

# Windows Phone 7 IN ACTION

Timothy Binkley-Jones  
Massimo Perga  
Michael Sync





## ***Windows Phone 7 in Action***

by Timothy Binkley-Jones  
Massimo Perga  
Michael Sync

### **Chapter 2**

Copyright 2013 Manning Publications

# *brief contents*

---

## **PART 1 INTRODUCING WINDOWS PHONE.....1**

- 1 ■ A new phone, a new operating system 3
- 2 ■ Creating your first Windows Phone application 29

## **PART 2 CORE WINDOWS PHONE.....55**

- 3 ■ Fast application switching and scheduled actions 57
- 4 ■ Launching tasks and choosers 93
- 5 ■ Storing data 121
- 6 ■ Working with the camera 149
- 7 ■ Integrating with the Pictures and Music +  
Videos Hubs 171
- 8 ■ Using sensors 199
- 9 ■ Network communication with push notifications  
and sockets 227

## **PART 3 SILVERLIGHT FOR WINDOWS PHONE.....257**

- 10 ■ ApplicationBar, Panorama, and Pivot controls 259
- 11 ■ Building Windows Phone UI with  
Silverlight controls 284

- 12 ■ Manipulating and creating media  
with MediaElement 310
- 13 ■ Using Bing Maps and the browser 341

**PART 4 SILVERLIGHT AND THE XNA FRAMEWORK .....369**

- 14 ■ Integrating Silverlight with XNA 371
- 15 ■ XNA input handling 399

# *Creating your first Windows Phone application*

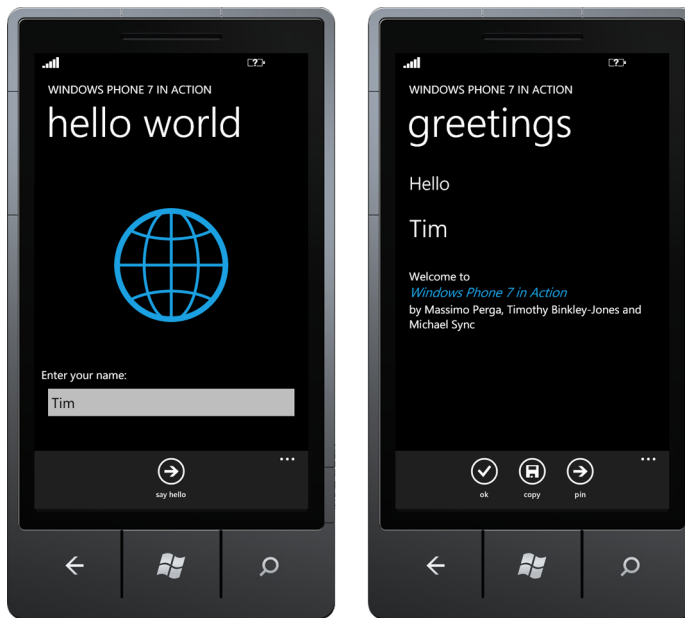


## ***This chapter covers***

- Creating your first Silverlight application
- Handling touch events
- Navigating between pages
- Trial licensing

Now that you have the necessary background on the Windows Phone platform and the Windows Phone Developer Tools, it's time to get down to business and start programming. You'll start by building a Hello World project. For developers experienced with Visual Studio, simple Hello World projects may seem unduly remedial. Windows Phone projects have several unique settings and features that you need to understand to build proper applications and games. The Hello World project in this chapter is designed to highlight these aspects of Windows Phone development.

You'll build a Hello World Silverlight application and explore a few of the phone-specific extensions to Silverlight. Silverlight applications have several project properties unique to Windows Phone. Two of these properties define the icons used in the phone's start screen and Applications List. Other properties determine the titles shown next to the start and application list icons. You'll learn how to use the Visual Studio project templates to generate a new application and how to



**Figure 2.1** The Silverlight Hello World application

use the item templates to generate a new page for your application. You'll also learn how to deploy the application to the emulator or a physical device and use the debugger to step through code.

**TIP** If you're new to Silverlight development, read the primers for Expression Blend and Silverlight in the appendices.

In most ways, building a Silverlight application for the phone is the same as building one for the browser or the desktop, but there are some minor differences. You'll see some of the differences as you build your application. The Hello World application that you'll create is shown in figure 2.1.

The application displays a title, draws a globe, and prompts the user to enter their name. When the user presses the toolbar button, the application navigates to a greeting page. You'll start building your application by creating a new Silverlight project.

## 2.1 **Generating the project**

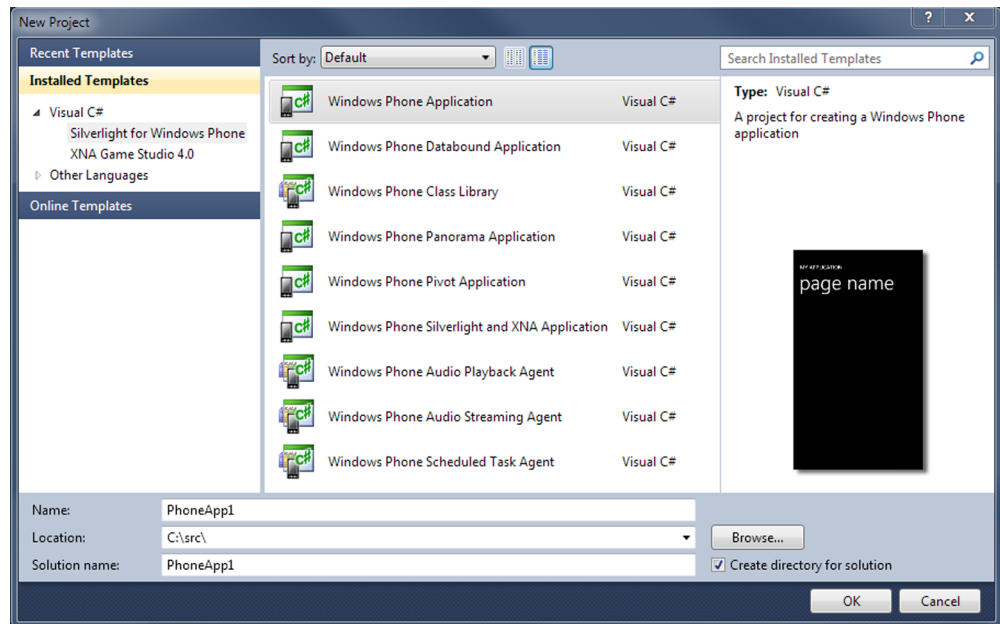
To start the Hello World application, you'll use the Windows Phone Application project template in Visual Studio. The Windows Phone Application project template is just one of the several Silverlight project templates that are installed with Visual Studio. Table 2.1 lists the available project templates.

You'll get started by opening Visual Studio and creating a new project. Figure 2.2 shows the new project dialog for the Hello World Silverlight application. Name the project SilverlightHello.

**Table 2.1 Windows Phone project templates**

Project template	Description
Windows Phone Application	A basic application skeleton with a single page.
Windows Phone Databound Application	An application demonstrating page navigation, databound list controls, and the MVVM pattern.
Windows Phone Class Library	A simple library for creating reusable components.
Windows Phone Panorama Application	An application demonstrating a databound <code>Panorama</code> control and the MVVM pattern. The <code>Panorama</code> control is covered in chapter 10.
Windows Phone Pivot Application	An application demonstrating a databound <code>Pivot</code> control and the MVVM pattern. The <code>Pivot</code> control is covered in chapter 10.
Windows Phone Silverlight and XNA Application	An application that mixes Silverlight and XNA Framework graphics. You'll build an application that uses both Silverlight and XNA in chapter 14.
Windows Phone Audio Playback Agent	A library containing an application's background audio logic. Audio Playback Agents are covered in chapter 7.
Windows Phone Audio Streaming Agent	A library containing an application's background streaming audio logic.
Windows Phone Scheduled Task Agent	A library containing an application's background processing logic. Scheduled Tasks Agents are covered in chapter 3.

Each of the project templates listed here are available for both C# and Visual Basic projects.

**Figure 2.2 Visual Studio's New Project dialog box**

Once you click OK, you'll be prompted with a dialog asking you to pick the target operating system version. This dialog can be confusing because it lists the Windows Phone SDK versions and not the operating system versions. If you're building an application that makes use of the new features in the Windows Phone 7.5 operating system, choose *Windows Phone OS 7.1* from the drop-down. After you click the OK button, a new Visual Studio solution and project are created. The IDE opens *MainPage.xaml* in the editor, and you're ready to begin. Before you start work, let's take a look at what Visual Studio created. Figure 2.3 shows the new project in the Solution Explorer.

The project structure mirrors that of a regular Silverlight project with Properties and References folders, *App.xaml*, *MainPage.xaml*, *AppManifest.xml*, and *AssemblyInfo.cs*. Along with the references to the Microsoft .Phone assemblies, a few additional files are present:

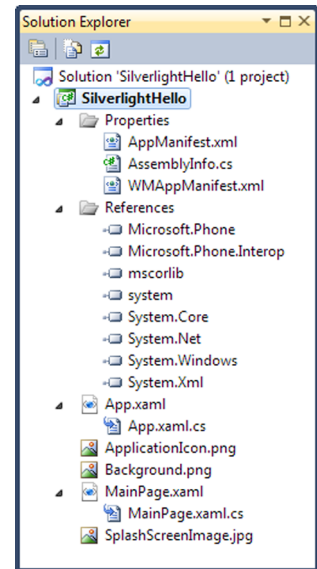
- *WAppManifest.xml*
- *ApplicationIcon.png*
- *Background.png*
- *SplashScreenImage.jpg*

The PNG image files are used by the operating system when displaying the application in the Start Experience, Application List, or Games Hub, and the splash image is shown when the Silverlight application starts up. We'll look at the image files in more depth later in the chapter.

**NOTE** *Background.png* is used as the background for the start experience tile. It's not intended to be the default background of the application.

*WAppManifest.xml* contains metadata for the application, providing important details about the application to the operating system. Information in *WAppManifest.xml* is also used by the Application Marketplace to validate and list an application. Visual Studio adds the *WAppManifest* file to the .xap file deployment package when it builds an application. The final *WAppManifest.xml* file that appears in the package downloaded to a user's phone will not necessarily contain the same information the developer specified when they built the application before submitting it to the Marketplace. During the marketplace certification, the application is examined and its manifest file is updated. A product identifier is added, the hub type or genre is set, and the security capabilities are confirmed.

Many of the settings in *WAppManifest.xml* are set via the project property pages. Open the *WAppManifest.xml* file and look for the *App* element, specifically the *Genre* attribute:



**Figure 2.3** Files in the Solution Explorer



```
<App xmlns="" ProductID="{65438a9e-0537-451f-aaec-6ff25ca0bf85}"
  Title="Hello World" RuntimeType="Silverlight" Version="1.0.0.0"
  Genre="Apps.Normal" Author="" Description="" Publisher="">
```

The Genre attribute declares whether the application appears in the Application List or the Games Hub. When developing and testing on the emulator, you should leave the genre set to `Apps.Normal` since the Games Hub isn't present on the emulator. If you want to test integration with the Games Hub on a real device, you can change the setting to `Apps.Games`.

Your new Hello World project is ready to be built and deployed to the emulator or a phone. Visual Studio's Debugger is used to debug running Windows Phone applications.

### 2.1.1 Debugging phone projects

Once you've built a project, you'll be able to debug it both in the emulator and on a real device. Before starting a debug session, you'll want to confirm the appropriate target is selected in the target deployment device combo box. In figure 2.4, you can see that Windows Phone Emulator is the target device, and the application will be launched in the emulator.

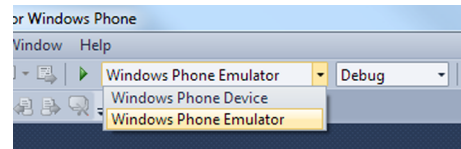


Figure 2.4 Target Deployment Device selector

The first time you launch an application in the emulator, it'll take some time to boot and initialize the emulator prior to starting the application. You can also start the emulator ahead of time from the *Windows Phone SDK* folder in the Start Menu. Once the application has been launched, you'll be able to debug and interact with it in the emulator. The Windows Phone emulator can be kept running between debugging sessions.

**TIP** An application can detect whether it's running in the emulator by checking the value of the `Microsoft.Devices.Environment.DeviceType` static property. If the value is `DeviceType.Emulator`, the application is running in the emulator.

Prior to launching an application on a real device, the phone must be plugged into the USB port and connected to your computer. The phone is considered connected when the Zune software is running. If you don't want to keep the Zune software running, you can connect the phone with the WPConnect tool. Before you can deploy and debug an application on a real phone, you must register your phone with the Developer Registration Tool. The WPConnect and Developer Registration tools were introduced in chapter 1.

When the application is being debugged, it'll automatically stop on the breakpoints you have set in the source code. You can add or remove breakpoints during the execution just like you would in any other desktop or Silverlight project. Finally, when you're done, you can stop debugging in the IDE or press the Back button on the device.

Visual Studio allows you to install or deploy the project on the device without starting a debugging session. You can do this by right-clicking on the project name in the Solution Explorer and selecting the Deploy option from the menu. The application will be copied to the device, and can be launched from the Applications List on the phone. When the application is launched, Silverlight for Windows Phone adds a few custom steps to the startup process that aren't found in Silverlight for the browser applications.

## 2.1.2 *Application startup*

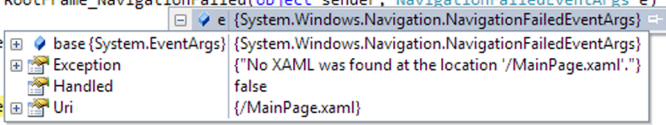
Like any other Silverlight application, the entry point is a `System.Windows.Application`-derived class found in `App.xaml`. All phone applications are Silverlight Navigation Applications. In your phone application, the `App` class creates an instance of `PhoneApplicationFrame`, which is used as the `RootVisual`. Behind the scenes, the application host calls the `NavigationService` directly to navigate to `MainPage.xaml` during initial launch. When an application is reactivated, the application host navigates directly to the active page's XAML—see chapter 3 for more details on application launching and activation.

All this magic navigation is well and good; the application starts and `MainPage` is automatically loaded. What happens when you decide to rename `MainPage.xaml`? Since the XAML filename doesn't have to match the name of the C# class that it contains, changing the name of `MainPage` is a two-step process. Fortunately, Visual Studio's refactoring features make this a simple operation. In the Solution Explorer, right-click `MainPage.xaml`, choose `Rename`, and change the filename to `HelloPage.xaml`. Next, open `MainPage.xaml.cs`, select the `MainPage` text in the class definition, and choose `Rename` from the `Refactor` menu, specifying `HelloPage` as the new name.

When you debug the application now, the `App.RootFrame_NavigationFailed` event handler is called. Figure 2.5 shows the `NavigationFailedEventArgs` properties sent to the event handler.

Even though you renamed `MainPage.xaml`, the application is still configured to use `MainPage.xaml` as the startup URI. The startup URI is declared in the `WMAppManifest.xml` file as the `NavigationPage` attribute of the `Task` named “\_default”. Though a number of the `WMAppManifest.xml` settings can be set using the project property editor, the default task URI must be specified by directly editing the XML

```
// Code to execute if a navigation fails
private void RootFrame_NavigationFailed(object sender, NavigationFailedEventArgs e)
{
    if (System.Windows.Navigation.NavigationFailedEventArgs e)
    {
        // A
        System.Windows.Navigation.NavigationFailedEventArgs e
    }
}
```



**Figure 2.5** `NavigationFailedEventArgs` properties after renaming `MainPage.xaml`

file. Open WMAppManifest.xml, update the URI attribute, and then save and run the application:

```
<Tasks>
  <DefaultTask Name="_default"
    NavigationPage="HelloPage.xaml"/>
</Tasks>
```

In this section you created a new Windows Phone Application project and examined the files created by the project template. You learned that a PhoneApplicationFrame is the root visual for the application and that the Silverlight runtime uses the navigation framework to load the startup page. Now that the application is running, and you've customized the name of the startup page, you'll customize the page contents.

## 2.2 Implementing Hello World

The project template created a default main page for your application, which you just renamed to HelloPage.xaml. In this section you're going to add a second page to the application that will display a greeting message. The second will employ a pair of TextBlock controls as well as a RichTextBox. You're also going to customize HelloPage by drawing a globe, as well as asking the user to input their name. First we'll take a closer look at the page created for you by the project template.

### 2.2.1 Customizing the startup page

The Windows Phone Application project template created the startup page with several elements meant to match the page design to the Metro style described in Microsoft's *User Experience Design Guidelines for Windows Phone*. The design guide, found on MSDN, details the expected look and feel of phone applications. The following listing shows the XAML markup added by the project template for HelloPage's content.

**Listing 2.1** HelloPage's content as created by the project template

```
<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="MY APPLICATION"
      Style="{StaticResource PhoneTextNormalStyle}" />
    <TextBlock x:Name="PageTitle"
      Text="page name"
      Margin="9,-7,0,0"
      Style="{StaticResource PhoneTextTitle1Style}" />
  </StackPanel>
```

**1** LayoutRoot Grid control with two rows

**2** TitlePanel with two TextBlocks

```

<!--ContentPanel - place additional content here-->
<Grid x:Name="ContentPanel"
      Grid.Row="1"
      Margin="12,0,12,0">
</Grid>
</Grid>

```

**3** ContentPanel for all other markup

The page's root layout panel is a grid control that has been split into two rows **1**. The first row contains TitlePanel **2**, which stacks two TextBlock controls for the application and page titles. The remainder of the page is allocated to the ContentPanel **3**. Figure 2.6 shows HelloPage.xaml as created by the project template.

Application and page titles aren't required by the design guidelines or marketplace specification but there are several rules that should be followed when they're used. The application title should be the name of the application, and should be all uppercase characters. The page title should be all lowercase characters and should describe the data or features displayed in the page. The titles shouldn't scroll or wrap; when the title doesn't fit on the screen, the text should appear truncated. If the title panel appears on the main page, it should appear on all pages to provide the user with a consistent experience.

In HelloPage.xaml, update the application title to "WINDOWS PHONE 7 IN ACTION" and the page title to "hello world":

```

<TextBlock x:Name="ApplicationTitle"
           Text="WINDOWS PHONE 7 IN ACTION"
           Style="{StaticResource PhoneTextNormalStyle}" />
<TextBlock x:Name="PageTitle"
           Text="hello world"
           Margin="9,-7,0,0"
           Style="{StaticResource PhoneTextTitle1Style}" />

```

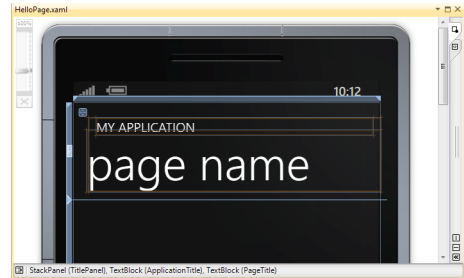
The title TextBlock controls each have their Style properties set to a static resource. The style resources used here won't be found anywhere in your project. They are styles injected into your application by the Silverlight framework so that your application can adhere to the user interface theme chosen by the user. Theme resources are covered in more depth in chapter 11. Theme resources are also used in the root PhoneApplicationPage tag to set the font and foreground color properties for the page:

```

FontFamily="{StaticResource PhoneFontFamilyNormal}"
FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"

```

PhoneApplicationPage also has a couple of orientation properties—Orientation and SupportedOrientations. The orientation property specifies whether the current



**Figure 2.6** HelloPage.xaml's TitlePanel in the Visual Studio Designer

orientation is portrait or landscape. The `SupportedOrientations` property declares which orientations are supported by the page. The visual designer supports both portrait and landscape and allows you to quickly switch between the two layouts, as shown in figure 2.7. You can read more about page orientation in chapter 11.

Windows Phone presents a status bar at the top edge of the screen in portrait layout. In landscape layout, the status bar is anchored to the edge opposite the Start button as it moves to the left or right, depending on the direction the user rotates the phone. The status bar displays the signal strength, battery, current time, and other indicators. The status bar consumes 32 pixels in portrait layout and 72 pixels in landscape layout. Screen designs should account for the space occupied by the status bar. Silverlight applications can hide the status bar with the `System-Tray.IsVisible` attached property. The project template sets this attached property to `True`. You can provide more room for your application's content and hide the status bar by setting the property's value to `False`. Before you choose to hide the status bar in your application, you should know that many users consider the status bar an essential element and dislike applications that hide it.

You're making good progress. You've gotten your hands dirty with XAML and started customizing your application. Along the way, you learned how to ensure your application fits into the system look and feel. Your next step is to add a globe and text box to the application content panel.

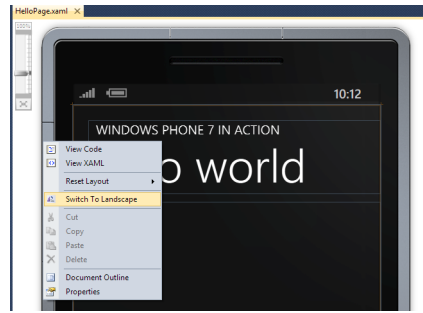
## 2.2.2 Adding application content

Remember that you want the first page of the application to draw a globe and prompt the user to enter a name. These visual elements will be added to the `ContentPanel` grid control that was created by the project template, as shown in the next listing. You'll start by dividing the `ContentPanel` into two rows, with one row using two thirds of the panel, and the remaining third allocated to the second row.

### Listing 2.2 Drawing the globe

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
  <Grid.Resources>
    <SolidColorBrush x:Key="GlobeBrush"
      Color="{StaticResource PhoneAccentColor}" />
  </Grid.Resources>
  <Grid.RowDefinitions>
    <RowDefinition Height="2*" />
    <RowDefinition Height="1*" />
  </Grid.RowDefinitions>
  <Canvas Width="200" Height="200" VerticalAlignment="Center"
```

← 1 **GlobeBrush is static resource**



**Figure 2.7** Using the context menu to switch between portrait and landscape layouts

```

Background="{StaticResource PhoneBackgroundBrush}" >
<Ellipse Width="200" Height="200"
    Stroke="{StaticResource GlobeBrush}"
    StrokeThickness="10" />
<Ellipse Width="100" Height="200"
    Canvas.Left="50"
    Stroke="{StaticResource GlobeBrush}"
    StrokeThickness="5" />
<Path Data="M 100,0 100,200"
    Stroke="{StaticResource GlobeBrush}"
    StrokeThickness="5" />
<Path Data="M 0,100 200,100"
    Stroke="{StaticResource GlobeBrush}"
    StrokeThickness="5" />
<Path Data="M 30,40 A 100,50 0 0 0 170,40"
    Stroke="{StaticResource GlobeBrush}"
    StrokeThickness="5" />
<Path Data="M 30,160 A 100,50 0 0 1 170,160"
    Stroke="{StaticResource GlobeBrush}"
    StrokeThickness="5" />
</Canvas>
</Grid>

```

2 Binding to theme brush

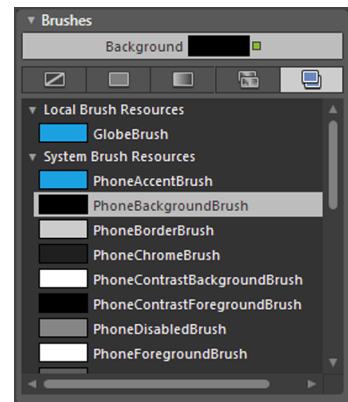
3 Binding to GlobeBrush

The globe is drawn using Silverlight's `Ellipse` and `Path` drawing primitives. These are just two examples of the drawing primitive classes found in the `System.Windows.Shapes` namespace. The sphere of the globe and the two arced meridians are drawn with ellipses. The straight meridian and the three parallels are drawn with paths. The drawing canvas is centered in the first row of the `ContentPanel`.

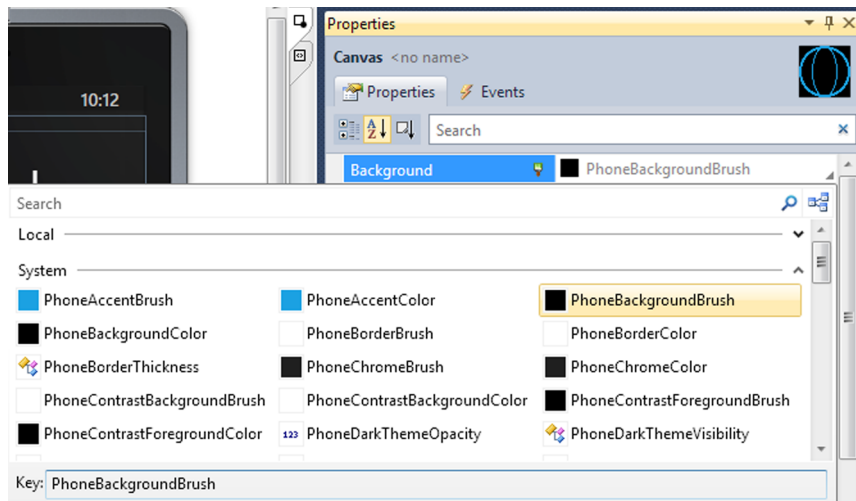
**TIP** To improve an application's performance, Microsoft recommends that complex XAML graphics be captured in a PNG or JPG and displayed with an image control.

Each of the shapes has its `Stroke` property bound ③ to a static resource you create named `GlobeBrush` ①. `GlobeBrush` has its `Color` property bound to another static resource named `PhoneAccentColor`. The canvas has its `Background` bound to a static resource named `PhoneBackgroundBrush` ②. `PhoneBackgroundBrush` and `PhoneAccentBrush` are other examples of the system theme resources that the Silverlight Framework injects into a Silverlight application. Both Expression Blend and the Visual Studio designer allow selecting system resources from their respective property windows. The Expression Blend resource menu, shown in figure 2.8, is accessed via the property editor.

The Visual Studio resource picker, shown in figure 2.9, is accessed from the `Apply Resource` option in a property's `Advanced Options` menu.



**Figure 2.8** Expression Blend's System Brush Resources selector



**Figure 2.9** Visual Studio's System resource menu

In this section, you added XAML markup to draw a globe, and bound the globe elements to system brushes to enable theme support. You still need to add UI controls to implement the remaining requirement, which is to navigate to the greeting page and display the user's name. First you need to create the greetings page.

### 2.2.3 Adding the greetings page

The second page of your application will display a greeting message to the user, using the name typed into the main page. Add the new page using the Windows Phone Portrait Page item template and name the file `GreetingPage.xaml`. The Portrait Page item template is one of several item templates that ship with the Windows Phone Developer Tools. Table 2.2 lists the Windows Phone item templates.

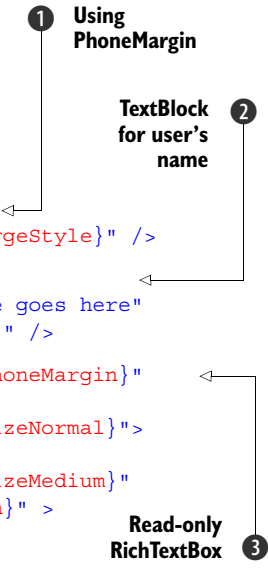
**Table 2.2** Windows Phone item templates

Page Template	Description
Windows Phone Portrait Page	A basic application page with title and description fields. The <code>Orientation</code> and <code>SupportedOrientations</code> properties are set to <code>Portrait</code> .
Windows Phone Landscape Page	An application page identical to a portrait page, except that the <code>Orientation</code> and <code>SupportedOrientations</code> properties are set to <code>Landscape</code> .
Windows Phone User Control	A starting point for creating reusable XAML-based controls.
Windows Phone Panorama Page	Adds an application page with <code>Panorama</code> control as its only content element.
Windows Phone Pivot Page	Adds an application page with <code>Pivot</code> control as its only content element.

The new greeting page contains controls for the application and page title. Following the same steps described for the hello page, change the application title to “WINDOWS PHONE 7 IN ACTION” and the page title to “greetings”.

The greetings page will use a couple of TextBlocks and a RichTextBox control to display the message. The XAML markup for the page’s content panel is shown in the next listing.

**Listing 2.3** GreetingPage’s content



```

<Grid x:Name="ContentPanel"
    Grid.Row="1"
    Margin="12,0,12,0">
    <Grid.RowDefinitions>
        <RowDefinition Height="72" />
        <RowDefinition Height="100" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <TextBlock Margin="{StaticResource PhoneMargin}"
        Text="Hello" Style="{StaticResource PhoneTextLargeStyle}" />
    <TextBlock x:Name="helloMessage" Grid.Row="1"
        Margin="{StaticResource PhoneMargin}" Text="name goes here"
        Style="{StaticResource PhoneTextExtraLargeStyle}" />
    <RichTextBox Grid.Row="2" Margin="{StaticResource PhoneMargin}"
        VerticalAlignment="Bottom">
        <Paragraph FontSize="{StaticResource PhoneFontSizeNormal}">
            Welcome to</Paragraph>
        <Paragraph FontSize="{StaticResource PhoneFontSizeMedium}"
            Foreground="{StaticResource PhoneAccentBrush}" >
            <Italic>Windows Phone 7 in Action</Italic>
        </Paragraph>
        <Paragraph FontSize="{StaticResource PhoneFontSizeNormal}">
            Written by Massimo Perga, Timothy Binkley-Jones
            and Michael Sync.</Paragraph>
    </RichTextBox>
</Grid>

```

**1 Using PhoneMargin**

**2 TextBlock for user's name**

**3 Read-only RichTextBox**

You start by dividing the content panel into three rows, specifying fixed heights for the first two rows. In the first row you place a TextBlock containing the text “Hello”. You use the PhoneMargin resource **1** to align the controls with the TextBlocks in the title panel. Next, you add a second TextBlock and give it the name helloMessage **2**. You’ll use this TextBlock to display the name of the user. Finally, you add a RichTextBox **3** which you use to display formatted text. On Windows Phone, the RichTextBox is read-only.

You now have the two pages in your Hello World application setup and ready to go. If you run the application now, you’ll see the hello page with the nice globe. Other than look at the globe, you can’t do anything in the application. You still need to add input controls to capture the user’s name. You also need something the user can use to navigate to the greetings page. Let’s take a look at how you interact with the user.



## 2.3 Interacting with the user

Silverlight for Windows Phone provides most of the core user input controls that are available to Silverlight for the browser. The input controls have been modified and restyled to work in a touch-only environment and have new events that are raised when the user touches the screen. To maintain compatibility with Silverlight for the browser, the Windows Phone controls also provide mouse-related events and automatically promote touch events into mouse events. Unless you're specifically looking for touch events, you'll work with the input controls in nearly the exact same way you did when building browser applications.

There will be situations where you want to work with the touch events and gestures. Raw touch events are decomposed into *start*, *delta*, and *stop* events. *Touch gestures* combine several raw touch events into well-known gestures such as Tap, Double Tap, Hold, Pinch, Pan, and Flick. In this section you'll learn how to capture Tap and Double Tap gestures to change the color of the globe. First let's take a closer look at how the common TextBox control operates on Windows Phone.

### 2.3.1 Touch typing

The Hello World application uses a TextBox control for text entry and a TextBlock for a label. These two controls are placed inside a StackPanel and added to the second row of the ContentPanel. Since you'll need to reference the TextBox from code when you display the greeting message, give it the name nameInput:

```
<StackPanel Grid.Row="1" Margin="{StaticResource PhoneMargin}">
    <TextBlock>Enter your name:</TextBlock>
    <TextBox x:Name="nameInput" InputScope="Text"/>
</StackPanel>
```

When you run the application (see figure 2.1), you'll notice that the font sizes for the two controls are different, even though you didn't specify any font information. The TextBlock adopts the FontSize of its parent containers, in this case from the page itself. Remember that the project template set the page's FontSize to the PhoneFontSizeNormal system resource.

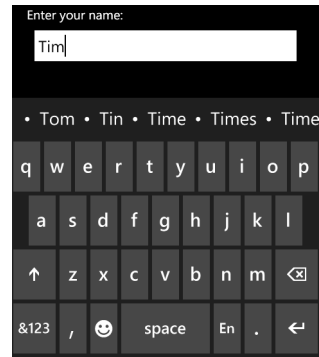
The TextBox retrieves its font information from the default TextBox control template. The TextBox control template sets FontSize to the PhoneFontSizeMediumLarge system resource:

```
<Setter Property="FontFamily"
    Value="{StaticResource PhoneFontFamilyNormal}"/>
<Setter Property="FontSize"
    Value="{StaticResource PhoneFontSizeMediumLarge}"/>
```

When the user touches inside the TextBox the on-screen keyboard is displayed if the device doesn't have a physical keyboard. By default, the standard QWERTY keyboard is displayed as shown in figure 2.10. We recommend that you always specify an InputScope, even if you just use the Text input scope that you've used here. The Text input scope provides word correction features that aren't available with the default input scope. Other

keyboard layouts, such as Number or Url, can be specified. InputScopes are covered in more depth in chapter 11.

The on-screen keyboard also exposes clipboard copy and paste operations. TextBox automatically supports the clipboard, and your application doesn't need to do anything special to enable clipboard operations. Developers can programmatically copy text to the system clipboard to share with other applications. Before we show you how to copy text to the clipboard, let's look at how touch gestures are supported. Your application can listen for Tap gestures and perform custom actions in response to gesture events.



**Figure 2.10** Inputting text with the on-screen keyboard

### 2.3.2 Touch gestures

The *User Experience Design Guidelines for Windows Phone* defines the touch gestures Tap, Double Tap, Hold, Pan, and Flick. The initial Windows Phone SDK didn't expose any gestures from Silverlight controls. The Windows Phone SDK 7.1 introduced three gesture events:

- Tap
- DoubleTap
- Hold

To demonstrate how touch gestures can be used in an application, you're going to change the color of the globe when it's tapped by the user. Changing the color of the globe can be accomplished by changing the color of the brush used to draw the globe's ellipse and path graphics. Remember that you bound all of the graphic elements to the static resource named `GlobeBrush`. To access the brush resource from code, you need to define a field and then initialize the field with the `SolidColorBrush` that's stored in the `ContentPanel`'s resource dictionary:

```
SolidColorBrush globeBrush;

public HelloPage()
{
    InitializeComponent();
    globeBrush = (SolidColorBrush)ContentPanel.Resources["GlobeBrush"];
}
```

Before you implement the Tap and DoubleTap event handlers, you need to add a couple of fields to enable color changes. The first is an array of colors and the second is an index of the current color:

```
Color[] colors = new Color[] { Colors.Red, Colors.Orange,
    Colors.Yellow, Colors.Green, Colors.Blue, Colors.Purple };
int colorIndex = 0;
```

Next, you hook up the Tap and DoubleTap events to the canvas panel containing the globe:

```
<Canvas Width="200" Height="200" VerticalAlignment="Center"
    Background="{StaticResource PhoneBackgroundBrush}"
    Tap="Canvas_Tap" DoubleTap="Canvas_DoubleTap">
```

In the Tap event handler you want to assign the globeBrush's Color property to the next color in the colors array. Don't forget to check the index and reset it to the beginning of the array:

```
private void Canvas_Tap(object sender, GestureEventArgs e)
{
    colorIndex++;
    if (colorIndex >= colors.Length)
        colorIndex = 0;
    globeBrush.Color = colors[colorIndex];
}
```

In the DoubleTap event handler you reset the brush color to the accent color provided by the system theme. The accent color can be obtained from the application resources:

```
private void Canvas_DoubleTap(object sender, GestureEventArgs e)
{
    globeBrush.Color = (Color)App.Current.Resources["PhoneAccentColor"];
}
```

If your application requires gestures beyond tap and hold, you'll need to process the raw manipulation events raised by the Silverlight controls. The UIElement class exposes ManipulationStarted, ManipulationDelta, and ManipulationCompleted events when a user touches, moves, and releases their finger from the screen. Converting manipulation events into gestures is beyond the scope of this book.

Now that you've learned about gestures, let's discuss how to copy text to the system clipboard. Your application will copy text to the clipboard when a toolbar button is pressed.

### 2.3.3 Adding a toolbar button

Windows Phone provides a built-in toolbar and menu control called the *application bar*. The Visual Studio project adds sample, commented out, application bar markup when the page is created. For your Hello World application, you need to add one button to HelloPage and three buttons to GreetingPage. In HelloPage.xaml, you'll replace the sample ApplicationBar buttons with your own button. Start by uncommenting the ApplicationBar markup and removing the second button and both example menu items:

```
<phone:PhoneApplicationPage.ApplicationBar>
    <shell:ApplicationBar IsVisible="True" IsMenuEnabled="False">
        <shell:ApplicationBarIconButton Text="say hello"
            IconUri="/Images/appbar.next.rest.png" />
    </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

Next, set text for the remaining button, named appBar\_button1, to "say hello". Finally, you need to specify an IconUri.

For GreetingPage, you also need to create an ApplicationBar and add buttons. You need three buttons: one labeled “ok”, a second labeled “copy”, and one more that’s labeled “pin”:

```
<phone:PhoneApplicationPage.ApplicationBar>
  <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
    <shell:ApplicationBarIconButton Text="ok"
      IconUri="/Images/appbar.check.rest.png" />
    <shell:ApplicationBarIconButton Text="copy"
      IconUri="/Images/appbar.save.rest.png" />
    <shell:ApplicationBarIconButton Text="pin"
      IconUri="/Images/appbar.next.rest.png" />
  </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

In this application, you’re using a few of the icons from the Windows Phone SDK, which are installed to c:\Program Files\Microsoft SDKs\Windows Phone\7.1\Icons\Dark. On 64-bit Windows, the SDK is installed in c:\Program Files (x86).

Create a project folder named Images and add the appbar.next.rest.png, appbar.check.rest.png, and appbar.save.rest.png files to the folder. For each of the images, set the Build Action to Content. More information about the application bar can be found in chapter 10.

Like any other button, ApplicationBarIconButtons raise a Click event when the user presses them. You’ll next register for the Click event on the copy button and implement the event handler. Update the button’s markup to declare the event handler:

```
<shell:ApplicationBarIconButton Text="copy"
  IconUri="/Images/appbar.save.rest.png" />
  Click="copyButton_Click" />
```

Next, add the event handler in GreetingPage.xaml.cs:

```
private void copyButton_Click(object sender, EventArgs e)
{
    string message = string.Format("Hello {0}!", helloMessage.Text);
    Clipboard.SetText(message);
}
```

The event handler constructs a message by concatenating the word Hello and the text in the helloMessage TextBlock. The greeting message is then copied to the Clipboard and is ready to be pasted into some other application.

In this section you learned how to receive typed text from the user, respond to touch gestures, and use the system application bar to display a toolbar buttons. You implemented a click handler for one of the toolbar buttons, but the other two buttons don’t perform any work. The unimplemented buttons will be used to navigate between the two pages, which we cover in the next section.

### The Model-View-ViewModel pattern

Many, but not all, Silverlight developers use the Model-View-ViewModel pattern (MVVM) to separate user interface markup and logic from application logic. The separation of UI and application logic promoted by MVVM is made possible with Silverlight's data binding, value converter, and commanding features. Input and TextBlock controls are bound to model objects, which often implement the INotifyPropertyChanged interface. Values are converted to strings using converter classes that implement IValueConverter. Click event handlers are eschewed in favor of command objects implementing the ICommand interface.

Though MVVM separates UI and business logic, it introduces complexity. We've intentionally avoided using the complexity of the MVVM pattern in the sample applications in the book. We've also avoided binding trivial properties such as messages displayed in a TextBlock, and have placed a great deal of our application logic in the page code behind. MVVM is a great pattern that's well suited for XAML applications but one criticism of MVVM is that it's overkill for simple applications.

This isn't a book about Silverlight, but about Windows Phone. The bits of Silverlight we use in the sample applications are intended to highlight the features of the Windows Phone SDK that aren't available to browser-based Silverlight applications.

## 2.4 Page navigation

A phone application is a modified version of a Silverlight Navigation Application. Silverlight Navigation Applications are composed of a navigation frame and one or more pages that interact with the NavigationService. The NavigationService interacts with the operating system to maintain a journal or history of pages visited by the user. In this section, you're going to add navigation to the Hello World application.

### 2.4.1 Navigating to another page

Page navigation is the process that takes the user from one page to another. One example is when the user presses a button to open a new page, and then after completing some work, presses another button to come back to the main page. Navigation is managed by the NavigationService class. The NavigationService.Navigate method is called to move to a new page. When Navigate is called, the current page is placed on the navigation stack, and a new instance of the target page is generated. The NavigationService.GoBack method removes the current page and restores the previous page that's on the navigation stack.

You'll now add page navigation to your Hello World application. Starting in HelloPage.xaml add a click event handler to the "say hello" button:

```
<shell:ApplicationBarIconButton Text="say hello"
    IconUri="/Images/appbar.next.rest.png"
    Click="navigateForwardButton_Click" />
```

You want to navigate to GreetingPage when the button is pressed, so you need to add code to the click handler:

```
private void navigateForwardButton_Click(object sender, RoutedEventArgs e)
{
    this.NavigationService.Navigate(
        new Uri("/GreetingPage.xaml", UriKind.Relative));
}
```

You access the `NavigationService` via the `PhoneApplicationPage`'s `NavigationService` property. The `Navigate` method accepts an `Uri`, which in this case is the name of the file containing the page you wish to load. You construct the `Uri` using `UriKind.Relative`, as it's part of the same XAP file.

Now you want to reopen `GreetingPage.xaml` and generate the click event handler for the OK application bar button:

```
<shell:ApplicationBarIconButton Text="ok"
    IconUri="/Images/appbar.check.rest.png"
    Click="navigateBackButton_Click" />
```

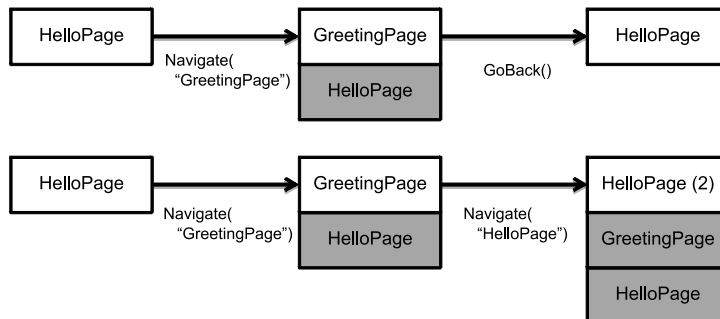
You implement the handler by calling the `GoBack` method:

```
private void navigateBackButton_Click(object sender, EventArgs e)
{
    this.NavigationService.GoBack();
}
```

Now press F5, or select the `Debug->Start Debugging` menu option, and debug the application. You've just linked your two pages using only two lines of code. Press the hello button and see the second page appear. When you press the OK button, the main page appears again.

It's worth noting that you use the `GoBack` method instead of the `Navigate` method to return to `MainPage.xaml`. When you call `GoBack` the current page is removed from the page stack. If you'd used `Navigate`, a new page would've been added on top of the page stack. Depending on the scenario you want to achieve, you can choose the approach more appropriate for your application, but you must be aware of the consequences. Both the cases are illustrated in Figure 2.11.

Let's examine the two scenarios presented in figure 2.11. In the top sequence the navigation uses `GoBack` to return to `HelloPage`, so the page stack is reduced. In the sequence on the bottom, the navigation uses `Navigate` to navigate to `HelloPage`, and a new page is added on top of the page stack and made visible.



**Figure 2.11** The navigation page stack resulting from `GoBack` (top) and `Navigate` (bottom) method calls. The white boxes represent the visible page, whereas the shaded boxes are the pages in the background.

Your Hello World application now moves from one page to another, and the greeting page starts up as expected. How do you get the user-entered name from the hello page to the greeting page? The Silverlight Navigation Framework provides features to enable passing data into a newly launched page.

## 2.4.2 Passing parameters between pages

In the previous example you concentrated on navigation between pages, but didn't pass any information to the greeting page. In theory, pages should be as self-contained as possible in order to maintain isolation between the pages, but it can be useful to pass parameters when navigating. You could choose to use some form of global data or data cached in the App class instead of passing data, but you should consider passing parameters in the Uri much as you would pass data to a constructor. As you'll learn later in the chapter, the operating system can call your page directly without ever constructing an instance of your main page.

In your sample main page, the user enters a name into a text box control named `nameInput` whose `Text` property will be used as a parameter passed to `GreetingPage`. `GreetingPage` will set the text block having name `helloMessage` with the parameter passed by `HelloPage`. Two changes are required in your code to pass a parameter—`HelloPage` must pass the parameter value to `GreetingPage` via the navigation Uri and `GreetingPage` must extract the parameter value from the query string.

Earlier you just created a URI with the hard-coded name of the greeting page in the `navigateForwardButton_Click` method. You could choose to hard-code the parameters as well, but now you have more magic strings in your code. What if you change the name of the greeting page, or change the name of the parameters passed in the query string? You'll next move Uri construction code into a static method of the `GreetingPage` class. Modify the Uri to pass a parameter in the same manner that you would if you were adding fields to a standard HTTP query string:

```
public static Uri BuildNavigationUri(string name)
{
    return new Uri("/GreetingPage.xaml?name=" + name, UriKind.Relative);
}
```

Update the `navigateForwardButton_Click` method in `HelloPage.xaml.cs` to call the new factory method, passing along the name entered by the user. The parameter value is obtained from the `nameInput` control:

```
NavigationService.Navigate(
    GreetingPage.BuildNavigationUri(nameInput.Text));
```

The data passed via the navigation Uri can be retrieved from the target page's `NavigationContext` property. The `NavigationContext` class has a single property named `QueryString`, which is an `IDictionary<string, string>` mapping parameter names to values.

You'll use the `NavigationContext` in the code behind for `GreetingPage.xaml`. The appropriate time to access the query string is after the page navigation has completed.

The navigation framework calls the `PhoneApplicationPage.OnNavigatedTo` virtual method when navigation is complete. In the sample application, you override `OnNavigatedTo` and obtain the parameter value by using the string "name" as a key into the `QueryString`. You set the returned value into `Text` property of `helloMessage`:

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    helloMessage.Text =
        this.NavigationContext.QueryString["name"];
}
```

`OnNavigatedTo` is one of the virtual methods defined by `PhoneApplicationPage` that are called when navigation events occurs. `OnNavigatedFrom` and `OnNavigatingFrom` are two other methods you can use to determine when the current page is changing.

In this section we've shown how you can navigate between pages from your software. Other activities, such as the user pressing the hardware Back key, can cause navigation changes in your application. Every Windows Phone is equipped with a hardware Back key. Its effect in an application is equivalent to calling the `NavigationService.GoBack` method.

### 2.4.3 *Changing the Back key behavior*

When the Back key is pressed, the navigation framework automatically performs a `GoBack` operation. The Back key behavior can be interrupted, for instance, to avoid moving off a page that has unsaved changes. To interrupt the automatic `GoBack`, the `PhoneApplicationPage` class provides an event named `BackKeyPress`. You'll see this event in action by wiring it to an event handler in your `GreetingPage` class. You first need to edit `GreetingPage.xaml` by adding an attribute to the `PhoneApplicationPage` tag:

```
BackKeyPress="Page_BackKeyPress"
```

Next, you add the event handler to `GreetingPage.xaml.cs`. This example prompts the user with a confirmation message:

```
private void Page_BackKeyPress(object sender, CancelEventArgs e)
{
    MessageBoxResult result = MessageBox.Show(
        "Press OK to return to the previous page.",
        "WP7 in Action", MessageBoxButton.OKCancel);
    if (result == MessageBoxResult.Cancel)
        e.Cancel = true;
}
```

If the user presses the Cancel button in the message box, you set the `CancelEventArgs.Cancel` property to true. This cancels the default behavior of the Back key. Failing to add this statement or setting `e.Cancel` to false would have maintained the default behavior, which is to move to the previous page or to terminate the application if no other pages are in the page stack.



**NOTE** The *Application Certification Requirements for Windows Phone* details appropriate application behavior when working with the back key. Specifically, when the Back button is pressed while the main page is visible, the application must exit.

Navigation relies on URI and query strings to navigate to specific locations within an application. Navigation strings can also be used to allow a user to launch to a specific location within your application with application tiles

#### 2.4.4 Navigating with tiles

Windows Phone users can pin an application's tile to the start screen. Tiles are large icons that display a background image and a title. We show you how to customize the main application tile in the next section. Starting with Windows Phone SDK 7.5, applications can also create secondary tiles that will navigate directly to a specific location in an application. Figure 2.12 shows the application and secondary tiles for your Hello World application.

When the user clicks the application tile for Hello World, the application is launched and the `NavigationService` is called with the URL `/HelloPage.xaml`. When the user clicks the secondary tile, the application is launched and the application host passes the URL associated with the secondary tile to the `NavigationService`. When you built the `GreetingPage`, you added a pin button to the application bar. Add a click handler to the pin button and implement code to create a secondary tile:

```
private void pinButton_Click(object sender, EventArgs e)
{
    StandardTileData tileData = new StandardTileData
    {
        BackgroundImage = new Uri("Background.png", UriKind.Relative),
        Title = string.Format("Hello {0}!", helloMessage.Text),
    };
    ShellTile.Create(BuildNavigationUri(helloMessage.Text), tileData);
}
```

The `StandardTileData` class has several properties that describe the tile. In your application you only use the `BackgroundImage` and the `Title` properties. You set the `BackgroundImage` property to use `Background.png`, the image specified in the project properties for your main tile. You set the `Title` property to be the greeting message. The `Create` method of the `ShellTile` class is used to create the new tile. You specify the `Url` to the greeting page, passing the same parameters that are specified by `HelloPage`. When `ShellTile.Create` is called, the application exits and the start screen is launched, showing the new secondary tile to the user.



**Figure 2.12** Application and secondary tiles for Hello World

Tiles have several other features that include flip side background images, content, and counters. Tiles can be dynamically updated or deleted by application code. These tile features are covered in chapter 9.

In this section you implemented the final requirement for your Hello World application—navigating to a second page and displaying a greeting to the user. You learned about the `NavigationService` and how to use query string parameters to pass data between pages. Finally, you learned how to use tiles to navigate directly to the greeting page from the start screen. Now you'll add some polish and customize the start-up experience with your own splash screen and other artwork.

## 2.5 **Application artwork**

The Windows Phone operating system expects your Silverlight application to provide a few different artwork files which it uses to represent your application to the user. Depending on how your application is built and configured, your application artwork can be displayed in the Start Screen, the Application List, the Games Hub, and the Music + Videos Hub. The Silverlight Framework also looks for a splash screen image when launching your application. In this section we discuss how to update or replace the artwork created by the project templates. We also discuss the image formats and sizes that are expected by Windows Phone. Let's begin with the splash screen.

### 2.5.1 **Customizing the splash screen**

When a Silverlight application is loaded, the application framework briefly displays a splash screen while constructing and navigating to the first page. If the page's constructor or the `OnNavigatedTo` methods perform lengthy operations, the splash screen will remain visible until the work is completed.

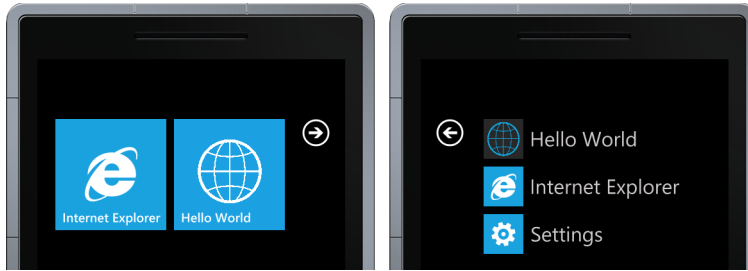
**NOTE** The *Application Certification Requirements for Windows Phone* recommends that you only provide a splash screen image when your application takes longer than 1 second to display the first page. The certification requirements also require that the first page be shown in less than five seconds.

The splash screen displayed by the framework is a static image, and can't be updated or replaced during runtime. The image used by the framework comes from the file named `SplashScreenImage.jpg`. The default 480 x 800 pixel image created by project templates is a dark gray background with a clock face. A custom splash screen image can be used by simply overwriting `SplashScreenImage.jpg` with another file of the same name.

There are two other image files that are created by the project template: the tile image and the application icon.

### 2.5.2 **Customizing tile images and application icons**

The tile image is used by the operating system when an application or game is pinned to the start screen. Application icons are used by the operating system when an application



**Figure 2.13** Custom images used in the start experience and the application list

appears in the application list. Figure 2.13 shows the start screen and application list custom globe images.

The Visual Studio project templates create default tile background images named `Background.png` for Silverlight applications. Silverlight projects are created with an application icon named `ApplicationIcon.png`. You can replace the default tile background image following these steps:

- Create a new 173 x 173 pixel PNG file.
- The image should have a 12-pixel margin on all edges.
- The image should reserve a 37 x 37 pixel area inside the top-right margin for tile notifications. (Tile notifications are covered in chapter 9.)
- Add the new image to the root of the project and specify the Content build action.
- In the Application tab of the project properties, select the new image in the Background image field.

The tile image is displayed in the Start Experience with the tile title. You can set a custom tile title with the Tile Title field in the Application tab of the project properties. An application can dynamically update its tile background image, which is covered in more depth in chapter 9. Applications implementing notifications commonly update the tile background image.

The tile images for the native phone application use the system theme's accent color as a background color. When the user changes the accent color, the tile background is updated to match. You can also design your tile images to use the theme's accent color by using transparency in your PNG file. Transparent pixels in the tile image will allow the accent color to show through.

You can replace the default application icon following these steps:

- Create a new 62 x 62 pixel PNG file for an application. If your application will be displayed in the Games Hub, the image should be 173 x 173 pixels.
- Add the new image to the root of the project and specify the Content build action.
- Open the Application tab of the project properties and select the new image in the Deployment Icon field.

The Games Hub expects the application icon/game thumbnail to be 173 x 173 pixels. The Application List expects the icon to be 62 x 62 pixels. ApplicationIcon.png is created by the project template as 62 x 62 pixel files. You should replace these files with larger images if your Silverlight application will appear in the Games Hub. Customizing the tile and application images improves an application's integration into the overall Windows Phone experience.

## 2.6 *Try before you buy*

Before committing hard-earned money to a purchase, many users like to use a demonstration or trial version of an application, and the Windows Application Marketplace provides support for limited trials. Applications use the `IsTrial` method of the `LicenseInformation` class to determine if the application is running under a trial license:

```
LicenseInformation licenseInfo = new LicenseInformation();
if (licenseInfo.IsTrial())
{
    // implement trial mode logic here...
}
```

The manner in which the trial mode is implemented is up to the discretion of the developer. Trial mode applications can be restricted to a certain number of days, a certain number of launches, have a restricted feature set, or some other limitation.

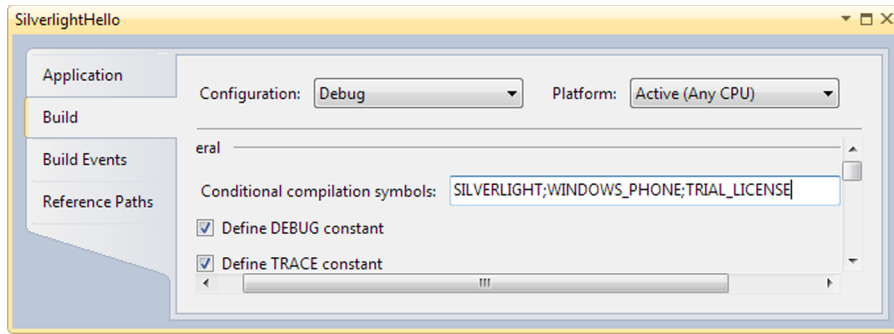
Applications running in trial mode should provide a “buy me” link to purchase the application from the Application Marketplace. The marketplace link can be implemented using the `MarketplaceDetailTask`, which is covered in section 4.2.4. Applications should re-check the trial when first launched, or when reactivated from a dormant or tombstoned state, as the user may have purchased the application while it was inactive. Checking the trial status can be time consuming and shouldn't be performed in a tight loop.

Developers should always test their applications in both trial and unlimited modes. Testing can be problematic as the `IsTrial` method always returns false when running in the emulator. Conditional compilation techniques can be used to test trial licensing:

```
LicenseInformation licenseInfo = new LicenseInformation();
#if TRIAL_LICENSE
    bool isInTrialMode = true;
#else
    bool isInTrialMode = licenseInfo.IsTrial();
#endif
if (isInTrialMode)
{
    // implement trial mode logic here...
}
```

To turn on trial licensing, all you need to do is add a conditional compilation symbol as shown in figure 2.14.

The application marketplace's trial licensing makes it easy to provide potential customers a preview of your application or game, without the need to build multiple



**Figure 2.14** Setting a conditional compilation symbol

versions of your project. Trial licensing eliminates the need to maintain and publish a separate free or light version of your product.

## 2.7 Summary

The Windows Phone Developer Tools help you build many different kinds of Silverlight applications. Project templates are provided for simple projects and class libraries as well as list, pivot, panorama, and Silverlight with XNA style projects. Chapter 10 covers panorama and pivot applications, whereas Silverlight with XNA applications are covered in chapter 14.

The Silverlight framework makes it easy to align your application with the system theme and style. The framework injects resources into applications so they can match the system theme (light versus dark, accent color) and the look and feel (fonts, colors, sizes). The visual designers and property editors in Expression Blend and Visual Studio expose theme resources.

Silverlight has been extended for Windows Phone with components built specifically for the platform. New navigation frame, page control, and application bar components are just a few of the additions. Other existing Silverlight controls have been modified to work on the phone. The chapters in part 3 of this book look at these new and modified components.

Finally you learned some of the procedures for integrating with the phone operating system. Live tiles and application icons can be customized to make your application stand out in the quick start, application list, and Game Hub. System capabilities must be declared in order to use many of the core phone APIs. The phone APIs provide access to the native applications, services, sensors, and media features of the phone. In part 2 of this book you read about how to use the phone APIs. In the next chapter you learn how Windows Phone implements application multitasking and how to design application-to-lifecycle events. We also look at how to create and run background agents to perform work while other applications run in the foreground.

# Windows Phone 7 IN ACTION

Binkley-Jones • Perga • Sync



**W**indows Phone 7 is a powerful mobile platform sporting the same Metro interface as Windows 8. It offers a rich environment for apps, browsing, and media. Developers code the OS and hardware using familiar .NET tools like C# and XAML. And the new Windows Store offers an app marketplace reaching millions of users.

**Windows Phone 7 in Action** is a hands-on guide to programming the WP7 platform. It zips through standard phone, text, and email controls and dives head-first into how to build great mobile apps. You'll master the hardware APIs, access web services, and learn to build location and push applications. Along the way, you'll see how to create the stunning visual effects that can separate your apps from the pack.

## What's Inside

- Full introduction to WP7 and Metro
- HTML5 hooks for media, animation, and more
- XNA for stunning 3D graphics
- Selling apps in the Windows Store

Written for developers familiar with .NET and Visual Studio. No WP7 or mobile experience is required.

**Timothy Binkley-Jones** is a software engineer with extensive experience developing commercial IT, web, and mobile applications. **Massimo Perga** is a software engineer at Microsoft and **Michael Sync** is a solution architect for Silverlight and WP7.

To download their free eBook in PDF, ePub and Kindle formats, owners of this book should visit [manning.com/WindowsPhone7inAction](http://manning.com/WindowsPhone7inAction)

“Definitely recommended!”

—Vipul Patel, Amazon.com

“Top resource for Windows Phone developers.”

—Loïc Simon, Solent SAS

“A great handbook for climbing the WP7 ladder.”

—Francesco Goggi  
Magneti Marelli

“Gives you a kickstart in Windows Phone development.”

—Mark Monster  
Monster Consultancy

