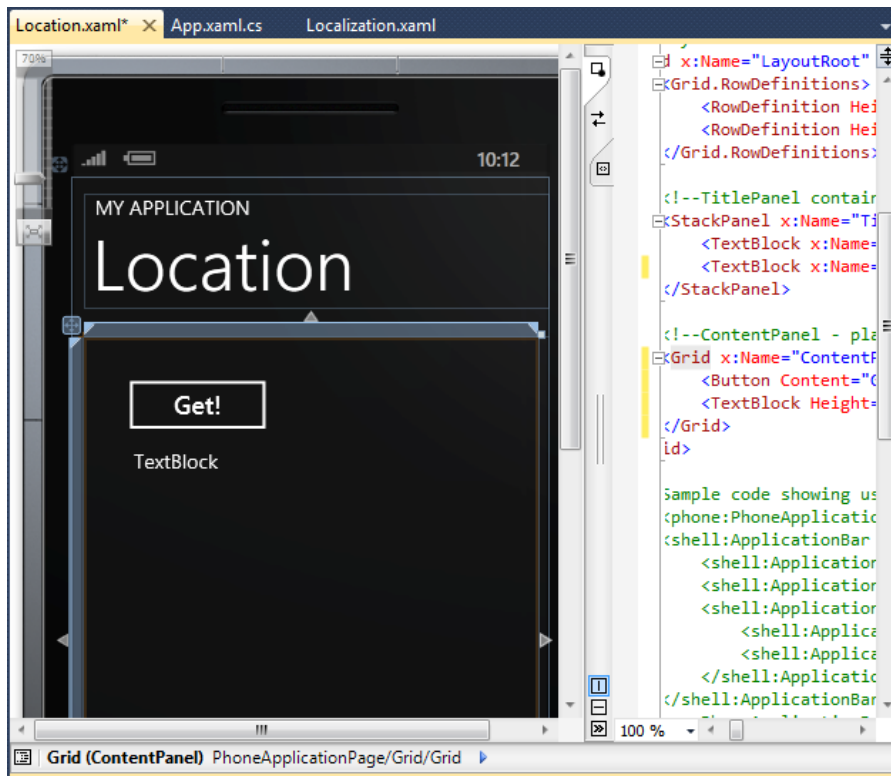
There are several things to take into account in using location sensors on the device. The first one is movement threshold. This is set in meters and we need to consider its effect to battery consumption and noise handling of movement changes due to GPS sensor. Sensor accuracy can also be managed depending on the needs of the application. Using higher accuracy will surely give a better result, but it will also cause a battery drain. Meanwhile battery consumption is highly critical in mobile application. Reckless handling regarding this matter will cause a bad user experience. This is how smart phones become dumb.

To study the ways of using location features on Windows Phone, observe the steps below:

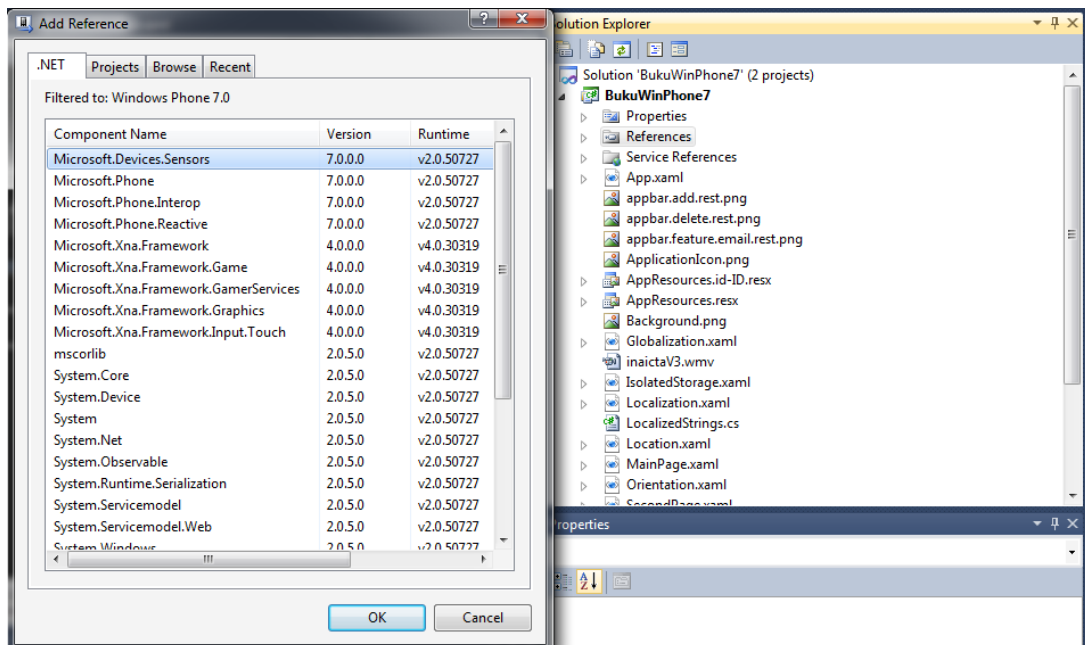
1. If you continue from the previously made project, then add a page to learn about Location. Otherwise, create a new project for this purpose. Having been doing exercises up to this page, it should be fairly easy to do. The following example uses a previously existing project. Right click on the project, **Add New Item**, select **Windows Phone Portrait Page** and rename the file, in this example **Location.xaml** then select **Add**



2. Insert a button and a TextBlock so that it looks like this:



3. To use GPS, we need to add a reference to **System.Device**. To do this, right click on References and select **Add Reference**.



4. Now we will add a reference and an instance of **GeoCoordinateWatcher** class which we will use to retrieve position from the device's GPS.

```
using System.Device.Location;
```

```
namespace BukuWinPhone7
{
```

```

public partial class Location : PhoneApplicationPage
{
    GeoCoordinateWatcher watcher;

    public Localization()
    {
        InitializeComponent();
    }
}

```

5. Use the **Start** function in the GeoCoordinateWatcher class instance to retrieve data from location service. In this example, data will be fetched when the button is pressed. Add an event handler on the button.

```

private void button1_Click(object sender, RoutedEventArgs e)
{
    if (watcher == null)
    {
        watcher = new GeoCoordinateWatcher(GeoPositionAccuracy.Default);
        watcher.MovementThreshold = 20; //to lower noise
        watcher.StatusChanged += new
        EventHandler<GeoPositionStatusChangedEventArgs>(watcher_StatusChanged);
        watcher.PositionChanged += new
        EventHandler<GeoPositionChangedEventArgs<GeoCoordinate>>(watcher_PositionChanged);
        watcher.Start();
    }
}

void watcher_PositionChanged(object sender,
GeoPositionChangedEventArgs<GeoCoordinate> e)
{
    throw new NotImplementedException();
}

void watcher_StatusChanged(object sender, GeoPositionStatusChangedEventArgs e)
{
    throw new NotImplementedException();
}

```

Remember that we need to implement an event handler for status change (whether or not the location service is available) and position change. Start function will call for service asynchronously so users can still use the application.

6. For status change handler, notice that this event handler will be called by a different thread from the active page. For this reason, we need to utilize **Dispatcher** to invoke functions in the page's thread.

```

void watcher_StatusChanged(object sender, GeoPositionStatusChangedEventArgs e)
{
    Deployment.Current.Dispatcher.BeginInvoke(() =>
        MyStatusChanged(e));
}

void MyStatusChanged(GeoPositionStatusChangedEventArgs e)
{
    switch (e.Status)
    {
        case GeoPositionStatus.NoData :
            MessageBox.Show("No Data Available");
            break;
        case GeoPositionStatus.Ready :
            //donothing
            break;
        case GeoPositionStatus.Disabled:
            MessageBox.Show("Location service is disabled");
            break;
    }
}

```

7. As the location service is active and receives data, it will invoke *PositionChanged* event, therefore the handling of position change can be done according to an application logic that we want. The same thing applies for status change event; we need to use Dispatcher to call the handler.

```

void watcher_PositionChanged(object sender,
GeoPositionChangedEventArgs<GeoCoordinate> e)
{
    Deployment.Current.Dispatcher.BeginInvoke(() => MyPosititonChanged(e));
}

void MyPosititonChanged(GeoPositionChangedEventArgs<GeoCoordinate> e)
{
    textBlock1.Text = e.Position.Location.Latitude.ToString("0.000") + " " +
e.Position.Location.Longitude.ToString("0.000");
}

```

8. Press F5 and observe the result. Press the button to see how the application works. Of course, since we are using an emulator, the data service will not be available, but the code will work on a real device.



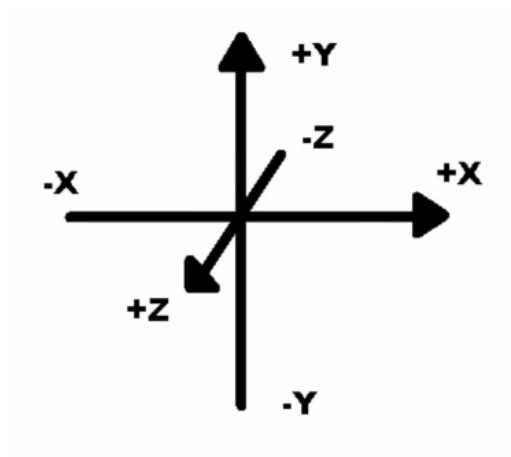
GETTING TO KNOW ACCELEROMETER

Accelerometer is a component to measure acceleration in such a way that it can detect changes in the device's position and the magnitude of the change. Microsoft requires every Windows Phone manufacturer to put this sensor in every device that supports Windows Phone. This way, a change in the device's physical position can be used as an alternative for users to interact with a Windows Phone application.

This component gives as a new user experience in interacting with the device's movement. This can be used not only for standard applications but especially for games also.

To use accelerometer, you need to import **Microsoft.Devices.Sensors** dll.

It's pretty simple to use; there are two basic functions, Start and Stop, and an event to handle, which is ReadingChanged. Accelerometer detects a change in the x, y, and z dimensions. Using the value it detects, we can write certain code that reacts depending on the change.



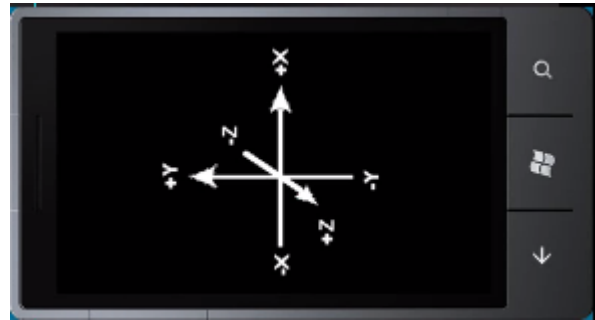
The device's axis does not change on orientation change. The Y axis is always from the top to the bottom of the device, perpendicular to the three hardware buttons, the X axis is always from one side of the device to another, parallel to the three hardware button, while the Z axis is a virtual axis that goes through the device, assuming we are holding the device and looking at it. The value we can get ranges from -1 to 1.



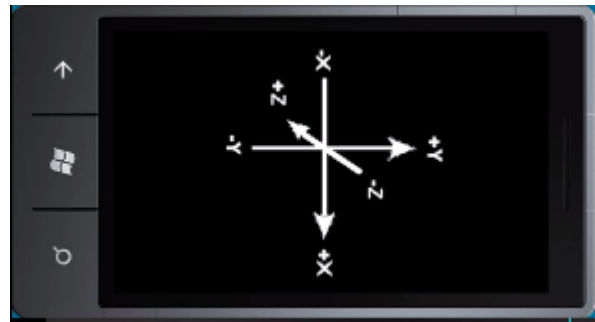
For reading schema, take a look at these illustrations of result values from the accelerometer:

Upright : 0 -1 0	A smartphone screen displaying a 3D coordinate system, identical to the one in the first image. The vertical axis is labeled +Y at the top and -Y at the bottom. The horizontal axis is labeled +X on the right and -X on the left. The diagonal axis is labeled +Z pointing towards the bottom-left and -Z pointing towards the top-right. The screen has a black background with white axes and labels. At the bottom of the screen, there are three touch-sensitive buttons: a back arrow, the Windows logo, and a search magnifying glass.	
------------------	--	--

Rotated counterclockwise : -1 0 0



Rotated clockwise : 1 0 0

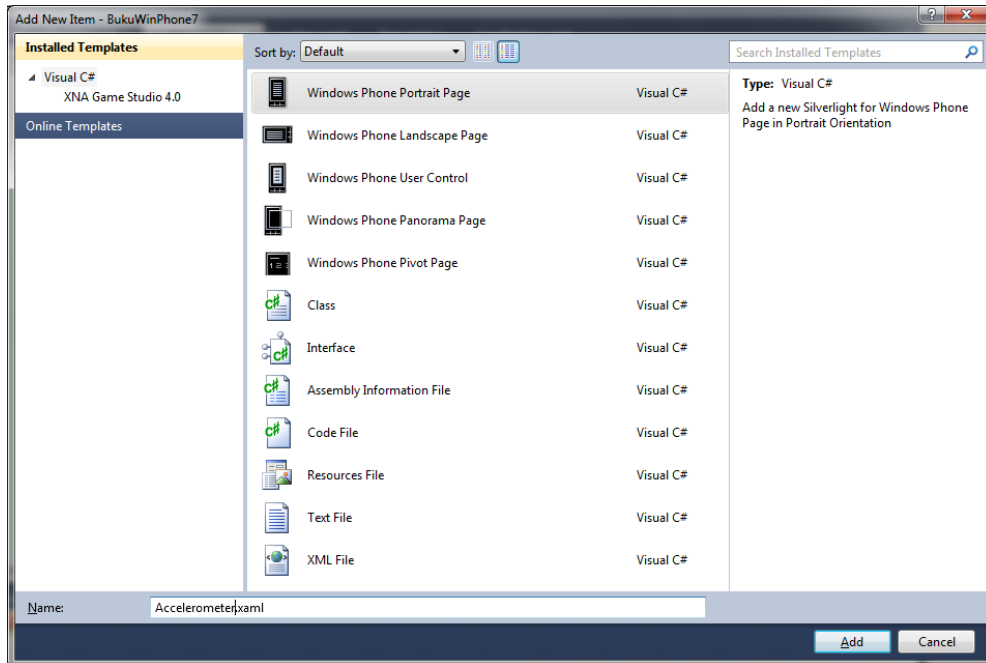


Lying flat : 0 0 -1

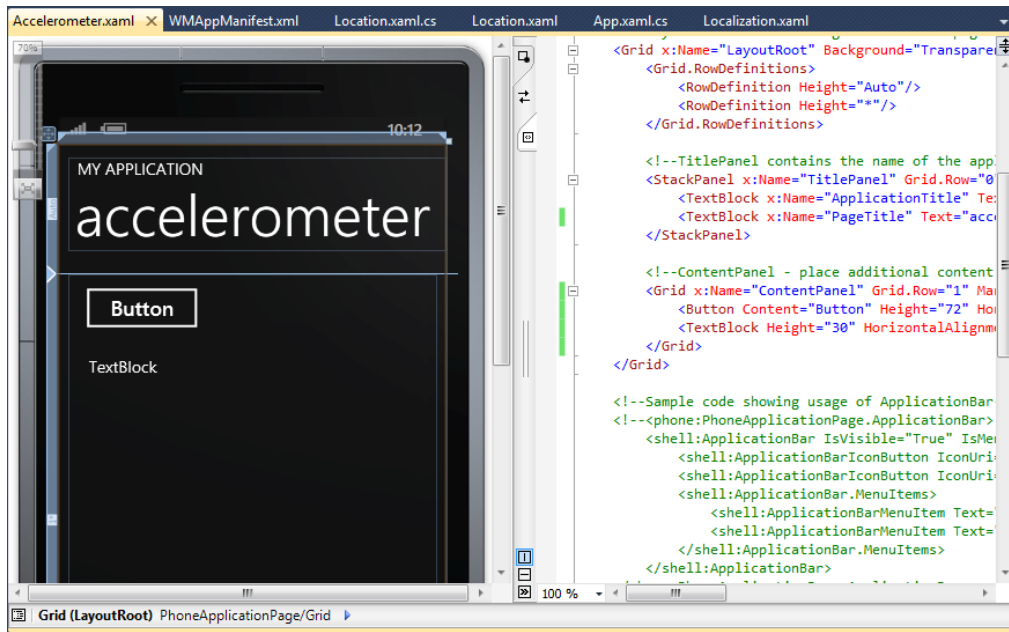


Now we will learn how to retrieve data from the sensor.

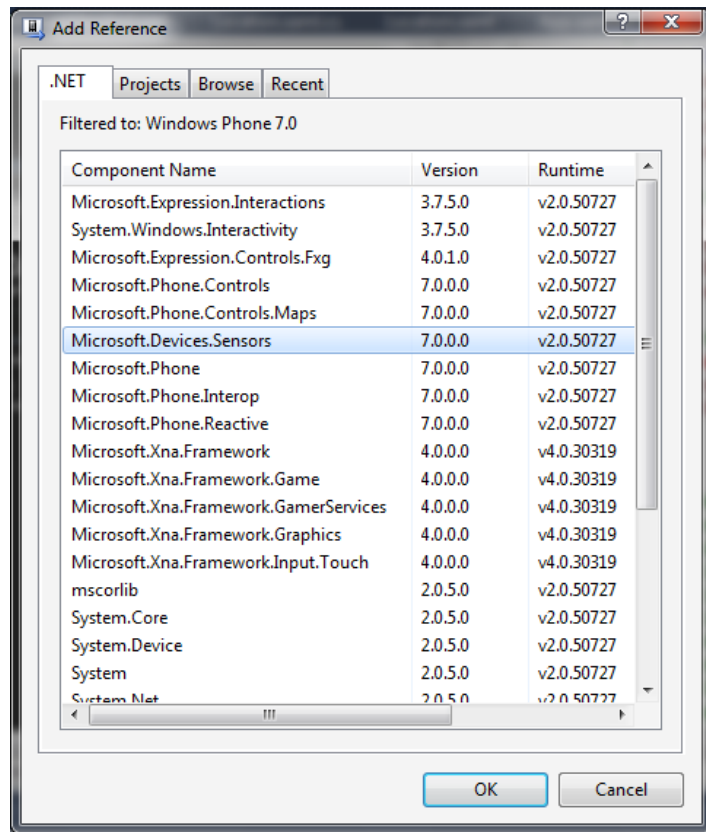
1. If you continue from the previously made project, then add a page to learn about Accelerometer. Otherwise, create a new project for this purpose. Having been doing exercises up to this page, it should be fairly easy to do. The following example uses a previously existing project. Right click on the project, **Add New Item**, select **Windows Phone Portrait Page** and rename the file, in this example **Accelerometer.xaml** then select **Add**.



2. Add a button and a TextBlock so that it looks like this:



3. Add a reference to **Microsoft.Devices.Sensors** assembly. On **Solution Explorer** window, right click on **Reference** and select **Add New Reference**



4. Add the following code:

```
using Microsoft.Devices.Sensors;

namespace BukuWinPhone7
{
    public partial class Accelerometer : PhoneApplicationPage
    {
        Accelerometer accelerometer;

        public Accelerometer()
        {
            InitializeComponent();
        }
    }
}
```

Don't forget to add assembly usage reference. On the next part the **Accelerometer** object id defined for us to use later.

5. Add an event handler on the button using this code:

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    if (accelerometer == null)
    {
```

```

        accelerometer = new Accelerometer();
        accelerometer.ReadingChanged += new
EventHandler<AccelerometerReadingEventArgs>(accelerometer_ReadingChanged);
    }

    accelerometer.Start();
}

void accelerometer_ReadingChanged(object sender,
AccelerometerReadingEventArgs e)
{
    throw new NotImplementedException();
}

```

After instantiating accelerometer, add a handler to handle data change from the sensor.

6. Because data changes are sent constantly and this may happen during thread runtime, in fetching data we need to use Dispatcher to invoke data change event handler.

```

void accelerometer_ReadingChanged(object sender, AccelerometerReadingEventArgs e)
{
    Deployment.Current.Dispatcher.BeginInvoke(() MyReadingChanged(e));
}

```

7. Insert a function to handle the data change.

```

void MyReadingChanged(AccelerometerReadingEventArgs e)
{
    textBlock1.Text = String.Format("{0} {1} {2}", e.X.ToString(),
e.Y.ToString(), e.Z.ToString());
}

```

Fill this part with the logic for the application you're developing.

8. Press F5 and press button, see what happens.



Accelerometer shows the values 0 0 -1, which means the emulator is in lying flat position. Since the emulator doesn't have built in support for accelerometer, we cannot test this function using emulator.

There are several things to remember. Getting the exact value of 1.0 is however very unlikely to ever happen because gravity does not work that way. It may astound you how standing on top of a mountain or shaking your hand too much can add pressure on the device.

Accelerometer has its own fault tolerance, so maybe you would want to experiment on the total value of the fault.

Imagine that you will receive a lot of new data changes, precisely 50 times per second. This means that the amount of data we will receive is humongous. Retrieved data is very likely to be unstable due to the nature of accelerometer sensor itself. Even when the device is lying on a table, any variation may occur. This means that any application that uses accelerometer needs a reading mechanism, whether it's calibration or smoothing for any data received from the component.

Further information regarding this topic can be obtained here.

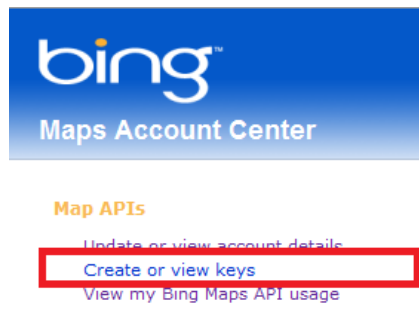
BING MAPS CONTROL FOR WINDOWS PHONE

Bing™ Maps Silverlight® Control for Windows® Phone combines the powers of Silverlight and Bing Maps to support applications. Developers can now use Bing Maps Silverlight Control, which includes location and search services. For those of you who are quite familiar with using this control in standard Silverlight applications, you surely won't see any difficulties to use the control in your Windows Phone applications.

To be able to use this service, you have to register in order to get a key to use control, SOAP Services, REST, and Bing Spatial Data Services. Without a valid key we will not be able to fetch data via web.

REGISTERING BING MAPS ACCOUNT

1. Open your browser and go to page <http://www.bingmapsportal.com>
2. Select **Create** to make a new account using Windows Live ID
3. Complete your registration data on the next page
4. After it's finished, create a new key for your application. Select **Create or view keys** from the links on the left



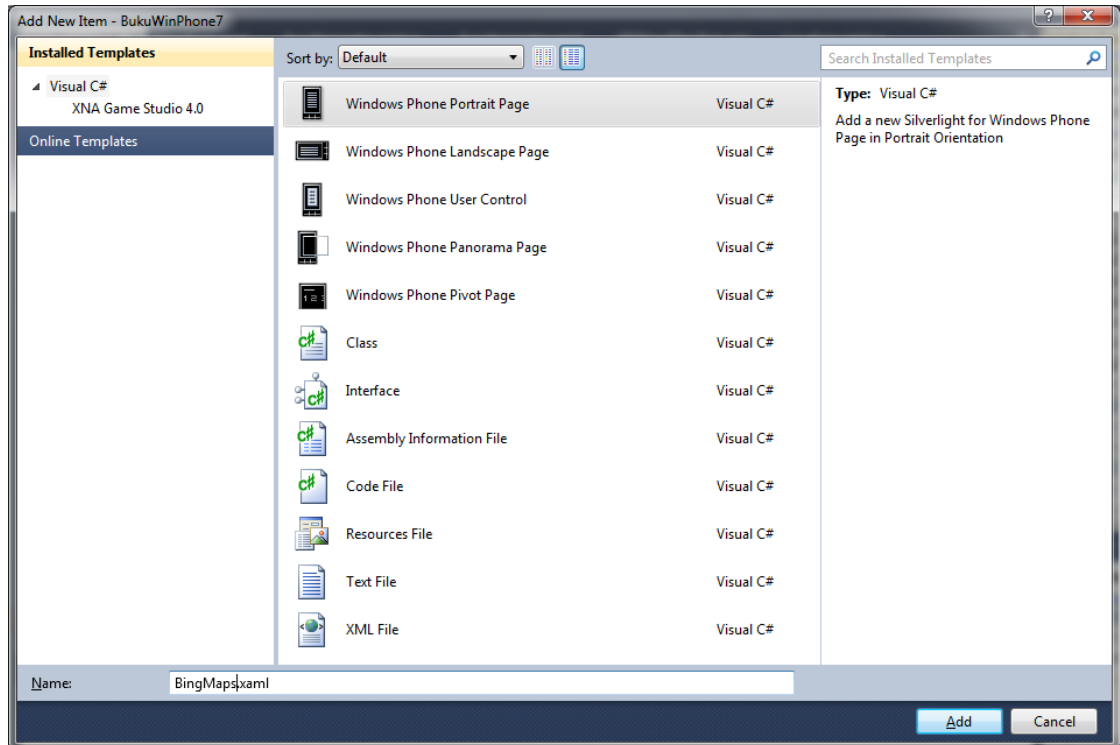
5. Complete the application details, and click **Create key**

Application name	Key / URL
Mosaic	 http://ganesh-project.co.cc/ Evaluation/Trial

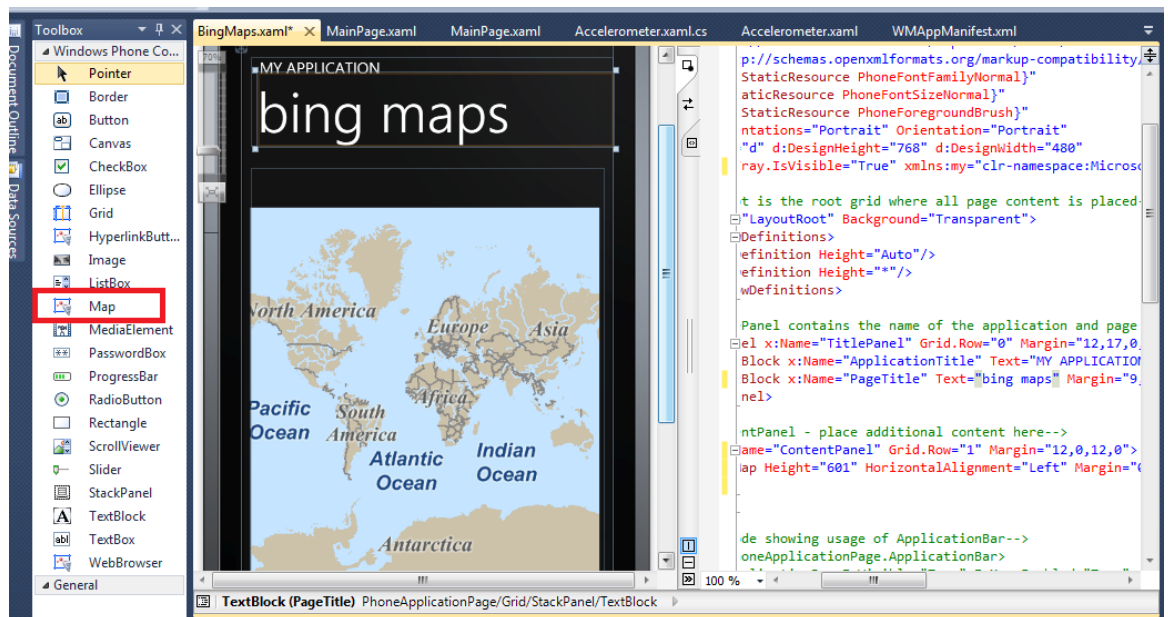
Secure this key for later purposes.

USING BING MAPS CONTROL

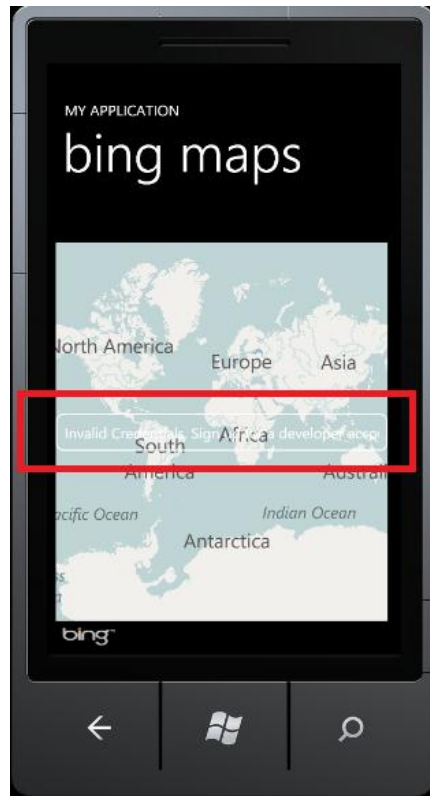
1. If you continue from the previously made project, then add a page to learn about Bing Maps. Otherwise, create a new project for this purpose. Having been doing exercises up to this page, it should be fairly easy to do. The following example uses a previously existing project. Right click on the project, **Add New Item**, select **Windows Phone Portrait Page** and rename the file, in this example **BingMaps.xaml** then select **Add**.



2. Add a map control from the toolbox



3. Change the properties in WMApManifest.xml so that BingMaps.xaml is the initial page for the application. Press F5 and see the results



On the center of the control you will see a warning: "Invalid Credentials. Sign up for a developer account". This indicates that you haven't inserted your key, which will allow you to use Bing maps services.

4. Open **BingMaps.xaml** file and insert your key.

```
<!--ContentPanel - place additional content here-->
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <my:Map CredentialsProvider="<insert your key here>" Height="601"
HorizontalAlignment="Left" Margin="0,6,0,0" Name="map1" VerticalAlignment="Top"
Width="450" />
    </Grid>
```

5. Press F5 and see that the warning has disappeared.



6. Now we add a simple functionality that will automatically add a marker, or more generally known as pushpin, whenever we click on an area on the map. Add an event handler to handle click event on the map.

```
private void map1_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
{
    Pushpin p = new Pushpin()
    {
        Location = map1.ViewportPointToLocation(e.GetPosition(sender as
Map)),
        Content = "marker"
    };
    (sender as Map).Children.Add(p);
}
```

7. Press F5 and see the result. Click anywhere to add a marker on the map.



Pretty simple, isn't it? The addition of map control on Windows Phone gives developers more freedom to enrich user experiences, especially on using maps and other Bing Maps services. The topic about Bing Maps Control Silverlight itself has a very wide scope which will not be discussed further here. If you are interested, you can see references on [MSDN](#) site or interactive SDK for Silverlight.

PART III

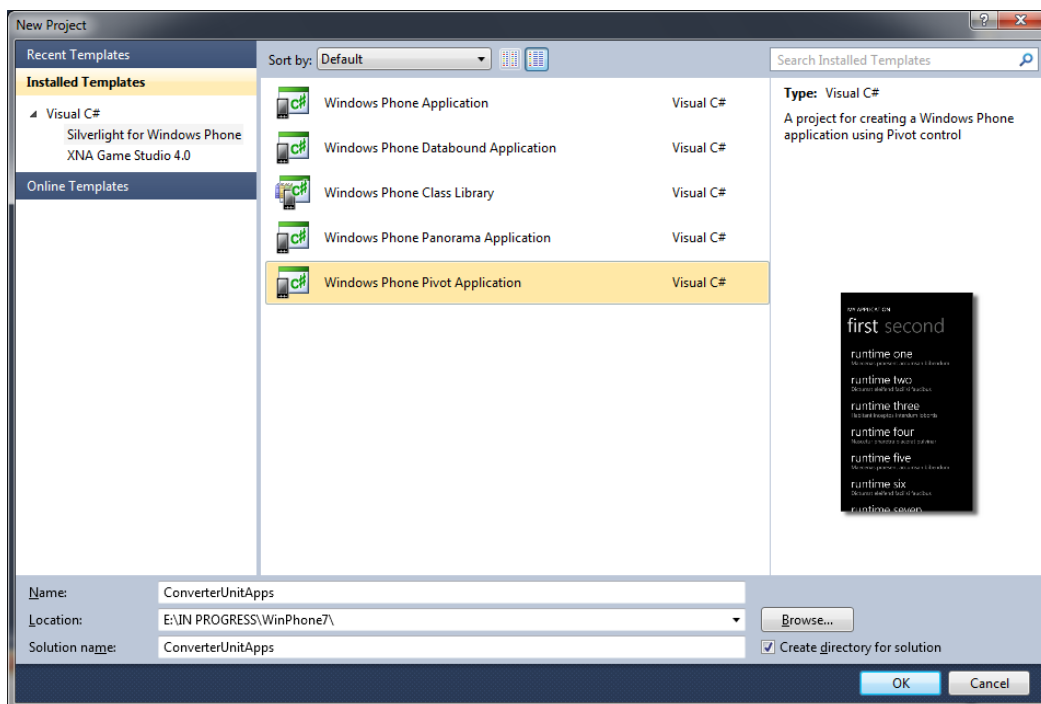
PRACTICE

#1 - UNIT CONVERTER

This Unit Converter application we are making is an application to convert values from one measurement unit to another, for example from *meter* to *feet*, or from *mile* to *kilometer*. Conceptually, we will use several aspect we discussed on the LEARN part, which are using SIP Layout (Digit), IsolatedStorage to store settings, Globalization and Localization, and Pivot as main layout.

PREPARING THE MAIN INTERFACE

1. Open your copy of Visual Studio. To be safe, do this in *Run as Administrator* mode.
2. Create a new project and select **Windows Phone Pivot Application**. Name the project as you like. In this example it's ConverterUnitApps.



3. Configure the **MainPage.xaml** file: name the application, name the PivotItem header, and delete the mark-up from PivotItem content. Now your code should look like this:

```
<!--LayoutRoot is the root grid where all page content is placed-->
<Grid x:Name="LayoutRoot" Background="Transparent">
  <!--Pivot Control-->
  <controls:Pivot Title="CONVERTER UNIT">
    <!--Pivot item one-->
    <controls:PivotItem Header="convert">

  </controls:PivotItem>

    <!--Pivot item two-->
    <controls:PivotItem Header="settings">
```

```

        </controls:PivotItem>
    </controls:Pivot>
</Grid>

```

4. Insert two TextBoxes and two PickerBoxes. Textbox will contain the measurement value input while list picker will provide the measurement units. Add these items into a **PivotItem** named *convert*.

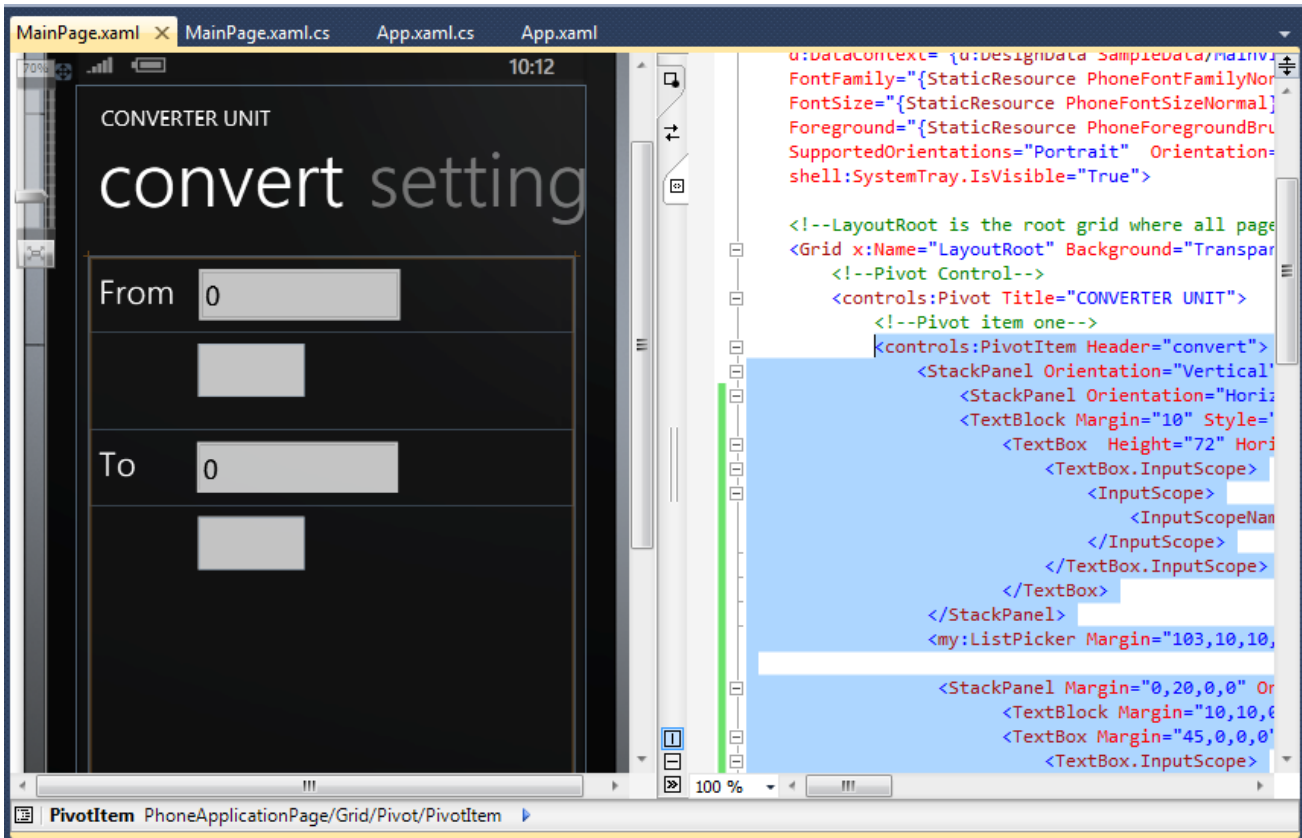
```

<controls:PivotItem Header="convert">
    <StackPanel Orientation="Vertical">
        <StackPanel Orientation="Horizontal">
            <TextBlock Margin="10" Style="{StaticResource
PhoneTextTitle2Style}" Text="From"></TextBlock>
            <TextBox Height="72" HorizontalAlignment="Left"
Margin="0,0,0,0" Name="unitone" Text="0" Width="213" Canvas.Left="0" Canvas.Top="0">
                </TextBox>
        </StackPanel>
        <my:ListPicker Margin="103,10,10,10" FontSize="30" Height="50"
HorizontalAlignment="Left" Name="measurementunitone" Width="100" Padding="5,0,0,0"/>

        <StackPanel Margin="0,20,0,0" Orientation="Horizontal">
            <TextBlock Margin="10,10,0,10" Style="{StaticResource
PhoneTextTitle2Style}" Text="To"></TextBlock>
            <TextBox Margin="45,0,0,0" HorizontalAlignment="Left"
Canvas.Left="0" Canvas.Top="0" Height="72" Name="unittwo" Text="0"
Width="213"></TextBox>
        </StackPanel>
        <my:ListPicker Margin="103,10,10,10" FontSize="30" Height="50"
HorizontalAlignment="Left" Name="measurementunittwo" Width="100" Padding="5,0,0,0" />

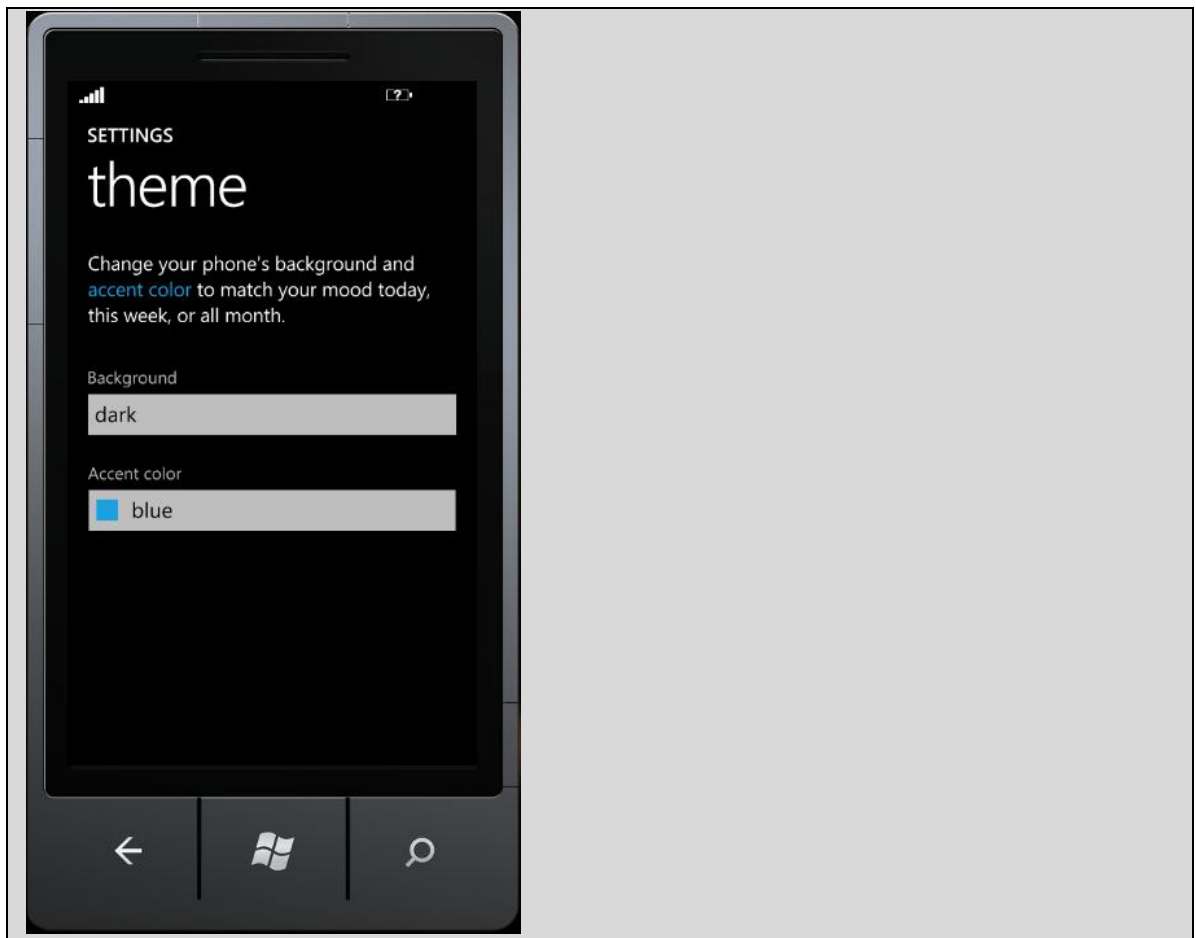
    </StackPanel>
</controls:PivotItem>

```



Note:

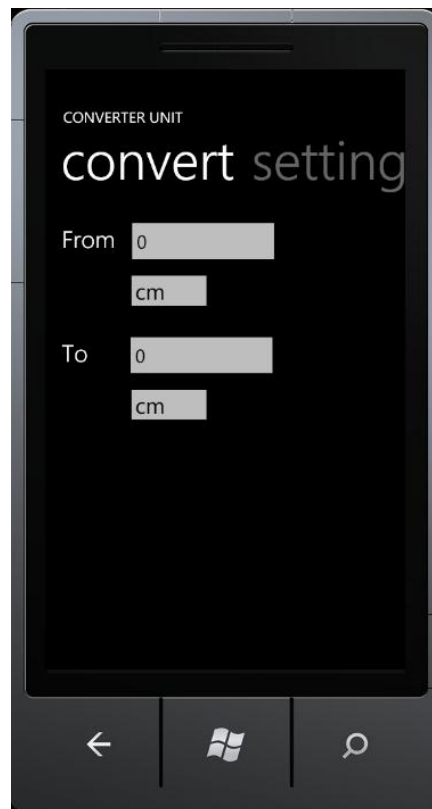
We intentionally don't use ComboBox for unit selection. Although ComboBox is not available in the toolbox, we can actually insert it using XAML code. However, based on Windows Phone UI Guidelines, this control is not a part of Windows Phone UX platform for its natural characteristic that requires mouse/stylus precision. Hence the use of list picker. This control is implemented by Alex Yakhnin here according to the list picker style available in the emulator's setting.



5. Open **MainPage.xaml.cs** file and add the following code to insert items to list picker:

```
public List<String> UnitList = new List<String>() { "mm", "inch", "mile",  
"feet" };  
  
// Constructor  
public MainPage()  
{  
    InitializeComponent();  
  
    // Set the data context of the ListBox control to the sample data  
    DataContext = App.ViewModel;  
    this.Loaded += new RoutedEventHandler(MainPage_Loaded);  
}  
  
// Load data for the ViewModel Items  
private void MainPage_Loaded(object sender, RoutedEventArgs e)  
{  
    if (!App.ViewModel.IsDataLoaded)  
    {  
        App.ViewModel.LoadData();  
    }  
    measurementunitone.ItemsSource = UnitList;  
    measurementunittwo.ItemsSource = UnitList;  
}
```


6. Press F5 and check whether or not the application layout has met our expectations.



CONVERTING

1. To do a conversion we need a converter table. This table stores scales for each units. There are many ways to implement this, but to keep it simple, in this example unit converters will be listed in a dictionary.

```
public Dictionary<string, double> ConverterList = new Dictionary<string, double>();
```

```
private void InitConveterList()
{
    //to cm
    ConverterList.Add("inch-cm", 2.54);
    ConverterList.Add("feet-cm", 30.48);
    ConverterList.Add("mile-cm", 160934);
    ConverterList.Add("cm-cm", 1.0);

    //from cm
    ConverterList.Add("cm-inch", 0.39);
    ConverterList.Add("cm-feet", 0.03);
    ConverterList.Add("cm-mile", 0.0000062);

    //to inch
```

```

ConverterList.Add("feet-inch", 11.8872);
ConverterList.Add("mile-inch", 62764.26);
ConverterList.Add("inch-inch", 1.0);

//from inch
ConverterList.Add("inch-feet", 0.0762);
ConverterList.Add("inch-mile", 0.00001578);

//to feet
ConverterList.Add("mile-feet", 4828.02);
ConverterList.Add("feet-feet", 1.0);

//from feet
ConverterList.Add("feet-mile", 0.0002);

//from mile
ConverterList.Add("mile-mile", 1.0);

}

```

- Next, we add a converter function using the table we've created.

```

private double ConvertMeasurement(String unitone, String unittwo, double value)
{
    double unit = 1.0;
    ConverterList.TryGetValue(unitone + unittwo, out unit);
    return unit * value;
}

```

- Double click on the first TextBox to add an event handler. This event handler processes the input on the TextBox and displays the result in the second TextBox.

```

private void unitone_TextChanged(object sender, TextChangedEventArgs e)
{
    unittwo.Text =
    ConvertMeasurement(measurementunitone.SelectedItem.ToString(),
    measurementunittwo.SelectedItem.ToString(), Double.Parse((sender as
    TextBox).Text)).ToString();
}

```

- Add an input scope and limit the valid input to numbers only. Do this for both TextBoxes.

```

<TextBox Height="72" HorizontalAlignment="Left" Margin="0,0,0,0" Name="unitone"
Text="0" Width="336" Canvas.Left="0" Canvas.Top="0"
TextChanged="unitone_TextChanged" >
    <TextBox.InputScope>

```

```
        <InputScope>
            <InputScopeName
NameValue="Digits"></InputScopeName>
        </InputScope>
    </TextBox.InputScope>
</TextBox>
```

5. To give a better user experience, add an update function when measurements are changed during runtime. Double click on the list picker and add the following code:

```
private void measurementunitone_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    double value;

    if (double.TryParse(unitone.Text, out value))
    {
        unittwo.Text =
ConvertMeasurement(measurementunitone.SelectedItem.ToString(),
measurementunittwo.SelectedItem.ToString(), value).ToString();
    }
    else
    {
        unittwo.Text = "";
    }
}
```

6. Press F5 and observe the results.



ADDING CULTURE LIST

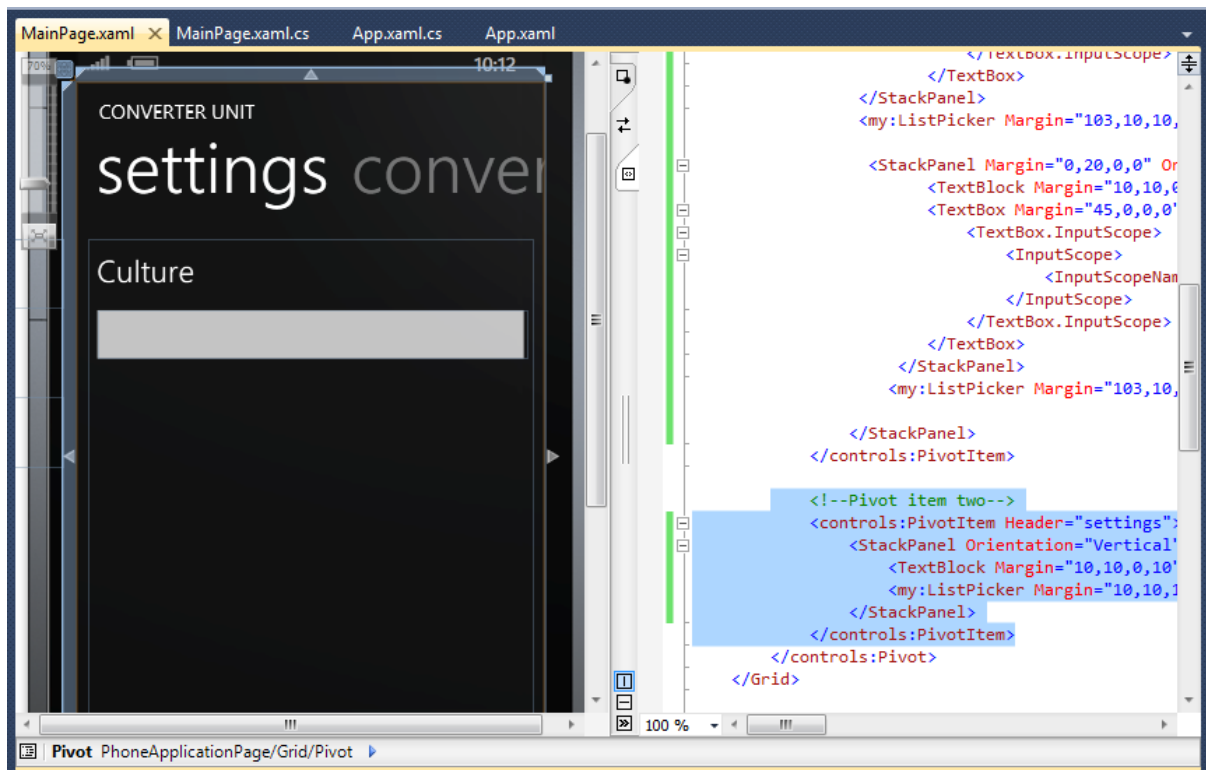
User preference is provided ultimately to adjust to language or culture selection that users want for the application. This will affect the application's presentation and surely the more adjustable it is to users' expectations, the better.

1. Let's prepare culture selections for users to select from. Insert a list picker on the second **PivotItem**.

```

<!--Pivot item two-->
    <controls:PivotItem Header="settings">
        <StackPanel Orientation="Vertical">
            <TextBlock Margin="10,10,0,10" Style="{StaticResource
PhoneTextTitle2Style}" Text="Culture"></TextBlock>
            <my:ListPicker Margin="10,10,10,10" FontSize="30" Height="50"
HorizontalAlignment="Left" Name="culturelist" Width="440" Padding="5,0,0,0" />
        </StackPanel>
    </controls:PivotItem>

```



2. We are going to add culture list for the application. In this example we will only handle two cultures (you can later add more, according to your application's needs), which are US and Indonesia.

```

        public Dictionary<string, string> CultureList = new Dictionary<string,
string>();

private void InitCultureList()
    {
        CultureList.Add("en-US", "United States of America");
        CultureList.Add("id-ID", "Indonesia");
        culturelist.ItemsSource = CultureList;
        culturelist.DisplayMemberPath = "Value";
    }

```

Call for InitCultureList function in **Main_Loaded()**

```

private void MainPage_Loaded(object sender, RoutedEventArgs e)
    {
        if (!App.ViewModel.IsDataLoaded)
        {
            App.ViewModel.LoadData();
        }
        measurementunitone.ItemsSource = UnitList;
        measurementunittwo.ItemsSource = UnitList;
        InitConveterList();
        InitCultureList();
    }

```

```
}
```

3. Press F5 to see results. Select the Setting tab and you can see that users can now select the culture they want to use in the application.

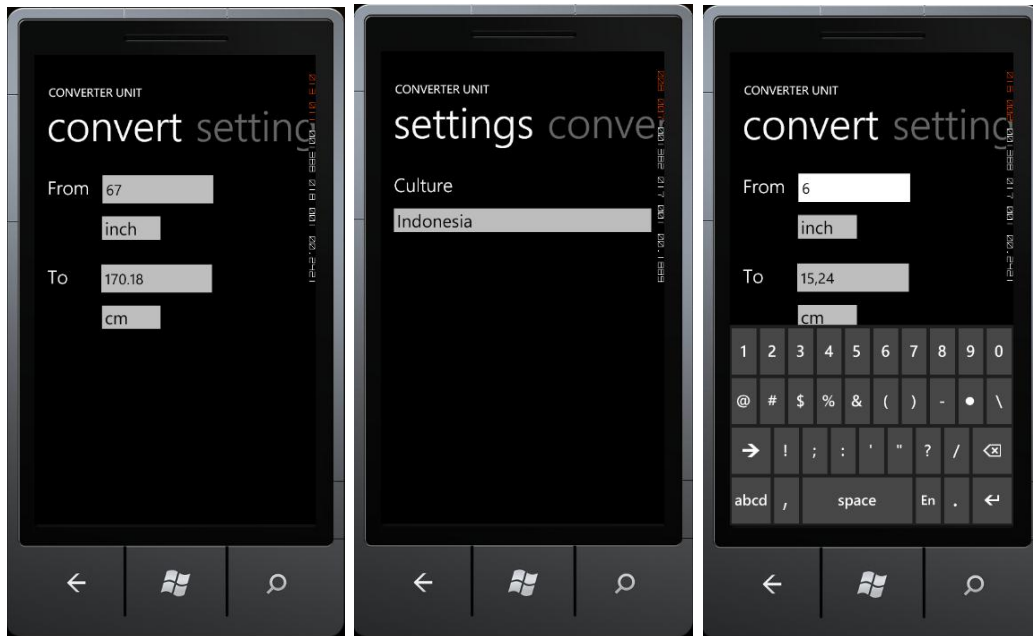


4. Add an event handler to change cultures according to users' selection. Double click on the list picker and add the following code:

```
private void culturelist_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    String culture = ((KeyValuePair<String,
String>)culturelist.SelectedItem).Key.ToString();

    CultureInfo cult = new CultureInfo(culture);
    Thread.CurrentThread.CurrentCulture = cult;
}
```

5. Press F5 and see results. Change the culture and return to the conversion page. If you try out the Indonesian culture, the decimal value separator will change from (.) to (,) .



SAVING USER PREFERENCES

To permanently keep the user preferences for later uses of the application, the data should be stored in IsolatedStorage. This will be done in the Application Closing event so that users don't have to bother saving the preferences manually.

1. Open **App.xaml.cs** and add the following code:

```
IsolatedStorageSettings settings = IsolatedStorageSettings.ApplicationSettings;
    if (!settings.Contains("culture"))
    {
        settings.Add("culture", Thread.CurrentThread.CurrentCulture.Name);
    }
    else
    {
        settings["culture"] = Thread.CurrentThread.CurrentCulture.Name;
    }
    settings.Save();
```

Don't forget to add directive

```
using System.IO.IsolatedStorage;
using System.Threading;
```

2. Next, we need to load the setting when the application is first started. To do this, still in **App.xaml.cs** we add a load function in Application_Loading event handler

```
private void Application_Launching(object sender, LaunchingEventArgs e)
{
```

```

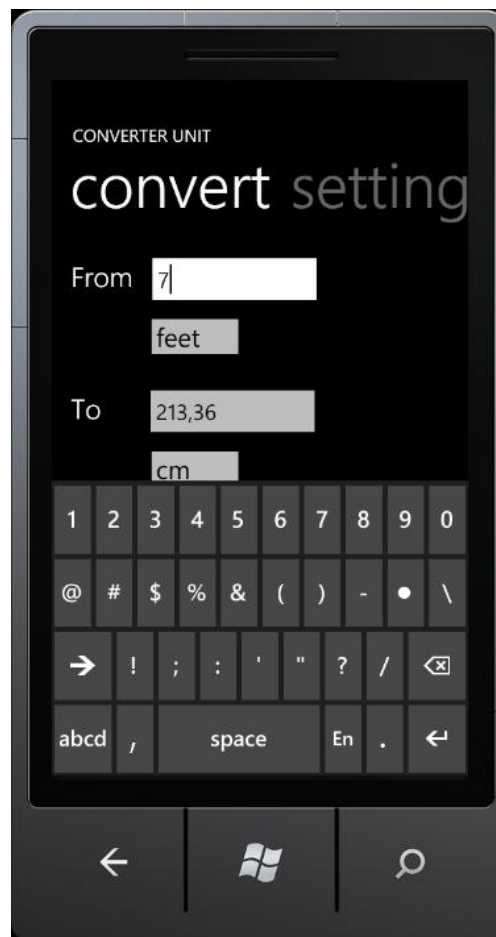
IsolatedStorageSettings settings =
IsolatedStorageSettings.ApplicationSettings;
    string value = "";

    try
    {
        settings.TryGetValue("culture", out value);
        Thread.CurrentThread.CurrentCulture = new
System.Globalization.CultureInfo(value);

    }
    catch
    {
        Debug.WriteLine("error load the culture info..");
    }
}

```

3. Press F5 and see results. Try changing the culture preferences, then close the application and start it again. Does it store your preferred culture?



Up to this point the simple Unit Converter application is successfully built. You can of course develop the application further, like add more measurement units (not only length but also weight et cetera), add more supported languages, or create a better interface.

#2 – STOCK SCREEN

Do you like to invest and try your luck with stocks? If your answer is yes, then you surely need an application that can help you survey companies' stock values in the stock market. In the following exercise we will develop a simple application to see several companies' stock values, do stock analysis, et cetera. For this application, the concepts we will implement are databinding, using application bar, navigating with parameters, and consuming services. The service consumption we will simulate in this example will be stored within our network so that it will not need internet connections.

PREPARING DATA FOR COMPANY STOCK VALUES

In this sample application, we will not use a real-time data, but a previously downloaded one. There's no need to fuss over this matter; if you want to use real-time data, what you need to do is find the data source, Yahoo Finance for example. The use of this downloaded data is just so that this application can be deployed in your machine.

For the data we will use, I would like to personally thank my colleague **Kaisar Siregar**, for lending the stock data and analysis based on the stock analysis technique which was a part of his final project. For those of you who would like to know further about said technique, please contact kaisar.siregar@gmail.com

Here are the data to prepare:

1. Company Stock Data

```
<?xml version="1.0" encoding="utf-8"?>
<Infos>
  <Info>
    <Symbol>MSFT</Symbol>
    <Name>Microsoft Corpora</Name>
    <Close>29.32</Close>
    <Date>4/7/2010</Date>
    <Change>0.1</Change>
  </Info>
  <Info>
    <Symbol>YHOO</Symbol>
    <Name>Yahoo! Inc.</Name>
    <Close>16.92</Close>
    <Date>4/7/2010</Date>
    <Change>-0.06</Change>
  </Info>
  <Info>
    <Symbol>AAPL</Symbol>
    <Name>Apple Inc.</Name>
    <Close>239.54</Close>
    <Date>4/7/2010</Date>
    <Change>1.61</Change>
  </Info>
  <Info>
    <Symbol>GOOG</Symbol>
    <Name>Google Inc.</Name>
```

```
<Close>568.22</Close>
<Date>4/7/2010</Date>
<Change>-2.994</Change>
</Info>
</Infos>
```

Save this under the name **CompanyInfo.xml**

2. Company Transaction Signal Data

```
<?xml version="1.0" encoding="utf-8"?>
<Infos>
  <Info>
    <Symbol>MSFT</Symbol>
    <Name>Microsoft Corpora</Name>
    <Date>4/6/2010</Date>
    <Signal>Buy</Signal>
    <Interval>Daily</Interval>
  </Info>
  <Info>
    <Symbol>YHOO</Symbol>
    <Name>Yahoo! Inc.</Name>
    <Date>4/6/2010</Date>
    <Signal>Buy</Signal>
    <Interval>Daily</Interval>
  </Info>
  <Info>
    <Symbol>AAPL</Symbol>
    <Name>Apple Inc.</Name>
    <Date>4/6/2010</Date>
    <Signal>Buy</Signal>
    <Interval>Daily</Interval>
  </Info>
  <Info>
    <Symbol>GOOG</Symbol>
    <Name>Google Inc.</Name>
    <Date>4/5/2010</Date>
    <Signal>Buy</Signal>
    <Interval>Daily</Interval>
  </Info>
</Infos>
```

Save under the name **CompanySignal.xml**

3. Company Transaction Signal Details

AAPL

```
<?xml version="1.0" encoding="utf-8"?>
<Signals>
  <Signal>
```

```
<Date>4/6/2010</Date>
  <Type>Buy Daily</Type>
</Signal>
<Signal>
  <Date>4/5/2010</Date>
  <Type>Buy Weekly</Type>
</Signal>
<Signal>
  <Date>4/1/2010</Date>
  <Type>Buy Monthly</Type>
</Signal>
<Signal>
  <Date>2/1/2010</Date>
  <Type>Sell Daily</Type>
</Signal>
<Signal>
  <Date>11/30/2009</Date>
  <Type>Sell Weekly</Type>
</Signal>
<Signal>
  <Date>9/21/2009</Date>
  <Type>Buy Weekly</Type>
</Signal>
<Signal>
  <Date>3/30/2009</Date>
  <Type>Sell Weekly</Type>
</Signal>
<Signal>
  <Date>3/9/2009</Date>
  <Type>Buy Weekly</Type>
</Signal>
<Signal>
  <Date>1/26/2009</Date>
  <Type>Sell Weekly</Type>
</Signal>
<Signal>
  <Date>6/2/2008</Date>
  <Type>Sell Monthly</Type>
</Signal>
<Signal>
  <Date>9/1/2006</Date>
  <Type>Buy Monthly</Type>
</Signal>
<Signal>
  <Date>5/1/2006</Date>
  <Type>Sell Monthly</Type>
</Signal>
</Signals>
```

MSFT

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<Signals>
  <Signal>
    <Date>4/6/2010</Date>
    <Type>Buy Daily</Type>
  </Signal>
  <Signal>
    <Date>4/5/2010</Date>
    <Type>Buy Weekly</Type>
  </Signal>
  <Signal>
    <Date>4/1/2010</Date>
    <Type>Sell Daily</Type>
  </Signal>
  <Signal>
    <Date>3/29/2010</Date>
    <Type>Sell Weekly</Type>
  </Signal>
  <Signal>
    <Date>3/29/2010</Date>
    <Type>Buy Daily</Type>
  </Signal>
  <Signal>
    <Date>3/22/2010</Date>
    <Type>Buy Weekly</Type>
  </Signal>
  <Signal>
    <Date>3/1/2010</Date>
    <Type>Buy Monthly</Type>
  </Signal>
  <Signal>
    <Date>2/22/2010</Date>
    <Type>Sell Weekly</Type>
  </Signal>
  <Signal>
    <Date>2/3/2010</Date>
    <Type>Sell Daily</Type>
  </Signal>
  <Signal>
    <Date>2/1/2010</Date>
    <Type>Sell Monthly</Type>
  </Signal>
  <Signal>
    <Date>12/1/2009</Date>
    <Type>Buy Monthly</Type>
  </Signal>
  <Signal>
    <Date>8/24/2009</Date>
    <Type>Buy Weekly</Type>
  </Signal>
  <Signal>
    <Date>4/13/2009</Date>
    <Type>Sell Weekly</Type>
  </Signal>
</Signals>
```

```
<Signal>
  <Date>4/6/2009</Date>
  <Type>Buy Weekly</Type>
</Signal>
<Signal>
  <Date>1/20/2009</Date>
  <Type>Sell Weekly</Type>
</Signal>
<Signal>
  <Date>4/2/2007</Date>
  <Type>Sell Monthly</Type>
</Signal>
<Signal>
  <Date>11/1/2006</Date>
  <Type>Buy Monthly</Type>
</Signal>
<Signal>
  <Date>9/1/2006</Date>
  <Type>Sell Monthly</Type>
</Signal>
</Signals>
```

GOOG

```
<?xml version="1.0" encoding="utf-8"?>
<Signals>
  <Signal>
    <Date>4/5/2010</Date>
    <Type>Buy Weekly</Type>
  </Signal>
  <Signal>
    <Date>4/5/2010</Date>
    <Type>Buy Daily</Type>
  </Signal>
  <Signal>
    <Date>4/1/2010</Date>
    <Type>Buy Monthly</Type>
  </Signal>
  <Signal>
    <Date>3/29/2010</Date>
    <Type>Sell Daily</Type>
  </Signal>
  <Signal>
    <Date>3/22/2010</Date>
    <Type>Sell Weekly</Type>
  </Signal>
  <Signal>
    <Date>3/8/2010</Date>
    <Type>Buy Weekly</Type>
  </Signal>
  <Signal>
    <Date>2/23/2010</Date>
```

```

    <Type>Buy Daily</Type>
  </Signal>
  <Signal>
    <Date>2/10/2010</Date>
    <Type>Sell Daily</Type>
  </Signal>
  <Signal>
    <Date>2/5/2010</Date>
    <Type>Buy Daily</Type>
  </Signal>
  <Signal>
    <Date>2/1/2010</Date>
    <Type>Sell Monthly</Type>
  </Signal>
  <Signal>
    <Date>2/1/2010</Date>
    <Type>Sell Daily</Type>
  </Signal>
  <Signal>
    <Date>12/1/2009</Date>
    <Type>Buy Monthly</Type>
  </Signal>
  <Signal>
    <Date>3/30/2009</Date>
    <Type>Sell Weekly</Type>
  </Signal>
  <Signal>
    <Date>11/17/2008</Date>
    <Type>Buy Weekly</Type>
  </Signal>
  <Signal>
    <Date>11/10/2008</Date>
    <Type>Sell Weekly</Type>
  </Signal>
  <Signal>
    <Date>10/1/2007</Date>
    <Type>Sell Monthly</Type>
  </Signal>
  <Signal>
    <Date>3/1/2007</Date>
    <Type>Buy Monthly</Type>
  </Signal>
  <Signal>
    <Date>9/1/2006</Date>
    <Type>Sell Monthly</Type>
  </Signal>
</Signals>

```

Store each file under the name **CompanySignal-<company_name>.xml**

4. Company Stock Trade Data

AAPL

```

<?xml version="1.0" encoding="utf-8"?>
<Quotes>
  <Ask>241.16</Ask>
  <AverageDailyVolume>22314300</AverageDailyVolume>
  <Bid>241.13</Bid>
  <Change>1.61</Change>
  <ChangeinPercent>0.67</ChangeinPercent>
  <DividendShare>0</DividendShare>
  <DividendYield>0</DividendYield>
  <ExDividendDate>21-Nov-95</ExDividendDate>
  <LastTradePriceOnly>241.15</LastTradePriceOnly>
  <MarketCapitalization>218.7B</MarketCapitalization>
  <Name>Apple Inc.</Name>
  <OneyrTargetPrice>267.54</OneyrTargetPrice>
  <PERatio>23.33</PERatio>
  <PreviousClose>239.54</PreviousClose>
  <TradeDate>4/7/2010 11:25:04 PM</TradeDate>
  <Volume>8277227</Volume>
  <YearHigh>240.24</YearHigh>
  <YearLow>114.19</YearLow>
</Quotes>

```

MSFT

```

<?xml version="1.0" encoding="utf-8"?>
<Quotes>
  <Ask>29.43</Ask>
  <AverageDailyVolume>58268100</AverageDailyVolume>
  <Bid>29.42</Bid>
  <Change>0.1</Change>
  <ChangeinPercent>0.34</ChangeinPercent>
  <DividendShare>0.52</DividendShare>
  <DividendYield>1.77</DividendYield>
  <ExDividendDate>Feb 16</ExDividendDate>
  <LastTradePriceOnly>29.42</LastTradePriceOnly>
  <MarketCapitalization>258.0B</MarketCapitalization>
  <Name>Microsoft Corpora</Name>
  <OneyrTargetPrice>33.73</OneyrTargetPrice>
  <PERatio>16.15</PERatio>
  <PreviousClose>29.32</PreviousClose>
  <TradeDate>4/7/2010 11:25:11 PM</TradeDate>
  <Volume>25815184</Volume>
  <YearHigh>31.5</YearHigh>
  <YearLow>18.47</YearLow>
</Quotes>

```

GOOG

```

<?xml version="1.0" encoding="utf-8"?>
<Quotes>
  <Ask>565.46</Ask>
  <AverageDailyVolume>3645340</AverageDailyVolume>

```



```

<Bid>565.17</Bid>
<Change>-2.994</Change>
<ChangeinPercent>-0.53</ChangeinPercent>
<DividendShare>0</DividendShare>
<DividendYield>0</DividendYield>
<ExDividendDate>N/A</ExDividendDate>
<LastTradePriceOnly>565.226</LastTradePriceOnly>
<MarketCapitalization>179.7B</MarketCapitalization>
<Name>Google Inc.</Name>
<OneyrTargetPrice>673.7</OneyrTargetPrice>
<PERatio>27.83</PERatio>
<PreviousClose>568.22</PreviousClose>
<TradeDate>4/7/2010 11:25:09 PM</TradeDate>
<Volume>904905</Volume>
<YearHigh>629.51</YearHigh>
<YearLow>355.31</YearLow>
</Quotes>

```

Save each file under the name **CompanyQuotes-<company_name>.xml**

5. Subscribed Companies Data

```

<?xml version="1.0" encoding="utf-8"?>
<Subscriptions>
  <Subscription>
    <Symbol>AAPL</Symbol>
    <Name>Apple Inc.</Name>
  </Subscription>
  <Subscription>
    <Symbol>GOOG</Symbol>
    <Name>Google Inc</Name>
  </Subscription>
  <Subscription>
    <Symbol>MSFT</Symbol>
    <Name>Microsoft </Name>
  </Subscription>
</Subscriptions>

```

Save under the name **SubscribtionList.xml**

6. Non-Subscribed Companies Data

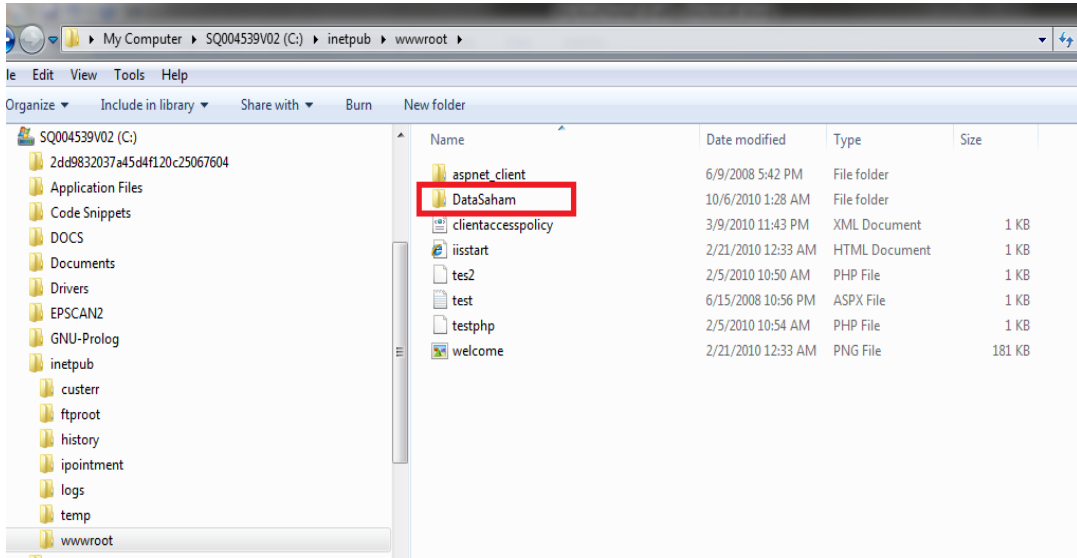
```

<?xml version="1.0" encoding="utf-8"?>
<Subscriptions>
  <Subscription>
    <Symbol>YHOO</Symbol>
    <Name>Yahoo! Inc</Name>
  </Subscription>
</Subscriptions>

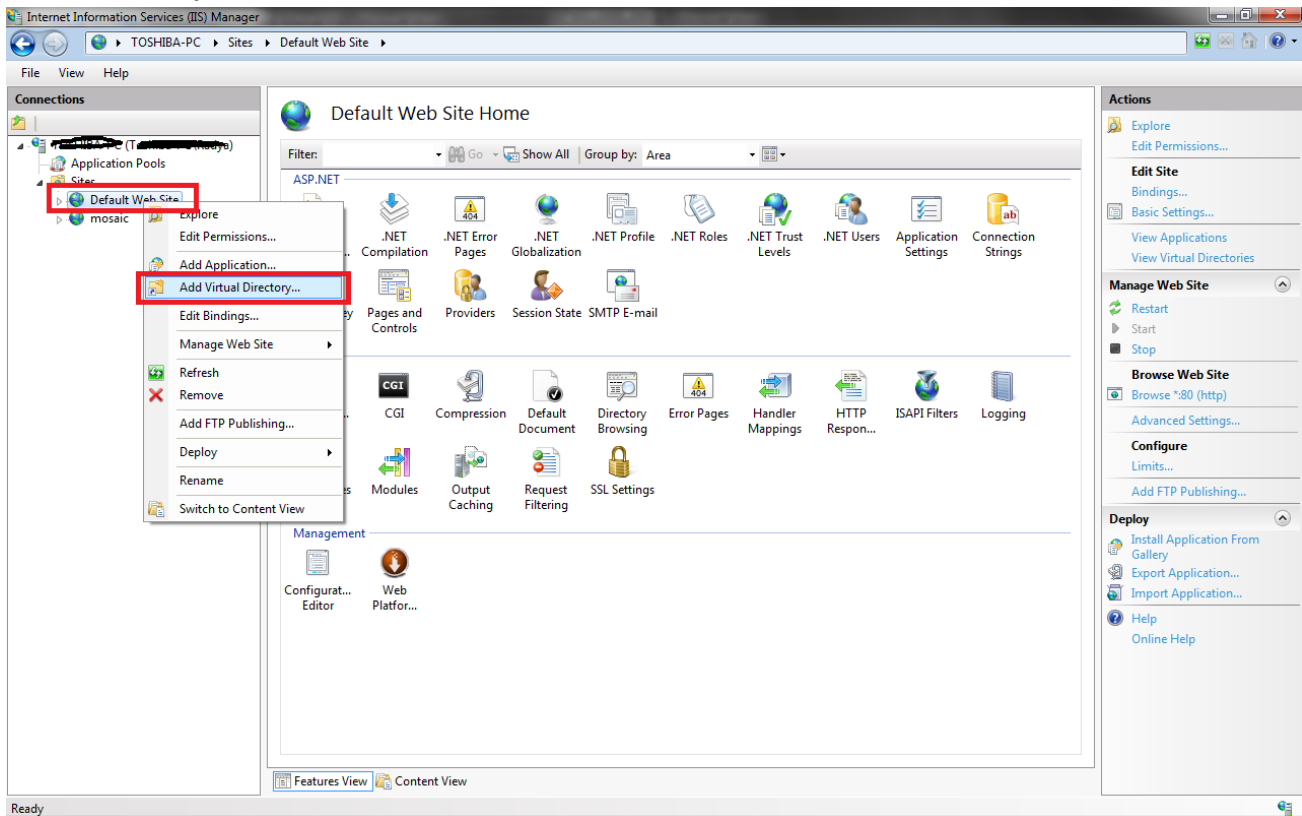
```

Save under the name **UnsubscriptionList.xml**.

- Put all files in one folder. Copy the folder to IIS Root folder. This folder can normally be found in **C:\inetpub\wwwroot**. In this example, the folder is named DataSaham

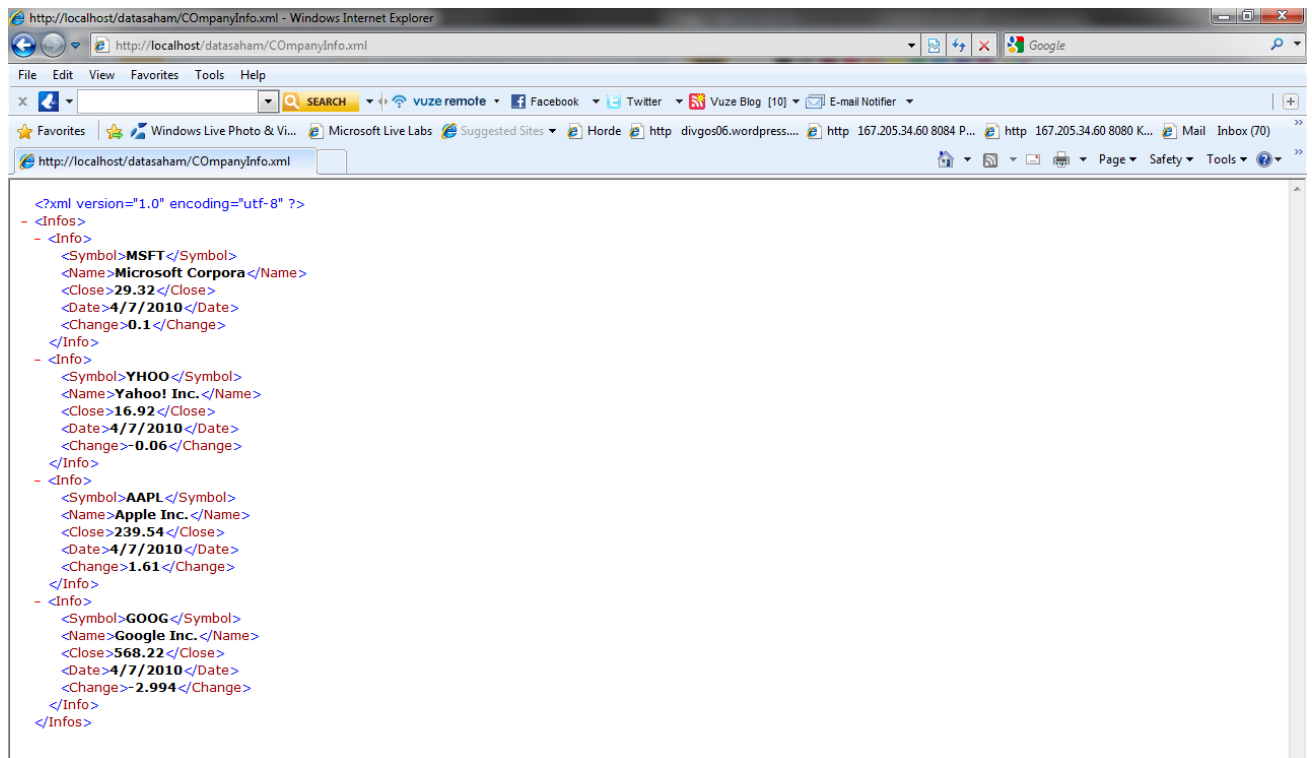


- Open **Internet Information Services (IIS) Manager**, then right click on **Default Web Site** -> **Add Virtual Directory**



Add an alias **data saham** and refer to the previously saved folder for the physical path. Then click OK.

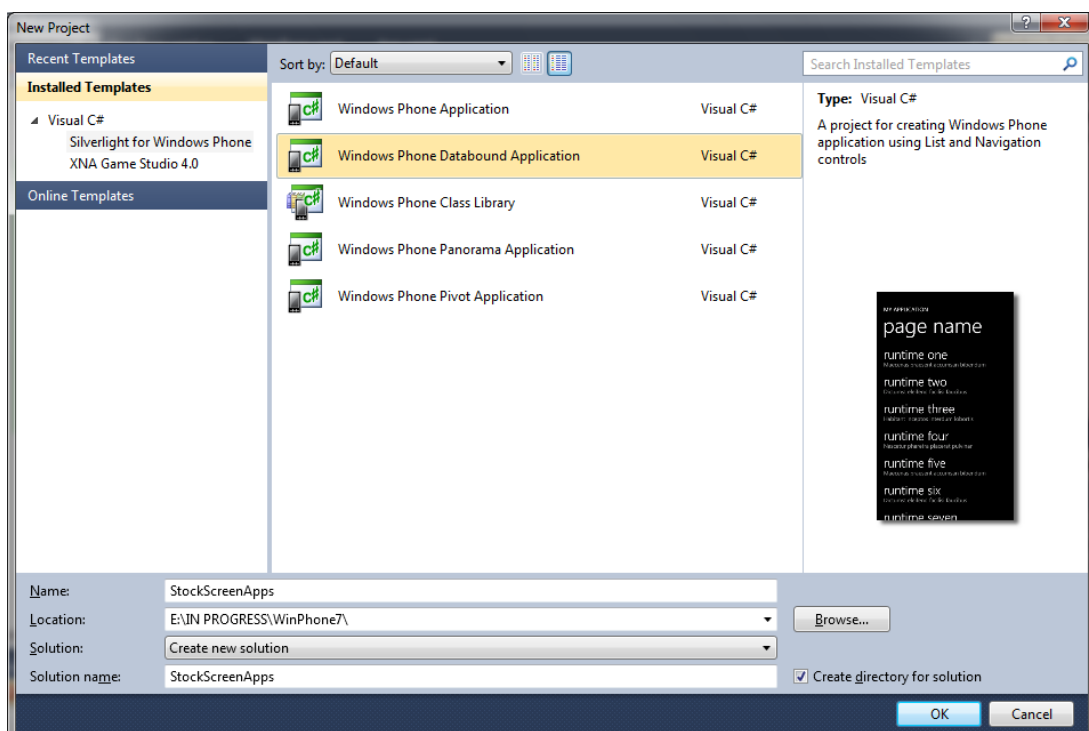
9. To check it, insert the following link to your browser:
<http://localhost/datasaham/CompanyInfo.xml>. This should display the CompanyInfo data.



At this point, data is ready for consumption.

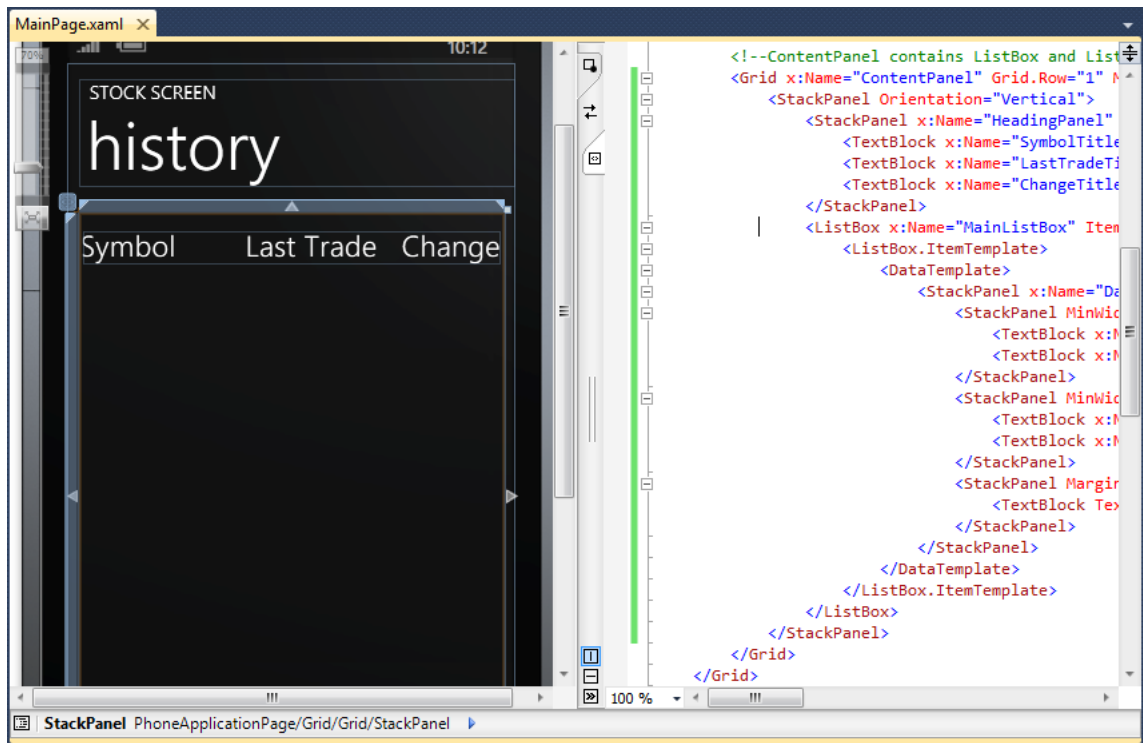
PREPARING COMPANY STOCK LIST PAGE

1. Create a new project, select **Windows Phone Databound Application**. Rename the project as you wish; in this example we name it StockScreenApps

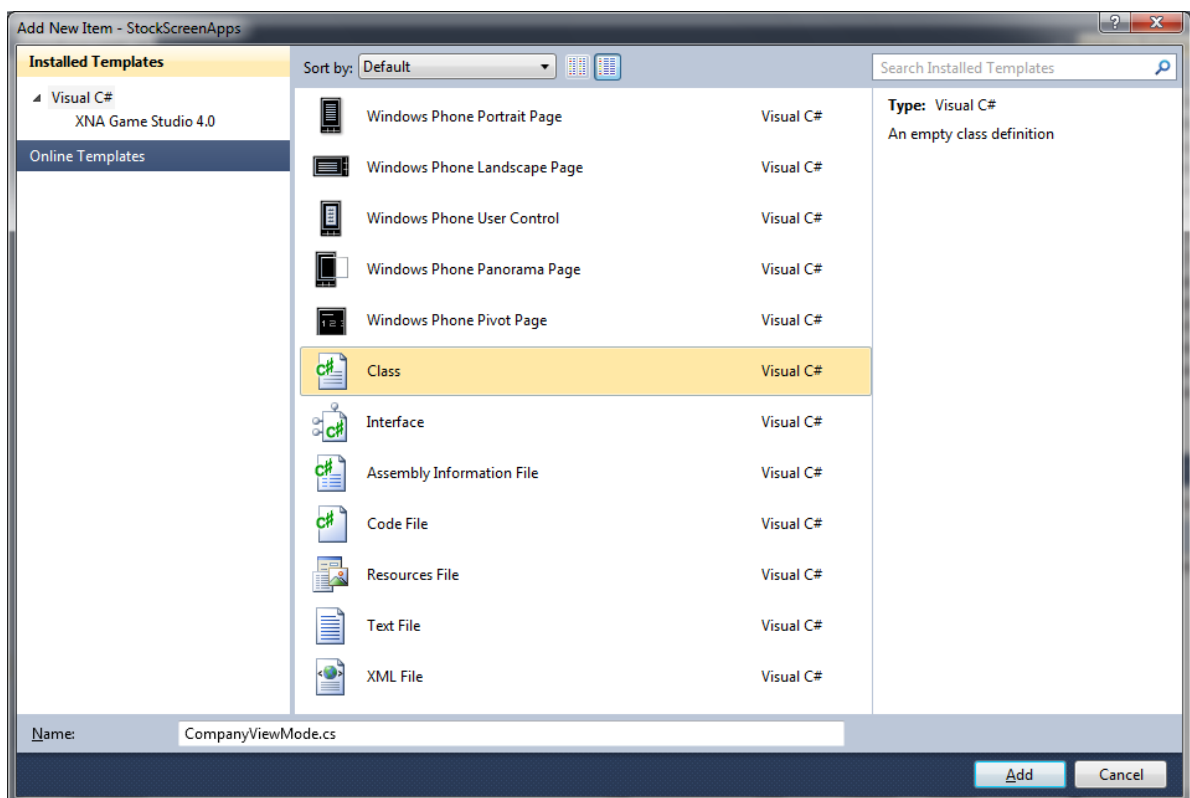


2. Add a title for our application. On the markup, in the content section, add the code below:

```
<StackPanel Orientation="Vertical">
    <StackPanel x:Name="HeadingPanel" Grid.Row="1"
Orientation="Horizontal" Margin="5,20,5,20">
        <TextBlock x:Name="SymbolTitle" Text="Symbol" MinWidth="180"
Margin="-3,-8,0,0" Style="{StaticResource PhoneTextLargeStyle}"/>
        <TextBlock x:Name="LastTradeTitle" Text="Last Trade"
MinWidth="170" Margin="-3,-8,0,0" Style="{StaticResource PhoneTextLargeStyle}"/>
        <TextBlock x:Name="ChangeTitle" Text="Change" Margin="-3,-8,0,0"
Style="{StaticResource PhoneTextLargeStyle}"/>
    </StackPanel>
    <ListBox x:Name="MainListBox" ItemsSource="{Binding Items}"
SelectionChanged="MainListBox_SelectionChanged">
        <ListBox.ItemTemplate>
            <DataTemplate>
                <StackPanel x:Name="DataTemplateStackPanel"
Orientation="Horizontal">
                    <StackPanel MinWidth="180" Margin="5">
                        <TextBlock x:Name="SymbolText" Text="{Binding
CompanySymbol}" Margin="-2,-13,0,0" Style="{StaticResource
PhoneTextExtraLargeStyle}"/>
                        <TextBlock x:Name="NameText" Text="{Binding Name}"
Margin="0,-6,0,3" Style="{StaticResource PhoneTextSubtleStyle}"/>
                    </StackPanel>
                    <StackPanel MinWidth="170" Margin="5">
                        <TextBlock x:Name="CloseText" Text="{Binding
Close}" Margin="-2,-13,0,0" Style="{StaticResource PhoneTextExtraLargeStyle}"/>
                        <TextBlock x:Name="DateText" Text="{Binding Date}"
Margin="0,-6,0,3" Style="{StaticResource PhoneTextSubtleStyle}"/>
                    </StackPanel>
                    <StackPanel Margin="5" MinWidth="80">
                        <TextBlock Text="{Binding Change}" FontSize="28"
HorizontalAlignment="Right"></TextBlock>
                    </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </StackPanel>
```



- Now we will start working with the data. On the previous part, it was stated that Silverlight for Windows Phone has an awesome databinding feature that can help us create a cleaner code. We should take an advantage from this, combined with MVVM pattern. Insert a new class in **ViewModels** folder, to do this, right click and select **Add New Class**. Name it **CompanyViewModel**.



4. This class contains the view model for CompanyInfo which consist of a number of properties. The contents of this class are:

Add `InotifyPropertyChanged` interface so that databinding can be done from two directions.

```
private String companySymbol;
public String CompanySymbol
{
    get { return companySymbol; }
    set
    {
        if (value != companySymbol)
        {
            companySymbol = value;
            NotifyPropertyChanged("CompanySymbol");
        }
    }
}

private String name;
public String Name
{
    get { return name; }
    set {
        if (value != name)
        {
            name = value;
            NotifyPropertyChanged("Name");
        }
    }
}

private Boolean isSubscribed;
public Boolean IsSubscribed
{
    get { return isSubscribed; }
    set
    {
        if (value != isSubscribed)
        {
            isSubscribed = value;
            NotifyPropertyChanged("IsSubscribed");
        }
    }
}

private String date;
public String Date
{
    get { return date; }
    set
```

```

    {
        if (value != date)
        {
            date = value;
            NotifyPropertyChanged("Date");
        }
    }

private String close;
public String Close
{
    get { return close; }
    set
    {
        if (value != close)
        {
            close = value;
            NotifyPropertyChanged("Close");
        }
    }
}

private String change;
public String Change
{
    get { return change; }
    set
    {
        if (value != change)
        {
            change = value;
            NotifyPropertyChanged("Change");
        }
    }
}

public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged(String propertyName)
{
    PropertyChangedEventHandler handler = PropertyChanged;
    if (null != handler)
    {
        handler(this, new PropertyChangedEventArgs(propertyName));
    }
}
}

```

5. Add a class that will act as MainClass for ViewModel Company Info. Right click on the ViewModel folder and add MainCompanyViewModel class. This class will contain the following two matters:

```
public MainCompanyViewModel()
```

```

{
    // Insert code required on object creation below this point
    Items = new ObservableCollection<CompanyViewModel>();
}

public ObservableCollection<CompanyViewModel> Items { get; set; }

public event PropertyChangedEventHandler PropertyChanged;
private void NotifyPropertyChanged(String propertyName)
{
    PropertyChangedEventHandler handler = PropertyChanged;
    if (null != handler)
    {
        handler(this, new PropertyChangedEventArgs(propertyName));
    }
}
}

```

This class contains an item collection of CompanyInfo type. This class will later be bound to a ListBox in the main page.

- Now we declare MainCompanyViewModel property in **MainPage.xaml.cs** class, and call for web service using WebClient by entering URI: <http://localhost/datasaham/CompanyInfo.xml>

```

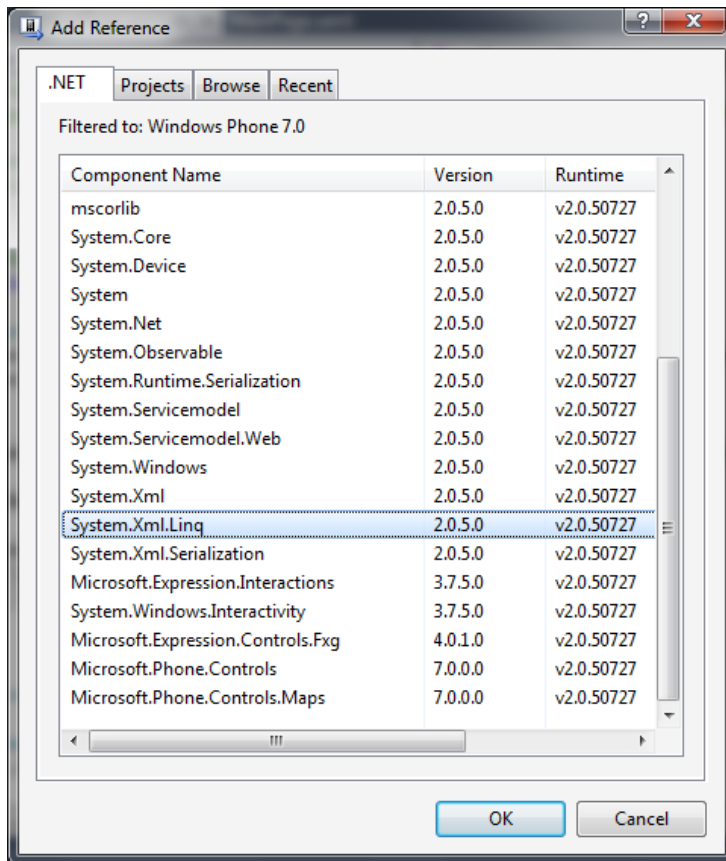
MainCompanyViewModel stockHistoryViewModel = new MainCompanyViewModel();
string uri = String.Format("http://localhost/datasaham/CompanyInfo.xml");

// Load data for the ViewModel Items
private void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    WebClient wc = new WebClient();
    wc.DownloadStringCompleted += new
DownloadStringCompletedEventHandler(wc_DownloadStringCompleted);
    wc.DownloadStringAsync(new Uri(uri));
}

void wc_DownloadStringCompleted(object sender, DownloadStringCompletedEventArgs
e)
{
    if (e.Result != null)
    {
        stockHistoryViewModel = ParseStockHistoryFromXML(e.Result);
        if (DataContext == null)
            DataContext = stockHistoryViewModel;
    }
}
}

```

- The data consumption will return an XML. Therefore we need to do parsing using LINQ. Add the **System.Linq.Xml.dll**, right click on Reference and select Add Reference.



8. Add the following code. What this function does is fetch data from CompanyInfo, parse the data, and store it in the related class.

```

public MainCompanyViewModel ParseStockHistoryFromXML(String result)
{
    MainCompanyViewModel retVal = new MainCompanyViewModel();
    retVal.Items.Clear();

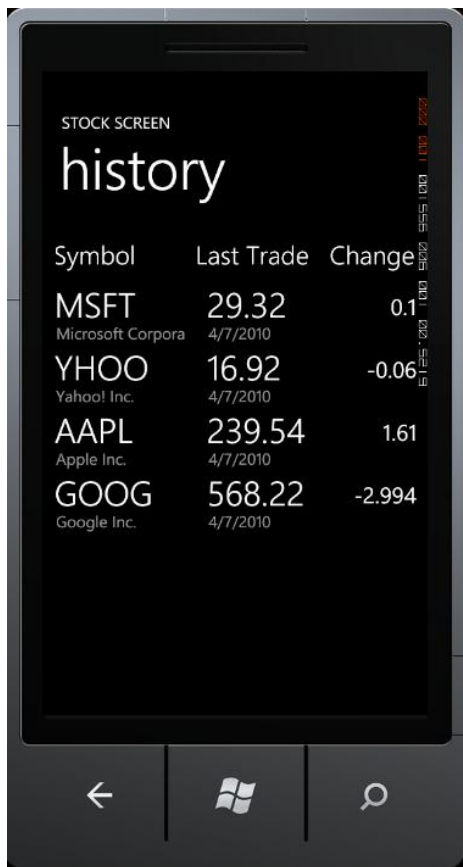
    XDocument xdoc = XDocument.Parse(result);

    int i = 0;
    foreach (var x in xdoc.Descendants("Info"))
    {
        i++;
        CompanyViewModel company = new CompanyViewModel()
        {
            CompanySymbol = x.Element("Symbol").Value,
            Name = x.Element("Name").Value,
            Close = x.Element("Close").Value,
            Date = x.Element("Date").Value,
            Change = x.Element("Change").Value
        };
        retVal.Items.Add(company);
    }
    return retVal;
}

```

```
}
```

9. Press F5 to see how the application works.

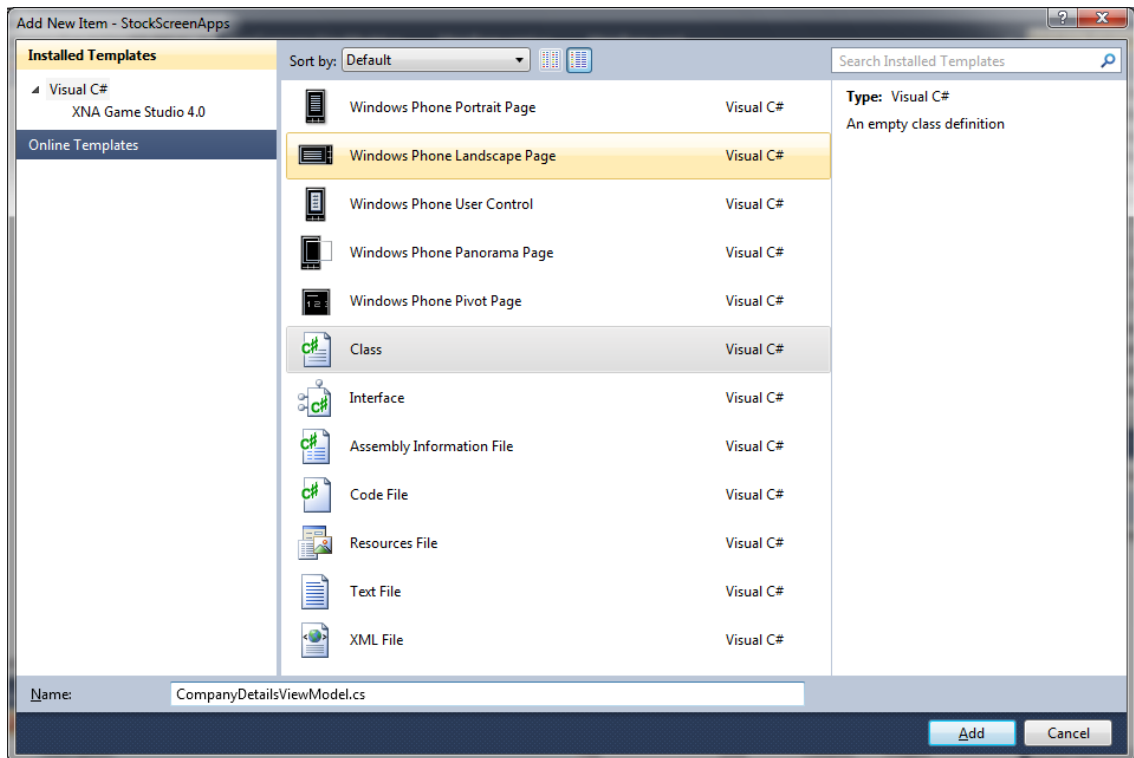


In a moment, your application will display the data you need for stock trading :) As you can see, with the use of databinding, displaying data is a really simple thing to do. Just set the data context accordingly, and Silverlight will do the rest.

COMPANY STOCK DETAIL NAVIGATION

The company stock detail will be displayed when users select one of the companies in the list. For this purpose we will pass parameters using Windows Phone navigation.

1. Prepare the `CompanyDetailsViewModel` class to store data to be displayed. Right click on the `ViewModel` folder and select **Add Class**. Name it `CompanyDetailsViewModel.cs`



2. Insert the following code:

```
public class CompanyDetailsViewModel : CompanyViewModel, INotifyPropertyChanged
{
    public CompanyDetailsViewModel()
    {
    }

    private String prevchange;
    public String PrevChange
    {
        get { return prevchange; }
        set
        {
            if (value != prevchange)
            {
                prevchange = value;
                NotifyPropertyChanged("PrevChange");
            }
        }
    }

    private String volume;
    public String Volume
    {
        get { return volume; }
        set
        {
```

```

        if (value != volume)
        {
            volume = value;
            NotifyPropertyChanged("Volume");
        }
    }

private String avgvolume;
public String AvgVolume
{
    get { return avgvolume; }
    set
    {
        if (value != avgvolume)
        {
            avgvolume = value;
            NotifyPropertyChanged("AvgVolume");
        }
    }
}

private String prevclose;
public String Prevclose
{
    get { return prevclose; }
    set
    {
        if (value != prevclose)
        {
            prevclose = value;
            NotifyPropertyChanged("PrevClose");
        }
    }
}

private String oneYearTarget;
public String OneYearTarget
{
    get { return oneYearTarget; }
    set
    {
        if (value != oneYearTarget)
        {
            oneYearTarget = value;
            NotifyPropertyChanged("OneYearTarget");
        }
    }
}

private String marketCapital;
public String MarketCapital
{
    get { return marketCapital; }
}

```

```

        set
        {
            if (value != marketCapital)
            {
                marketCapital = value;
                NotifyPropertyChanged("MarketCapital");
            }
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    private void NotifyPropertyChanged(String propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (null != handler)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}

```

This creates a container class that inherits several properties from CompanyViewModel class.

3. Open **DetailsPage.xaml** file. Change XAML markup in **ContentPage** so that it looks like this:

```

<Grid x:Name="ContentPanel" Grid.Row="1">
    <ScrollView HorizontalScrollBarVisibility="Auto">
        <StackPanel Orientation="Vertical" >
            <StackPanel Margin="0,5,0,5" Orientation="Horizontal">
                <TextBlock MinWidth="200" Text="Company" Style="{StaticResource
PhoneTextLargeStyle}"></TextBlock>
                <TextBlock Text="{Binding Name}" Style="{StaticResource
PhoneTextLargeStyle}" Foreground="Gray"></TextBlock>
            </StackPanel>
            <StackPanel Margin="0,5,0,5" Orientation="Horizontal">
                <TextBlock MinWidth="200" Text="Last Trade" Style="{StaticResource
PhoneTextLargeStyle}"></TextBlock>
                <TextBlock Text="{Binding Close}" Style="{StaticResource
PhoneTextLargeStyle}" Foreground="Gray"></TextBlock>
            </StackPanel>
            <StackPanel Margin="0,5,0,5" Orientation="Horizontal">
                <TextBlock MinWidth="200" Text="Trade Date" Style="{StaticResource
PhoneTextLargeStyle}"></TextBlock>
                <TextBlock Text="{Binding Date}" Style="{StaticResource
PhoneTextLargeStyle}" Foreground="Gray"></TextBlock>
            </StackPanel>
            <StackPanel Margin="0,5,0,5" Orientation="Horizontal">
                <TextBlock MinWidth="200" Text="Change" Style="{StaticResource
PhoneTextLargeStyle}"></TextBlock>
                <TextBlock Text="{Binding Change}" Style="{StaticResource
PhoneTextLargeStyle}" Foreground="Gray"></TextBlock>
            </StackPanel>
        </StackPanel>
    </ScrollView>
</Grid>

```

```

        <StackPanel Margin="0,5,0,5" Orientation="Horizontal">
            <TextBlock MinWidth="200" Text="Change (%)" Style="{StaticResource
PhoneTextLargeStyle}"></TextBlock>
            <TextBlock Text="{Binding PrevChange}" Style="{StaticResource
PhoneTextLargeStyle}" Foreground="Gray"></TextBlock>
        </StackPanel>
        <StackPanel Margin="0,5,0,5" Orientation="Horizontal">
            <TextBlock MinWidth="200" Text="Volume" Style="{StaticResource
PhoneTextLargeStyle}"></TextBlock>
            <TextBlock Text="{Binding Volume}" Style="{StaticResource
PhoneTextLargeStyle}" Foreground="Gray"></TextBlock>
        </StackPanel>
        <StackPanel Margin="0,5,0,5" Orientation="Horizontal">
            <TextBlock MinWidth="200" Text="Avg Volume"
Style="{StaticResource PhoneTextLargeStyle}"></TextBlock>
            <TextBlock Text="{Binding AvgVolume}" Style="{StaticResource
PhoneTextLargeStyle}" Foreground="Gray"></TextBlock>
        </StackPanel>
        <StackPanel Margin="0,5,0,5" Orientation="Horizontal">
            <TextBlock MinWidth="200" Text="Prev Close"
Style="{StaticResource PhoneTextLargeStyle}"></TextBlock>
            <TextBlock Text="{Binding Prevclose}" Style="{StaticResource
PhoneTextLargeStyle}" Foreground="Gray"></TextBlock>
        </StackPanel>
        <StackPanel Margin="0,5,0,5" Orientation="Horizontal">
            <TextBlock MinWidth="200" Text="1 Year Target"
Style="{StaticResource PhoneTextLargeStyle}"></TextBlock>
            <TextBlock Text="{Binding OneYearTarget}"
Style="{StaticResource PhoneTextLargeStyle}" Foreground="Gray"></TextBlock>
        </StackPanel>
        <StackPanel Margin="0,5,0,5" Orientation="Horizontal">
            <TextBlock MinWidth="200" Text="Market Capital"
Style="{StaticResource PhoneTextLargeStyle}"></TextBlock>
            <TextBlock Text="{Binding MarketCapital}"
Style="{StaticResource PhoneTextLargeStyle}" Foreground="Gray"></TextBlock>
        </StackPanel>
    </StackPanel>
</ScrollViewer>
<!--<TextBlock x:Name="ContentText" Text="{Binding LineThree}"
TextWrapping="Wrap" Margin="24,10,24,24" Style="{StaticResource
PhoneTextTitle3Style}"/>-->
</Grid>

```

Don't forget to change the title for the page that we will bind into the selected company name.

```

<!--TitlePanel contains the name of the application and page title-->
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="PageTitle" Text="STOCK SCREEN" Style="{StaticResource
PhoneTextNormalStyle}"/>
        <TextBlock x:Name="ListTitle" Text="{Binding CompanySymbol}" Margin="9,-
7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

```

4. Open **DetailsPage.xaml.cs** file, then add the code below:

```
string selectedCompany = "";
```

Declare the selected company.

```
// When page is navigated to, set data context to selected item in list
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    if (NavigationContext.QueryString.TryGetValue("selectedItem", out
selectedCompany))
    {
        WebClient wc = new WebClient();
        String uri = String.Format("http://localhost/datasaham/CompanyQuote-
{0}.xml", selectedCompany);
        wc.DownloadStringCompleted += new
DownloadStringCompletedEventHandler(wc_DownloadStringCompleted);
        wc.DownloadStringAsync(new Uri(uri));
    }
}
```

Calling for web service using WebClient. We fetch the parameter from the previous page using string query on navigation. Here we use the navigation ability of Silverlight for Windows Phone.

```
void wc_DownloadStringCompleted(object sender, DownloadStringCompletedEventArgs e)
{
    if (e.Result != null)
    {
        CompanyDetailsViewModel compDetail =
ParseCompanyInfoFromXML(e.Result);
        DataContext = compDetail;
        LayoutRoot.Visibility = System.Windows.Visibility.Visible;
    }
}

public CompanyDetailsViewModel ParseCompanyInfoFromXML(String result)
{
    CompanyDetailsViewModel companyDetail = null;
    XDocument xdoc = XDocument.Parse(result);

    try
    {
        companyDetail = new CompanyDetailsViewModel()
        {
            Name = xdoc.Element("Quotes").Element("Name").Value,
            AvgVolume =
xdoc.Element("Quotes").Element("AverageDailyVolume").Value,
            Change = xdoc.Element("Quotes").Element("Change").Value,

```

```

        Prevclose = xdoc.Element("Quotes").Element("PreviousClose").Value,
        Date = xdoc.Element("Quotes").Element("TradeDate").Value,
        MarketCapital =
xdoc.Element("Quotes").Element("MarketCapitalization").Value,
        OneYearTarget =
xdoc.Element("Quotes").Element("OneyrTargetPrice").Value,
        Volume = xdoc.Element("Quotes").Element("Volume").Value,
        PrevChange =
xdoc.Element("Quotes").Element("ChangeinPercent").Value,
        Close =
xdoc.Element("Quotes").Element("LastTradePriceOnly").Value,
        CompanySymbol = selectedCompany
    };
    return companyDetail;
}
catch (Exception e)
{
    return null;
}
}

```

At this point, we parse the result data from the consumed web service using LINQ to XML. This also sets the necessary data context.

5. Open **MainPage.xaml.cs** page. In function **MainListBox_SelectionChanged** add the following code:

```

// If selected index is -1 (no selection) do nothing
if (MainListBox.SelectedIndex == -1)
    return;

// Navigate to the new page
NavigationService.Navigate(new Uri("/DetailsPage.xaml?selectedItem=" +
(MainListBox.SelectedItem as CompanyViewModel).CompanySymbol, UriKind.Relative));

// Reset selected index to -1 (no selection)
MainListBox.SelectedIndex = -1;

```

What this function does is fetch the company name which item was selected by user and pass it to **DetailsPage.xaml** page.

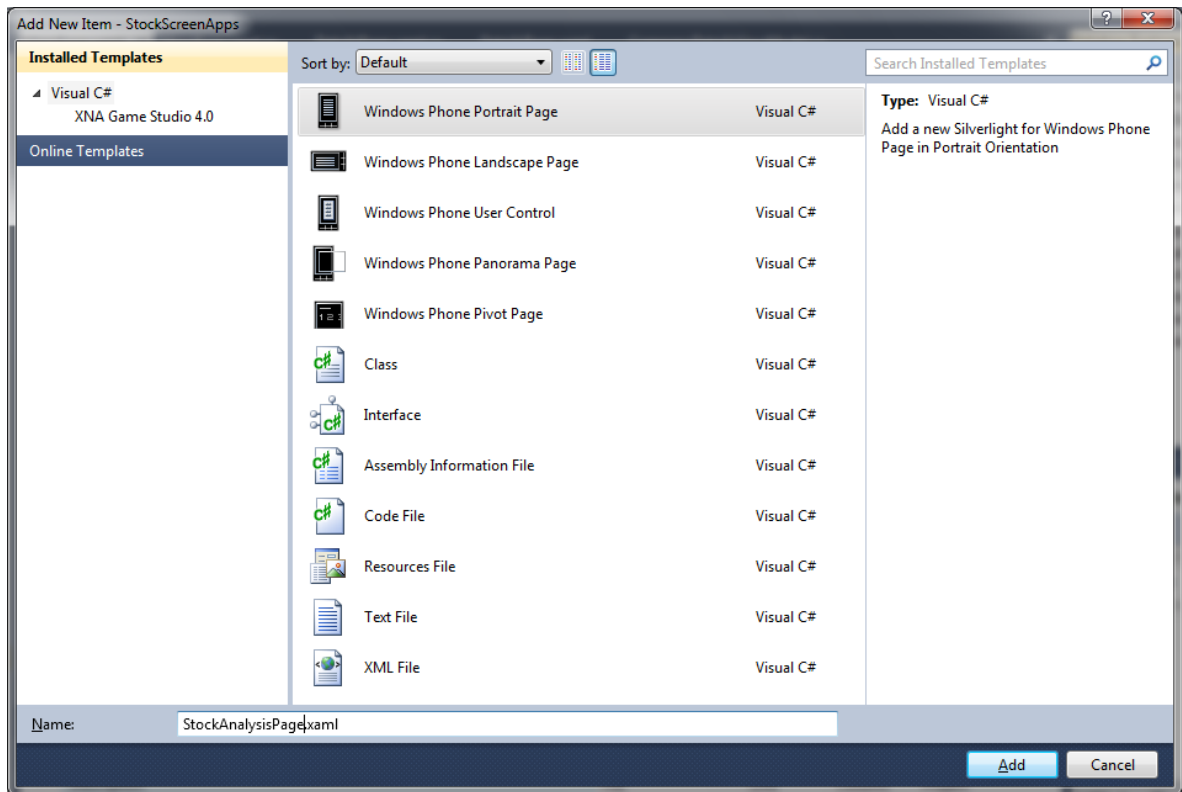
6. Press F5 for result. Select one of the companies whose stock data we want to view.



PREPARING THE STOCK TRANSACTION SIGNAL PAGE

Next we will add a page to show transaction signals which will be the guide whether or not we should buy a company's stock. Transaction signal used in this application is processed using MESA Sine-wave technique.

1. Add a **Windows Phone Portrait Page** and name it **StockAnalysisPage.xaml**



2. Configure application name, page title, and page content. Use the XAML code below:

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>

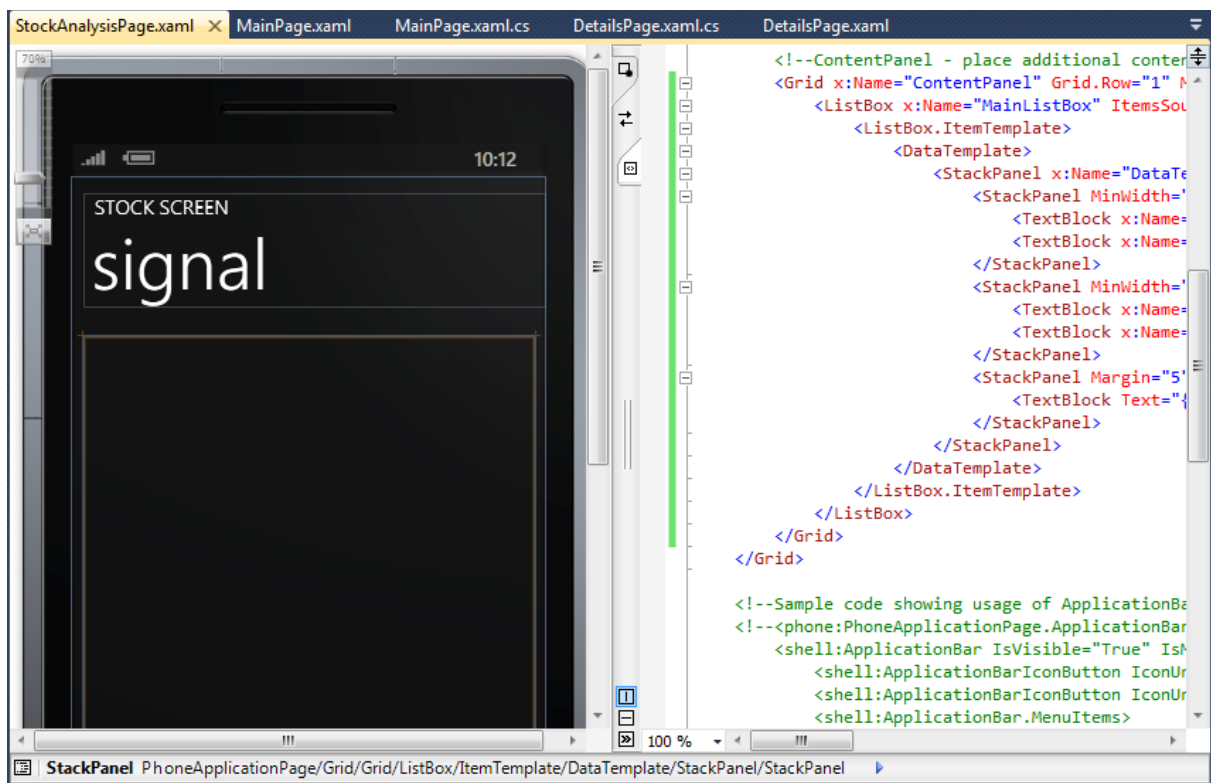
    <!--TitlePanel contains the name of the application and page title-->
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="STOCK SCREEN"
Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="signal" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <!--ContentPanel - place additional content here-->
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <ListBox x:Name="MainListBox" ItemsSource="{Binding Items}"
SelectionChanged="MainListBox_SelectionChanged">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel x:Name="DataTemplateStackPanel"
Orientation="Horizontal">
                        <StackPanel MinWidth="180" Margin="5">
                            <TextBlock x:Name="SymbolText" Text="{Binding
CompanySymbol}" Margin="-2,-13,0,0" Style="{StaticResource
PhoneTextExtraLargeStyle}"/>
                            <TextBlock x:Name="NameText" Text="{Binding Name}"
Margin="0,-6,0,3" Style="{StaticResource PhoneTextSubtleStyle}"/>
                        </StackPanel>
                    </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </Grid>
</Grid>
```

```

        </StackPanel>
        <StackPanel MinWidth="170" Margin="5">
            <TextBlock x:Name="CloseText" Text="{Binding Close}"
Margin="-2,-13,0,0" Style="{StaticResource PhoneTextExtraLargeStyle}"/>
            <TextBlock x:Name="DateText" Text="{Binding Date}"
Margin="0,-6,0,3" Style="{StaticResource PhoneTextSubtleStyle}"/>
        </StackPanel>
        <StackPanel Margin="5" MinWidth="80">
            <TextBlock Text="{Binding Change}" FontSize="28"
HorizontalAlignment="Right"></TextBlock>
        </StackPanel>
    </StackPanel>
</DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
</Grid>
</Grid>

```



3. Create a class to contain company signal data we fetch so that it can be easily bound to UI. Since the data structure stored is similar to company data in **CompanyViewModel.cs** class, then for this purpose let's reuse the class. For the real deal this is of course not recommended.
4. Open **StockAnalysisPage.xaml.cs** and add the following code:

Declare *MainViewModel* to contain analysis data.

```
public MainCompanyViewModel StockAnalysis;
```

Fetch data using WebClient

```
void StockAnalysisPage_Loaded(object sender, RoutedEventArgs e)
```

```

    {
        if (StockAnalysis == null)
        {
            StockAnalysis = new MainCompanyViewModel();
            WebClient webClient = new WebClient();
            webClient.DownloadStringCompleted += new
DownloadStringCompletedEventHandler(webClient_DownloadStringCompleted);
            webClient.DownloadStringAsync(new
Uri("http://localhost/datasaham/CompanySignal.xml"));

        }
    }

    void webClient_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
    {
        if (e.Result != null)
        {
            StockAnalysis = ParseStockAnalysisFromXML(e.Result);
            DataContext = StockAnalysis;
        }
    }
}

```

Parse the fetched data so that it matches the container class's structure. Use LINQ to XML.

```

public MainCompanyViewModel ParseStockAnalysisFromXML(String result)
{
    MainCompanyViewModel retVal = new MainCompanyViewModel();
    retVal.Items.Clear();

    XDocument xdoc = XDocument.Parse(result);

    int i = 0;
    foreach (var x in xdoc.Descendants("Info"))
    {
        i++;
        CompanyViewModel company = new CompanyViewModel()
        {
            CompanySymbol = x.Element("Symbol").Value,
            Name = x.Element("Name").Value,
            Close = x.Element("Signal").Value,
            Date = x.Element("Date").Value,
            Change = x.Element("Interval").Value

        };
        retVal.Items.Add(company);
    }
    return retVal;
}

```

CREATING APPLICATION NAVIGATION USING APPLICATION BAR

Now we have two pages and therefore it is mandatory to have a means for navigating through pages. Let's use our knowledge on Application Bar that we have discussed in previous section. Since we want an Application Bar that is consistent in every page, we will use Global Application Bar.

1. Open **App.xaml** and add the code below:

```
<!--Application Resources-->
<Application.Resources>
  <shell:ApplicationBar x:Name="globalAppBar" x:Key="globalAppBar"
  IsVisible="True" IsMenuEnabled="True" Opacity="1">
    <shell:ApplicationBar.MenuItems>
      <shell:ApplicationBarMenuItem x:Name="stockHistoryItem"
  Click="stockHistoryItem_Click" Text="Stock History"></shell:ApplicationBarMenuItem>
      <shell:ApplicationBarMenuItem x:Name="stockAnalysisItem" Text="Stock
  Analysis" Click="stockAnalysisItem_Click"></shell:ApplicationBarMenuItem>
      <shell:ApplicationBarMenuItem x:Name="subscribtionItem"
  Click="subscribtionItem_Click" Text="Subscription"></shell:ApplicationBarMenuItem>
    </shell:ApplicationBar.MenuItems>
  </shell:ApplicationBar>
</Application.Resources>
```

2. Add an event handler in **App.xaml.cs** for each menu bar.

```
private void subscribtionItem_Click(object sender, EventArgs e)
{
    this.RootFrame.Navigate(new Uri("/Subscription.xaml", UriKind.Relative));
}

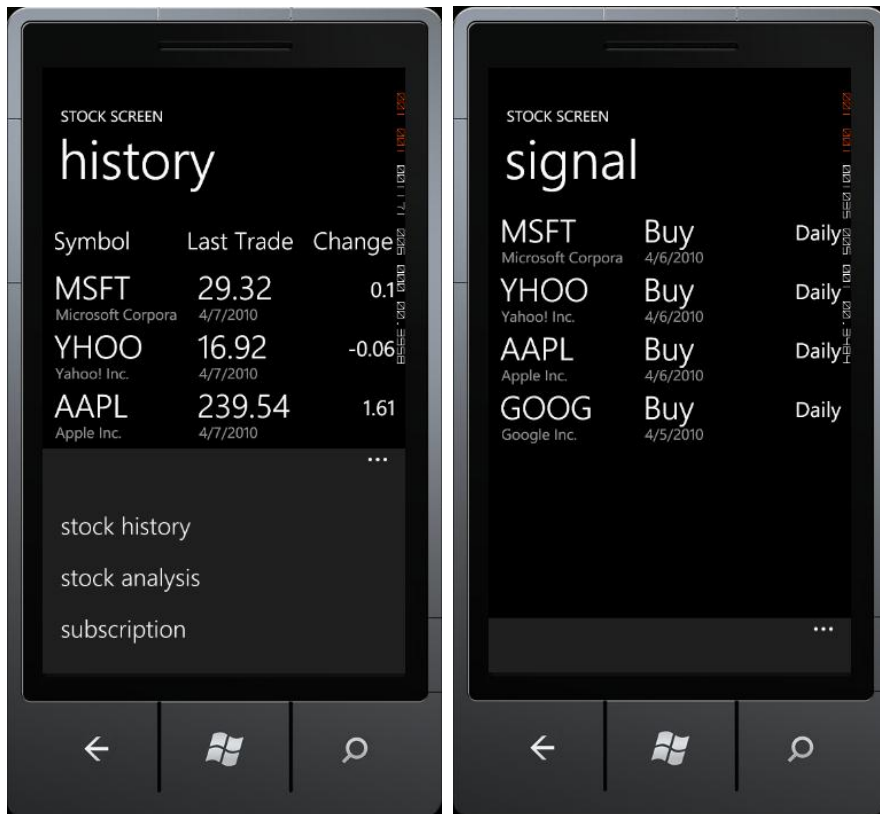
private void stockHistoryItem_Click(object sender, EventArgs e)
{
    this.RootFrame.Navigate(new Uri("/MainPage.xaml", UriKind.Relative));
}

private void stockAnalysisItem_Click(object sender, EventArgs e)
{
    this.RootFrame.Navigate(new Uri("/StockAnalysisPage.xaml",
UriKind.Relative));
}
```

3. Open **MainPage.xaml** and **StockAnalysisPage.xaml**. Add the following code to both:

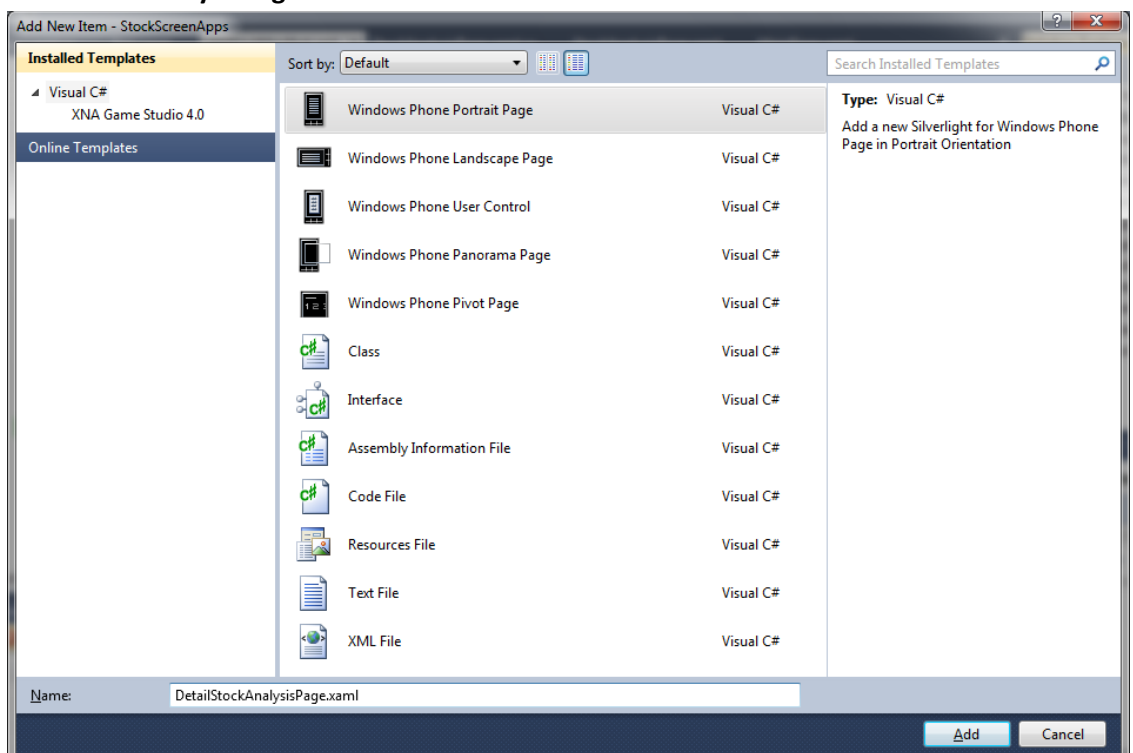
```
<phone:PhoneApplicationPage
...
shell:SystemTray.IsVisible="True"
  ApplicationBar="{StaticResource globalAppBar}">
```

4. Press F5 for results.



STOCK TRANSACTION DETAIL PAGE

1. Add a new page, right click on the project, select **Add New Page** and name it **DetailStockAnalysisPage.xaml**.



2. Modify the XAML code so that it looks like this:

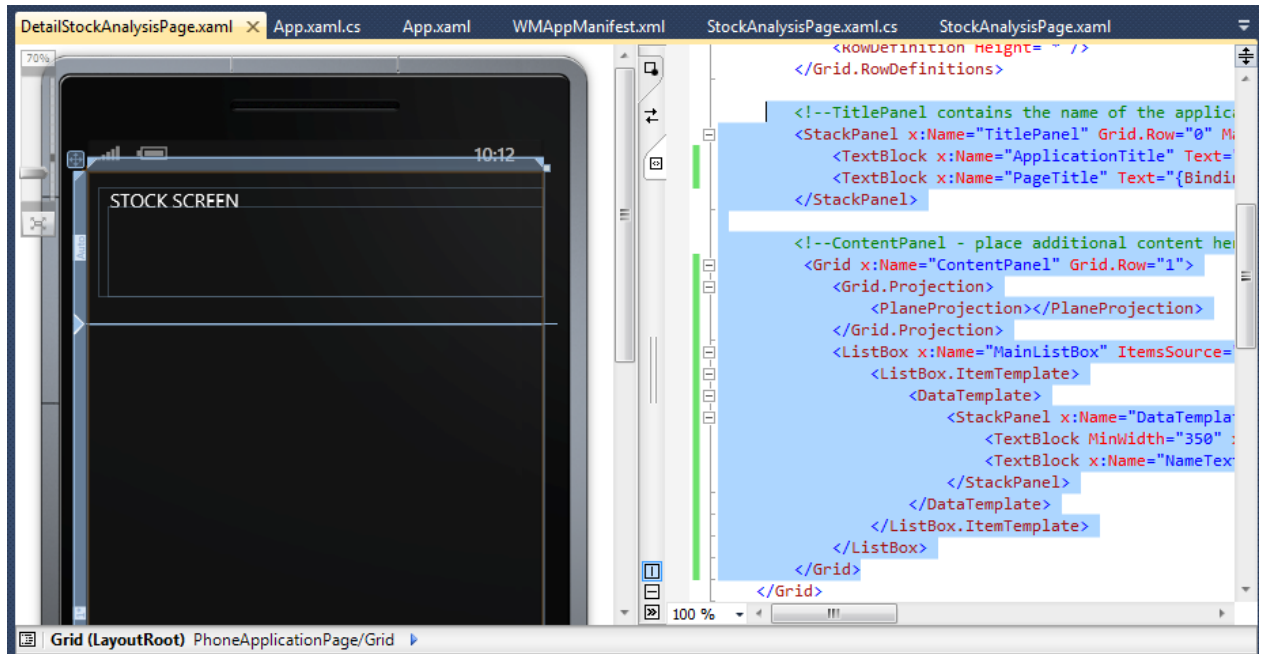
```

<Grid x:Name="LayoutRoot" Background="Transparent">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*/>
  </Grid.RowDefinitions>

  <!--TitlePanel contains the name of the application and page title-->
  <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
    <TextBlock x:Name="ApplicationTitle" Text="STOCK SCREEN"
Style="{StaticResource PhoneTextNormalStyle}"/>
    <TextBlock x:Name="PageTitle" Text="{Binding CompanyName}" Margin="9,-
7,0,0" Style="{StaticResource PhoneTextTitle1Style}"/>
  </StackPanel>

  <!--ContentPanel - place additional content here-->
  <Grid x:Name="ContentPanel" Grid.Row="1">
    <Grid.Projection>
      <PlaneProjection></PlaneProjection>
    </Grid.Projection>
    <StackPanel Orientation="Vertical">
      <StackPanel Orientation="Horizontal" Margin="5,20,5,20">
        <TextBlock x:Name="SymbolTitle" Text="Signal Type" MinWidth="350"
Margin="-3,-8,0,0" Style="{StaticResource PhoneTextLargeStyle}"/>
        <TextBlock x:Name="DateTitle" HorizontalAlignment="Right" Text="Date"
Margin="-3,-8,0,0" Style="{StaticResource PhoneTextLargeStyle}"/>
      </StackPanel>
      <ListBox x:Name="MainListBox" ItemsSource="{Binding Items}">
        <ListBox.ItemTemplate>
          <DataTemplate>
            <StackPanel x:Name="DataTemplateStackPanel"
Orientation="Horizontal">
              <TextBlock MinWidth="350" x:Name="SymbolText"
Text="{Binding Type}" Margin="5,5,0,5" Style="{StaticResource PhoneTextLargeStyle}"/>
              <TextBlock x:Name="NameText" Text="{Binding Date}"
Margin="0,5,0,5" Style="{StaticResource PhoneTextSubtleStyle}"/>
            </StackPanel>
          </DataTemplate>
        </ListBox.ItemTemplate>
      </ListBox>
    </StackPanel>
  </Grid>
</Grid>

```



3. Prepare a ViewModel to display the company transaction data of a company. Right click on project and select **Add Class**, name it **SignalViewModel.cs**. Type the code below:

```
public class SignalViewModel : INotifyPropertyChanged
{
    private String type;
    public String Type
    {
        get { return type; }
        set
        {
            if (value != type)
            {
                type = value;
                NotifyPropertyChanged("Type");
            }
        }
    }
    private String date;
    public String Date
    {
        get { return date; }
        set
        {
            if (value != date)
            {
                date = value;
                NotifyPropertyChanged("Date");
            }
        }
    }
}
```



```

    }

    public event PropertyChangedEventHandler PropertyChanged;

    private void NotifyPropertyChanged(String propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (null != handler)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}

```

4. For SignalModel main class, create a new class. Right click on project, select **Add Class**, and name it **MainSignalViewModel.cs**. Insert the following code:

```

public class MainSignalViewModel : INotifyPropertyChanged
{
    public ObservableCollection<SignalViewModel> Items { get; set; }
    public String CompanyName { get; set; }

    public MainSignalViewModel()
    {
        Items = new ObservableCollection<SignalViewModel>();
    }

    public event PropertyChangedEventHandler PropertyChanged;
    private void NotifyPropertyChanged(String propertyName)
    {
        PropertyChangedEventHandler handler = PropertyChanged;
        if (null != handler)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}

```

5. Open **DetailStockAnalysisPage.xaml** page and modify the code:

Declare a variable to store parameters from the previous page

```
string selectedCompany = "";
```

When the application is navigated to the said page, it will call a web service. The parameters for the web service are retrieved by doing a string query on the previous page.

```

// When page is navigated to, set data context to selected item in list
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);
}

```

```

        if (NavigationContext.QueryString.TryGetValue("selectedItem", out
selectedCompany))
        {
            WebClient wc = new WebClient();
            String uri = String.Format("http://localhost/datasaham/CompanySignal-
{0}.xml", selectedCompany);
            wc.DownloadStringCompleted += new
DownloadStringCompletedEventHandler(wc_DownloadStringCompleted);
            wc.DownloadStringAsync(new Uri(uri));
        }
    }

    void wc_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
    {
        if (e.Result != null)
        {
            this.DataContext = ParseDetailStockAnalysisFromXML(e.Result,
selectedCompany);
        }
    }
}

```

Then parse the fetched data using LINQ to XML.

```

public MainSignalViewModel ParseDetailStockAnalysisFromXML(String result, String
symbol)
{
    MainSignalViewModel retVal = new MainSignalViewModel() { CompanyName =
symbol };

    XDocument xdoc = XDocument.Parse(result);

    int i = 0;
    foreach (var x in xdoc.Descendants("Signal"))
    {
        i++;
        SignalViewModel company = new SignalViewModel()
        {
            Type = x.Element("Type").Value,
            Date = x.Element("Date").Value
        };
        retVal.Items.Add(company);
    }
    return retVal;
}

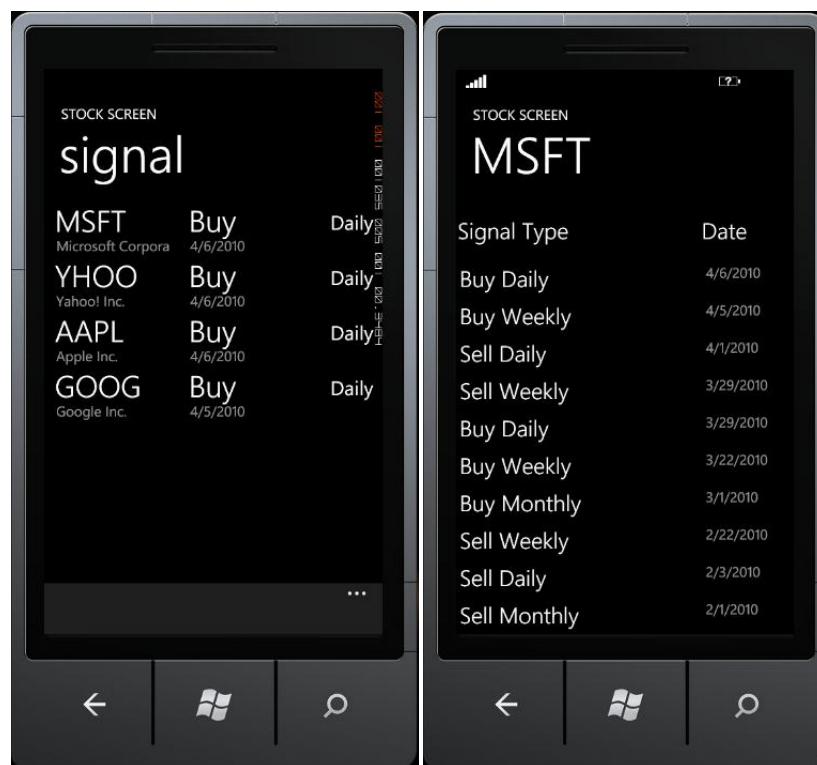
```

6. Open **StockAnalysisPage.xaml** page and add the code below in **MainListBox_SelectionChanged** event handler:

```
private void MainListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    //// Navigate to the new page
    NavigationService.Navigate(new
Uri("/DetailStockAnalysisPage.xaml?selectedItem=" + (MainListBox.SelectedItem as
CompanyViewModel).CompanySymbol, UriKind.Relative));

    // Reset selected index to -1 (no selection)
    MainListBox.SelectedIndex = -1;
}
```

7. Press F5 for results. You can select a company from the list and see the analysis for its stock.

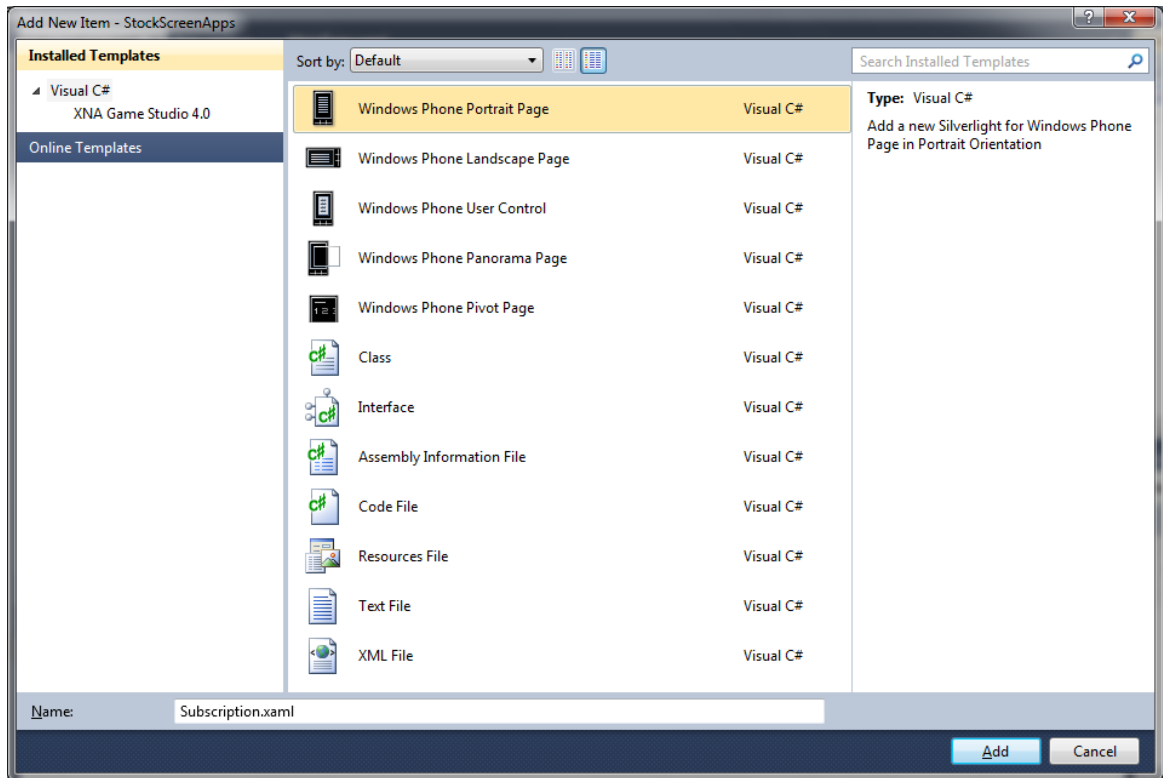


ADDING COMPANY LIST

An interesting feature in Windows Phone (one which will not be discussed in this e-book) is the ability to do Push Notification using Microsoft service, Push Notification Server. Using this push, developers can send new data to an application without forcing the application to do constant polling to data provider. This means that without having to be active, application can still fetch the newest data.

This scenario fits perfectly for application like stock screen. Assume this application uses the service, then we will create a mechanism how users can select companies to subscribe, so that they will receive information actually from the selected companies.

1. Insert a page, right click on project, select **Add Page -> Windows Phone Portrait Page** and name it **Subscription.xaml**.



2. Add the following code so that the page layout will look like the figure below.

```

<phone:PhoneApplicationPage
    x:Class="StockScreenApps.Subscription"
    ....
    ApplicationBar="{StaticResource globalAppBar}"
    <phone:PhoneApplicationPage.Resources>
        <DataTemplate x:Key="subscription">
            <StackPanel x:Name="DataTemplateStackPanel" Orientation="Horizontal">
                <Image x:Name="ItemImage"
                    Source="/StockScreenApps;component/appbar.delete.rest.png" Height="43" Width="43"
                    VerticalAlignment="Top" Margin="10,0,20,0"
                    MouseLeftButtonDown="ItemImage_MouseLeftButtonDown"/>
                <StackPanel>
                    <TextBlock x:Name="CompanyText" Text="{Binding Name}" Margin="-2,-
                    13,0,0" Style="{StaticResource PhoneTextExtraLargeStyle}"/>
                    <TextBlock x:Name="CompanySymbolText" Text="{Binding
                    CompanySymbol}" Margin="0,-6,0,3" Style="{StaticResource PhoneTextSubtleStyle}"/>
                </StackPanel>
            </StackPanel>
        </DataTemplate>
        <DataTemplate x:Key="unsubscription">
            <StackPanel x:Name="DataTemplateStackPanel" Orientation="Horizontal">
                <Image x:Name="ItemImage"
                    Source="/StockScreenApps;component/appbar.add.rest.png" Height="43" Width="43"
                    VerticalAlignment="Top" MouseLeftButtonDown="AddImage_MouseLeftButtonDown"
                    Margin="10,0,20,0"/>
                <StackPanel>

```

```

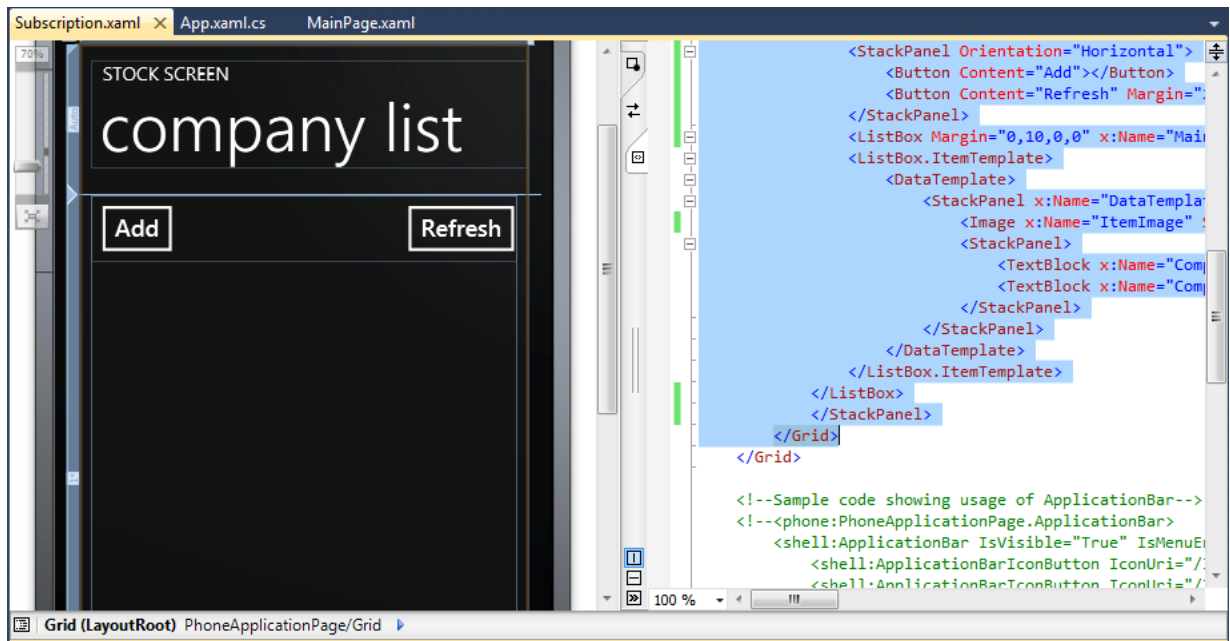
        <TextBlock x:Name="CompanyText" Text="{Binding Name}" Margin="-2,-
13,0,0" Style="{StaticResource PhoneTextExtraLargeStyle}"/>
        <TextBlock x:Name="CompanySymbolText" Text="{Binding
CompanySymbol}" Margin="0,-6,0,3" Style="{StaticResource PhoneTextSubtleStyle}"/>
    </StackPanel>
    </StackPanel>
</DataTemplate>
</phone:PhoneApplicationPage.Resources>
<!--LayoutRoot is the root grid where all page content is placed-->
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*/>
    </Grid.RowDefinitions>

    <!--TitlePanel contains the name of the application and page title-->
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="STOCK SCREEN"
Style="{StaticResource PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="company list" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

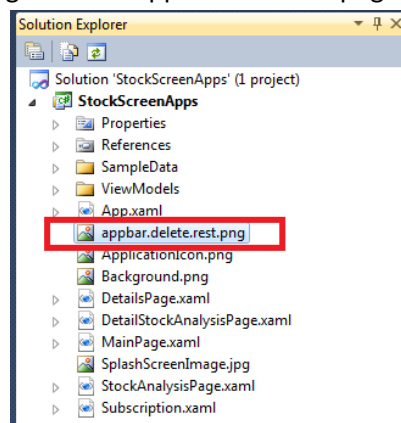
    <!--ContentPanel - place additional content here-->
    <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
        <StackPanel Orientation="Vertical">
            <StackPanel Orientation="Horizontal">
                <Button Content="Add"></Button>
                <Button Content="Refresh" Margin="230,0,0,0"></Button>
            </StackPanel>
            <ListBox Margin="0,10,0,0" x:Name="MainListBox"
SelectionMode="Multiple" ItemsSource="{Binding Items}">
        </ListBox>
    </StackPanel>
</Grid>
</Grid>
</phone:PhoneApplicationPage>

```

What you should notice from the code above is the declaration of two different data templates. The first data template is used to displayed the list of subscribed companies, along with a delete button to remove a company from the list. The second template is used to display available companies along with an add (+) button to add the company to our subscription list.



3. Add the resource icon that will be used, obtainable from *C:\Program Files\Microsoft SDKs\Windows Phone\v7.0\Icons*. Do this by right clicking on project, select **Add Existing Item** and find *app.bar.delete.rest.png* icon and *app.bar.add.rest.png* from the said folder.



4. Open **Subscription.xaml.cs**. Add the following line of codes:

Declare a variable to contain the list of subscribed companies and non-subscribed companies.

```
MainCompanyViewModel subscriptionlist;
MainCompanyViewModel unsubcriptionlist;
```

On *Loaded()* event handler add a function to fetch data from available services. Then parse the return value.

```
public Subscription()
{
    InitializeComponent();
    this.Loaded += new RoutedEventHandler(Subscription_Loaded);
}
```

```

void Subscription_Loaded(object sender, RoutedEventArgs e)
{
    //set item template
    MainListBox.ItemTemplate =
(DataTemplate)this.Resources["subscription"];

    //get data

    WebClient wc = new WebClient();
    wc.DownloadStringCompleted += new
DownloadStringCompletedEventHandler(wc_DownloadStringCompleted);
    wc.DownloadStringAsync(new
Uri("http://localhost/datasaham/SubscribtionList.xml"));

    WebClient wc2 = new WebClient();
    wc2.DownloadStringCompleted += new
DownloadStringCompletedEventHandler(wc2_DownloadStringCompleted);
    wc2.DownloadStringAsync(new
Uri("http://localhost/datasaham/UnsubscriptionList.xml"));

}

void wc_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
{
    if (e.Result != null)
    {
        subscribtionlist = ParseFromXML(e.Result);
        DataContext = subscribtionlist;
    }
}

```

Use LINQ to XML.

```

public MainCompanyViewModel ParseFromXML(String result)
{
    MainCompanyViewModel retVal = new MainCompanyViewModel();
    retVal.Items.Clear();

    XDocument xdoc = XDocument.Parse(result);

    foreach (var x in xdoc.Descendants("Subscription"))
    {
        CompanyViewModel company = new CompanyViewModel()
        {
            CompanySymbol = x.Element("Symbol").Value,
            Name = x.Element("Name").Value
        };
        retVal.Items.Add(company);
    }
}

```

```
        return retVal;
    }
```

5. Double click the **Add** button and add an event handler. When this button is pressed, the screen will display a list of companies the user has not subscribed to.

```
bool AddMode;
private void Button_Click(object sender, RoutedEventArgs e)
{
    if (!AddMode)
    {
        AddMode = true;
        (sender as Button).Content = "OK!";

        MainListBox.ItemTemplate =
(DataTemplate)this.Resources["unsubscription"];

        //change datacontext
        if (unsubscriptionlist == null)
        {

        }
        else
        {
            DataContext = unsubscriptionlist;
        }
    }
    else
    {
        AddMode = false;
        (sender as Button).Content = "Add";
        MainListBox.ItemTemplate =
(DataTemplate)this.Resources["subscription"];
        DataContext = subscriptionlist;
    }
}

void wc2_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
{
    if (e.Result != null)
    {
        unsubscriptionlist = new MainCompanyViewModel();
        unsubscriptionlist = ParseFromXML(e.Result);
    }
}
}
```


6. Now add a function to handle deletion/addition to the list. We add a handler to handle MainListBox_SelectionChanged event.

```
bool isDelete = false;
    bool isAdd = false;

    private void MainListBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
    {

        if (isDelete)
        {
            string symbol = ((sender as ListBox).SelectedItem as
CompanyViewModel).CompanySymbol;
            MessageBoxResult result = MessageBox.Show(((sender as
ListBox).SelectedItem as CompanyViewModel).Name, "delete", MessageBoxButtons.OKCancel);
            isDelete = false;
            if (result == MessageBoxResult.OK)
            {
                CompanyViewModel temp = subscribtionlist.GetCompany(symbol);
                CompanyViewModel company = new CompanyViewModel()
                {
                    CompanySymbol = temp.CompanySymbol,
                    Name = temp.Name
                };

                unsubscribtionlist.Items.Add(company);
                subscribtionlist.Items.Remove(temp);
            }
        }
        else if (isAdd)
        {
            string symbol = ((sender as ListBox).SelectedItem as
CompanyViewModel).CompanySymbol;
            MessageBoxResult result = MessageBox.Show(((sender as
ListBox).SelectedItem as CompanyViewModel).Name, "add
company", MessageBoxButtons.OKCancel);
            isAdd = false;
            if (result == MessageBoxResult.OK)
            {
                CompanyViewModel temp = unsubscribtionlist.GetCompany(symbol);
                CompanyViewModel company = new CompanyViewModel()
                {
                    CompanySymbol = temp.CompanySymbol,
                    Name = temp.Name
                };

                subscribtionlist.Items.Add(company);
                unsubscribtionlist.Items.Remove(temp);
            }
        }
    }
}
```

```

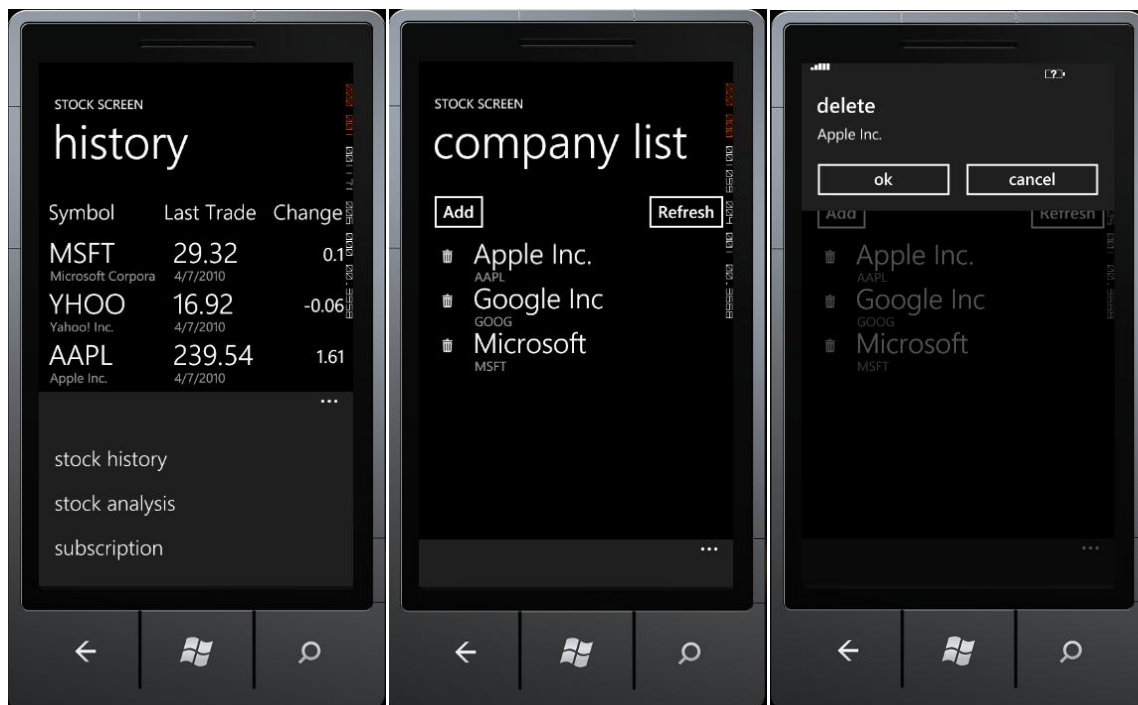
private void ItemImage_MouseLeftButtonDown(object sender, MouseButtonEventArgs
e)
{
    isDelete = true;
}

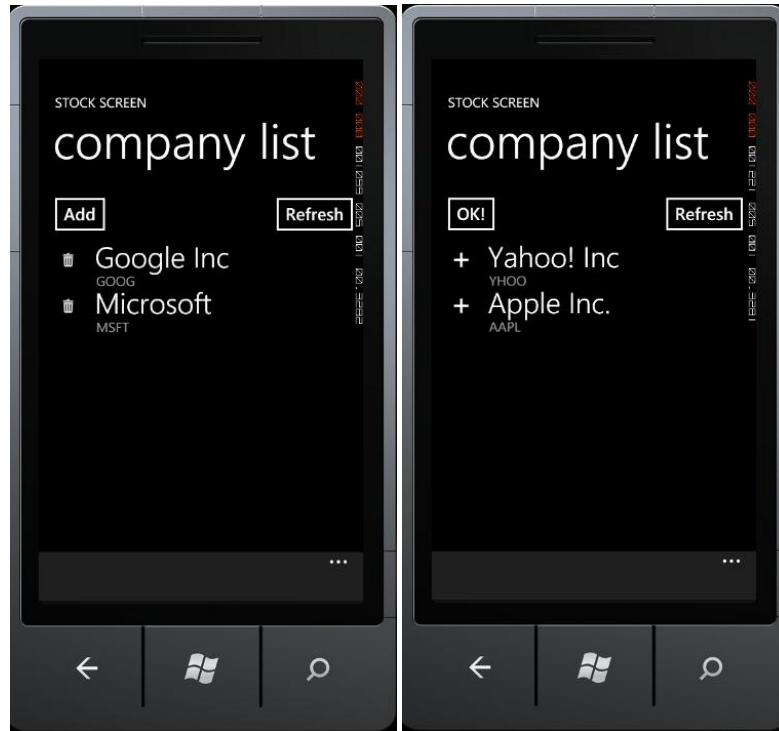
private void AddImage_MouseLeftButtonDown(object sender, MouseButtonEventArgs
e)
{
    isAdd = true;
}

```

Note: The code above is not the only solution to handle deletion and addition of data. Let's just say **"it works"** but it isn't necessarily the best solution. At the very least it is enough for current purpose. You should consider not using a MessageBox in the real application.

7. Press F5 and see how the application works. Press the delete icon to remove a company from the list. To add a company, click Add and select one of the available company. We will feel the advantage of using MVVM schema and INotifyPropertyChanged, with which we can delete an item in an observable collection and the application's interface will automatically updated to the latest condition.





CLOSING

Finally we have reached the end of this e-book :) Such a fun journey of exploration, don't you agree? We have learned a lot of things about Windows Phone, Silverlight platform for Windows Phone application development, specific features on Silverlight for Windows Phone, up to practicing by developing two simple applications. One thing for sure, the interface design aspect is not discussed too far in this e-book because I personally don't have an expertise in the subject.

Windows Phone is a very broad platform. There are still so many topics yet to be discussed in this e-book, such as:

- Push Notification
- Manipulation (Multi-touch)
- Bing Maps for Windows Phone
- Launcher and Chooser
- ... and many more

Reference list in this e-book can be used to study about said topics. Also, every source code related to this book, including the two simple applications, can be downloaded for free in Silverlight for Windows Phone homepage: <http://slforwp7.codeplex.com/releases> .

I especially want to thank [Halida Astatin](#), for helping me translate this ebook, to provide easy to read and such a convenient experience for one who use English as the main language.

Lastly, I humbly apologize for any typing or layout mistakes. Enjoy your study :)

Regards,

Puja Pramudya
puja.pramudya@gmail.com
@poedja_p

REFERENCES

- [1] [Windows Phone in MSDN](#)
- [2] [Windows Phone 7 Jump Start Training](#)
- [3] [Windows Phone 7 in 7](#)
- [4] <http://i.msdn.microsoft.com/dynimg/IC430124.png>
- [5] <http://i.msdn.microsoft.com/dynimg/IC425813.jpg>
- [6] <http://i.msdn.microsoft.com/dynimg/IC425811.jpg>