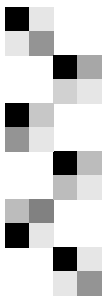
Essential Windows® Phone 8

Shawn **Wildermuth**

Essential Windows Phone 8

This page intentionally left blank



Essential Windows Phone 8

Shawn Wildermuth

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

The Library of Congress cataloging-in-publication data is on file.

ISBN-13: 978-0-321-90494-2
ISBN-10: 0-321-90494-X

Text printed in the United States on recycled paper at Edwards Brothers in Ann Arbor, Michigan.
First printing May 2013

To Resa Shwarts—for her patience and education on the passive voice.

This page intentionally left blank

Contents at a Glance

This page intentionally left blank

Contents

Preface

I never owned a Palm Pilot. But I did have Palm tops and smartphones. I dived into writing software for a plethora of devices but never got very far. My problem was that the story of getting software onto the phones was chaotic and I didn't see how the marketing of software for phones would lead to a successful product. In the intervening years, I got distracted by Silverlight and web development. I didn't pay attention as the smartphone revolution happened. I was happily neck deep in data binding, business application development, and teaching XAML.

The smart revolution clearly started with the iPhone. What I find interesting is that the iPhone is really about the App Store, not the phone. It's a great device, but the App Store is what changed everything. A simple way to publish, market, and monetize applications for these handheld powerhouses that people all wanted. Of course, Apple didn't mean to do it. When the original iPhone shipped, Apple clearly said that Safari (its web browser) was the development environment. With the pressure of its OSX developer community, Apple relented and somewhat accidently created the app revolution.

When it was clear that I had missed something, I dove headlong into looking at development for phones again. I had an Android phone at the time, so that is where I started. Getting up to speed with Eclipse and Java wasn't too hard, but developing for the phone was still a bit of a chore. The development tools just didn't seem to be as easy as the development I was

used to with Visual Studio and Blend. In this same timeframe, I grabbed a Mac and tried my hand at Objective-C and Xcode to write something simple for the iPhone. That experience left me bandaged and bloody. I wanted to write apps, but because it was a side effort, the friction of the toolsets for Android and iPhone left me wanting and I put them aside.

Soon after my experience with the iPhone and Android, Microsoft took the cover off its new phone platform: the Windows Phone. For me, the real excitement was the development experience. At that point, I'd been teaching and writing about Silverlight since it was called WPF/E, so the ability to take my interest in mobile development and marry it to my Silverlight knowledge seemed like a perfect match.

I've enjoyed taking the desktop/web Silverlight experience I have and applying the same concepts to the phone. By using Visual Studio and Blend to craft beautiful user interface designs and quickly go from prototype to finished application, the workflow of using these tools and XAML makes the path of building my own applications much easier than on other platforms.

After my experience writing the first edition of this book, I was excited about the newest version of the Windows Phone that Microsoft unveiled: Windows Phone 8. It was more than just a few added features—it was a real change to the underlying operating system. This was a big change, but how did it affect Windows Phone 7 and 7.5 developers? Microsoft could have really affected those developers by just shuttering the entire back catalog and changing the APIs in the new edition. I am happy to tell you that Microsoft walked the razor-thin line between change and backward compatibility. But how would that change the book I wrote?

Coming back to a book for a second edition is a challenge for any author. At the face of it, I could have just done a quick search-and-replace to change Windows Phone 7.5 to Windows Phone 8 and moved on, but I felt like I could really highlight and improve the first edition. My goal was to make the book approachable for both developers new to the Windows Phone, as well as show the new features. I hope I've been able to accomplish this.

While the changes are peppered all over this edition, for me the most striking change comes in the very last chapter: enterprise development.

Microsoft allows you to build your own applications for the phone that can be delivered to your own employees without certification or validation. Microsoft is finally opening the door to the power of using the Windows Phone as a platform for your employees.

Acknowledgments

This is the second edition of this book, and that changes the way you write a book. Writing is hard…editing is even more difficult. It is easy to simply miss the obvious changes that should be caught. To that end, three people have been pivotal in getting this book correct (or perhaps will shoulder the blame for any errata that got missed ;):

I want to thank Joan Murray at Addison-Wesley for maintaining her incredible patience. When I am writing, I tend to vacillate between an excited pixie and an angry imp. She handles that swinging pendulum better than most.

In addition, I want to again thank Christopher Cleveland for the job of developmental editor. He's been great at keeping the little things in mind and making sure the *i*'s are dotted and the *t*s are crossed.

I'd also like to specially thank Jeff Wilcox (of 4th and Mayor fame) for doing an exemplary job at tech reviewing the chapters. He's been critical at getting the phone story right. He's been in the thick of building a large Windows Phone application that has weathered the storms of a high number of users, and his experience shows in every chapter.

To the litany of people on the Windows Phone Advisor's Mailing Lists, I would like to thank you for your patience as I pestered the lists with endless questions and hyperbolic rants.

Like the first edition, my blog's readers and my followers on Facebook and Twitter have helped immeasurably with their passion, wit, and knowledge.

For anyone else I forgot, I apologize.

Shawn Wildermuth

May 2013

http://wildermuth.com

@shawnwildermuth

About the Author

During his 27 years in software development, **Shawn Wildermuth** has experienced a litany of shifts in software development, shaping how he understands technology. Shawn is a 10-time Microsoft MVP, a member of the INETA Speaker's Bureau, and an author of several books on .NET. He has spoken at a variety of international conferences, including TechEd, MIX, VSLive, OreDev, SDC, WinDev, MIX, DevTeach, DevConnections, and DevReach. Shawn has written dozens of articles for a variety of magazines and websites, including *MSDN*, *DevSource*, *InformIT*, *CoDe Magazine*, *ServerSide.NET*, and *MSDN Online*. He is currently helping companies with coaching and training through his company Wilder Minds LLC (http://wilderminds.com).

2

Writing Your First Phone Application

While the press might have you believe that becoming a phone-app millionaire is a common occurrence, it's actually pretty rare, but that doesn't mean you won't want to create applications for the phone. Hopefully the days of cheap and useless but popular phone apps are over, and we can start focusing on phone-app development as being a way to create great experiences for small and large audiences. Microsoft's vision of three screens is becoming a reality, as the phone is joining the desktop and the TV as another vehicle for you to create immersive experiences for users.

Although understanding Windows Phone capabilities and services is a good start, you are probably here to write applications. With that in mind, this chapter will walk you through setting up a machine for authoring your very first Windows Phone application.

Preparing Your Machine

Before you can start writing applications for the phone, you must install the Windows Phone Developer Tools. Go to https://dev.windowsphone.com/ to download the tools called Windows Phone SDK. This website is the starting point for downloading the tools as well as accessing the forums if you have further questions about creating applications.

To install the Windows Phone SDK, you must meet the minimum system requirements shown in Table 2.1.

TABLE 2.1 Windows Phone Developer Tools Requirements

Requirement	Description
Operating system	Windows 7, x86 or x64 (all but Starter Edition); or Windows Vista SP2, x86, or x64 (all but Starter Edition).
Memory	3GB RAM.
Disk space	4GB free space.
Graphics card	DirectX 10-capable card with a WDDM 1.1 driver.

Once you meet the requirements, you can run the vm_web.exe file that you downloaded from the website to install the Windows Phone SDK. The SDK installer includes Microsoft Visual Studio 2012 Express for Windows Phone, Microsoft Blend Express for Windows Phone (the Express version of Microsoft Expression Blend), and the Software Development Kit (SDK). Visual Studio Express is the coding environment for Windows Phone. Blend Express is the design tool for phone applications. And the SDK is a set of libraries for creating phone applications and an emulator for creating applications without a device.

In addition, the Windows Phone SDK's phone emulator has additional requirements. This is because the Windows Phone SDK for Windows Phone 8 includes an all-new emulator that is a Hyper-V image (instead of the old virtual machine technology). This matters because the emulator has steeper requirements than the SDK itself. These requirements are shown in Table 2.2.

TABLE 2.2 Windows Phone Developer Tools Requirements

Requirement	Description
Operating system	Windows 8 Professional, 64-bit version
Memory	4GB RAM
Hyper-V	Installed and running
BIOS settings	Hardware Assisted Virtualization, Secondary Level Address Translation (SLAT) and Data Execution Protection (DEP) all enabled
Group membership	Must be member of both local Administrator and Hyper-V Administrator groups

> **TIP**
>
> The Windows Phone emulator does not work well in a virtual machine (for example, Virtual PC, VMware, and so on) and is not officially supported. The emulator is a virtual machine of the phone, so running a virtual machine in a virtual machine tends to cause problems, especially slow performance.

Visual Studio is the primary tool for writing the code for your phone applications. Although the Windows Phone SDK installs a version of Visual Studio 2012 Express specifically for phone development, if you already have Visual Studio 2012 installed on your machine, the phone tools will also be integrated into this version of Visual Studio. The workflow for writing code in both versions of Visual Studio is the same. Although both versions offer the same features for developing applications for the phone, in my examples I will be using Visual Studio Express Edition for Windows Phone. In addition, I will be using Blend Express, not the full version of Blend (that is, Expression Blend).

Creating a New Project

To begin creating your first Windows Phone application, you will need to start in one of two tools: Visual Studio or Expression Blend. Visual Studio is where most developers start their projects, so we will begin there; however, we will also discuss how you can use both applications for different parts of the development process.

Visual Studio

As noted earlier, when you install the Windows Phone SDK you get a version of Visual Studio 2010 Express that is used to create Windows Phone applications only. When you launch Visual Studio 2012 Express, you will see the main window of the application, as shown in Figure 2.1.

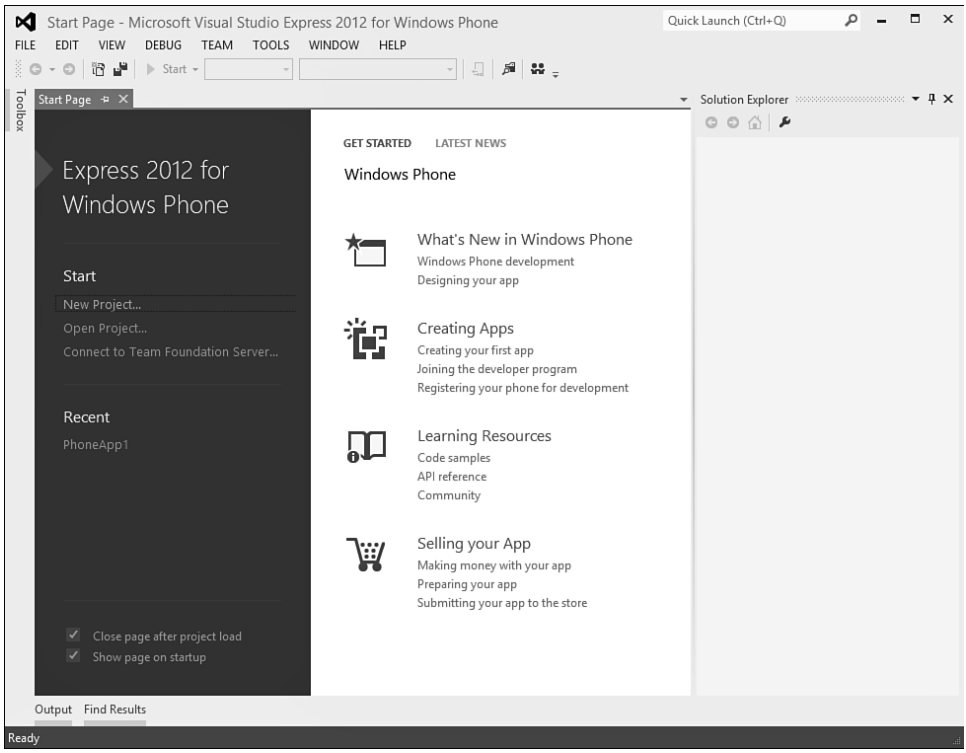


FIGURE 2.1 Microsoft Visual Studio 2012 Express for Windows Phone

Click the New Project link on the Start page; you will be prompted to start a new project. Visual Studio 2012 Express only supports creating applications for Window Phone. The New Project dialog box shows only Windows Phone and XNA projects (see Figure 2.2). For our first project we will start with a new project using the Windows Phone App template and name it HelloWorldPhone.

When you click the OK button to create the project, Visual Studio will prompt you with a dialog box where you can pick the version of the phone to target (version 7.1 or 8.0), as shown in Figure 2.3.

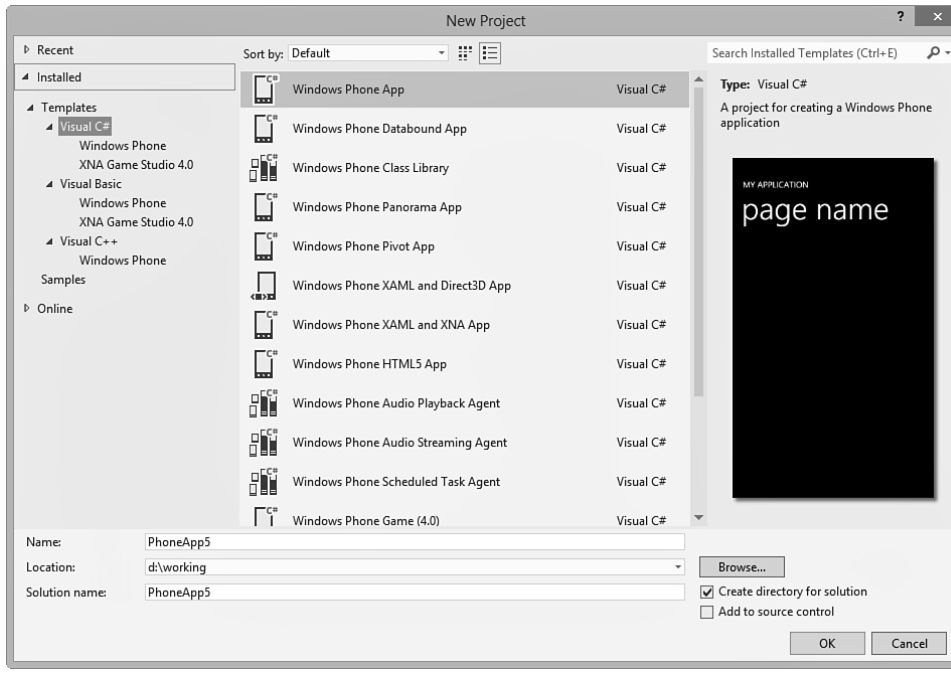


FIGURE 2.2 New Project dialog box

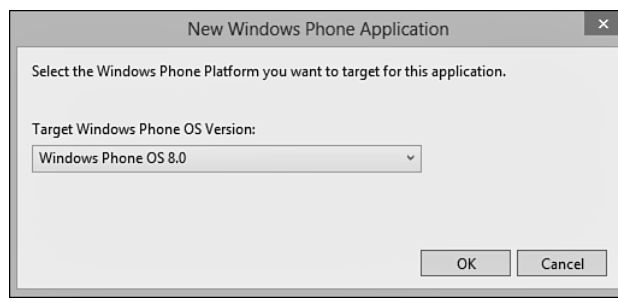


FIGURE 2.3 Picking the phone version to target

After Visual Studio creates the new project, you can take a quick tour of the user interface (as shown in Figure 2.4). By default, Visual Studio shows two main panes for creating your application. The first pane (labeled #1 in the figure) is the main editor surface for your application. In this pane, every edited file will appear separated with tabs as shown. By default, the MainPage.xaml file is shown when you create a new Windows Phone application; this is the main design document for your new application.

The second pane (#2 in the figure) is the Solution Explorer pane, and it displays the contents of the new project.

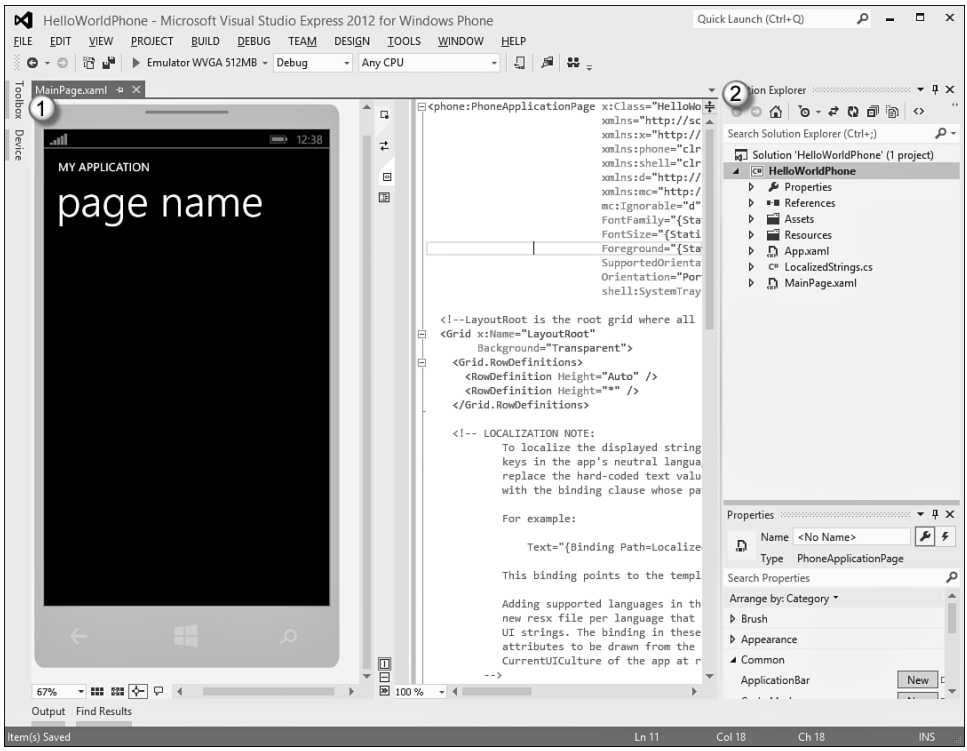


FIGURE 2.4 The Visual Studio user interface

Another common pane you will use is the toolbar; it is collapsed when you first use Visual Studio. On the left side of the main window is a Toolbox tab that you can click to display the Toolbox, as shown in Figure 2.5.

You also might want to click the pin icon to keep the toolbar shown at all times (as highlighted in Figure 2.5).

Before we look at how to create the application into something that is actually useful, let's see the application working in the device. You will notice that in the toolbar (not the Toolbox) of Visual Studio there is a bar for debugging. On that toolbar is a drop-down box for specifying what to do to debug your application. This drop-down should already display the words "Emulator WVGA 512MB," as that is the default when the tools are installed (as shown in Figure 2.6).

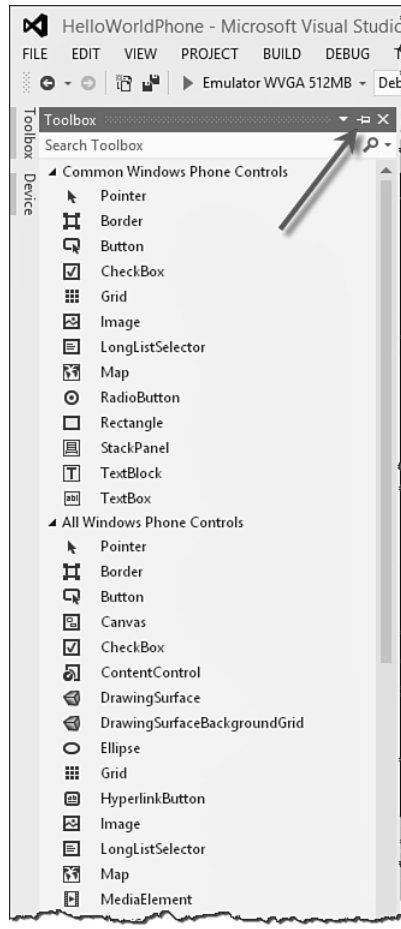


FIGURE 2.5 Enabling the toolbar

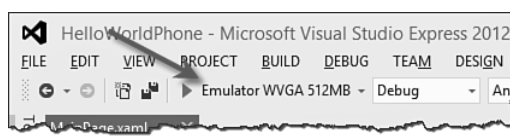


FIGURE 2.6 Using the emulator

At this point, if you press the F5 key (or click the triangular play button on the debugging toolbar), Visual Studio will build the application and start the emulator with our new application, as shown in Figure 2.7.

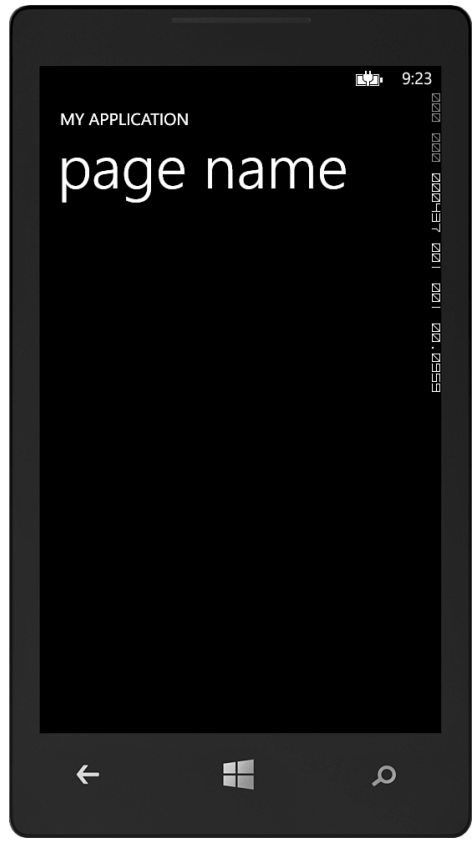


FIGURE 2.7 The emulator

This emulator will be the primary way you will debug your applications while developing applications for Windows Phone. Our application does not do anything, so you can go back to Visual Studio and click the square stop button on the debugging toolbar (or press Shift+F5) to end your debugging session. You should note that the emulator does not shut down. It is meant to stay running between debugging sessions.

XAML

In Silverlight, development is really split between the design and the code. The design is accomplished using a markup language called eXtensible Application Markup Language (XAML). XAML (rhymes with *camel*) is an XML-based language for representing the look and feel of

your applications. Because XAML is XML-based, the design consists of a hierarchy of elements that describe the design. At its most basic level, XAML can be used to represent the objects that describe the look and feel of an application.[1] These objects are represented by XML elements, like so:

```
<Rectangle />

<!-- or -->

<TextBox />
```

You can modify these XML elements by setting attributes to change the objects:

```
<Rectangle Fill="Blue" />

<!-- or -->

<TextBox Text="Hello World" />
```

Containers in XAML use XML nesting to imply ownership (a parent-child relationship):

```
<Grid>
  <Rectangle Fill="Blue" />
  <TextBox Text="Hello World" />
</Grid>
```

Using this simple XML-based syntax, you can create complex, compelling designs for your phone applications. With this knowledge in hand, we can make subtle changes to the XAML supplied to us from the template. We could modify the XAML directly, but instead we will start by using the Visual Studio designer for the phone. In the main editor pane of Visual Studio, the MainPage.xaml file is split between the designer and the text editor for the XAML. The left pane of the MainPage.xaml file is not just a preview but a fully usable editor. For example, if you click on the area containing the words "page name" on the design surface, it will select that element in the XAML, as shown in Figure 2.8.

1 This is an oversimplification of what XAML is. Chapter 3, "XAML Overview," will explain the nature of XAML in more detail.

FIGURE 2.8 Using the Visual Studio XAML design surface

When you have that element selected in the designer, the properties for the element are shown in the Properties window (which shows up below the Solution Explorer). If the window is not visible, you can enable it in the View menu by selecting "Properties window" or by pressing the F4 key. This window is shown in Figure 2.9.

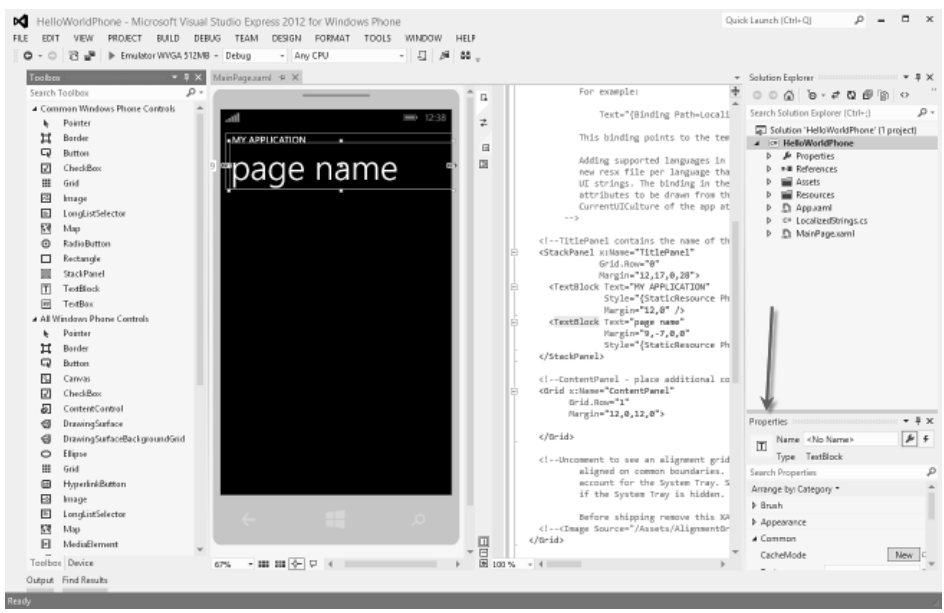


FIGURE 2.9 Location of the Properties window

The Properties window consists of a number of small parts containing a variety of information, as shown in Figure 2.10.

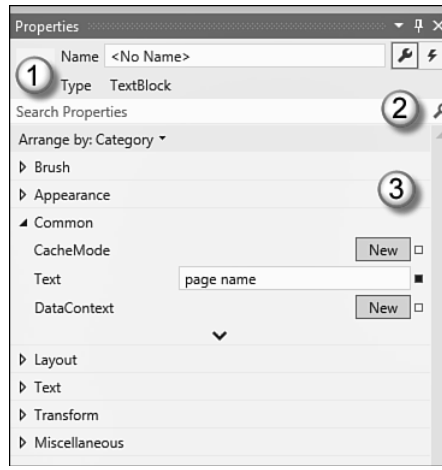


FIGURE 2.10 Contents of the Properties window

The section near the top (#1 in Figure 2.10) shows the type of object you have selected (in this example, a `TextBlock`) and the name of the object, if any (unspecified here so shown as <No Name>). This should help you ensure that you have selected the correct object to edit its properties. The next section down (#2) contains a Search bar where you can search for properties by name, as well as buttons for sorting and grouping the properties. The third section (#3) is a list of the properties that you can edit.

> **NOTE**
>
> You can also use the Properties window to edit events, but we will cover that in Chapter 3.

From the Properties window you can change the properties of the selected item. For example, to change the text that is in the `TextBlock`, you can simply type in a new value for the `Text` property. If you enter "hello world" in the `Text` property and press Return, the designer will change to display the new value. Changing this property actually changes the XAML in the MainPage.xaml file. The design surface is simply reacting to the change in the XAML. If you look at the XAML, the change has been affected there as well, as shown in Figure 2.11.

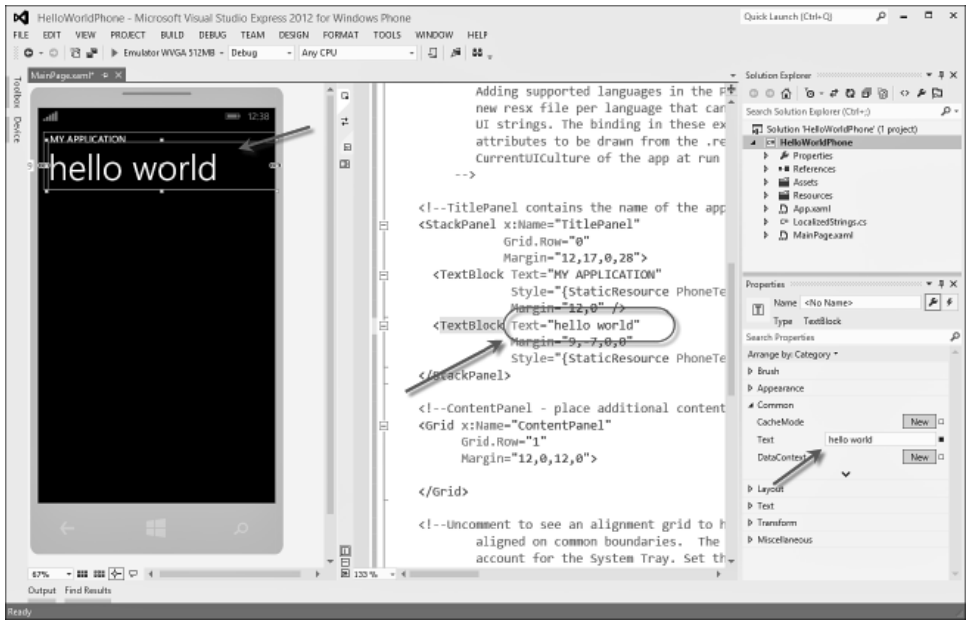


FIGURE 2.11 The changed property

You can edit the XAML directly as well if you prefer. If you click on the `TextBlock` above the `PageTitle` (the one labeled "ApplicationTitle"), you can edit the `Text` attribute directly. Try changing it to "MY FIRST WINDOWS PHONE APP" to see how it affects the designer and the Properties window:

```
...
<TextBlock x:Name="ApplicationTitle"
           Text="MY FIRST WINDOWS PHONE APP"
           Style="{StaticResource PhoneTextNormalStyle}" />
...
```

Depending on their comfort level, some developers find it easier to use the Properties window while others will be more at ease editing the XAML directly. There is no wrong way to do this.

Although the Visual Studio XAML designer can create interesting designs, the real powerhouse tool for designers and developers is Blend. Let's use it to edit our design into something useful for our users.

Designing with Blend

As noted earlier, in addition to offering an Express version of Visual Studio, the Windows Phone SDK includes an Express version of Expression Blend specifically for use in developing phone applications. You can launch Blend by looking for the shortcut key, or you can open it directly with Visual Studio. If you right-click the MainPage.xaml file, you will get a context menu like the one shown in Figure 2.12.

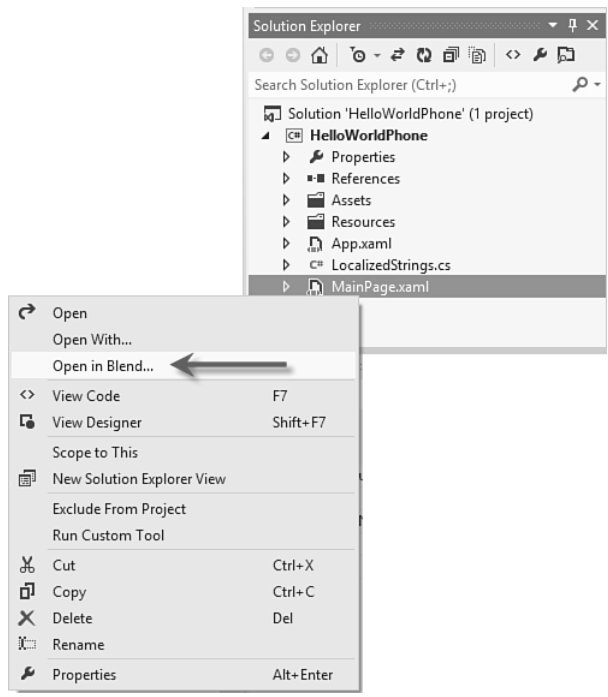


FIGURE 2.12 Opening Blend directly in Visual Studio

When you select Open in Expression Blend, Blend will open the same solution in the Expression Blend tool with the selected XAML file in the editor, as shown in Figure 2.13. You should save your project before going to Blend to make sure Blend loads any changes (Ctrl+Shift+S).

FIGURE 2.13 The Blend user interface

Although Expression Blend is thought of as purely a design tool, designers and developers alike can learn to become comfortable with it. And although Visual Studio and Expression Blend share some of the same features, both developers and designs will want to use Blend to build their designs. Some tasks are just simpler and faster to do in Blend. Chapter 5, "Designing for the Phone," covers which tasks are better suited to Expression Blend.

Like Visual Studio, Blend consists of a number of panes that you will need to get familiar with.

> **NOTE**
>
> Blend and Visual Studio both open entire solutions, not just files. This is a significant difference from typical design tools.

The first pane (labeled #1 in Figure 2.13) contains multiple tabs that give you access to several types of functionality. By default, the first tab (and the one in the foreground) is the Projects tab (although a different tab could be showing by default). This tab displays the entire solution of projects. The format of this tab should look familiar; it's showing the same information as the Solution Explorer in Visual Studio. The next pane (#2) is the editor pane. This pane contains tabs for each opened file (only one at this point). MainPage.xaml should be the file currently shown in the editor. Note that the editor displays the page in the context of the phone so that you can better visualize the experience on the phone. On the right side of the Blend interface is another set of tabs (#3) that contain information about selected items in the design surface. The selected tab should be the Properties tab. This tab is similar to the Properties window in Visual Studio but is decidedly more designer-friendly. As you select items on the design surface, you'll be able to edit them in the Properties tab here. Finally, the Objects and Timeline pane (#4) displays the structure of your XAML as a hierarchy.

Let's make some changes with Blend. First (as shown in Figure 2.14); select the "hello world" text in the designer.

FIGURE 2.14 Selecting an object in Blend

After it's selected, you can see that eight squares surround the selection. These are the handles with which you can change the size or location of the `TextBlock`. While this object is selected, the Objects and Timeline pane shows the item selected in the hierarchy; as well, the item is shown in the Properties tab so you can edit individual properties (as shown in Figure 2.15).

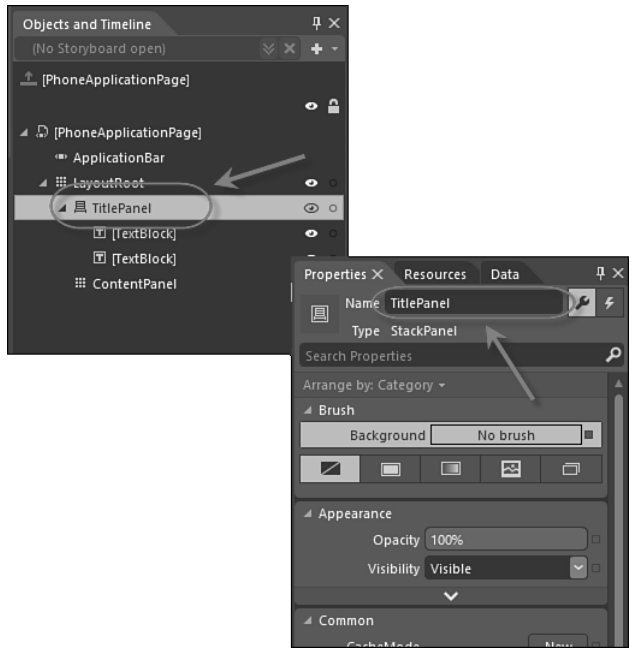


FIGURE 2.15 Selecting an object to edit in the Properties pane

If you type "text" into the search bar of the Properties pane, the properties that have that substring in them will appear (to temporarily reduce the number of properties in the Properties pane). You can change the title by changing the `Text` property, as shown in Figure 2.16.

FIGURE 2.16 Updating a property in Blend

After you're done changing the text, you might want to click the "X" in the Search bar to clear the search criteria. This will remove the search and show all the properties of the `TextBlock` again.

Selecting items and changing properties seems similar to what you can do in Visual Studio, but that's just where the design can start. Let's draw something. Start by selecting a container for the new drawing. In the Objects and Timeline pane, select the `ContentPanel` item. This will show you that it is a container that occupies most of the space below our "hello world" text on the phone's surface.

We can draw a rectangle in that container by using the left toolbar. On the toolbar is a rectangle tool (as shown in Figure 2.17). Select the tool and draw a rectangle in the `ContentPanel` to create a new rectangle (also shown in Figure 2.17). If you then select the top arrow tool (or press the V key), you'll be able to modify the rectangle.

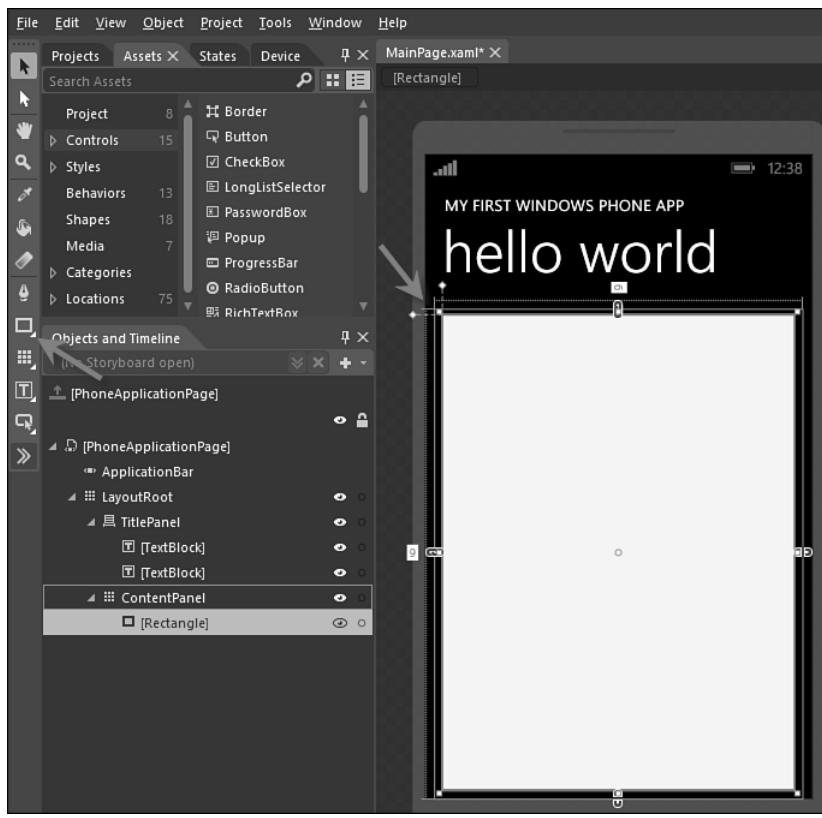


FIGURE 2.17 Drawing in a container

The rectangle you created has eight control points (the small squares at the corners and in the middle of each side). In addition, the rectangle has two small control points in the upper-left side (outside the surface area of the rectangle). These controls are used to round the corners of rectangles. Grab the top one with your mouse and change the corners to be rounded slightly, as shown in Figure 2.18.

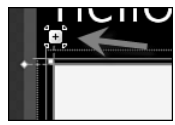


FIGURE 2.18 Rounding the corners

Now that you have rounded the corners, you can use the Properties pane to change the colors of the rectangle. In the Properties pane is a Brushes section showing how the various brushes for the rectangle are painted. The rectangle contains two brushes: a fill brush and a stroke brush. Selecting one of these brushes will allow you to use the lower part of the brush editor to change the look of that brush. Below the selection of brush names is a set of tabs for the various brush types, as shown in Figure 2.19.

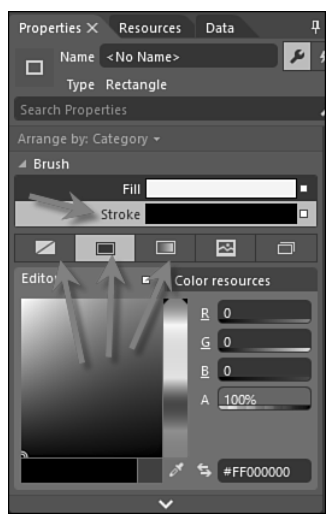


FIGURE 2.19 Editing brushes

The first four tabs indicate options for brushes. These include no brush, solid color brush, gradient brush, and tile brush. Select the stroke brush, and then select the first tab to remove the stroke brush from the new rectangle. Now select the fill brush, and change the color of the brush by selecting a color within the editor, as shown in Figure 2.20.

FIGURE 2.20 Picking a color

Now let's put some text in the middle of our design to show some data. More specifically, let's put a `TextBlock` on our design. Go back to the toolbar and double-click the TextBlock tool (as shown in Figure 2.21). Although we drew our rectangle, another option is to double-click the toolbar, which will insert the selected item into the current container (in this case, the `ContentPanel`). The inserted `TextBlock` is placed in the upper left of our `ContentPanel`, as also shown in Figure 2.21.

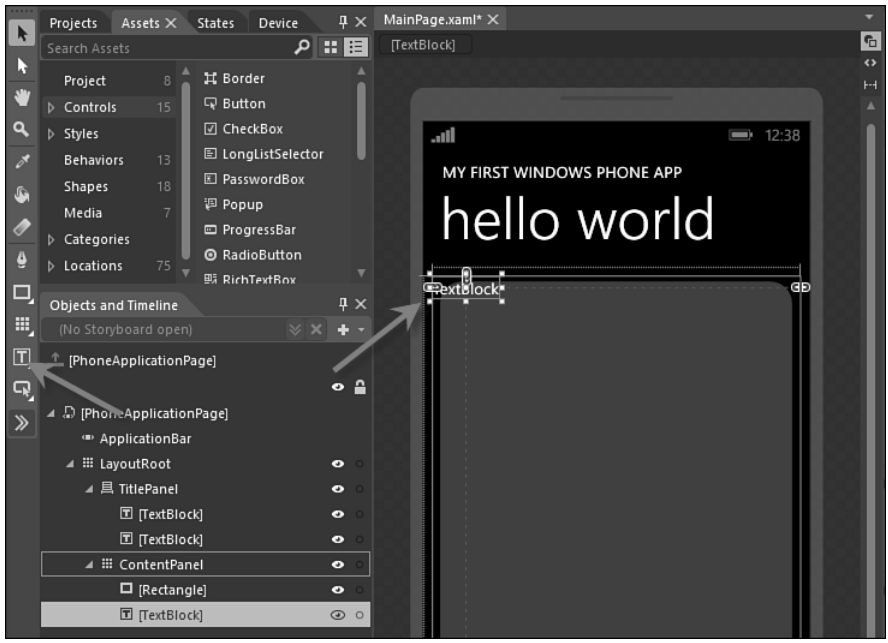


FIGURE 2.21 Inserting a `TextBlock`

After the new `TextBlock` is inserted, you can simply type to add some text. Type "Status" just to have a placeholder for some text we will place later in this chapter. You should use the mouse to click the Selection tool (the top arrow on the toolbar) so that you can edit the new `TextBlock`. You could use the mouse to place the `TextBlock` exactly where you like, but you could also use the Properties pane to align it. In the Properties pane, find the Layout section and select the horizontal center alignment and vertical bottom alignment, as shown in Figure 2.22. You might need to set your margins to zero as well to achieve the effect (because Blend might put a margin on your item depending on how you draw it).

Next you can edit the font and size of the `TextBlock` using the Text section of the Properties pane. You will likely need to scroll down to reach the Text section. From there, you can change the font, font size, and text decoration (for example, bold, italic, and so on). Change the font size to 36 points and make the font bold, as shown in Figure 2.23.

FIGURE 2.22 Centering the `TextBlock`

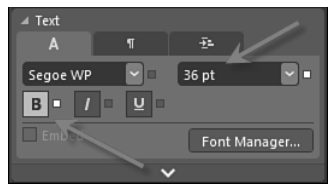


FIGURE 2.23 Changing the text properties

At this point our application does not do much, but hopefully you have gotten your first taste of the basics of using Blend for design. To get our first application to do something, we will need to hook up some of the elements with code. So we should close Blend and head back to Visual Studio.

When you exit Blend you will be prompted to save the project. Upon returning to Visual Studio, your changes will be noticed by Visual Studio; allow Visual Studio to reload the changes.

TIP

Blend is great at a variety of design tasks, such as creating animations, using behaviors to interact with user actions, and creating transitions. In subsequent chapters we will delve much further into using those parts of the tool.

Adding Code

This first Windows Phone application is not going to do much, but we should get started and make something happen with the phone. Because this is your first Windows Phone application, let's not pretend it is a desktop application but instead show off some of the touch capabilities.

First, if you look at the text of the XAML you should see that the first line of text shows the root element of the XAML to be a `PhoneApplicationPage`. This is the basic class from which each page you create will derive. The `x:Class` declaration is the name of the class that represents the class. If you open the code file, you will see this code was created for you:

```
<phone:PhoneApplicationPage x:Class="HelloWorldPhone.MainPage"
  ...
```

> **NOTE**
>
> The "phone" alias is an XML alias to a known namespace. If you're not familiar with how XML namespaces work, we will cover it in more detail in Chapter 3.

You will need to open the code file for the XAML file. You can do this by right-clicking the XAML page and picking View Code, or you can simply press F7 to open the code file. The initial code file is pretty simple, but you should see what the basics are. The namespace and class name match the `x:Class` definition we see in the XAML. This is how the two files are related to each other. If you change one, you will need to change the other. You should also note that the base class for the `MainPage` class is the same as the root element of the XAML. They are all related to each other. Here is the initial code file:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Navigation;
using Microsoft.Phone.Controls;
```

```
using Microsoft.Phone.Shell;
using HelloWorldPhone.Resources;

namespace HelloWorldPhone
{
  public partial class MainPage : PhoneApplicationPage
  {
    // Constructor
    public MainPage()
    {
      InitializeComponent();

      // Sample code to localize the ApplicationBar
      //BuildLocalizedApplicationBar();
    }
  }
  // ...
}
```

These two files (the .xaml and the code files) are closely tied to each other. In fact, you can see that if you find an element in the XAML that has a name, it will be available in the code file. If you switch back to the .xaml file, click the `TextBlock` that you created in Blend. You will notice in the Properties window that it does not have a name (as shown in Figure 2.24).

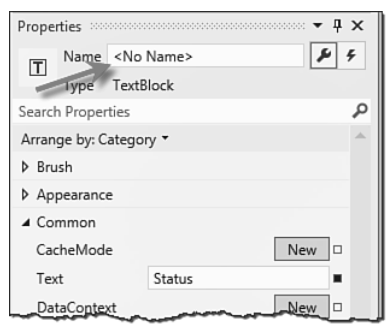


FIGURE 2.24 Naming an element in the Properties window

If you click the text "<no name>", you can enter a name. Name the `TextBlock` "theStatus." If you then switch over to the code file, you will be able to use that name as a member of the class:

```
...
public partial class MainPage : PhoneApplicationPage
{
  // Constructor
  public MainPage()
  {
    InitializeComponent();

    theStatus.Text = "Hello from Code";
  }
}
...
```

At this point, if you run the application (pressing F5 will do this), you will see that this line of code is being executed as the `theStatus TextBlock` is changed to show the new text (as seen in Figure 2.25).

FIGURE 2.25 Running the application

There is an important fact you should derive from knowing that named elements in the XAML become part of the class: The job of the XAML is to build an object graph. The hierarchy of the XAML is just about creating the hierarchy of objects. At runtime, you can modify these objects in whatever way you want.

When you stop your application, the emulator will continue to run. You can leave the emulator running across multiple invocations of your application. You should not close the emulator after debugging your application.

Working with Events

Because you are building a phone application, let's show how basic events work. You can wire up events just as easily using standard language (for example, C#) semantics.[2] For example, you could handle the `Tap` event on `theStatus` to run code when the text is tapped:

```
...
public partial class MainPage : PhoneApplicationPage
{
  // Constructor
  public MainPage()
  {
    InitializeComponent();

    theStatus.Text = "Hello from Code";

    theStatus.Tap += theStatus_Tap;

    // Sample code to localize the ApplicationBar
    //BuildLocalizedApplicationBar();
  }

  void theStatus_Tap(object sender,
                     System.Windows.Input.GestureEventArgs e)
  {
    theStatus.Text = "Status was Tapped";
  }
}
...
```

2 For Visual Basic, you would just use the `handles` keyword instead of the C# event handler syntax.

When you tap on `theStatus` the `Tap` event will be fired (which is what causes the code in the event handler to be called). All events work in this simple fashion, but the number and type of events in Silverlight for Windows Phone vary widely.

Debugging in the Emulator

If clicking the user interface was not working the way we would like, it might help if we could stop the operation during an event to see what was happening during execution. We can do this by debugging our operation. We can use the debugger to set breakpoints and break in code while using the emulator. Place the text cursor inside the event handler and press F9 to create a breakpoint. When you run the application (again, press F5), you can see that when you click the `theStatus` `TextBlock` the debugger stops inside the event handler. You can hover your mouse over specific code elements (for example, `theStatus.Text`) to see the value in a pop-up (as shown in Figure 2.26).

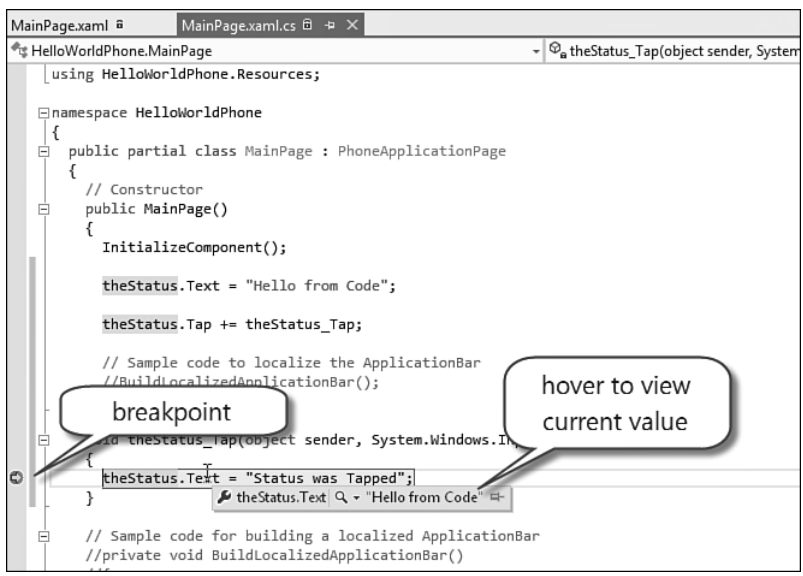


FIGURE 2.26 Using the Visual Studio debugger

Pressing the F5 key while stopped at a breakpoint will cause the application to continue running. There are other ways to walk through the code, but for now that should be sufficient to get you started. Using the emulator is the most common way you will develop your applications, but there are some interactions that are difficult to do with the emulator (for example, multitouch, using phone sensors, and so on) for which debugging directly on a device would be very useful. Luckily, debugging on the device is supported and works pretty easily.

Debugging with a Device

If you have a phone with which you want to do your development, you will need to be able to deploy and debug directly on the phone itself. First, you need to connect your phone to your development machine. All you need to do is connect your phone to your computer by a USB cable.

'Now that your device is connected, you can use it to browse to the directories for music, photos, and so on. However, before you can use a phone as a development device, you will need to register the phone for development. This lifts the requirements that applications be signed by Microsoft and allows you to deploy your applications directly to the phone so that you can debug applications.

Before you can enable your phone as a developer phone, you will need to have an account at the Windows Phone App Hub (http://developer.windowsphone.com). After you have done that, you can enable your phone to be used for development. To do this you will need the Windows Phone Developer Registration tool, which is installed when you install the Windows Phone SDK. When you run this application, it detects the device. You will need to ensure that the device is unlocked and turned on. At that point the Windows Phone Developer Registration tool will enable the Register button, as shown in Figure 2.27.

FIGURE 2.27 Windows Phone Developer Registration tool

Next, it will ask you for your Windows Live ID that you used to register with the developer portal, as shown in Figure 2.28.

If your phone is successfully attached to your computer, the Status area will tell you that it is ready to register your device for development. At this point, just click the Register button to register with the developer portal. After it registers the phone, it changes the status to show you that the phone is ready.

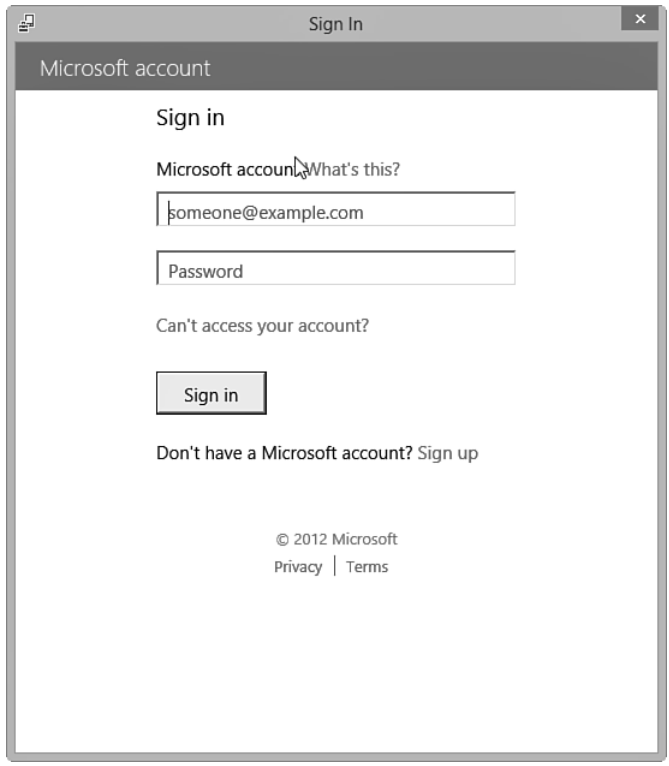


FIGURE 2.28 Signing in with your Microsoft ID

When you use a device to debug, you will find it much easier to change the default time-out of the device to be longer than the default (usually one minute, but it depends on the device manufacturer and carrier). To do this, go to the settings on your phone. In the settings is an option called "lock screen", as shown in Figure 2.29.

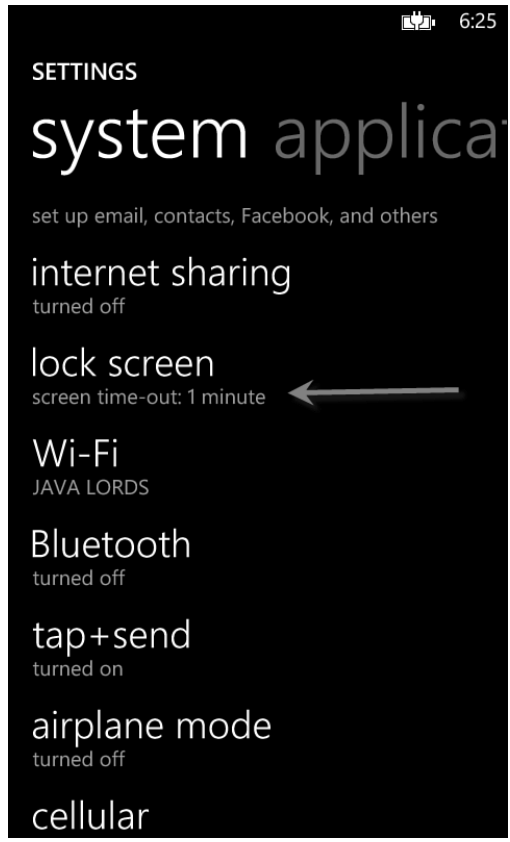


FIGURE 2.29 The Lock Screen option on the settings screen

When you're in this option, you can scroll down to find the "Screen times out after" option, open the option, and select the longest time-out you can tolerate (this will affect battery life when you're not debugging on the device, so be careful to choose a time-out you can live with if it's not a testing-only device). You can see this in Figure 2.30.

Now that you've registered your device, you can deploy and debug your applications using Visual Studio. The key to using the device instead of the emulator is to change the deployment using the drop-down list of deployment options. The drop-down is located in the toolbar of Visual Studio, as shown in Figure 2.31.

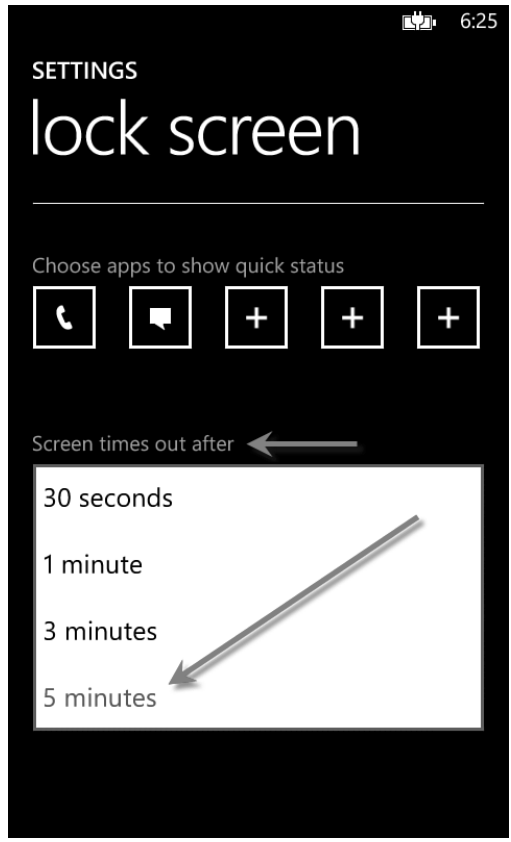


FIGURE 2.30 Changing the default device time-out

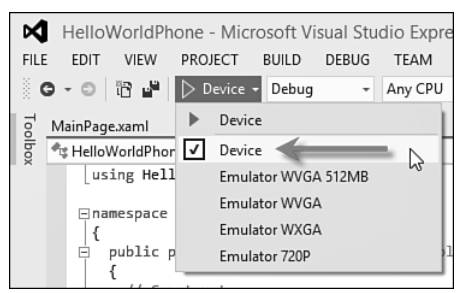


FIGURE 2.31 Changing the deployment to use a development phone

After you change the deployment target, you can debug just like you did with the emulator. When you run the application, it will deploy your application to the device and run it so that you can debug it in the same way as you did with the emulator.

Using Touch

Even though the touch interactions do fire mouse events, other events enable you to design your application for touch. Because touch is so important to how applications on the phone work, this first application should give you a taste of that experience. To show touch working, let's add an ellipse to the application that the user can move around by dragging it with her finger. To get started, you should open the MainPage.xaml file and add a new ellipse in the center of the page. To do this, find the `TextBlock` called `theStatus` and place a new `Ellipse` element after it, like so:

```
...
    <Grid x:Name="ContentGrid"
          Grid.Row="1">
      <Rectangle Fill="#FF7E0505"
                 Margin="8"
                 RadiusY="24"
                 RadiusX="24" />
      <TextBlock HorizontalAlignment="Center"
                 TextWrapping="Wrap"
                 Text="Status"
                 VerticalAlignment="Bottom"
                 FontSize="48"
                 FontWeight="Bold"
                 Name="theStatus" />
      <Ellipse x:Name="theEllipse"
               Fill="White"
               Width="200"
               Height="200">
      </Ellipse>
    </Grid>
...
```

We need to be able to move the ellipse (named `theEllipse`) as the user drags it. To allow us to do this, we must use something called a **transform**. In XAML, a transform is used to change the way an object is rendered without having to change properties of the ellipse. Although we could

change the margins and/or alignments to move it around the screen, using a transform is much simpler. You should use a `TranslateTransform` to allow this movement. A `TranslateTransform` provides `X` and `Y` properties, which specify where to draw the element (as a delta between where it originally exists and where you want it). You can specify this transform by setting the `RenderTransform` property with a `TranslateTransform` (naming it in the process):

```
...
<Ellipse x:Name="theEllipse"
         Fill="White"
         Width="200"
         Height="200">
  <Ellipse.RenderTransform>
    <TranslateTransform x:Name="theMover" />
  </Ellipse.RenderTransform>
</Ellipse>
...
```

Now that we have a way to move our ellipse around the page, let's look at dealing with touch. In Silverlight, there are two specific types of touch interactions that are meant to allow the user to change onscreen objects. These are when the user drags her finger on the screen and when she uses a pinch move to resize objects. These types of interactions are called **manipulations.** Silverlight has three events to allow you to use this touch information:

- `ManipulationStarted`
- `ManipulationDelta`
- `ManipulationCompleted`

These events let you get information about the manipulation as it happens. For example, let's handle the `ManipulationDelta` event to get information about when the user drags on the screen. This event is called as the manipulation happens, and it includes information about the difference between the start of the manipulation and the current state (for example, how far the user has dragged her finger):

```
...
public partial class MainPage : PhoneApplicationPage
{
  // Constructor
  public MainPage()
  {
    InitializeComponent();

    theStatus.Text = "Hello from Code";

    theStatus.Tap += theStatus_Tap;

    theEllipse.ManipulationDelta += theEllipse_ManipulationDelta;

    // Sample code to localize the ApplicationBar
    //BuildLocalizedApplicationBar();
  }

  void theEllipse_ManipulationDelta(object sender,
                                    System.Windows.Input.
                                    ManipulationDeltaEventArgs e)
  {
    // As a manipulation is executed (drag or resize), this is called
    theMover.X = e.CumulativeManipulation.Translation.X;
    theMover.Y = e.CumulativeManipulation.Translation.Y;
  }

  ...
}
...
```

The event is fired while the user either pinches or drags within the `theEllipse` element. In this case the code is only concerned with the dragging. In the event handler for `ManipulationDelta`, the `ManipulationDeltaEventArgs` object contains information about the extent of the manipulation. The `CumulativeManipulation` property of the event args has a property called `Translation`, which contains the extent of the drag operation (the complete delta). We are just changing `theMover`'s properties to match the manipulation. This means we can now drag the `theEllipse` element around and see it change position under our dragging, as shown in Figure 2.32.

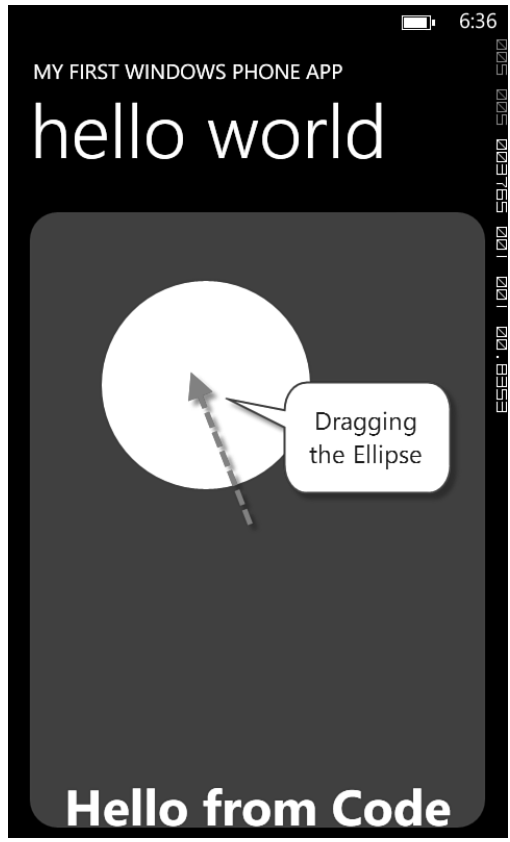


FIGURE 2.32 Dragging the ellipse

Working with the Phone

This first application is a program that can be pretty self-sufficient, but not all applications are like that. Most applications will want to interact with the phone's operating system to work with other parts of the phone. From within your application, you might want to make a phone call, interact with the user's contacts, take pictures, and so on. The Windows Phone SDK calls these types of interactions **tasks.** Tasks let you leave an application (and optionally return) to perform a number of phone-specific tasks. Here is a list of some of the most common tasks:

- CameraCaptureTask
- EmailAddressChooserTask
- EmailComposeTask
- PhoneCallTask
- SearchTask
- WebBrowserTask

These tasks allow you to launch a task for the user to perform. In some of these tasks (for example, CameraCaptureTask, EmailAddressChooserTask), after the task is complete the user expects to return to your application; while in others (for example, SearchTask), the user might be navigating to a new activity (and might come back via the Back key, but might not).

Let's start with a simple task, the SearchTask. Add a using statement to the top of the code file for Microsoft.Phone.Tasks to ensure that the SearchTask class is available to our code file. Next, create an event handler for the Tap event on theEllipse. Then, inside the handler for the Tap event, you can create an instance of the SearchTask, set the search criteria, and call Show to launch the task:

```
...
using Microsoft.Phone.Tasks;
...
public partial class MainPage : PhoneApplicationPage
{
  // Constructor
  public MainPage()
  {
    ...

    theEllipse.Tap += theEllipse_Tap;

  }

  void theEllipse_Tap(object sender,
                      System.Windows.Input.GestureEventArgs e)
  {
    SearchTask task = new SearchTask();
    task.SearchQuery = "Windows Phone";
    task.Show();
  }

  ...
}
```

If you run your application, you'll see that when you tap on the `theEllipse` element it will launch the phone's Search function using the search query you supplied (as shown in Figure 2.33). The results you retrieve for the search query can vary because it is using the live version of Bing for search.

FIGURE 2.33 The `SearchTask` in action

Although this sort of simple task is useful, the more interesting story is being able to call tasks that return to your application. For example, let's pick an email address from the phone and show it in our application. The big challenge here is that when we launch our application, we might get tombstoned (or deactivated). Remember that, on the phone, only one application can be running at a time. To have our task wired up when our application is activated (remember, it can be deactivated or even unloaded

if necessary), we have to have our task at the page or application level and wired up during construction. So, in our page, we create a class-level field and wire up the `Completed` event at the end of the constructor for it, like so:

```
public partial class MainPage : PhoneApplicationPage
{

  EmailAddressChooserTask emailChooser =
    new EmailAddressChooserTask();

  // Constructor
  public MainPage()
  {
    ...

    emailChooser.Completed += emailChooser_Completed;
  }

  ...
}
```

In the event handler, we can simply show the email chosen using the MessageBox API:

```
...
void emailChooser_Completed(object sender, EmailResult e)
{
  MessageBox.Show(e.Email);
}
...
```

Now we need to call the task. To do this, let's hijack the event that gets called when the `theEllipse` element is tapped. Just comment out the old `SearchTask` code and add a call to the `emailChooser`'s `Show` method, like so:

```
...
void theEllipse_MouseLeftButtonUp(object sender,
                                  MouseButtonEventArgs e)
{
  //SearchTask task = new SearchTask();
  //task.SearchQuery = "Windows Phone";
  //task.Show();

  // Get an e-mail from the user's Contacts
  emailChooser.Show();
}
...
```

After you run the application, a list of contacts will be displayed and you will be able to pick a contact (and an address, if there is more than one), as shown in Figure 2.34. The emulator comes prepopulated with several fake contacts to test with.

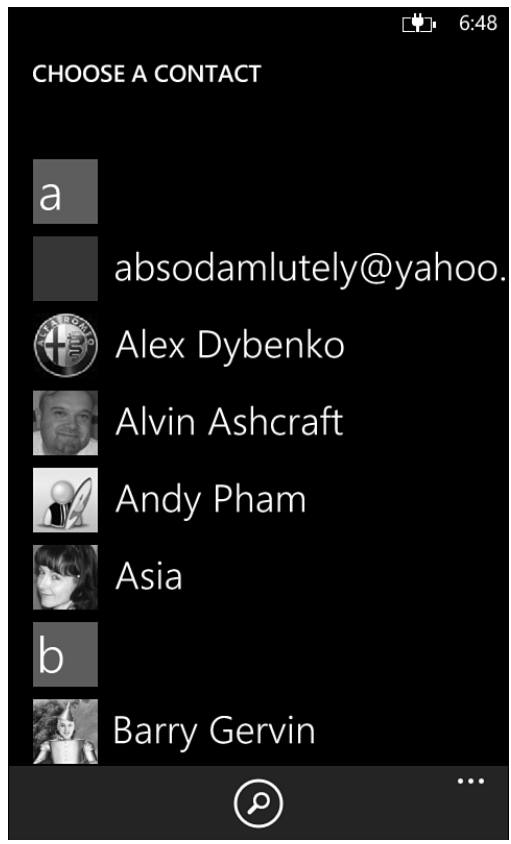


FIGURE 2.34 Choosing a contact to retrieve an email address via the `EmailAddressChooserTask`

After the user selects the contact, the phone returns to your application. You will be returned to your application (and debugging will continue). The event handler should be called when it is returned to the application, as shown in Figure 2.35.

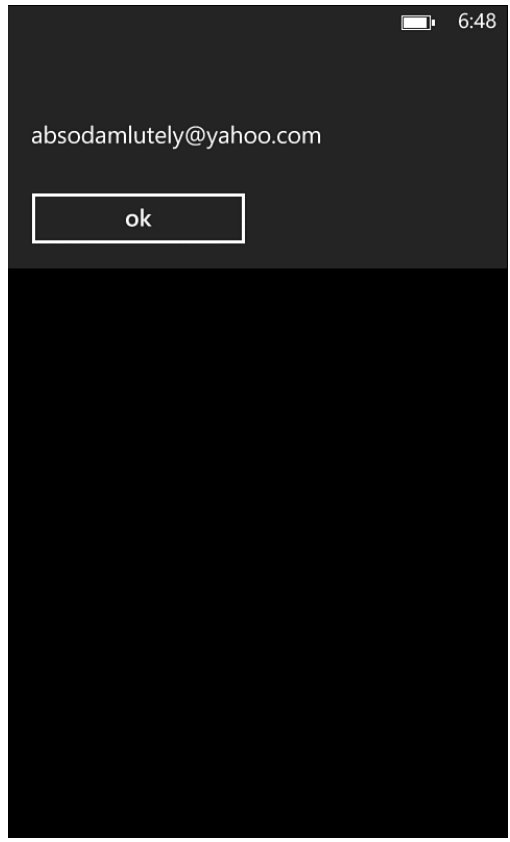


FIGURE 2.35 Showing the selected email in a `MessageBox`

Where Are We?

You have created your first Windows Phone application using Silverlight. Although this example has very little real functionality, you should have a feel for the environment. Taking this first stab at an application and applying it to your needs is probably not nearly enough. That's why, in subsequent chapters, we will dive deep into the different aspects of the phone application ecosystem. You will learn how to build rich applications using Silverlight, services, and the phone itself.

Index

B

C

D

E

F

G

H

I

M

N

O

P

Q

R

S

T

U

V

W

X-Y

Z